

PhD Dissertation

**Secure Multi-Party Computation Based on
 (k, n) Threshold Secret Sharing with
 $n < 2k - 1$ and Application into Searchable
Encryption**

$n < 2k - 1$ における (k, n) 閾値秘密分散法を用いた
秘密計算法及び秘匿検索への応用

MARCH 2022

AHMAD AKMAL AMINUDDIN BIN MOHD KAMAL

Department of Electrical Engineering
Tokyo University of Science

Supervisor:
Professor Keiichi IWAMURA

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Keiichi Iwamura, for his continuous support, advice, and patience during my doctoral studies at Tokyo University of Science. He has supported me patiently and extensively throughout the writing of this dissertation. His immense knowledge and expertise in information security technologies helped formulate the research questions, objectives, and methodology, and helped to bring my work to a higher level. He has been an ideal teacher, mentor, and supervisor, and I am very grateful for my time working with him.

I would also like to extend my sincere thanks to Associate Professor Masaki Inamura (Hiroshima City University) and Associate Professor Hyunho Kang (National Institute of Technology, Tokyo College) for their invaluable advice and support throughout my student life at Iwamura Laboratory. I am also profoundly grateful to the previous Assistant Professor of Iwamura Laboratory, Dr. Koya Sato, for all the advice and support. Their immense knowledge and great experience encouraged me in my doctoral studies and helped in my daily life in Japan.

Additionally, I would like to express my sincere gratitude to my dissertation committee members, Professor Takayuki Hamamoto (Tokyo University of Science), Professor Mikio Hasegawa (Tokyo University of Science), Professor Takahiro Yoshida (Tokyo University of Science), Professor Yukinobu Taniguchi (Tokyo University of Science) and Dr. Yuji Suga (Internet Initiative Japan, Inc.), not only for their precious time, but also for all the insightful comments and suggestions given during the process of writing and revising my dissertation.

Furthermore, I would like to express my sincere gratitude to Sato Yo International Scholarship Foundation (SISF) for supporting me throughout my academic research life. I would also like to offer my sincere thanks to the members of SISF for their continuous support and for their time reading and commenting on my monthly report over the past three years. The constant encouragement and belief in me given by SISF allowed me to concentrate on my research and complete this dissertation.

I am also grateful to all the members of Iwamura Laboratory, especially members of the Secure Computation Research Group, for their discussions and for their kindness to me, despite the fact that I was the only foreigner in the research group. Their warmheartedness allowed me to have a wonderful and memorable five years of research life in Iwamura Laboratory. I would also like to extend my gratitude to my seniors, Shingu Takeshi and Ken Aoi, for their part in completing this dissertation.

Finally, I would like to thank all of my friends and family members for encouraging and supporting me whenever I needed them. I am incredibly grateful, especially to my parents, for their wise counsel and sympathetic ear and for their regular encouragement at every step and decision in my life. Without their support, this dissertation would not have been possible.

Contents

Acknowledgements	iii
1 Introduction	1
1.1 Background of Research	1
1.2 Concept of Multiparty Computation	4
1.2.1 Introduction and Brief Timeline	4
1.2.2 MPC Use Cases	6
1.3 Techniques of Realizing MPC	9
1.4 Problem of MPC using (k, n) Threshold Secret Sharing	10
1.5 Objective and Contributions	12
1.6 Dissertation Outline	17
2 Basic Definitions and Building Blocks	21
2.1 Definitional Parameters in MPC	21
2.2 Finite Field	22
2.3 Computation in the Finite Field	23
2.4 Lagrange Interpolation	26
2.5 Secret Sharing	26
2.5.1 Additive Secret Sharing	27
2.5.2 (k, n) Threshold Secret Sharing	28
3 A Conditionally Secure MPC using (k, n) Threshold Secret Sharing	31
3.1 Introduction	31
3.2 Related Work: SPDZ Method	32
3.3 Proposed Method: TUS 1 Method	33
3.3.1 Overview of TUS 1 Method	33
3.3.2 Protocol of TUS 1 Method	33
3.3.3 Security of TUS 1 Method	36
3.4 Extension of TUS 1 Method	41
3.4.1 Many-Inputs Multiplication	42
3.4.2 Many-Inputs Addition/Subtraction	44
3.5 Limitation of the TUS 1 Method	45

3.6	Discussion	47
3.6.1	Computational and Communication Costs of TUS 1 Method	47
3.6.2	Qualitative Comparison with SPDZ Method	48
3.6.3	Quantitative Comparison with SPDZ Method	49
3.7	Chapter Summary	52
4	An Improved Conditionally Secure MPC	53
4.1	Introduction	53
4.2	Proposed Method: TUS 2 Method	54
4.2.1	Overview of TUS 2 Method	54
4.2.2	Protocol of TUS 2 Method	55
4.2.3	Security of TUS 2 method	57
4.3	Extension of TUS 2 Method: Combination of Multiple Product-Sum Operation	62
4.3.1	Extended Method	62
4.3.2	Security of Extended Method	63
4.4	Discussion	66
4.4.1	Qualitative Comparison with TUS 1 and SPDZ Methods	66
4.4.2	Quantitative Comparison with TUS 1 and SPDZ Methods	67
4.4.3	Discussion about Conditions	68
4.5	Chapter Summary	70
5	Application of MPC: Searchable Encryption of Documents	71
5.1	Introduction	71
5.2	Building Block: Overview of Secure MPC	74
5.3	Related Work	75
5.3.1	SE Using Symmetric Key Encryption	75
5.3.2	SE Using Public Key Encryption	76
5.3.3	SE Using Secret Sharing	77
5.4	Proposed Method: Conjunctive Search	77
5.4.1	Overview of Conjunctive Search	77
5.4.2	Protocol of Conjunctive Search (when $n = k$)	78
5.4.3	Security of Conjunctive Search	84
5.4.4	Extension of Conjunctive Search (when $n > k$)	89
5.5	Proposed Method: Conjunctive and Disjunctive Searches	90
5.5.1	Overview of Conjunctive and Disjunctive Searches	90
5.5.2	Protocol of Conjunctive and Disjunctive Searches (when $n = k$)	90
5.5.3	Security of Conjunctive and Disjunctive Searches	95
5.6	Discussion	97
5.6.1	Comparison with Conventional SEs	97

5.6.2	Adaptation of the Proposed Methods	99
5.6.3	Acceptable Information Leakage of SE	102
5.7	Chapter Summary	104
6	Multiplication of Polynomials with $N < 2k - 1$ Servers	105
6.1	Introduction	105
6.2	System Model and Adversary	107
6.3	Related Work	108
6.3.1	Two-party Multiplication Using Shamir's (k, n) Method	108
6.3.2	Multiplication of Shares Using the Recombination Vector	108
6.3.3	Watanabe et al.'s Method	109
6.4	Proposed Method: Multiplication of Two Polynomials (when $N = k$)	110
6.4.1	Overview of Proposed Method	110
6.4.2	Protocol of Proposed Method	111
6.4.3	Security of Proposed Method	113
6.5	Extension of Proposed Method (when $N > k$)	115
6.5.1	Protocol of the Extended Method	115
6.5.2	Security of the Extended Method	118
6.6	Limitation of the Proposed Method	119
6.7	Discussion	122
6.7.1	Computational and Communication Costs	122
6.7.2	Experimental Implementation of Proposed Method	124
6.7.3	Repetition of Multiplication	124
6.7.4	Comparison with Conventional Methods	128
6.8	Chapter Summary	130
7	Conclusion and Future Works	133
7.1	Conclusions	133
7.2	Future Works	135
A	List of Publications	137
A.1	Refereed Publication	137
B	Example of computation	139
B.1	Computation of protocols 6.2, 6.3 and 6.4	139
	References	143

List of Figures

1.1	Basic example of secure computation	3
1.2	Example of secure computation based on additive secret sharing	11
1.3	Our construction of client-server model MPC with using secret sharing	12
1.4	Problem of multiplication in MPC based on secret sharing	13
1.5	Scalar multiplication with an encrypted secret input αa	15
1.6	Dissertation outline	19
2.1	Addition operation in a finite field	23
2.2	A secret map can be divided into smaller pieces to be distributed and stored	27
4.1	Basic computation of TUS 1 method	58
4.2	Basic computation of TUS 2 method	58
4.3	Computation of a by combining multiple product-sum operations	64
4.4	General computation of a	65

List of Tables

1.1	Methods for realizing secure computation	9
1.2	Time taken to sort 20 bit 1,000,000 data as simulated in [44]	10
3.1	Communication cost and rounds of the TUS 1 method	48
3.2	Computational cost of the TUS 1 method	48
3.3	Comparison with the SPDZ method	50
3.4	Comparison with the SPDZ method (computational cost)	52
3.5	Comparison with the SPDZ method (communication cost)	52
3.6	Comparison with the SPDZ method (round)	52
4.1	Comparison with previous works	67
4.2	Comparison with previous works (computation cost)	68
4.3	Comparison with previous works (communication cost)	69
4.4	Comparison with previous works (round)	69
5.1	Example of ASCII code and corresponding characters	75
6.1	Communication and number of rounds of the proposed method (when $N = k$)	122
6.2	Communication and number of rounds of the extended method (when $N > k$)	123
6.3	Computational cost of proposed method (when $N = k$)	123
6.4	Computational cost of the extended method (when $N > k$)	123
6.5	Computational time of proposed method when $N = k = 2$ (for m multiplications)	124
6.6	Comparison with conventional methods (Computational cost)	130
6.7	Comparison with conventional methods (Communication cost)	131
6.8	Comparison with conventional methods (rounds)	131

List of Abbreviations

MPC	MultiParty Computation
FHE	Fully Homomorphic Encryption
SHE	Somewhat Homomprphic Encryption
MAC	Message Authentication Code
SE	Searchable Encryption
SSE	Symmetric Searchable Encryption
PKE	Public Key Encryption
ASCII	American Standard Code (for) Information Interchange
IoT	Internet of Things
TUS	Tokyo University (of) Science
TTP	Trusted Third Party
HE	Homomorphic Encryption
RSA	Rivest Shamir Adleman
NSS	Nearest Neighbor Search

Chapter 1

Introduction

This dissertation describes comprehensive studies on the technology of secure computation, particularly **secure multi-party computation (MPC) using (k, n) threshold secret sharing**, which can realize all four arithmetic operations with $n < 2k - 1$ participating computing servers while achieving information-theoretic security against a semi-honest adversary.

In addition, this dissertation also includes studies on utilizing the proposed secure computation to realize **searchable encryption (SE) of documents**, which allows users to outsource the storage of their data (documents) to online cloud storage in a private manner while maintaining the ability to selectively perform searches.

This chapter first introduces the research background, introduction, and use case of secure MPC. In sequence, the challenges in conventional methods of secure MPC using (k, n) threshold secret sharing are described. Finally, the research motivations, contributions, and the organization of this dissertation are detailed.

1.1 Background of Research

In recent years, innovations in information and communication technology and the advancement of technology for the collection of data such as big data and the Internet of Things (IoT) have enabled us to collect and analyze large amounts of diverse data. Furthermore, technologies that can use an individual's personal information embedded in the big data for value creation have been anticipated. In particular, various efforts have been recently made to enable the use of personal data for solving various social problems (i.e., reducing the wage gap between genders [1]) and continuously develop new values and services (i.e., better health-care service [2]).

Google, Apple, Facebook, and Amazon (also known as "GAFA"), which are now the four most valuable and dominant public tech companies in the world by market population, have fully utilized personal data collected from their users to enhance the speed and precision of searches and improve recommendation and matching features such as personalized

advertisements for their users [3]. In 2020, with the worldwide outbreak of the COVID-19 pandemic, researchers focused on developing applications (apps) for “contact tracing” to identify individuals who had close contact with positive carriers, which required data such as the location of an individual for the past 14–21 days. The introduction of these apps enabled the automation of the contact tracing process and quick and reliable identification of contacts with significant infection risk [4].

However, collecting a large amount of personal data also introduces a new concern, particularly on the management, security, and privacy of these data. Moreover, the concentration of personal data in a centralized ecosystem also becomes an attractive target for cyberattacks, raising the users’ risk and concern that their data might be leaked and misused. For example, with the introduction of the “contact tracing” apps, there was also a considerable debate about the privacy of users’ information, as the user’s location information or identity could be revealed without the user permission, or the collected data could be used to track the users even after the end of the pandemic [5] [6].

In the latest amended Act on the Protection of Personal Information [7], which aims to protect an individual’s rights and interests while considering the utility of personal information to the creation of new industries for a better quality of life for the people of Japan, the term “personal information” refers to the information about a living individual in Japan whose identity can be identified. This includes some of the most obvious information, such as the name and date of birth. In this dissertation, we believe that the definition of “personal information” is not limited to information about private or family life but extends to any information or opinion about the individual from which they can be identified. For example, information such as the history of online purchases using a credit card or web browsing history can also be included as personal information as it can be used to infer or produce an opinion about a person, such as their tastes and preferences.

Therefore, in the age where data collection is essential to improve the life quality, the trade-off between the use of data and the increase in the risk of personal information leakage should be reduced. In other words, there is a need for a technology that allows data to be used without infringing users’ privacy. One of the available technologies that to address this issue is called *secure computation*, wherein processes such as statistical processing and other types of data processing can be performed on data that remains in a secure (or encrypted) form.

For example, let us consider the situation in Figure 1.1, in which an institution (School A) wants to store its records of academic results into a database. In this case, to protect the students’ personal information, including the students’ names and IDs, all data are encrypted before being sent to the database. However, conventional encryption methods prohibit any form of data processing on encrypted data. Thus, if an analyst needs the average score of Subject A from the data stored in the database, the database must first decrypt all the data, compute the average result, and send it to the analyst, which creates a risk for data leakage

during the computation phase.

When using a secure computation, the analyst first sends its request to the database (in this case, to find the average score of Subject A), as illustrated in Figure 1.1. In sequence, the database will perform secure computation on the encrypted data and send the correct result back to the analyst without decrypting the data. Thus, the concept of secure computation has attracted considerable attention as a technological solution for data analysis while maintaining privacy.

Secure computation allows a group of parties who do not trust each other to jointly compute an arbitrary function that depends on their private inputs without revealing these inputs to each other [8]. In this dissertation, we performed a study on the secure computation technology that relies on the interworking of multiple computing servers (also called computing parties) that communicate with each other. Nowadays, this is a common scenario with the widespread of distributed computing, where a number of connected computing servers who hold a distributed database system are required to conduct a joint computation of a type of function on the database in a secure manner. Here, the computing servers perform the processing while exchanging the anonymized data among themselves [9]. This type of secure computation is also known as secure MPC. In this study, the notions of secure computation and MPC are considered as synonyms as they significantly overlap.

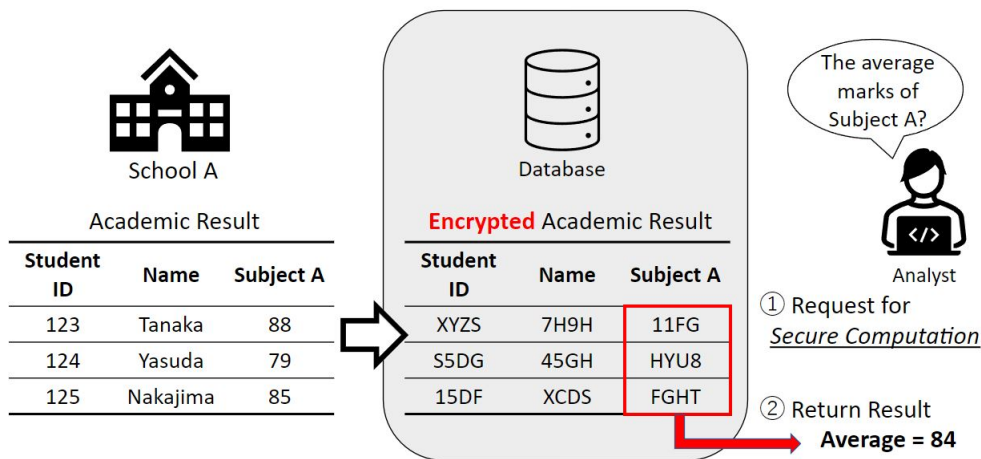


FIGURE 1.1: Basic example of secure computation.

1.2 Concept of Multiparty Computation

1.2.1 Introduction and Brief Timeline

Andrew Yao first proposed the concept of MPC in 1982 to solve the following millionaire problem [10]:

“Two millionaires wish to know who is richer; however, they do not want to find out any additional information about each other’s wealth.”

Yao introduced the first general notion of MPC, where m parties wish to compute the value of a function $f(x_1, x_2, \dots, x_m)$, which is an integer-valued function of m integer variables x_i of bounded range [10]. Here, the millionaires’ problem corresponds to the case of $m = 2$ parties P_1 and P_2 , each with inputs (wealth) x_1 and x_2 , who wish to realize the following secure computation of $f(x_1, x_2)$:

$$f(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 < x_2 \\ 1 & \text{otherwise} \end{cases}$$

In 1982, Yao described manners of solving the problem of secure computation by using one-way functions (i.e., functions that are easy to evaluate but difficult to invert). In particular, one of the methods proposed by Yao uses the approach of “garbled circuit” to perform secure computation between two parties (to solve the aforementioned millionaire problem). A garbled circuit is a method to encrypt a computation by taking the desired computation and expressing it as a circuit, where the computation will only reveal the output of the computation and no information about the inputs or intermediate values [11]. The use of garbled circuits enables the protocol to have a constant number of rounds of communication, but they are large and costly in bandwidth compared to other techniques proposed afterward.

In 1988, Ben-Or et al. [12] proposed a method of secure MPC using Shamir’s (k, n) threshold secret sharing [13], where a secret input was divided into n different values (known as shares) and distributed to n computing servers. Here, the authors also introduced the limitation and theorem for secure MPC using (k, n) threshold secret sharing, in which secure MPC under the setting of $n < 2k - 1$ is impossible due to the increase in the polynomial degree of the multiplication result (explained in detail in Section 1.4). Here, k is the threshold for the number of shares required to reconstruct the original secret input, and n is the number of shares generated from the (k, n) threshold secret sharing. Because of this limitation, all secure MPC that uses (k, n) threshold secret sharing must assume $n \geq 2k - 1$. For example, if the threshold is $k = 2$, the minimum number of shares n must be $n = 3$, and as each computing server holds one share, three will be required. In other words, the most significant disadvantage of a secure MPC using (k, n) threshold secret sharing is that it must be composed of at least three computing servers, which increases the overall initial cost of setting up the system.

In 2004, Malkhi et al. introduced Fairplay [14], the first reported full-fledged system that implements Yao's garbled circuit. This system was designed to present the first evaluation of an overall secure computation in real-world settings, examine its components, and identify potential bottlenecks. Since the introduction of the concept of secure computation, it had not been applied in practice and was typically considered to have theoretical significance. This motivated Malkhi et al. to develop a system that allowed two participants to implement a joint computation without any trusted third party (TTP). However, this system only implemented secure computation between two parties and could not handle MPC. Moreover, the execution time for secure computation was very slow, mainly due to the invocations of the oblivious transfer protocol.

In 2008, Bogdanov et al. introduced Sharemind [15], a virtual machine for privacy-preserving data processing that relies on additive secret sharing (refer to Section 2.5 for further details) technique using three computing servers. Sharemind framework uses an additive secret sharing and supports addition, multiplication, and greater-than-or-equal comparison of two shared values. The choice of using additive secret sharing in this implementation instead of the garbled circuit made the Sharemind almost 1,000 times faster than the Fairplay introduced by Malkhi et al. in 2004 [14]. Moreover, in the first implementation of Sharemind, the most efficient setting was chosen, where only one of three computing servers could be corrupted. However, as the multiplication operation used in this implementation requires a sub-protocol of reduction method based on Du-Atallah protocol [16] where all three computing servers are needed, if one of the computing servers fails, secure computation will no longer be possible.

In 2009, Gentry introduced a secure computation method known as fully homomorphic encryption (FHE) [17]. Since the first introduction of secure computation by Yao et al. [11], most of the methods had utilized more than one computing server to realize secure computation between multiple clients. In contrast, the FHE introduced by Gentry was the first known method to use only one computing server to perform secure computation, being able to compute all operations, including multiplication. This immediately solved the problem of secure MPC using (k, n) threshold secret sharing where at least three computing servers are required. However, FHE requires a considerable computational overhead to realize computation such as multiplication, particularly when compared with MPC based on (k, n) threshold secret sharing. Therefore, FHE is not suitable for use in applications with large amounts of data, such as the statistical analysis of the genome [18].

In 2012, Damgård et al. introduced the first SPDZ (pronounced as "speedz") method [19], a secure MPC using additive secret sharing that realizes multiplication with the parameter $n = k$, where the protocol is secure against up to $n - 1$ corrupted computing servers. In 1988, Ben-Or et al. [12] had proposed a theorem stating that an unconditionally secure MPC using secret sharing is impossible when $n < 2k - 1$ (where more than half of the computing servers are corrupted). The SPDZ method bypassed this by using computational assumptions

to construct a secure MPC against a dishonest majority adversary (where all but one of the servers are corrupted by the adversary). This method works by introducing a preprocessing phase to supply the required raw material for the actual computation without knowing the function to be computed or the inputs. The preprocessing phase is based on the circuit randomization technique of Beaver [20] to produce shares of a , b , and c , where $c = ab$, using a somewhat homomorphic encryption (SHE), which is an encryption technique that can perform only a limited number of addition and multiplication operations [21]. This method only achieves computational security due to the use of SHE, and the computational overheads for computing shares of a , b , and c are very large.

In 2013, Hamada et al. introduced a secure MPC system MEVAL (**m**ulti-**p**arty **e**valuator) for statistical analysis [22]. This system was built using Shamir's $(2, 3)$ threshold secret sharing with three computing servers and can tolerate one corrupted server. Moreover, this method is secure against a semi-honest adversary. However, as the multiplication is based on Gennaro et al.'s method of multiplication [23], the same limitation of $n \geq 2k - 1$ remains. Therefore, this system requires a minimum of three computing servers to securely perform multiplication operation. In addition, as MEVAL was set up with parameters $n = 3$, $k = 2$, it cannot be used for other settings of n, k .

In 2016, Araki et al. introduced a new secure MPC using Shamir's $(2, 3)$ threshold secret sharing with $n = 3$, $k = 2$ [24]. This protocol could realize a fast computation with minimal communication required. In particular, the multiplication protocol in this method only requires one time of communication between the computing servers. During multiplication, the protocol assumes that the three computing servers are given correlated randomness α , β , γ where $\alpha + \beta + \gamma = 0$. This correlated randomness can be prepared beforehand by the three computing servers. However, as the protocol was specifically designed for only three computing servers with at most one corrupted server, the protocol does not work for any number of computing servers.

1.2.2 MPC Use Cases

Since the introduction of the MPC concept in 1982 [11], there have been many theoretical examples of the possible uses of MPC. The potential applications of MPC in privacy-preserving computations are enormous. For instance, MPC can be used to perform a double auction in a privacy-preserving manner, perform statistic computation revealing only the aggregate result, realize secure machine learning, etc.

Until very recently, most of these examples were only in theory. However, since the introduction of Sharemind in 2008, MPC has transitioned from a theoretical study in the 1990s to be used in multiple real-world use cases. In this section, some examples of MPC applications that have been deployed are detailed.

- **The Danish Sugar Beet Auctions [25]**

This is the first real-world, large-scale application of secure MPC using secret sharing. In Denmark, around 5,000 Danish farmers produce sugar beets, which are sold to Danisco, the only Danish sugar beets processor. Farmers have their own contracts that give them the production rights, allowing for a certain amount of beets production. However, as the European Union (EU) significantly reduced subsidies and Danisco closed one of its factories, there was an immediate need for a nationwide market for trading production rights. The auction's goal was to find the market clearing price (MCP), which is the price per unit of the commodity being traded [25][26]. However, as the MCP is computed based on sealed bids, there was a need to decide who would be the "trusted" auctioneer. As the bids by the farmer reveal information such as the farmer's economic position, selecting Danisco as the auctioneer would cause concerns of might of information.

In contrast, trusting the auction to a consultancy house would cost a considerable amount of money. In the end, it was decided that three-party computation would be used, where the role of the auctioneer would be changed to a virtual auctioneer and played by all three parties: the company Danisco, the association of sugar beet growers, and the researcher from the Secure Information Management and Processing (SIMAP) project. The use of MPC was crucial in this case because there were multiple parties with conflicting interests, and MPC could ensure that no single party would have access to the sensitive information of the bids. This project in 2008 had led to the formation of a new company named Partisia [27], which uses MPC to support auctions for industries.

- **Boston Gender/Racial Wage Gap Study**

The Boston Women's Workforce Council (BWWC) started to use MPC in 2016 to measure the gender/racial wage gap for the Greater Boston area [28]. BWWC believes that in a city where women represent more than half the workforce, equity is crucial, and the first step to realize this is to measure it. However, many companies could not provide their raw data to the BWWC due to privacy concerns. Therefore, to ensure the privacy and security of data of over 250 Boston-area employers that had signed the 100% Talent Compact, BWWC partnered with Hariri Institute for Computing of Boston University to implement secure MPC in the wage gap analysis process [29].

Employers submitted their wage data through a web-based software program that employed encryption using secure MPC during the submission process. Using MPC, individual compensation data did not leave the server of each organization, allowing the BWWC to receive the aggregate data unconnected to any firm. In the 2019 Boston Wage Gap Report, BWWC analyzed data from 125 companies and 140,000 employees, representing 12.2 billion dollars in annual earnings, and found that, in 2018, the women workforce in Boston, on average, only earned 70 cents to a man's dollar [28]. This is an example of the use of MPC to solve social problems.

- **Cryptographic Key Management**

Another use-case of MPC is in the protection and management of cryptographic keys. Typically, a cryptographic key is handled by a single individual or organization. This poses a risk to the security of the cryptographic key, as it may be leaked if the individual is attacked. Therefore, rather than trusting a single individual, secure MPC allows the cryptographic key to be split into multiple pieces and placed on different servers and devices, such that an attacker would have to breach them all to steal the key, therefore enhancing the security [9]. Moreover, secure MPC also enables the use of cryptographic keys for any operations without the need to reconstruct them in a single place.

One of the companies that provide this service is Unbound, which offers solutions for cryptographic key management by using secure MPC [30]. In this application, a single organization uses MPC to generate split keys, and the key shares are placed in separate locations such as on-premises and cloud servers. Moreover, through the use of threshold cryptography [31], all cryptographic computations, including decryption, authentication, and digital signing, can be performed between multiple servers without revealing their respective shares of the cryptographic key. Thus, cryptographic keys are never united or exposed in a single place, even when the key is in use.

- **Privacy-preserving Analysis in Medical Research – Genome Information Analysis**

Another important application of secure MPC is in medical research, particularly genome information analysis. Genome information analysis includes identifying, measuring, or comparing genome features such as DNA sequence, structural variation, gene expression, or regulatory and functional element annotation. This analysis is essential to elucidate the relationship between genomes and diseases to develop effective medicines based on a person's genome information [32]. However, genome data involves the personal information of an individual. Unlike information such as passwords or email, genome information cannot be changed and will remain the same for the entire life. Leaking of such genome information will reveal information such as the susceptibility to a certain disease, which can be misused against the individual. Therefore, a method for securely sharing and analyzing genome data is required.

In 2019, NEC corporation collaborated with the research group of Professor Akihiro Nakaya from the Graduate School of Medicine at Osaka University to demonstrate the practicality of using secure MPC in genome analysis of genome data from multiple medical institutes without revealing the actual content of genome information. The results indicated that the genome information of approximately 8,000 individuals could be analyzed in approximately 1 s [18], proving the practical use of MPC in applications involving a huge amount of data. In the future, this MPC application will enable researchers to collect and analyze

TABLE 1.1: Methods for realizing secure computation

Approach	FHE	Secret Sharing
Confidentiality of information	Confidentiality is ensured by encrypting with an encryption key	Confidentiality is ensured by dividing the data and physically dividing the computational space
Advantage	Requires one computing server, no communication is required	Low computational overhead
Disadvantage	Large computational overhead	Requires multiple computing servers, communication between servers is required

large volumes of genome information that was previously limited to their respective medical institutions.

1.3 Techniques of Realizing MPC

In Section 1.2, we introduced the concept of MPC and showed its transition from a theoretical idea to real-world applications. In this section, we will explain several techniques for constructing MPC. Generally, there are several methods to realize MPC, such as using a garbled circuit, homomorphic encryption, oblivious transfer, and secret sharing. Each of these techniques will produce MPC with different properties for different settings and applications.

Two of the most commonly used approaches are homomorphic encryption (HE) [33] [34] [19] [35] [17] [36] and secret sharing [12] [15] [37] [38] [39] [16] [23] [40] [41] [42] [43]. However, HE often relies on complex processes and is very expensive in terms of computational cost, requiring a considerably longer computation time. For example, as shown in Table 1.2, in a simulation performed by NTT researchers comparing the efficiency of FHE, secret sharing, and other techniques, it was demonstrated that MPC using FHE was almost 50,000 times slower than using secret sharing to compute the same process [44]. Therefore, approaches with lower computational costs are preferable to FHE when considering the utilization of MPC into cloud computing to process a large amount of data such as big data.

However, as conventional MPC using secret sharing requires multiple computing servers to perform secure computation, multiple communications between multiple computing servers are needed during the computation process. In contrast, FHE does not require communication because it can perform secure computation using a single computing server. This comparison is summarized in Table 1.1.

In this dissertation, to realize a secure MPC with a faster processing speed, we focused on using secret sharing instead of FHE. Let us consider the previous situation of Figure 1.1, in which data are encrypted using the simplest form of secret sharing known as additive secret

TABLE 1.2: Time taken to sort 20 bit 1,000,000 data as simulated in [44]

Method used for secure computation	Processing time [s]
Secret sharing	1.1
Garbled circuit	243
Homomorphic encryption	3,564
FHE	48,877

sharing (where the secret input is broken into fragments that add up to the original secret input).

For example, the analyst wants to obtain the average score for Subject A from the encrypted database. First, School A will distribute the test score to the database by using additive secret sharing. Specifically, as shown in Figure 1.2, when there are $N = 2$ computing servers, each score is split into $n = 2$ different shares, whose sums are equal to the original score (for example, score ‘88’ is divided into ‘40’ and ‘48’), and sent to each computing server (e.g., Database 1 and 2). In sequence, each database computes the average score from the received values and send to the analyst. To obtain the final result of the average score of Subject A, the analyst only needs to add the two values, which result in the score ‘84’ (which is equal to the average score of Subject A). By using secret sharing to divide the original scores, the values seen by each database are entirely unrelated to the actual scores of each student, and the analyst only learns about the final result. Therefore, the privacy and security of information of each student is preserved.

The implementation depicted in Figure 1.2 is an example of the most basic use of secure MPC using secret sharing. In this scenario, MPC can be translated as a cryptographic protocol that distributes computation across multiple computing servers (in this case, Databases 1 and 2), where no individual server will be able to see the actual data, enabling the analysis of data without compromising privacy.

In this dissertation, we utilized a secure MPC using secret sharing. However, instead of the simplest form of additive secret sharing previously mentioned, we opted for the (k, n) threshold secret sharing, which allows us to realize a more flexible computation method and provides resistance toward server loss. The exact definitional parameters of MPC, variations of secret sharing, and its security requirements will be discussed in detail in Chapter 2.

1.4 Problem of MPC using (k, n) Threshold Secret Sharing

MPC is a method that enables sensitive data to be brought together securely and helps achieving the dual goal of putting data to beneficial use while also avoiding misuse. As described in Section 1.3, many different techniques have been developed for realizing MPC with different

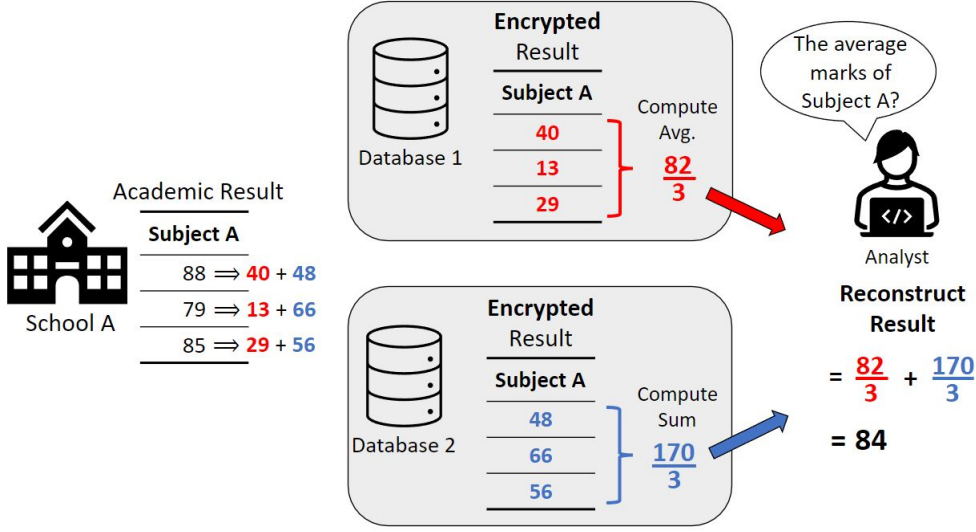


FIGURE 1.2: Example of secure computation based on additive secret sharing.

properties and settings, such as FHE and secret sharing. However, FHE requires more computational overhead, whereas secret sharing has a relatively low computational cost. Therefore, secret sharing is preferable in a cloud system environment with multiple users.

In this study, we focused on the client-server model of MPC using (k, n) threshold secret sharing as shown in Figure 1.3. In this model, we assumed a situation where the clients (other than computing servers, such as the owner of secret inputs) generate and distribute shares of their secret inputs to multiple computing servers using (k, n) threshold secret sharing. The computing servers jointly compute the desired function using MPC and return the results without learning the secret inputs. This model has been widely used in many cloud computing environments and is the business model used in Cybernetica [15][45].

An example of (k, n) threshold secret sharing is that of Shamir. To share a secret input s , the dealer chooses a random polynomial $f(x)$ of degree $(k - 1)$ under the constraint that $f(0) = s$ as follows:

$$f(x) = s + \alpha_1 x + \alpha_2 x^2 + \dots + \alpha_{k-1} x^{k-1}$$

For every $i = 0, 1, \dots, n - 1$ computing servers, the dealer provides the i th server with the share $f(i + 1)$. As the polynomial has a degree $(k - 1)$, reconstruction is possible with a subset of any threshold k or more shares. Reconstruction works by interpolating the polynomial to compute $f(x)$ and deriving the secret input $s = f(0)$. However, any subset of $k - 1$ or fewer shares reveal no information about the secret input as they have $k - 1$ or fewer points on the polynomial. We will discuss the formal definitions and protocol for secret sharing in Chapter

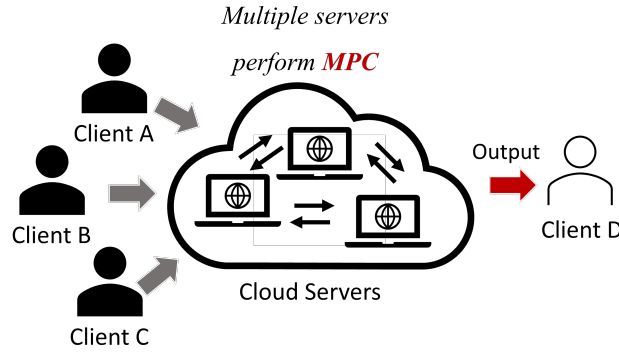


FIGURE 1.3: Our construction of client-server model MPC with using secret sharing.

2.

The classical result of secure MPC using (k, n) threshold secret sharing is that n computing servers can compute any arbitrary function such that any subset of up to $k - 1 < n/2$ computing servers obtains no information about the inputs of other servers, except for what can be derived from public information [12]. Conventional methods of secure MPC using Shamir's (k, n) threshold secret sharing perform addition by locally adding shares together. However, this is not the case for multiplication.

For example, let secret inputs a and b be encoded by $f(x)$ and $g(x)$, two polynomials of degrees $(k - 1)$. As shown in Figure 1.4, multiplying $f(x)$ and $g(x)$ will result in polynomial $h(x)$, whose free coefficient is ab .

However, by using $h(x)$ to encode the product of $a \times b$, the degree of $h(x)$ increases from $(k - 1)$ to $(2k - 2)$. In most conventional methods, this poses no problem for interpolating $h(x)$ from its n shares because it is assumed that $n \geq 2k - 1$. Each server holds only one share for each secret; hence, for each multiplication performed, the number of required servers increases from k to $2k - 1$.

Therefore, the construction of information-theoretically secure multiplication under the dishonest majority setting ($k - 1 \geq n/2$) is considered impossible. Some possible manners to avoid this impossibility result are: (1) giving up information-theoretic security [19] [35] (typically, generating a Beaver triple [20] between computing parties via computationally secure primitives such as SHE or oblivious transfer); and (2) assuming a trusted setup by other than computing parties.

1.5 Objective and Contributions

In a secure MPC using (k, n) threshold secret sharing, the result of the multiplication of two polynomials of degree $(k - 1)$ is a polynomial of degree $(2k - 2)$. Therefore, unconditionally

$$\begin{array}{l}
 f(x) = \mathbf{a} + a_1x_i + \dots + a_{k-1}x_i^{k-1} \\
 \times g(x) = \mathbf{b} + b_1x_i + \dots + b_{k-1}x_i^{k-1} \\
 \hline
 h(x) = \mathbf{ab} + \dots + (a_{k-1}b_{k-1})x_i^{2k-2}
 \end{array}$$



 Degree change
from
(k-1) to (2k-2)

FIGURE 1.4: Problem of multiplication in MPC based on secret sharing.

secure MPC using (k, n) threshold secret sharing with fairness and guaranteed output delivery can be achieved for any function only when $n \geq 2k - 1$ (i.e., an honest majority). Thus, for each multiplication performed, the number of shares n required will increase, and as each server only holds one share, the total cost for setting up the required servers N will also increase from $N \geq k$ to $N \geq 2k - 1$. One of the most utilized approaches to overcome this impossibility result is to give up information-theoretic security by implementing Beaver triple [20] during multiplication, such as in the SPDZ method [19]. However, this will result in a weaker security guarantee achieved by the protocol.

As previously mentioned, unconditionally secure MPC is only possible when $n \geq 2k - 1$ is assumed. As each server typically holds one share, the number of servers N needed for realizing unconditionally secure MPC will also be $N \geq 2k - 1$. In contrast, the phrase “*unconditionally secure MPC is only possible when $n \geq 2k - 1$* ” also means a secure MPC when $n < 2k - 1$ is possible with certain *conditions*.

Therefore, the main objective of this dissertation was to determine the conditions required to overcome the impossibility result and realize a conditionally secure MPC using Shamir’s (k, n) threshold secret sharing with $n < 2k - 1$, while maintaining the information-theoretic security nature of (k, n) threshold secret sharing.

There are several approaches to realize multiplication in MPC using (k, n) threshold secret sharing with $n < 2k - 1$ such as (1) scalar multiplication and (2) introduction of degree reduction. In this study, we first explored the use of scalar multiplication to realize secure MPC using (k, n) threshold secret sharing with $n < 2k - 1$. In particular, we multiplied a polynomial with a scalar value to prevent the increase of polynomial degrees of the multiplication result. Moreover, as unconditionally secure computation is impossible when $n < 2k - 1$, we also studied the conditions required to realize MPC with $n < 2k - 1$, showing that it can be used in real-world use cases such as in SE of encrypted documents even with certain conditions required. Finally, in the latter half of this dissertation, we explored the approach of introducing the degree reduction sub-protocol to the resulting polynomial to realize multiplication with $n < 2k - 1$. Real-world use case and applications of this approach will be addressed in future studies.

This dissertation is divided into three parts as follows:

Part I: Chapters 3 and 4

In the first part of this dissertation, we focused on realizing a conditionally secure MPC using (k, n) threshold secret sharing when $n < 2k - 1$, while keeping the information-theoretic security nature of (k, n) threshold secret sharing. The contribution of this part is as follows:

- **First conditionally secure MPC when $n < 2k - 1$ with information-theoretic security.**


The easiest method of realizing the multiplication of shares represented by polynomials without increasing the polynomial degree of the multiplication result is by using the *scalar multiplication* approach. That is, instead of the typical process of (polynomial \times polynomial) as shown in Figure 1.4, multiplication is performed by computing (polynomial \times scalar value), where one of the secret input is reconstructed momentarily and multiplied with the other polynomial. Conventional methods such as the SPDZ method proposed by Damgård et al. [19] also use this approach to compute multiplication without increasing the polynomial degree of the multiplication result. However, most conventional methods only achieve computational security instead of information-theoretical security and are only secure in specific limited parameters of k and n .

In this dissertation, we propose the first solution for multiplication that can achieve information-theoretic security even when $n < 2k - 1$. As previously mentioned, to realize multiplication operation through scalar multiplication, one of the secret inputs must be reconstructed as a scalar value. However, if the original secret input (e.g., input a) is reconstructed by the computing servers, the value of the secret input a will be leaked. Therefore, in the proposed method, instead of the “normal share” of (k, n) threshold secret sharing, we introduce a new functionality of “*encrypted share*” where the share is generated from the secret input that had been encrypted with a random number. For example, the secret input (e.g., input a) is first encrypted with a random number (e.g., α) to produce an encrypted secret (e.g., αa) and distributed to $N = n$ computing servers. During multiplication, the encrypted secret is momentarily restored as a scalar value (e.g., αa) and multiplied with the polynomial of secret input b (as shown in Figure 1.5). As secret input a is encrypted with a random number α , information of secret input a will not be leaked even if αa is known or made public.

Moreover, as stated in [12], unconditionally secure MPC is considered impossible under the setting of $n < 2k - 1$. Therefore, in this dissertation, we also considered the *conditions* needed to achieve information-theoretically secure MPC using (k, n) threshold secret sharing with $n < 2k - 1$, even when computation involving a combination of different types of operations such as $f(a, b, c) = ab + c$ is performed.

Advantages of secure MPC with $n < 2k - 1$

There are two significant advantages of realizing multiplication with parameter $n < 2k - 1$. The first advantage is that it is unnecessary to increase the number of computing servers for MPC each time a multiplication operation is performed, and the parameters n and k can

$$\begin{array}{r}
 \alpha a \\
 \times g(x) = \beta(b + b_1x_i + \dots + b_{k-1}x_i^{k-1}) \\
 \hline
 h(x) = \alpha\beta a(b + b_1x_i + \dots + b_{k-1}x_i^{k-1})
 \end{array}$$


Degree remains
at (k-1)

FIGURE 1.5: Scalar multiplication with an encrypted secret input αa .

be selected to achieve the optimum performance. In general, parameter k is determined as the threshold value where the system is resistant toward a subset of $k - 1$ or fewer corrupted servers. Based on this, the optimum parameter n is set such that $n \geq k$ ($n > k$ to realize resistance toward server loss/failure). Here, resistance toward server loss refers to the ability of the system to function even when a number of computing servers are lost (or broken). However, in conventional MPC using (k, n) threshold secret sharing, n must be set such that $n \geq 2k - 1$. Moreover, as one server is usually assumed to hold only one share each, the number of servers required (represented by parameter N) will also increase for each multiplication operation.

Therefore, the initial cost for setting up the system will increase. Nowadays, there are methods of MPC that focus on reducing the communication cost by fixing the parameters $n = 3, k = 2$, etc. However, in this example, even if one of the three servers fails, the consecutive computation cannot be performed because the information required is lost along with the broken server. Therefore, these methods do not provide resistance toward server loss. In addition, some conventional methods may introduce computational security as a trade-off for realizing computation in $n < 2k - 1$. As far as we know, the proposed method is the only method of MPC that can realize information-theoretic security even when $n < 2k - 1$.

The second advantage is that n can be set such that $n = k$. In this scenario, if the owner of the secret input participates in the MPC as a computing server and manages its share securely, the secret input will not be leaked even if all other computing servers other than the owner collude. In contrast, in the case of $n \geq 2k - 1$, even if the owner manages its share safely, all secret inputs will be leaked if a subset of k or more participants other than the owner collude.

Part II: Chapter 5

In Part I, we propose the generic conditionally secure MPC using (k, n) threshold secret sharing for computing all four arithmetic operations when $n < 2k - 1$. However, it is also essential that a secure MPC can be utilized in various applications. Therefore, in Part II, we focus on applications of secure MPC to realize SE of documents. The contribution of Part II is as follows:

- **Application of secure MPC into SE of document**

One of the natural applications of secure MPC is secure searching over encrypted data (SE). For example, suppose a scenario where Alice stores her data on a cloud server such that she can conveniently access it from anywhere. As there is a risk of leakage of her private data, information must be encrypted before storage to protect the privacy and confidentiality of data. However, searching encrypted data on a server is not possible without first decrypting the encrypted data. To solve this, we propose a method of SE using (k, n) threshold secret sharing that allows for searching over encrypted data. Several methods with this principle have been proposed, such as public key and symmetric key encryption. However, these methods often come with sizeable computational overhead, colossal storage requirements, etc.

As far as we know, the method in this dissertation is the first example of SE using (k, n) threshold secret sharing. In this method, searching is performed per character. The difference between the characters of the registered document and the search query is computed using the MPC method described in Part I. In addition, we also introduce manners of easing the conditions required in the MPC described in Part I. Moreover, as (k, n) threshold secret sharing requires a very low computational overhead, we realized SE of documents very efficiently with high flexibility.

Part III: Chapter 6

In Part I, we study the solutions for a secure MPC with $n < 2k - 1$ using the approach of (polynomial \times scalar value). However, as previously mentioned, there are other approaches to realize multiplication with $n < 2k - 1$. Thus, in Part III, we study an alternative method of introducing degree reduction to the polynomial of the multiplication result. Here, we also present a new method of distribution of secret input to allow for multiplication between polynomials with only $N \geq k$ computing servers. Therefore, the final contribution of this dissertation is as follows:

- **MPC with the multiplication of polynomials is possible with only $N \geq k$ servers**

First, we realized a two-input-one-output multiplication using the typical approach of (polynomial \times polynomial) with only $N \geq k$ computing servers. In this method, we implemented the process of degree reduction to reduce the degree of the polynomial of the multiplication result. As mentioned in Section 1.4, multiplication of two polynomials of degree $(k - 1)$ will result in a polynomial of degree $(2k - 2)$. However, by reducing the degree of polynomial from $(2k - 2)$ back to $(k - 1)$, the number of shares n required to reconstruct the multiplication result will also return to $n \geq k$. Conventional methods such as those by Ben-Or et al. [12] and Chaum et al. [37] also use this approach to prevent the further increase of the polynomial degree. However, as the initial stage of multiplication will produce a polynomial with a $(2k - 1)$ degree, the number of shares required will remain at $n \geq 2k - 1$; therefore,

the number of servers N initially required will also be $N \geq 2k - 1$, because each server only hold one share.

In this part, we overcome this limitation by using the new functionality of distributing multiple encrypted shares of the same secret input to each computing server. This was implemented by encrypting each share with a different random number before sending it to the computing servers. For example, instead of sending one encrypted share of secret input a to one server, we send two encrypted shares to one server. Typically, this will violate the security definition of (k, n) threshold secret sharing because more than k shares will be leaked from $k - 1$ computing servers. However, no information will be revealed as each share is encrypted with a different random number. Moreover, we also realized a new multiplication and degree reduction method for multiplying $(k - 1)$ sharing of encrypted shares of two inputs a, b and reducing the degree of resulting shares from $(2k - 2)$ to $(k - 1)$ using the recombination vector with only $N \geq k$ servers. In addition, we also compared this method with those in Part I to evaluate the differences between the approaches.

1.6 Dissertation Outline

Figure 1.6 shows the focus of each chapter of the dissertation. First, we describe the building block for all of our methods in Chapter 2. Next, Chapters 3 and 4 discuss the method to overcome the impossibility result of multiplication when $n < 2k - 1$ by using the polynomial \times scalar value approach. These chapters also introduce the necessary conditions to realize information-theoretic security when $n < 2k - 1$. Chapter 5 explains the application of our proposed method of MPC into SE of documents. Chapter 6 describes the approach of computing multiplication using the polynomial \times polynomial approach without increasing the number of computing servers required. Finally, Chapter 7 provides the conclusions. The detailed organization of this dissertation is summarized below.

Chapter 2: Basic definitions and building blocks

In this chapter, we introduce the necessary notation, definitions, and some fundamental mathematical theories required to understand the research in this dissertation. Furthermore, we also include a formal definitional parameter of secure MPC, notably MPC that uses secret sharing. We also show the definition of secret sharing and protocols for (k, n) threshold secret sharing that formed the research base.

Chapter 3: First conditionally secure MPC using (k, n) threshold secret sharing with $n < 2k - 1$

In this chapter, first, we briefly reiterate the problem of the conventional method of MPC using (k, n) threshold secret sharing and explain the reason for the unconditional impossibility of multiplication when $n < 2k - 1$. In sequence, we propose a new method of multiplication using the approach of scalar value \times polynomial to realize multiplication when $n < 2k - 1$.

We also evaluate the security of the proposed method against three semi-honest adversaries and explain the condition required to securely compute multiplication operation when $n < 2k - 1$. Finally, in the discussion of the proposed method, we show that the method is not secure against computation with a combination of different types of operation, such as the product-sum operation of $ab + c$, when assuming a semi-honest adversary that knows one of the inputs (e.g., input b) and output (e.g., output $ab + c$), in addition to information from $k - 1$ corrupted servers.

Chapter 4: MPC that is secure against the combination of different operations

In Chapter 3, it was shown that a conditionally secure MPC using (k, n) threshold secret sharing when $n < 2k - 1$ is possible when using the approach of scalar value \times multiplication, but also not secure against computation involving a combination of different types of operations. This chapter proposes an improved method of conditionally secure MPC using (k, n) threshold secret sharing capable of computing a product-sum operation of $ab + c$. We also show that the proposed method is secure against a semi-honest adversary. Moreover, we introduce three additional conditions required to realize computation involving different types of operations and extend the protocol for a single operation of product-sum $ab + c$ to combine multiple product-sum operations and realize a more complex computation. Finally, we perform a detailed analysis of the proposed method and discuss the realizability of each condition.

Chapter 5: Searchable encryption of documents

This chapter implements the MPC method in Chapter 4 to realize SE, where encrypted information can be searched without decrypting the encrypted data. In the MPC method in Chapter 4, three conditions are required to achieve information-theoretic security against semi-honest adversaries. Therefore, in this chapter, we also propose manners of easing the conditions and show that although certain conditions are required, they can be easily solved depending on the application assumed. Next, we show the detailed algorithm for realizing conjunctive search of a keyword with multiple characters and describe the algorithm for the disjunctive search of multiple search queries. Finally, we compare our method with conventional SE methods using public and symmetric key encryption.

Chapter 6: Multiplication of polynomials with $N \geq k$ servers

In Chapter 4, we study the approach of scalar multiplication to realize multiplication in MPC. In this chapter, we propose a new method of MPC using (k, n) threshold secret sharing that uses a different approach to realize multiplication with only $N \geq k$ computing. First, we show the new method of distributing multiple shares of the same secret input to a single computing server. In sequence, we show the multiplication operation using the typical polynomial \times polynomial approach with only $N \geq k$ computing servers. An improved method to reduce the degree of the resulting polynomial to enable the reconstruction of the

multiplication result with only $n < 2k - 1$ shares is also introduced. Finally, we discuss the limitation of this method compared to conventional methods.

Chapter 7: Conclusion and Future Works

We conclude the research contribution of the dissertation and discuss future works.

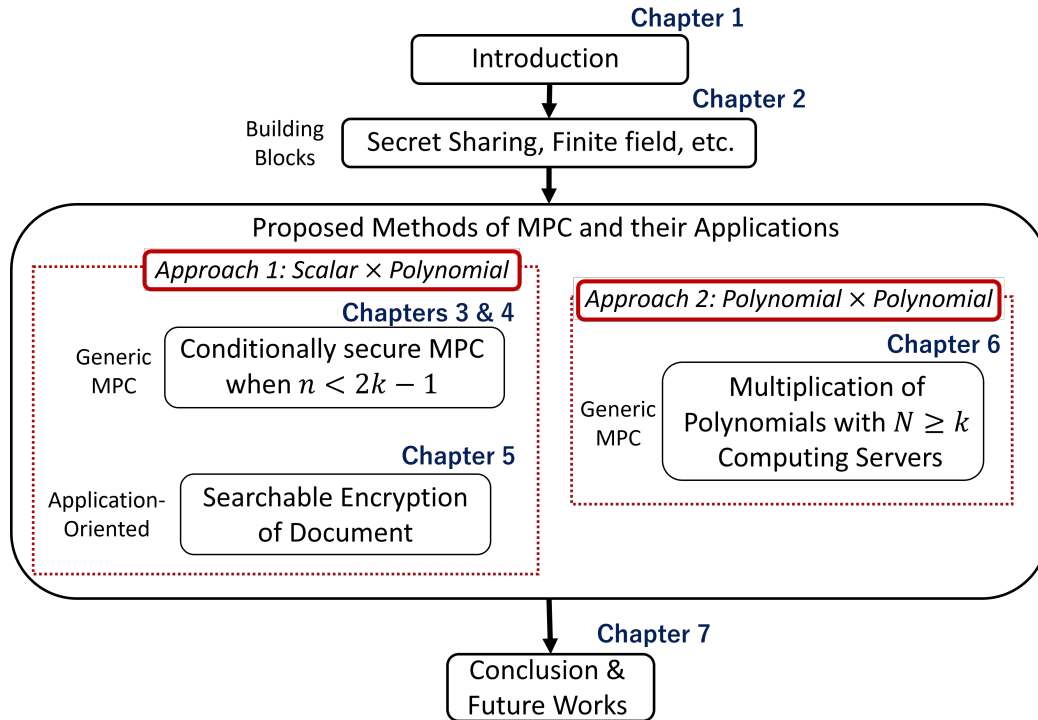


FIGURE 1.6: Dissertation outline.

Chapter 2

Basic Definitions and Building Blocks

This chapter discusses the definitional parameters and terms commonly used in the study of MPC and describes the protocol and mathematical theory of the building blocks used in this dissertation.

2.1 Definitional Parameters in MPC

In a secure MPC, a protocol may be attacked by an adversarial entity controlling some subsets of the computing parties. The computing parties under the adversary's control are called *corrupted* and follow the adversary's instructions. In the case of MPC using (k, n) threshold secret sharing, the adversary is assumed to be able to corrupt a subset of $k - 1$ or fewer computing servers.

To claim that an MPC is secure against any adversarial attack (the adversary's capability will be explained later), many different definitions and requirements have been proposed. This section will only focus on two of the most essential requirements on any MPC protocols: privacy and correctness. MPC protocol is considered secure if it fulfills these two requirements. For more information on other requirements of MPC, we highly suggest reading the survey work by Lindell [9] for a highly comprehensive clarification of all properties.

- **Privacy:** No computing parties should be able to learn more than their prescribed information. In the case of MPC using multiple computing servers, each server should not be able to learn about the actual secret input from the share prescribed to it. Moreover, when the adversary controls the player who reconstructed the output, the MPC protocol must also ensure that the adversary does not learn more than its prescribed output. However, there are extreme cases where MPC protocol might reveal some sensitive information from the output. For example, suppose a scenario where two people wish to compute the average of their ages in a private manner. The MPC protocol will ensure that this process is secure. However, if one of the participants is corrupted, given the output average of both ages and the person's own age, the exact age of the other

honest participant can be easily derived. Therefore, although MPC ensures privacy, in some cases, where information might be revealed from the output itself.

- **Correctness:** Each participant is guaranteed to receive a correct output. In the case of computation using MPC with multiple computing servers, the MPC protocol must ensure that the computing servers will correctly perform the computation, and the participant is guaranteed to receive the correct result or shares. Therefore, even if the adversary corrupted a subset of computing servers, the MPC protocol must ensure that the adversary cannot influence the computation to produce a false result.

Next, we will explain the power of the adversary that may attack a protocol execution. In particular, we will define the actions that the adversary is allowed to perform. In general, there are two most common types of adversaries (recently, there is also another type of adversary known as covert adversary [9]):

- **Semi-honest adversaries:** In this model, the adversary correctly follows the specification of the MPC protocol but may try to learn more than what is allowed from the information received throughout the computation. In the case of MPC using multiple computing servers, the adversary may corrupt a subset of computing servers, obtain their internal state, and use this information to learn about the participant’s secret input. MPC protocol with security against this type of adversarial behavior guarantees no accidental data leakage through the execution of the protocol. Semi-honest adversaries are also called “honest-but-curious” and “passive”.
- **Malicious adversaries:** In this model, the adversary can instruct the corrupted computing servers to arbitrarily deviate from the prescribed protocol specification. For example, in MPC using secret sharing where each computing server is specified with a share of the secret input, the adversary may instruct the corrupted servers to change the shares given, resulting in failure or incorrectness of the computation of reconstruction of secret input. Malicious adversaries are also called “active”.

In this dissertation, we based our method on the *semi-honest adversarial model*. This model is weaker than the malicious adversarial model, but guarantees no accidental data leakage in the protocol, which is sufficient in most current applications.

2.2 Finite Field

Finite fields (also known as Galois fields) are a very important aspect of modern cryptography. A finite field refers to a field in which there are many elements finitely. The number of elements of a finite field is called order or size. The element of finite field GF is defined as follows, where p is a prime number, and m is a positive integer:

$$\begin{aligned}
 GF(p^m) = & (0, 1, 2, \dots, p-1) \cup (p, p+1, p+2, \dots, p+p-1) \\
 & \cup (p^2, p^2+1, p^2+2, \dots, p^2+p-1) \cup \dots \\
 & \cup (p^{n-1}, p^{n-1}+1, p^{n-1}+2, \dots, p^{n-1}+p-1)
 \end{aligned}$$

In this dissertation, we only focused on finite fields with prime order ($m = 1$). The elements of the prime field of order p can be represented by integers in the range $(0, \dots, p-1)$.

2.3 Computation in the Finite Field

As previously mentioned, the notation of finite field $GF(p)$ indicates that numbers within $(0, \dots, p-1)$ will be addressed. For example, when a finite field of $GF(11)$ is chosen, the integers will be within $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$. Therefore, based on the property of finite field mentioned before, any computation performed within the finite field $GF(11)$ will also return values between $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$.

The easiest manner to understand this is to imagine a spinning wheel with numbers between 0 and 10. Suppose that the initial number is at 9 as shown in Figure 2.1, and that the number 5 is added. As the wheel only shows values between 0–10, the resulting value will be 3 and not 14. This is a basic example of an addition operation in a finite field. Next, we will explain in detail the operation of all computations in the finite field.

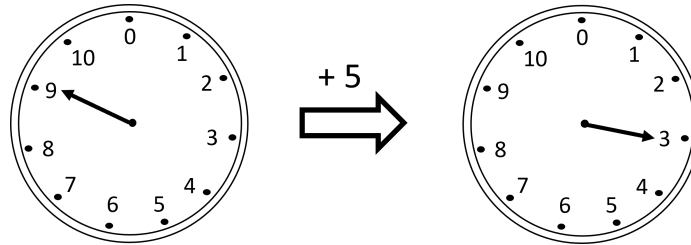


FIGURE 2.1: Addition operation in a finite field.

Addition/Subtraction operation in the finite field $GF(p)$

Addition and subtraction in the finite field can be computed very easily. Suppose that we have inputs a and b , with finite field $GF(p)$. The addition and subtraction between a and b can be computed as follows.

- *Addition operation of a and b*

$$(a + b) \bmod p = ((a \bmod p) + (b \bmod p)) \bmod p$$

- Subtraction operation of a and b

$$(a - b) \bmod p = ((a \bmod p) - (b \bmod p)) \bmod p$$

(Example) Suppose that $a = 5$, $b = 6$, and prime number $p = 7$.

The addition operation between a and b in $GF(7)$ will be as follows:

$$(5 + 6) \bmod 7 = 11 \bmod 7 = 4$$

And the subtraction is as follows:

$$(1 - 6) \bmod 7 = (-5) \bmod 7 = 2$$

Multiplication operation in the finite field $GF(p)$

Multiplication can also be easily computed in the finite field $GF(p)$.

- Multiplication operation of a and b

$$(a \times b) \bmod p = ((a \bmod p) \times (b \bmod p)) \bmod p$$

(Example) Suppose that $a = 5$, $b = 6$, and prime number $p = 7$.

The multiplication operation between a and b in $GF(7)$ will be as follows:

$$(5 \times 6) \bmod 7 = 30 \bmod 7 = 2$$

Division operation in the finite field $GF(p)$

A division in a finite field will require more tedious work. It can be performed by finding the inverse of a number $\bmod n$. For example, suppose that we want to compute the value $1/a = a^{-1}$ in finite field $GF(p)$. Thus, we need to find the solution for the following problem:

$$a^{-1}(\text{Inverse of } a) \Rightarrow \text{Find } b \text{ such that } (a \times b) \bmod p = 1.$$

Here, b is known as the multiplicative inverse of $a \bmod p$, and can be computed by using the *extended Euclidean algorithm* [46]. Let us suppose that we want to find the inverse of 5 in finite field $GF(7)$. Thus, we only need to compute the value of multiplicative inverse b such that $(5 \times b) = 1 \bmod 7$:

$$\begin{aligned} (5 \times b) &= 1 \bmod 7 \\ b &= 5^{-1} \bmod 7 \end{aligned}$$

To find the solution using the extended Euclidean algorithm, first, we need to verify the $GCD(5, 7) = 1$ (the numbers are relatively prime) by using the Euclidean algorithm. This algorithm is a continual repetition of the division algorithm until the remainder is equal to 0, and the GCD will be the last non-zero remainder of the computation. The example below shows the computation to find the GCD between $a = 5$ and $b = 7$.

Find $GCD(5, 7)$

Step 1:

$$7 = 5 \times 1 + 2 \text{ (Remainder = 2)} \quad (2.1)$$

Step 2:

$$5 = 2 \times 2 + 1 \text{ (Remainder = 1)} \quad (2.2)$$

Step 3:

$$2 = 1 \times 2 + 0 \text{ (Remainder = 0)} \quad (2.3)$$

As the last *nonzero* remainder is 1, $GCD(5, 7) = 1$

Thus, we verified that the $GCD(5, 7) = 1$. To find the multiplicative inverse b , where $(5 \times b) = 1 \pmod{7}$, the previous process needs to be reversed.

From equations (2.1) and (2.2), we learn the following:

$$2 = 7 - 5 \quad (2.4)$$

$$1 = 5 - (2 \times 2) \quad (2.5)$$

By substituting equation (2.4) into (2.5), we obtain the following:

$$1 = 5 - (2 \times (7 - 5)) \pmod{7}$$

$$1 = 5 - (2 \times 7) + (2 \times 5) \pmod{7} \quad (2.6)$$

Since $(2 \times 7) \pmod{7} = 0$, therefore

$$1 = 5 \times (1 + 2) \pmod{7} \quad (2.7)$$

By substituting equation (2.4) into (2.7), we obtain the following:

$$1 = 5 \times (1 + (7 - 5)) \pmod{7} \quad (2.8)$$

$$1 = 5 \times 3 \pmod{7} \quad (2.9)$$

As $(5 \times b) = (5 \times 3) = 1 \pmod{7}$, the multiplicative inverse for $a = 5$ in finite field $GF(7)$ can be written as follows:

$$5^{-1} \bmod 7 = 3 \quad (2.10)$$

In the computation for division or to find the multiplicative inverse, a problem will arise with the value of 0 (division by 0). Therefore, during computation for division operation, this value needs to be eliminated. However, to differentiate from the standard finite field, we use $GF^*(p)$ to represent the field without 0 rather than $GF(p)$. In this study, $GF^*(p)$ refers to a finite field that does not include the value 0.

2.4 Lagrange Interpolation

The Lagrange interpolation method is a manner to find a polynomial that takes certain values at arbitrary points [46]. For example, suppose a set of $n + 1$ points $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, where no two x_l are the same. The interpolation polynomial in the Lagrange form is a linear combination of Lagrange basis polynomials as shown below:

$$f(x_l) = \sum_{l=0}^n \lambda_l(x_l) y_l = \lambda_0(x_0) y_0 + \dots + \lambda_n(x_n) y_n \quad (2.11)$$

Here, $\lambda_j(x_l)$ is given as follows (λ_j is also known as the recombination vector, and will be very important in Chapter 6):

$$\lambda_j(x) = \prod_{l=1, l \neq j}^{n+1} \frac{(x - x_l)}{(x_j - x_l)} \quad (2.12)$$

2.5 Secret Sharing

Let us imagine the following situation:

"Alice has a treasure map with the exact location of a precious treasure that she wants to keep hidden from other people. If anyone looks at the map, the exact coordinate of the treasure will be known, and the treasure will be found. Therefore, to keep the treasure safe, the map can be divided into smaller pieces and stored such that the location of the treasure will not be known unless all pieces are gathered."

The aforementioned problem can be easily solved by using secret sharing. Secret sharing works by splitting a secret into multiple smaller pieces to be stored. In this case, as shown in Figure 2.2, Alice can divide her treasure map into multiple smaller pieces and distribute the pieces amongst her family and friends. Thus, the treasure map will only be complete when all the pieces are gathered together; however, no information can be learned from each individual piece.

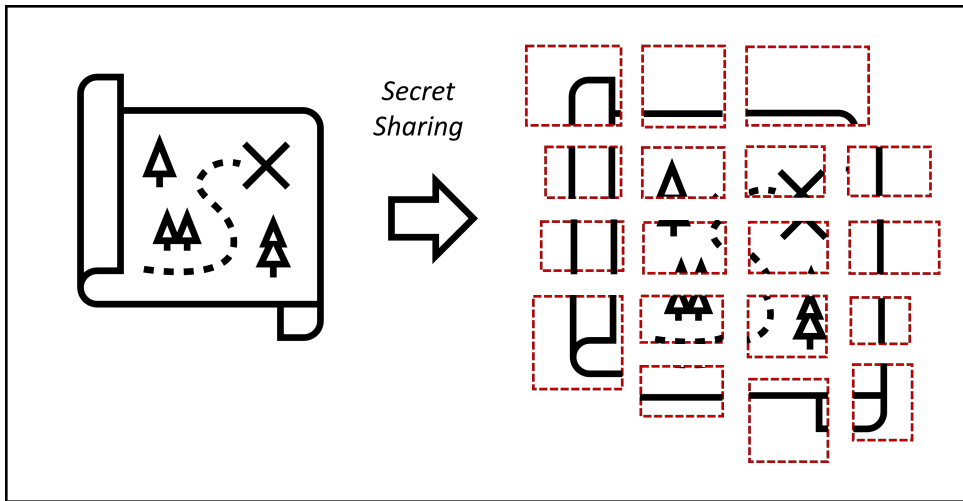


FIGURE 2.2: A secret map can be divided into smaller pieces to be distributed and stored.

In cryptography, secret sharing consists of splitting a private information into multiple small fragments (also known as *shares*) and distributing amongst a distributed network of untrusted users. One of the earliest methods in theory was proposed by Shamir [13] and Blakley [47] in 1979. In a conventional method of keeping a secret secure using an encryption key (e.g., public key encryption and symmetric key encryption), an encryption key is used to encrypt a secret, and any single user with the correct decryption key can decrypt and reconstruct the secret. In contrast, with secret sharing, as no encryption/decryption keys are used, anyone (even users without an encryption key) can distribute (=encrypt) a secret input, but it can only be reconstructed if a sufficient number of shares are collected.

There are multiple methods of secret sharing to distribute and reconstruct a secret, such as (k, n) threshold secret sharing, additive secret sharing, and XOR-based secret sharing [48]. Two of the most frequently used are additive secret sharing and (k, n) threshold secret sharing, which are briefly explained in the following sections.

2.5.1 Additive Secret Sharing

One of the simplest secret sharing methods is known as additive secret sharing. In this method, a secret input s is divided into n number of different shares, such that the original secret input s can only be reconstructed when all shares are gathered, that is, when all participating servers agree to contribute with their shares of the secret.

However, if one or more servers decides not to contribute with their shares or goes missing, the original secret input s cannot be recovered. To solve this problem, (k, n) threshold secret sharing is often used, as it only requires a certain threshold k number of shares for the reconstruction of the secret. We will discuss this in the following section.

Additive secret sharing uses the following protocols for distributing and reconstructing the secret input s . Note that all computations are performed in the finite field $GF(p)$. Moreover, the shares of secret input s are denoted by $\llbracket s \rrbracket_i$.

Protocol 2.1: Distribution of secret input s

1. Let the secret input be s . The dealer first selects $n - 1$ random numbers r_j ($j = 0, \dots, n - 2$) and computes the following:

$$\llbracket s \rrbracket_j = r_j \quad (j = 0, \dots, n - 2) \quad (2.13)$$

$$\llbracket s \rrbracket_{n-1} = s - \sum_{j=0}^{n-2} r_j \text{ mod } p \quad (2.14)$$

2. In sequence, the dealer distributes $\llbracket s \rrbracket_i$ ($i = 0, \dots, n - 1$) to n computing servers S_i .

Protocol 2.2: Reconstruction of secret input s

1. The player who wants to reconstruct the secret input s collects n shares $\llbracket s \rrbracket_i$ ($i = 0, \dots, n - 1$) from all servers S_i and reconstructs the secret input s as follows:

$$s = \sum_{i=0}^{n-1} \llbracket s \rrbracket_i \text{ mod } p \quad (2.15)$$

2.5.2 (k, n) Threshold Secret Sharing

A secret sharing is known as (k, n) threshold secret sharing when it satisfies the following conditions.

- Any $k - 1$ or fewer shares reveal no information about the original secret input s ;
- Any k or more shares enables the reconstruction of the original secret input s .

The classic method for (k, n) threshold secret sharing is Shamir's (k, n) threshold secret sharing (referred to as Shamir's (k, n) method) [13]. In this method, all computations are performed in the finite field $GF(p)$. Moreover, the shares of secret input s are denoted by $\llbracket s \rrbracket_i$.

Shamir's (k, n) method uses the following protocols for the distribution and reconstruction of secret input s .

Protocol 2.3: Distribution of secret input s

1. The dealer selects $k - 1$ random numbers $\alpha_1, \dots, \alpha_{k-1}$ and generates a random polynomial $f(x_i)$ as follows:

$$f(x_i) = s + \alpha_1 x_i + \alpha_2 x_i^2 + \dots + \alpha_{k-1} x_i^{k-1} \text{ mod } p \quad (2.16)$$

2. In sequence, the dealer inserts the ID $(i + 1)$ of server S_i ($i = 0, \dots, n - 1$) into $f(x_i)$, calculates the shares $\llbracket s \rrbracket_i = f(i + 1)$ corresponding to each ID, and distributes them to all servers.

(Example) Let $k = 2$, $n = 3$, $p = 19$, and secret input $s = 3$

1. The dealer selects a random number $\alpha_1 = 2$ and generates the following polynomial:

$$f(x) = 3 + 2x \text{ mod } 19$$

2. The dealer computes the $n = 3$ shares for each server by substituting their ID into x as follows:

$$\text{ID of server } S_0 = 1 : \text{Share } \llbracket s \rrbracket_0 = f(x_0 = 1) = 3 + 2(1) = 5$$

$$\text{ID of server } S_1 = 2 : \text{Share } \llbracket s \rrbracket_1 = f(x_1 = 2) = 3 + 2(2) = 7$$

$$\text{ID of server } S_2 = 3 : \text{Share } \llbracket s \rrbracket_2 = f(x_2 = 3) = 3 + 2(3) = 9$$

Protocol 2.4: Reconstruction of secret input s

1. The player who wants to restore the secret input s collects any k shares $\llbracket s \rrbracket_j$ ($j = 0, \dots, k - 1$) and their pair of IDs;
2. The player restores the original secret input s by using Lagrange's interpolation as follows:

$$s = \sum_{i=0}^{n-1} \prod_{j=0, j \neq i}^{n-1} \frac{x_j}{x_j - x_i} \llbracket s \rrbracket_i \text{ mod } p \quad (2.17)$$

(Example) Let $k = 2$, $p = 19$, and the shares held by each server are as shown in the example of Protocol 2.3.

1. The player collects $k = 2$ shares $\llbracket s \rrbracket_1 = (1, 5)$, $\llbracket s \rrbracket_2 = (2, 7)$ from servers S_0 and S_1 , and solves $k = 2$ simultaneous equation to obtain secret input s using Lagrange Interpolation as follows:

$$\begin{aligned} s &= \sum_{i=0}^{k-1} \prod_{j=0, j \neq i}^{k-1} \frac{x_j}{x_j - x_i} \\ &= \frac{2}{2-1} 5 + \frac{1}{1-2} 7 = 3 \text{ mod } p \end{aligned}$$

Chapter 3

A Conditionally Secure MPC using (k, n) Threshold Secret Sharing

In this chapter, we study the problem of the increase in the degree of the polynomial when performing multiplication between polynomials. In particular, we focus on finding a method of multiplication that does not increase the polynomial degree. In this chapter, instead of polynomial \times polynomial, we introduce the approach of multiplication using scalar value \times polynomial. We also evaluate the security of the proposed method and perform a comparison with the conventional SPDZ method by Damgård et al [19].

3.1 Introduction

In recent years, with the advancement of big data and the IoT ecosystem, technologies to utilize personal information to obtain valuable statistical data have been anticipated. However, this utilization could affect the individuals' privacy if their personal information is leaked. Therefore, a large amount of research has been conducted on utilizing big data while ensuring the protection of sensitive material, such as individuals' personal information. As mentioned in Chapter 1, one of the technologies that could realize this is known as secure computation. In this dissertation, we focus on the version of secure computation using multiple computing servers, also known as MPC. MPC can utilize big data while preserving privacy, which enables various statistical computations and processing of data with personal information.

In this chapter, we propose a conditionally secure MPC using Shamir's (k, n) method to preserve these data while performing various arithmetic computations, such as to utilize big data while the individuals' personal information is still protected. In Shamir's (k, n) method, even if part of the shares is lost due to a server or network failure, the original secret input can be restored if it is $n - k$ or less, realizing a resistance toward up to $n - k$ server losses. However, as discussed in Section 1.4, in conventional MPC methods that use Shamir's (k, n) method, the number of shares required to reconstruct the multiplication result will increase

from k to $2k - 1$ for each multiplication performed due to the increase of polynomial degree from $k - 1$ to $2k - 2$.

In this chapter, we propose a different functionality for sharing secret input. Instead of the “normal share” generated directly from the secret inputs as shown in Section 2.5.2, we realize an “*encrypted share*” functionality. Here, the secret input is multiplied by a random number to produce an encrypted secret input. In sequence, the encrypted secret input is distributed by the polynomial to n computing servers as shares.

During multiplication, the shares of the encrypted secret input are collected and temporarily restored as a scalar value. The multiplication operation between polynomials is transformed in the form of scalar value \times polynomial. Our proposed multiplication method does not change the degree of the resulting polynomial or the number of shares required to reconstruct the result. In addition, we also propose the protocol for addition operation using the encrypted secret inputs and construct a method that does not change the resulting degree of the polynomial for all four arithmetic operations while keeping the information-theoretic security nature of Shamir’s (k, n) method. This enables us to build a system that does not limit the minimum number of computing servers, even if the computation includes multiplication.

3.2 Related Work: SPDZ Method

For comparison with our proposed method, we will explain the conventional work of MPC using secret sharing proposed by Damgård et al. [19], known as the SPDZ method. Damgård et al. proposed a secure MPC called the SPDZ method that can be used even when the number of adversaries is more than half of the participants (dishonest majority) in the setting of $n = k$. The SPDZ method assumes an MPC model in which the owner of the secret input is also one of the computing players (or servers, when comparing with our method); even if all players ($n - 1$) other than the owner collude, leakage of the secret input does not occur. In addition, the confidentiality of secret input in the SPDZ method is realized by using additive secret sharing.

Multiplication using the SPDZ method is also realized using the approach of multiplication with a scalar value to prevent changes in the degree of polynomials. Specifically, to multiply the shares $\llbracket x \rrbracket_i, \llbracket y \rrbracket_i$ of the secret inputs x, y , respectively, shares $\llbracket a \rrbracket_i, \llbracket b \rrbracket_i, \llbracket c \rrbracket_i$ of random numbers a, b, c , respectively, that satisfy $ab = c$, proposed in Beaver’s circuit randomization method [20], are used. First, the values of $d = \text{open}(\llbracket x \rrbracket_i - \llbracket a \rrbracket_i), e = \text{open}(\llbracket y \rrbracket_i - \llbracket b \rrbracket_i)$ are reconstructed, and the multiplication is computed using the following equation:

$$\llbracket xy \rrbracket_i = de + e\llbracket a \rrbracket_i + d\llbracket b \rrbracket_i + \llbracket c \rrbracket_i \quad (3.1)$$

However, in the preprocessing phase of the SPDZ method, SHE is used to generate the shares of $[[a]]_i, [[b]]_i, [[c]]_i$. As shown in Table 1.2, SHE requires considerably more computational overhead than secret sharing. Therefore, although the operation for multiplication can be easily realized, the amount of computation required to perform SHE is very large in the SPDZ method. In other words, the SPDZ method can avoid the limitation of $n \geq 2k - 1$ related to multiplication using secret sharing at the expense of an increase of computation overhead for SHE. Moreover, the implementation of SHE also means that the SPDZ method only realizes weaker computational security instead of information-theoretic security.

3.3 Proposed Method: TUS 1 Method

3.3.1 Overview of TUS 1 Method

In this chapter, we describe our proposed method to conditionally secure MPC when $n < 2k - 1$. For simplicity, this method is known as the Tokyo University of Science 1 (TUS 1) method. In our proposed method, a secret input, encrypted by being multiplied by a random number, is called an encrypted secret input, and distributed to n computing servers as encrypted shares. During the process of multiplication, the encrypted secret input is reconstructed momentarily as a scalar value and multiplied with the other polynomial to realize multiplication without increasing the degree of the resulting polynomial. Note that the proposed method can be applied to any secret sharing method that is secure for addition and scalar multiplication operations. Here, Shamir's (k, n) method was assumed.

3.3.2 Protocol of TUS 1 Method

This section describes the proposed protocol of TUS 1 method. Here, the secret inputs $a, b \in GF(p)$, random numbers generated throughout the protocol, are also elements of $GF^*(p)$ uniformly distributed (exclude 0). However, during multiplication, if the secret input is equal to 0 (for example, input $a = 0$), its scalar value (we assume αa) reconstructed in the multiplication protocol will be 0 (in this case, $\alpha a = 0$); therefore, the value of the secret input as 0 will be leaked because random number α is generated from random numbers $\alpha_0, \dots, \alpha_{k-1}$ that do not include 0. To prevent this, we included a condition that the value of the secret input must not include 0 for the multiplication operation. Moreover, all computations, including distribution, are performed in finite field $GF(p)$.

Notations:

- $[[\alpha a]]_i$: Share of encrypted secret input αa held by server S_i .

Protocol 3.1: Distribution of secret input a

1. The dealer generates k random numbers $\alpha_0, \dots, \alpha_{k-1}$ and computes random number α as follows:

$$\alpha = \prod_{j=0}^{k-1} \alpha_j$$

2. The dealer computes the encrypted secret input $\alpha a = \alpha \times a$, and distributes the following using Shamir's (k, n) method to n computing servers S_i ($i = 0, \dots, n-1$):

$$\alpha a, \alpha_0, \dots, \alpha_{k-1}$$

3. Each server S_i ($i = 0, \dots, n-1$) holds the following shares of secret input a :

$$\llbracket \alpha a \rrbracket_i, \llbracket \alpha_0 \rrbracket_i, \dots, \llbracket \alpha_{k-1} \rrbracket_i$$

Protocol 3.2: Reconstruction of secret input a

1. The player collects the following shares from k servers S_j ($j = 0, \dots, k-1$), and reconstructs $\alpha a, \alpha_0, \dots, \alpha_{k-1}$ using Shamir's (k, n) method:

$$\llbracket \alpha a \rrbracket_j, \llbracket \alpha_0 \rrbracket_j, \dots, \llbracket \alpha_{k-1} \rrbracket_j \quad (j = 0, \dots, k-1)$$

2. The player computes random number α and reconstructs input a as follows:

$$\alpha = \prod_{j=0}^{k-1} \alpha_j$$

$$a = \alpha a \times \alpha^{(-1)}$$

Protocol 3.3: Multiplication of a and b

In sequence, we show the protocol to compute encrypted shares of multiplication result ab from encrypted shares of inputs a, b . The reconstruction of multiplication result ab from the encrypted shares can be performed through Protocol 3.2. Here, k servers S_j ($j = 0, \dots, k-1$) that reconstruct random numbers α_j, β_j are specified in advance out of n servers S_i ($i = 0, \dots, n-1$). In addition, in the multiplication protocol, the input does not include the value 0.

Input:

- Shares of secret input a from Protocol 3.1: $\llbracket \alpha a \rrbracket_j, \llbracket \alpha_0 \rrbracket_j, \dots, \llbracket \alpha_{k-1} \rrbracket_j$ ($j = 0, \dots, k-1$)
- Shares of secret input b from Protocol 3.1: $\llbracket \beta b \rrbracket_j, \llbracket \beta_0 \rrbracket_j, \dots, \llbracket \beta_{k-1} \rrbracket_j$ ($j = 0, \dots, k-1$)

Output: $\llbracket \alpha \beta ab \rrbracket_i, \llbracket \alpha_0 \beta_0 \rrbracket_i, \dots, \llbracket \alpha_{k-1} \beta_{k-1} \rrbracket_i$ ($i = 0, \dots, n-1$)

1. One of the servers (here, we assume server S_0) collects $[[\alpha a]]_j$ from k servers, reconstructs αa and sends it to all servers S_i .
2. Each server S_i ($i = 0, \dots, n-1$) computes the following:

$$[[\alpha \beta ab]]_i = \alpha a \times [[\beta b]]_i$$

3. Each server S_j ($k = 0, \dots, k-1$) collects k shares of the following from k servers and reconstructs random numbers α_j, β_j :

$$\begin{aligned} & [[\alpha_j]]_0, \dots, [[\alpha_j]]_{k-1}, \\ & [[\beta_j]]_0, \dots, [[\beta_j]]_{k-1} \end{aligned}$$

4. Each server S_j ($k = 0, \dots, k-1$) computes the following and distributes to all servers using Shamir's (k, n) method:

$$\alpha_j \beta_j = \alpha_j \times \beta_j$$

5. Each server S_i ($i = 0, \dots, n-1$) holds the following shares for ab :

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

Protocol 3.4: Division of a and b

For the division operation, the computation in Steps 2 and 4 of Protocol 3.3 is replaced with the following:

Step 2: Each server S_i ($i = 0, \dots, n-1$) computes the following:

$$\left[\frac{\beta b}{\alpha a} \right]_i = \frac{[[\beta b]]_i}{\alpha a}$$

Step 4: Each server S_j ($j = 0, \dots, k-1$) computes the following and distributes to all servers.

$$\frac{\beta_j}{\alpha_j} = \beta_j \div \alpha_j$$

The result of the division protocol will produce the following shares for the result b/a :

$$\left[\frac{\beta b}{\alpha a} \right]_i, \left[\frac{\beta_0}{\alpha_0} \right]_i, \dots, \left[\frac{\beta_{k-1}}{\alpha_{k-1}} \right]_i \quad (i = 0, \dots, n-1)$$

Typically, during the division operation, the solution cannot be defined when the divisor is 0. Therefore, we need to exclude the computation when the divisor equals 0. In the proposed method, the result of αa in Step 1 will be $\alpha a = 0$ if the divisor is 0; thus, we can stop the process here.

Protocol 3.5: Addition/Subtraction of a and b **Input:**

- Shares of secret input a from Protocol 3.1: $[\alpha a]_j, [\alpha_0]_j, \dots, [\alpha_{k-1}]_j$ ($j = 0, \dots, k-1$)
- Shares of secret input b from Protocol 3.1: $[\beta b]_j, [\beta_0]_j, \dots, [\beta_{k-1}]_j$ ($j = 0, \dots, k-1$)

Output: $[\gamma(a \pm b)]_i, [\gamma_0]_i, \dots, [\gamma_{k-1}]_i$ ($i = 0, \dots, n-1$)

1. Each server S_j ($j = 0, \dots, k-1$) collects k shares of the following and reconstructs random numbers α_j, β_j :

$$\begin{aligned} & [\alpha_j]_0, \dots, [\alpha_j]_{k-1}, \\ & [\beta_j]_0, \dots, [\beta_j]_{k-1} \end{aligned}$$

2. Each server S_j ($j = 0, \dots, k-1$) generates random number γ_j , computes the following and sends to one of the servers (here, we assume server S_0):

$$\frac{\gamma_j}{\alpha_j}, \frac{\gamma_j}{\beta_j}$$

3. Server S_0 computes $\gamma/\alpha, \gamma/\beta$ as follows and sends to all servers:

$$\begin{aligned} \frac{\gamma}{\alpha} &= \prod_{j=0}^{k-1} \frac{\gamma_j}{\alpha_j}, \\ \frac{\gamma}{\beta} &= \prod_{j=0}^{k-1} \frac{\gamma_j}{\beta_j} \end{aligned}$$

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

$$[\gamma(a \pm b)]_i = \left(\frac{\gamma}{\alpha} \times [\alpha a]_i \right) \pm \left(\frac{\gamma}{\beta} \times [\beta a]_i \right)$$

5. Each server S_j ($j = 0, \dots, k-1$) distributes random number γ_j to all servers using Shamir's (k, n) method.
6. Each server S_i ($i = 0, \dots, n-1$) holds the following for $a \pm b$:

$$[\gamma(a \pm b)]_i, [\gamma_0]_i, \dots, [\gamma_{k-1}]_i \quad (i = 0, \dots, n-1)$$

3.3.3 Security of TUS 1 Method

Our proposed method assumes a semi-honest adversary that honestly follows the protocol, secure communication between all players and servers, and that one server always handles

the same random number. For example, server S_2 will always reconstruct, combine, distribute random numbers α_2 and β_2 . Moreover, the following four types of adversaries are assumed:

Adversary 1: The adversary corrupts a subset of $k - 1$ computing servers, knows the information from a subset of $k - 1$ servers, and attempts to learn about the inputs or output of the computation.

Adversary 2: When one of the players who inputted the secret input is the adversary, it will be able to know one of the secret inputs and the random numbers used to encrypt the input. Moreover, the adversary has information from $k - 1$ servers and attempts to learn about the input of the other player or the output of the computation.

Adversary 3: When the player who reconstructs the computation result is the adversary, it will be able to learn about the information sent by k servers required to reconstruct the result. Moreover, the adversary has information from $k - 1$ servers and attempts to learn about the secret inputs.

Adversary 4: When the player who inputted one of the inputs and the player who reconstructed the result is the adversary, it will be able to know about one of the inputs and the random numbers used to encrypt the input, in addition to the information sent by k servers for the reconstruction of the computation result. Moreover, the adversary has information from $k - 1$ servers and attempts to learn about the remaining input.

However, in a two-input-one-output computation such as $a + b = c$, regardless of the types of secure computation used, Adversary 4, who knows one of the inputs (e.g., a) and the output (e.g., c), will be able to learn about the remaining input (e.g., $b = c - a$). Therefore, in the case of two-input-one-output computation, we only consider security against Adversaries 1–3. If Adversaries 1–3 manage to learn the information that they attempt to learn, the attack is considered a success. In other words, suppose two inputs a, b , and output c , and the information that each adversary knows to be Y . The attack is considered a success if Adversaries 1–3 manage to learn all the inputs a, b and output c from Y . However, if no information is learned about a, b , and c from Y , the attack is considered unsuccessful and the following equations will hold:

$$H(a) = H(a|Y)$$

$$H(b) = H(b|Y)$$

$$H(c) = H(c|Y)$$

Security of Protocol 3.3: Multiplication of a and b

Evaluation of security against Adversary 1

Adversary 1 knows the value of αa from the protocol and learns random numbers α_l, β_l ($l = 0, \dots, k - 2$) from $k - 1$ servers. Therefore, it knows the information of $\alpha a, \alpha_l, \beta_l$ ($l =$

$0, \dots, k-2$) and attempts to learn about secret inputs a, b , and output ab . First, random numbers α, β will not be leaked from $k-1$ values of α_l, β_l . Therefore:

$$\begin{aligned} H(\alpha) &= H(\alpha | \alpha_0, \dots, \alpha_{k-2}) \\ H(\beta) &= H(\beta | \beta_0, \dots, \beta_{k-2}) \end{aligned}$$

In addition, as long as the secret input a or the computed random number α are not equal to 0, α, a will not be leaked from αa . Therefore, the following is also true:

$$\begin{aligned} H(\alpha) &= H(\alpha | \alpha a, \alpha_0, \dots, \alpha_{k-2}) \\ H(a) &= H(a | \alpha a, \alpha_0, \dots, \alpha_{k-2}) \end{aligned}$$

Moreover, as random numbers β_l ($l = 0, \dots, k-2$) are independent of αa and α_l ($l = 0, \dots, k-2$), secret inputs a, b will not be leaked. Therefore, the following statements hold:

$$\begin{aligned} H(a) &= H(a | \alpha a, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}) \\ H(b) &= H(b | \alpha a, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}) \end{aligned}$$

In other words, even if Adversary 1 manages to learn about information from $k-1$ servers and information obtained through the execution of Protocol 3.3, secret inputs a, b will not be leaked.

Next, we discuss whether the result of multiplication ab will be leaked or not. In the multiplication protocol, the value of αa is multiplied with shares of $\llbracket \beta b \rrbracket_i$ and stored. However, $\alpha \beta ab$ cannot be reconstructed even if information from $k-1$ servers is leaked. Therefore, the following is true:

$$H(\alpha \beta ab) = H(\alpha \beta ab | \alpha a, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2})$$

In addition, as the value of $\alpha \beta$ is not leaked to Adversary 1, the multiplication result ab will also not be leaked. Therefore, the following statement is also true:

$$H(\alpha \beta) = H(\alpha \beta | \alpha a, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2})$$

Thus, even if Adversary 1 learns about information from $k-1$ servers in addition to public information leaked through the execution of Protocol 3.3, it will not be able to learn about the multiplication result ab , and:

$$H(ab) = H(ab | \alpha a, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2})$$

Evaluation of security against Adversary 2

For ease of understanding, suppose that Adversary 2 controls the player who inputted secret b . In this case, Adversary 2 will learn about random number β (and $\beta_0, \dots, \beta_{k-1}$ that compose it) and secret input b , in addition to information of Adversary 1 shown previously. However, as α, a and β, b are completely independent to each other, even with information of β, b , Adversary 2 will not be able to learn any information about random number α and secret input a . Moreover, it will not be able to learn the multiplication result ab from $k - 1$ servers. Therefore, the following statements are true:

$$\begin{aligned} H(a) &= H(a|\alpha a, \beta, b, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-1}) \\ H(ab) &= H(ab|\alpha a, \beta, b, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-1}) \end{aligned}$$

In addition, even if Adversary 2 controls the player who inputted secret input a instead of the player who inputted b , the same can be said, as it will not be able to learn about random number β and secret input b because α, a , and β, b are completely independent to each other.

Evaluation of security against Adversary 3

In addition to information from $k - 1$ servers (Adversary 1), Adversary 3, who controls the player that reconstructs the multiplication result, has information of random number $\alpha\beta$ (and $\alpha_0\beta_0, \dots, \alpha_{k-1}\beta_{k-1}$ that compose it), encrypted output $\alpha\beta ab$, and the multiplication result ab . Therefore, the information known by Adversary 3 (which is equal to the ciphertext) are $\alpha a, \alpha_l, \beta_l$ ($l = 0, \dots, k - 2$), $\alpha\beta, \alpha\beta ab$. Here, the multiplication result ab can be obtained from $\alpha\beta$ and $\alpha\beta ab$. However, secret inputs a, b and random numbers α, β cannot be separated from ab and $\alpha\beta$, respectively. In addition, as discussed before, random numbers α, β cannot be obtained from αa and α_l, β_l ($l = 0, \dots, k - 2$). Therefore, the following statements hold:

$$\begin{aligned} H(a) &= H(a|ab, \alpha\beta, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}) \\ H(b) &= H(b|ab, \alpha\beta, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}) \end{aligned}$$

From the argument above, even with information from $k - 1$ servers and information received during the reconstruction process in Protocol 3.2, Adversary 3 will not be able to learn about the secret inputs a, b .

However, when $a = b$, the player who reconstructs the multiplication result $ab = a^2$ will also learn about the value of input a , which occurs for any secure computation method including MPC using secret sharing; therefore, we did not consider this situation (the same is also true for the addition/subtraction protocol).

As the evaluations discussed above also hold for the division protocol (Protocol 3.4), we can state that our proposed protocol is information-theoretical secure against Adversaries 1–3 for a single multiplication or division operation.

Security of Protocol 3.5: Addition/Subtraction of a and b *Evaluation of security against Adversary 1*

Adversary 1 knows the public values of γ/α and γ/β from executing Protocol 3.5. In addition, it also knows information $\gamma_l, \alpha_l, \beta_l$ ($l = 0, \dots, k-2$) from $k-1$ servers. Therefore, the information that Adversary 1 knows (that are equal to the ciphertext) are $\gamma/\alpha, \gamma/\beta$ and $\gamma_l, \alpha_l, \beta_l$ ($l = 0, \dots, k-2$), and the adversary will attempt to learn about secret inputs a, b , and output $a \pm b$.

Here, random numbers $\gamma_l, \alpha_l, \beta_l$ are chosen individually, and random numbers γ, α, β will not be leaked from $k-1$ number of $\gamma_l, \alpha_l, \beta_l$. In addition, each individual random number γ, α, β cannot be separated from γ/α and γ/β . Therefore, the following statements are true:

$$\begin{aligned} H(\gamma) &= H\left(\gamma \left| \frac{\gamma}{\alpha}, \frac{\gamma}{\beta}, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}, \gamma_0, \dots, \gamma_{k-2} \right.\right) \\ H(\alpha) &= H\left(\alpha \left| \frac{\gamma}{\alpha}, \frac{\gamma}{\beta}, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}, \gamma_0, \dots, \gamma_{k-2} \right.\right) \\ H(\beta) &= H\left(\beta \left| \frac{\gamma}{\alpha}, \frac{\gamma}{\beta}, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}, \gamma_0, \dots, \gamma_{k-2} \right.\right) \end{aligned}$$

Moreover, from the information of γ/α and γ/β , which was sent to all servers, the adversary will be able to learn about the ratio of random numbers α, β used to encrypt secret inputs a, b (for example, α/β). However, unlike in Protocols 3.3 and 3.4, Protocol 3.5 does not require the reconstruction of scalar values of αa or βb . Therefore, even if Adversary 1 learns about information from $k-1$ servers in addition to the information obtained during the execution of Protocol 3.5, it will not be able to learn about secret inputs a, b .

Next, we discuss whether the computation result $a \pm b$ will leak or not. The computation result is not reconstructed in the middle of the computation process; for example, even when information from $k-1$ servers are leaked, Adversary 1 will not be able to reconstruct the computation result. Therefore, the following statements are true:

$$H(a+b) = H\left(a+b \left| \frac{\gamma}{\alpha}, \frac{\gamma}{\beta}, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}, \gamma_0, \dots, \gamma_{k-2} \right.\right)$$

From the arguments above, even with the information from $k-1$ servers and information learned during the execution of the protocol, Adversary 1 will not be able to learn about the result of $a \pm b$.

Evaluation of security against Adversary 2

Suppose that Adversary 2 controls the player who inputted secret input a . Because Adversary 2 knows the information of random number α , it will learn about random numbers γ and β from γ/α and γ/β , respectively. The same occurs when Adversary 2 controls the player who inputted b ; Adversary 2 will be able to learn random numbers γ and α from random number β .

Moreover, Adversary 2 will be able to learn all random numbers α_j, β_j ($j = 0, \dots, k-1$) from $k-1$ number of α_l, β_l ($l = 0, \dots, k-2$) and the values of α, β . However, encrypted secret inputs $\alpha a, \beta b$ will not be leaked from $k-1$ shares of $[\alpha a]_l$ and $[\beta b]_l$. Moreover, in Protocol 3.5, the values of encrypted secret inputs αa or βb are not reconstructed during the computation. Therefore, although Adversary 2 will learn about the random numbers γ, α, β , it will not be able to learn about the secret inputs a or b . This can be said to be the same situation as in the computation of addition/subtraction using the original Shamir's (k, n) method when random numbers $\alpha = \beta = \alpha_j = \beta_j = 1$ are known values.

Similarly, the secret inputs a, b will not be leaked from $k-1$ shares. Therefore, the attack will not be possible even if the random numbers used to encrypt the secret inputs are leaked, ensuring security of the secret inputs and computation results.

Evaluation of security against Adversary 3

Adversary 3 knows the information of random number γ and the encrypted result $\gamma(a \pm b)$ from the player who reconstructed the result of the computation, in addition to information from $k-1$ servers. Moreover, it also learns about $\gamma/\alpha, \gamma/\beta$ during the execution of Protocol 3.5. Therefore, Adversary 3 knows $\gamma, \gamma(a \pm b)$ and $\gamma/\alpha, \gamma/\beta, \alpha_l, \beta_l$ ($l = 0, \dots, k-2$). By using this information, Adversary 3 will attempt to learn about the secret inputs a, b (which are equal to the plaintext). First, from the random number γ , the adversary will be able to learn about random numbers α, β from $\gamma/\alpha, \gamma/\beta$, respectively. However, in Protocol 3.5, as the encrypted secret inputs αa and βb are not reconstructed, Adversary 3 will not be able to learn secret inputs a, b . Therefore, the following statements are true:

$$\begin{aligned} H(a) &= H\left(a \mid a+b, \gamma, \frac{\gamma}{\alpha}, \frac{\gamma}{\beta}, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}\right) \\ H(b) &= H\left(b \mid a+b, \gamma, \frac{\gamma}{\alpha}, \frac{\gamma}{\beta}, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}\right) \end{aligned}$$

Even with the information known, Adversary 3 will not be able to learn about secret inputs a, b . Therefore, we can state that Adversary 3 learns about random numbers γ, α, β , but is not able to learn about the secret inputs a, b . This is the same as in the computation using Shamir's (k, n) method, where all random numbers are known by the adversary.

As a conclusion, the arguments above indicate that our proposed protocol for addition/subtraction is information-theoretical secure against Adversaries 1–3.

3.4 Extension of TUS 1 Method

In Sections 3.2 and 3.3, we showed the protocol and explained the security for a single two-inputs computation. However, there are also times when a secure MPC needs to address

computation with many inputs. Thus, we will show the extension and security of our method when performing computation with many inputs such as $\prod a$ and $\sum a$.

In case the adversary corrupted and learned one of the players who inputted the secret input, if the individual inputs that compose the remaining input cannot be identified individually, the secret input is still secure. Therefore, here, we also assume the existence of Adversary 4 and evaluate the security against it.

3.4.1 Many-Inputs Multiplication

Multiplication of multiple inputs can be realized through the continuous execution of two-inputs-one-output Protocol 3.3 explained in Section 3.3. However, it would be very ineffective. Here, we show a more effective manner of realizing multiplication with many inputs by extending Protocol 3.3 as follows. The reconstruction process performed by the server S_0 in Steps 1 and 2 of the following protocol can also be shared between multiple servers.

For example, multiple servers can execute the process in Steps 1 and 2 in parallel to realize a faster computation. As the reconstructed products are shared with all servers, sharing the process in Steps 1 and 2 will not result in any problem regarding the security of the secret input and output.

Protocol 3.6: Many Inputs Multiplication Protocol

Input:

- Shares of secret input a from Protocol 3.1: $\llbracket \alpha a \rrbracket_j, \llbracket \alpha_0 \rrbracket_j, \dots, \llbracket \alpha_{k-1} \rrbracket_j$ ($j = 0, \dots, k-1$)
- Shares of secret input b from Protocol 3.1: $\llbracket \beta b \rrbracket_j, \llbracket \beta_0 \rrbracket_j, \dots, \llbracket \beta_{k-1} \rrbracket_j$ ($j = 0, \dots, k-1$)
- Shares of secret input z from Protocol 3.1: $\llbracket \zeta z \rrbracket_j, \llbracket \zeta_0 \rrbracket_j, \dots, \llbracket \zeta_{k-1} \rrbracket_j$ ($j = 0, \dots, k-1$)

Output: $\llbracket \alpha \cdots \zeta a \cdots z \rrbracket_i, \llbracket \alpha_0 \cdots \zeta_0 \rrbracket_i, \dots, \llbracket \alpha_{k-1} \cdots \zeta_{k-1} \rrbracket_i$ ($i = 0, \dots, n-1$)

1. Server S_0 collects k shares $\llbracket \beta b \rrbracket_j, \dots, \llbracket \zeta z \rrbracket_j$ from k servers.
2. Server S_0 reconstructs encrypted secret inputs $\beta b, \dots, \zeta z$ and sends them to all servers.
3. Each server S_i computes the following:

$$\llbracket \alpha \cdots \zeta a \cdots z \rrbracket_i = \beta b \times \cdots \times \zeta z \times \llbracket \alpha a \rrbracket_i$$

4. Each server S_j collects k shares of the following and reconstructs random numbers α_j, \dots, ζ_j .

$$\llbracket \alpha_j \rrbracket_0, \dots, \llbracket \alpha_j \rrbracket_{k-1}, \dots, \llbracket \zeta_j \rrbracket_0, \dots, \llbracket \zeta_j \rrbracket_{k-1}$$

5. Each server S_j computes the following and distributes to all servers using Shamir's (k, n) method.

$$\alpha_j \cdots \zeta_j = \alpha_j \times \cdots \times \zeta_j$$

6. Each server S_i holds the following for the result $a \cdots z$:

$$\llbracket \alpha \cdots \zeta a \cdots z \rrbracket_i, \llbracket \alpha_0 \cdots \zeta_0 \rrbracket_i, \dots, \llbracket \alpha_{k-1} \cdots \zeta_{k-1} \rrbracket_i$$

Evaluation of security against Adversaries 1–4

Adversary 1 knows all the reconstructed encrypted secret inputs of $\beta b, \dots, \zeta z$ (except αa) and $k - 1$ random numbers α_l, \dots, ζ_l ($l = 0, \dots, k - 2$). As the same arguments in Section 3.3.3 hold regarding the information that is leaked through the encrypted secret inputs and its partial random numbers, we can say that each random number α, \dots, ζ will not be leaked. In addition, considering the arguments in Section 3.3.3, we can say that the secret input and the result of the computation will also not be leaked from the combination of the known values. Moreover, even in the case of Adversaries 2 and 3, the same arguments still hold, and we can state that the secret inputs will not be leaked. Therefore, Protocol 3.6 is secure against Adversaries 1–3.

Next, we will explain the evaluation of security against Adversary 4, where the adversary knows the computation result and one of the inputs. First, let us assume that Adversary 4 controls the player who inputted the secret input a , and the player who reconstructed the output $a \cdots z$.

Here, Adversary 4 will learn $a, \alpha, \beta b, \dots, \zeta z, \alpha \cdots \zeta, \alpha \cdots \zeta a \cdots z$. In this case, Adversary 4 will be able to learn about the remaining random number $\beta \cdots \zeta$ from random numbers α and $\alpha \cdots \zeta$. Moreover, Adversary 4 will also be able to learn the partial result $b \cdots z$ from input a and the result $a \cdots z$. However, as each secret input and random number cannot be individually separated, the individual secret inputs b, \dots, z will not be leaked. In addition, the same arguments also hold even when Adversary 4 controls a player other than the player who inputted a .

Therefore, we can state that even for Protocol 3.6 that realizes many-inputs multiplication, information-theoretic security can be achieved against Adversary 4.

In contrast, in conventional methods such as the SPDZ method, an additional process for generating shares of multiplication triple using SHE will be required for each multiplication performed. Therefore, conventional methods cannot efficiently perform exponential computation.

3.4.2 Many-Inputs Addition/Subtraction

Computation for addition/subtraction with many inputs can also be computed using Protocol 3.5 shown in Section 3.3; however, this will also be very inefficient. In this section, we will show a more efficient method of realizing addition/subtraction with many inputs by extending Protocol 3.5 as follows:

Protocol 3.7: Many Inputs Addition / Subtraction Protocol

Input:

- Shares of secret input a from Protocol 3.1: $[\alpha a]_j, [\alpha_0]_j, \dots, [\alpha_{k-1}]_j$ ($j = 0, \dots, k-1$)
- Shares of secret input b from Protocol 3.1: $[\beta b]_j, [\beta_0]_j, \dots, [\beta_{k-1}]_j$ ($j = 0, \dots, k-1$)
- Shares of secret input z from Protocol 3.1: $[\zeta z]_j, [\zeta_0]_j, \dots, [\zeta_{k-1}]_j$ ($j = 0, \dots, k-1$)

Output: $[\gamma(a \pm \dots \pm z)], [\gamma_0]_i, \dots, [\gamma_{k-1}]_i$ ($i = 0, \dots, n-1$)

1. Each server S_j collects k shares of the following and reconstructs random numbers α_j, \dots, ζ_j :

$$[\alpha_j]_0, \dots, [\alpha_j]_{k-1}, \dots, [\zeta_j]_0, \dots, [\zeta_j]_{k-1}$$

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

$$\frac{\gamma_j}{\alpha_j}, \dots, \frac{\gamma_j}{\zeta_j}$$

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

$$\frac{\gamma}{\alpha} = \prod_{j=0}^{k-1} \frac{\gamma_j}{\alpha_j}, \dots, \frac{\gamma}{\zeta} = \prod_{j=0}^{k-1} \frac{\gamma_j}{\zeta_j}$$

4. Each server S_i computes the following:

$$[\gamma(a \pm \dots \pm z)]_i = \left(\frac{\gamma}{\alpha} \times [\alpha a]_i \right) \pm \dots \pm \left(\frac{\gamma}{\zeta} \times [\zeta z]_i \right)$$

5. Each server S_j distributes random number γ_j to all servers using Shamir's (k, n) method.
6. Each server S_i holds the following:

$$[\gamma(a \pm \dots \pm z)]_i, [\gamma_0]_i, \dots, [\gamma_{k-1}]_i$$

Evaluation of security against Adversary 1-4

Adversary 1 knows about $\gamma/\alpha, \dots, \gamma/\zeta$ from Step 3 of Protocol 3.7. From this information, it will also learn about the ratio of each random number α, \dots, ζ . In addition, Adversary 1 will learn random numbers $\gamma_l, \alpha_l, \dots, \zeta_l$ ($l = 0, \dots, k-2$) from $k-1$ servers. Therefore, this adversary will learn about the ratio of the remaining partial random numbers from the ratio of each random number α, \dots, ζ and $\gamma_l, \alpha_l, \dots, \zeta_l$ ($l = 0, \dots, k-2$).

However, as the information of each encrypted secret and the computation result cannot be learned from $k-1$ servers, as discussed in Section 3.3.3, Adversary 1 will not be able to learn about each individual secret input and output. In the case of Adversaries 2 and 3, the adversaries will learn about random numbers $\gamma, \alpha, \dots, \zeta$; however, as the encrypted secrets and the computation result are not reconstructed during the computation, as discussed in Section 3.3.3, the secret inputs will not be leaked.

Finally, Adversary 4 knows each random number $\gamma, \alpha, \dots, \zeta$, encrypted result $\gamma(a \pm \dots \pm z)$ and the result of the computation ($a \pm \dots \pm z$); however, because each secret input cannot be individually separated from this information, Adversary 4 will only be able to learn its own input and the result of the addition/subtraction. In other words, we can also state that Protocol 3.7 for many-inputs addition/subtraction is also information-theoretical secure against Adversaries 1–4.

3.5 Limitation of the TUS 1 Method

In this section, we discuss the limitation of the TUS 1 Method. Section 3.3.3 showed that the individual protocol for computing multiplication and addition is secure against Adversaries 1–3. Moreover, in Sections 3.4.1 and 3.4.2, we showed that multiplication and addition with multiple inputs are also secure against Adversaries 1–4 as long as only a single type of operation is performed at any time. However, a question remains regarding the computation that involves the combination of two types of different operations, such as multiplication and addition, to realize a more complex computation (e.g., the product-sum operation of $ab + c$). Here, we will discuss computation that involves the combination of two protocols in Section 3.3.2 and corresponding security using the proposed method.

As follows, we show the protocol for computing the product-sum operation of $ab + c$ by combining the protocols in Section 3.3.2. In this protocol, steps 1–5 perform multiplication of ab , whereas steps 6–10 perform the addition of ab and c . Note that all secret inputs are elements of finite field $GF(p)$, all random numbers used and generated are uniformly distributed from $GF(p)$, and 0 is excluded. In addition, all computations including the distribution of secret inputs are performed in a finite field with prime p .

Protocol 3.8: Product-sum operation $ab + c$

Input:

- Shares of secret input a from Protocol 3.1: $[\![\alpha a]\!]_j, [\![\alpha_0]\!]_j, \dots, [\![\alpha_{k-1}]\!]_j$ ($j = 0, \dots, k-1$)
- Shares of secret input b from Protocol 3.1: $[\![\beta b]\!]_j, [\![\beta_0]\!]_j, \dots, [\![\beta_{k-1}]\!]_j$ ($j = 0, \dots, k-1$)
- Shares of secret input c from Protocol 3.1: $[\![\lambda c]\!]_j, [\![\lambda_0]\!]_j, \dots, [\![\lambda_{k-1}]\!]_j$ ($j = 0, \dots, k-1$)

Output: $[\![\gamma(ab+c)]\!]_i, [\![\gamma_0]\!]_i, \dots, [\![\gamma_{k-1}]\!]_i$ ($i = 0, \dots, n-1$)

1. Server S_0 collects k shares $[\![\alpha a]\!]_j$ from k servers, restores αa , and sends it to all servers.
2. Each server S_i computes the following:

$$[\![\alpha\beta ab]\!]_i = \alpha a \times [\![\beta b]\!]_i$$

3. Each server S_j collects k shares of the following, restores random numbers α_j, β_j , and computes $\alpha_j\beta_j$:

$$\begin{aligned} &[\![\alpha_j]\!]_0, \dots, [\![\alpha_j]\!]_{k-1}, \\ &[\![\beta_j]\!]_0, \dots, [\![\beta_j]\!]_{k-1} \end{aligned}$$

4. Servers S_j distribute $\alpha_j\beta_j$ to all servers S_i using Shamir's (k, n) method.
5. Each server S_i now holds the following as shares for the result ab :

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

6. Each server S_j collects k shares of the following and restores $\alpha_j\beta_j, \lambda_j$. Server S_j then generates a random number γ_j , compute $\gamma_j/\alpha_j\beta_j, \gamma_j/\lambda_j$, and send them to server S_0 .

$$\begin{aligned} &[\![\alpha_j\beta_j]\!]_0, \dots, [\![\alpha_j\beta_j]\!]_{k-1}, \\ &[\![\gamma_j]\!]_0, \dots, [\![\gamma_j]\!]_{k-1} \end{aligned}$$

7. Server S_0 computes the values of $\gamma/\alpha\beta, \gamma/\lambda$ as follows and sends them to all servers:

$$\begin{aligned} \frac{\gamma}{\alpha\beta} &= \prod_{j=0}^{k-1} \frac{\gamma_j}{\alpha_j\beta_j} \\ \frac{\gamma}{\lambda} &= \prod_{j=0}^{k-1} \frac{\gamma_j}{\lambda_j} \end{aligned}$$

8. Each server S_i computes the following:

$$[\![\gamma(ab+c)]\!]_i = \frac{\gamma}{\alpha\beta} \times [\![\alpha\beta ab]\!]_i + \frac{\gamma}{\lambda} \times [\![\lambda c]\!]_i$$

9. Server S_j distributes random number γ_j to all servers S_i using Shamir's (k, n) method.
10. Each server S_i holds the following as shares of $ab + c$:

$$\llbracket \gamma(ab + c) \rrbracket_i, \llbracket \gamma_0 \rrbracket_i, \dots, \llbracket \gamma_{k-1} \rrbracket_i$$

Evaluation of security against Adversary 4

As Protocol 3.8 for the product-sum operation is a three-input-one-output operation, in which three players inputted secret inputs a, b, c and one player reconstructed the result $ab + c$, we also need to assume the existence of Adversary 4, where the adversary controls one of the players who inputted a secret and the player who reconstructed the output.

Let us assume that Adversary 4 controls the player who inputted secret b (and random number β) and the player who reconstructed output $ab + c$ (and random number γ). In addition, Adversary 4 also has information of $\alpha a, \gamma/\alpha\beta, \gamma/\lambda$ obtained from executing Protocol 3.8. By using the information of $\beta, \gamma, \gamma/\alpha\beta$, Adversary 4 can gain information of random number α used to encrypt secret input a . With the information of encrypted secret input αa and random number α , Adversary 4 will be able to decrypt secret input a . Furthermore, with information of secret inputs a, b , and output $ab + c$, information of secret input c will also be leaked to Adversary 4.

Thus, we can conclude that, when Adversary 4 has information about secret input b , output $ab + c$, and additional information from $k - 1$ servers, the remaining two secret inputs a, c will be eventually leaked. Therefore, the proposed method in Section 3.3.2 is not secure against Adversary 4 when combined to perform a more complex computation such as the product-sum operation of $ab + c$. However, Protocol 3.8 shown above remains information-theoretic secure against Adversaries 1–3.

3.6 Discussion

3.6.1 Computational and Communication Costs of TUS 1 Method

The quantitative evaluation regarding communication cost, round number, and computational cost of the TUS 1 method is shown in Tables 3.1 and 3.2. The number of communications was evaluated in terms of the round according to the direction of the communication. First, we define the parameters used in our evaluation.

Definition of Parameters:

- d_1 : Size of a share from Shamir's (k, n) method.
- C_1 : Computational cost of Shamir's (k, n) method.
- M : Computational cost of multiplication.

TABLE 3.1: Communication cost and rounds of the TUS 1 method

Process		Communication Cost	Total	Round
Distribution of a, b	Step 2	$2nd_1(k+1)$	$2nd_1(k+1)$	1
	Step 1	kd_1, nd_1		
Multiplication of ab	Step 3	$2k^2d_1$	$(k+n+2k^2+nk)d_1$	4
	Step 4	nkd_1		
	Step 1	$2k^2d_1$		
Addition of $a+b$	Step 2	$2kd_1$	$(2k^2+2k+2n+nk)d_1$	4
	Step 3	$2nd_1$		
	Step 5	nkd_1		
	Step 1	$k(k+1)d_1$		
Reconstruction	Step 1	$k(k+1)d_1$	$k(k+1)d_1$	1

TABLE 3.2: Computational cost of the TUS 1 method

Process		Computation Cost	Total
Distribution of a, b	Step 1	$2(k-1)M$	$2kM+2(k+1)C_1$
	Step 2	$2M, 2(k+1)C_1$	
Multiplication of ab	Step 1	C_1	$(n+k)M+(3k+1)C_1$
	Step 2	nM	
	Step 3	$2kC_1$	
	Step 4	kM, kC_1	
Addition of $a+b$	Step 1	$2kC_1$	$2(k-1+n)M+2kD+nA+3kC_1$
	Step 2	$2kD$	
	Step 3	$2(k-1)M$	
	Step 4	$2nM+nA$	
Reconstruction	Step 5	kC_1	$kM+(k+1)C_1$
	Step 2	$(k+1)C_1, kM$	

- D : Computational cost of division.
- A : Computational cost of addition.

Note that in Shamir's (k, n) method, share d_1 usually has almost the same size as the original secret, and the computational costs are different for the distribution and the reconstruction processes. However, we considered these computational costs as the same for ease of understanding.

3.6.2 Qualitative Comparison with SPDZ Method

Conventional methods for realizing computation in the setting of $n = k$ are proposed by Damgård et al. in 2012 and 2013 in the SPDZ [19] and SPDZ 2 [35] methods, respectively. The SPDZ 2 method is the improved version of the SPDZ method with the same process

performing multiplication ab . Therefore, we performed a comparison only with the SPDZ method.

The SPDZ method is only limited to the setting of $n = k$, where the owner of the secret inputs is also one of the computing players (or servers), and the protocol is secure even if $n - 1$ out of n participating players collude together. In particular, the SPDZ method realizes security against a malicious adversary with dishonest majority. The SPDZ method is composed of an offline phase that performs a preprocessing process using SHE and an online phase. Therefore, the security of this method is computational security and requires a considerable amount of computation in the offline phase. In addition, the SPDZ method is secure against the combination of operations such as the combination of multiplication and addition to realize the product-sum operation of $ab + c$. When performing division from shares of secrets a and b , shares of $1/a$ and b are required; however, to find the shares of $1/a$ from shares of a , an additional computation will be required in addition to the preprocessing process (for example, distribute the secret random number r , multiply with shares of a , broadcast the value of ar , multiply $1/ar$ with shares of random number r to obtain shares of $1/a$).

In contrast, in our proposed method in Section 3.3, the parameters n, k are not limited to when $n = k$, but they also effective when $n \geq k$ (we demonstrated the effectiveness of our method for $n < 2k - 1$, but it is still applicable when $n \geq 2k - 1$). The adversary will attempt to analyze the information obtained to learn about the secret inputs, but we only assume a semi-honest adversary and realize information-theoretic security against it. In addition, in our method, as long as the combination of computation only involves a single type of operation (such as addition or multiplication), the computation can be repeated without any limit to the number of inputs. Our proposed method has a limitation related to the security against computations that include a combination of different operations such as $ab + c$. However, as shown in Section 3.3, our method could realize division directly from the shares of a, b .

Furthermore, MPC of multiplication with the setting of $n < 2k - 1$ can be performed with the condition that the secret input does not include the value of 0. Information that does not include the value 0 exists in several fields. For instance, in medical data, the pulse and blood pressure are positive values, and the value 0 means that the individual is no longer alive and is not used for medical statistical computation; moreover, the blood glucose level is also a positive value. Therefore, in general, not including the value 0 is not a problem in the case of secure computation of information such as patients in a hospital or under treatment. The comparison explained above is included in Table 3.3.

3.6.3 Quantitative Comparison with SPDZ Method

The comparison of communication and computational costs between the TUS 1 method and the SPDZ method is shown in Tables 3.4, 3.5 and 3.6. We defined the parameters used in the comparison as follows.

TABLE 3.3: Comparison with the SPDZ method

	Proposed Method	SPDZ Method
Parameters n and k	$n \geq k$	$n = k$
Type of adversary	Semi-honest	Malicious
Security	Information-theoretic	Computational
Combination of operation?	Yes (same type only)	Yes
Division?	Directly	Additional processes are required
Condition(s) required	Exclude 0 for multiplication	No

Definition of Parameters

- d_1 : Size of share from secret sharing
- d_2 : Size of share from SHE
- C_1 : Computational cost of secret sharing
- C_2 : Computational cost of SHE

Here, the share size of Shamir's (k, n) method is d_1 . In our proposed method, for each secret input, $k + 1$ times of distribution are performed, and the total share size for the distribution process is $d_1(k + 1)$. Moreover, the data size of a SHE is d_2 , the computational cost for secret sharing is C_1 , and the computational cost for SHE is C_2 . In addition, the computational cost for a typical multiplication/division is M , and that for addition is represented by A . The shares size d_1 of Shamir's (k, n) method can be made to be almost the same as the size of the secret input; however, the data size d_2 of SHE is typically considerably larger than the secret input. Therefore, $d_2 > d_1$. However, because the total share size of our proposed method is $k + 1$ times larger than that of conventional methods of computation that use Shamir's (k, n) method, it is larger than the data size d_2 of SHE.

In the SPDZ method, in addition to the shares d_1 generated from additive secret sharing, additional information of multiplication triple needed during the multiplication process also need to be considered. Therefore, when comparing the total share size, we can state that our proposed method still requires less share size than the SPDZ method. In addition, C_1 and C_2 typically have the relation of $C_1 \ll C_2$, and C_2 is considerably larger than C_1 . In secret sharing, the computational costs required for distribution and reconstruction are different; as both costs are considerably lower than C_2 , we represented both using C_1 for ease of understanding.

In addition, M and A are both considerably lower than C_1, C_2 ; therefore, we included them in Tables 3.1 and 3.2. When comparing the total cost, the costs for M and A are omitted when either C_1 or C_2 is present. In addition, because our proposed method does not include

any verification process, we omitted all costs for verification such as zero-knowledge proof and message authentication code (MAC) during the comparison.

From Table 3.4, because our proposed method requires the distribution and reconstruction of k random numbers during the distribution protocol and computation process, the computation cost during addition is inferior to the SPDZ method. However, during multiplication, as our method does not require any preprocessing of SHE with the cost C_2 , we can state that our computation cost is far lower than that of the SPDZ method. In addition, when comparing the total costs of addition and multiplication between our method and the SPDZ method, our method could realize a faster computation than the SPDZ method because it does not include the cost of C_2 overall.

In addition, regarding the communication cost, the merits and demerits of each method depend on the parameters n, k, d_1, d_2 . However, regarding the round number, the total round number of our method was higher than that of the SPDZ method because it includes the reconstruction and distribution of k random numbers.

From the described above, when comparing our proposed method and the SPDZ method, our proposed could realize a lighter process. In contrast, when considering the operation model, in the SPDZ method, the player who inputted the secret input is also one of the computing players. However, in modern data analysis, it is not practical for the player who inputted the secret input to directly participate in the computation. In addition, when performing computation using a huge number of inputs, as the owner of the information also participates in the computation as a computing player, there is a need to set for a huge parameter of $n = k$. Generally, in big data analysis, a large amount of information is encrypted and stored in cloud servers, and their statistical values such as their average value and variance need to be calculated without reconstructing them. In this case, as shown in Section 3.5, every owner of inputs $a - z$ can distribute them to the cloud servers using our proposed method, and only k out of the n computing servers will perform computation. This scenario can be operated more efficiently in big data analysis and is more realistic.

Moreover, in the proposed method, to minimize the size of the computing servers needed, the parameters can be set to $n = k = 2$. In the conventional method of computation that uses Shamir's (k, n) method, as $n \geq 2k - 1$ servers are required, multiplication is not possible when $n = k = 2$.

In contrast, the SPDZ method requires a huge computation cost in the offline phase. In addition, in our proposed method, when considering the resistance toward server loss in the cloud servers, the parameters can also be set to $k = 3, n = 4$ or $k = 4, n = 5$ (in conventional methods where $n \geq 2k - 1$: if $k = 3, n \geq 5$; if $k = 4, n \geq 7$). In addition, the SPDZ method cannot address cases other than $n = k$. From the above, the proposed method can set the number of computing servers at the minimum value of two regardless of the multiplication operation and execute the secure computation with a small amount of computation cost. Therefore, our proposed method can enable a more flexible process than the SPDZ method.

TABLE 3.4: Comparison with the SPDZ method (computational cost)

Process	Proposed Method	SPDZ Method
Multiplication of ab	$(6k + 4)C_1$	$6C_1 + 2C_2$
Addition of $a + b$	$(6k + 3)C_1$	$3C_1$

TABLE 3.5: Comparison with the SPDZ method (communication cost)

Process	Proposed Method	SPDZ Method
Multiplication of ab	$(3n(k + 1) + k(3k + 2))d_1$	$10nd_1 + 2nd_2$
Addition of $a + b$	$(n(3k + 4) + 3k(k + 1))d_1$	$3nd_1$

3.7 Chapter Summary

In this chapter, we propose a new method of conditionally secure MPC by using the (k, n) threshold secret sharing, which is valid even for $n < 2k - 1$ under the condition that the secret inputs of the computation do not include the value of 0 during multiplication. This method can execute secure computation even when $n < 2k - 1$, and can realize information-theoretic security. As a result, the number of servers does not need to be changed even if multiplication is included, allowing for a more flexible operation. However, our next task is to study a method that can safely realize a combination of different types of operations (for example, product-sum operation of $ab + c$) and evaluate its security.

TABLE 3.6: Comparison with the SPDZ method (round)

Process	Proposed Method	SPDZ Method
Multiplication of ab	6	5
Addition of $a + b$	6	2

Chapter 4

An Improved Conditionally Secure MPC

In Chapter 3, we proposed a method of secure MPC when $n < 2k - 1$ with the condition that the secret input for multiplication does not include 0. Moreover, it was shown that the proposed method is secure against Adversaries 1–3, and that computation with many inputs such as many-inputs addition is also secure against Adversaries 1–4. However, the following question may arise here:

“We also need to realize a secure MPC that is capable of computing computations that involve a combination of different operations, such as the computation of product-sum $ab + c$. This is particularly important when considering use in big data analysis where computations such as variance and average often include combinations of different operations.”

In this chapter, we consider the aforementioned problem. This chapter mainly discusses an approach to securely compute computations involving different operations and introduces two more conditions required to realize information-theoretic security against semi-honest adversaries.

4.1 Introduction

Conventional methods of MPC using secret sharing can easily perform addition (and subtraction). However, this is not the case of multiplication, where the degree of polynomial changes from $k - 1$ to $2k - 2$ for each multiplication performed. To restore the multiplication result, the number of shares required increases from k to $2k - 1$. As one server usually holds only one share, the number of computing servers required will increase.

In Chapter 3, we showed an MPC method using (k, n) threshold secret sharing. However, in the method described, when computation involving a combination of operations, such as that of product-sum $ab + c$, is performed, if the adversary has information about one of the inputs and output, the values of the remaining two inputs can be specified.

This chapter proposes an improved MPC using (k, n) threshold secret sharing that is secure even when computation involving a combination of multiple different operations is performed. We also considered the conditions needed to achieve an information-theoretic secure MPC using secret sharing scheme in the setting of $n < 2k - 1$. If the conditions can be realized, we can state that the proposed MPC method is practical for real-world applications. However, we only assumed a semi-honest adversary. In addition, we verified the effectiveness of our proposed method by comparing it with the SPDZ method and the method in Chapter 3.

4.2 Proposed Method: TUS 2 Method

4.2.1 Overview of TUS 2 Method

In this chapter, we describe our improved proposed method of secure MPC when $n < 2k - 1$ that enables combinations of different operations. For simplicity, this method is known as TUS 2 method, considered to overcome the problem of the method in Chapter 3. As explained in Section 3.5, when multiplication and addition are combined to compute product-sum operation $ab + c$, the remaining input can be leaked when the adversary knows one of the inputs and outputs of the computation.

To prevent the remaining information from being leaked we propose an approach in which random numbers that are not known to the adversary are implemented. Therefore, we need to define a new condition in which sets of shares of random numbers that are unknown to the adversary are prepared and stored in the computing servers beforehand. To facilitate this, we assume that sets of shares derived from random numbers δ_j, η_j ($j = 0, 1, \dots, k-1$) are not known to the adversary, as shown below. These shares are called *set of conversion random numbers*, where δ and η are random numbers unknown to the adversary.

- Set of conversion random number δ : $[\delta]_i = ([\delta]_i, [\delta_0]_i, \dots, [\delta_{k-1}]_i)$
- Set of conversion random number η : $[\eta]_i = ([\eta]_i, [\eta_0]_i, \dots, [\eta_{k-1}]_i)$

For example, this set of shares can be easily prepared by using the protocol below:

Protocol 4.1: Generation of a set of conversion random number δ

1. Generate k random numbers $\delta_0, \delta_1, \dots, \delta_{k-1}$.
2. Calculate random number δ as follows:

$$\delta = \prod_{j=0}^{k-1} \delta_j$$

3. Distribute random numbers $\delta, \delta_0, \dots, \delta_{k-1}$ to servers S_i ($i = 0, \dots, n-1$) using Shamir's (k, n) method.

4. Each server S_i holds the following as a set of conversion random number δ .

$$[\delta]_i = ([\delta]_i, [\delta_0]_i, \dots, [\delta_{k-1}]_i)$$

Hence, we define condition (2) as follows.

2. There are sets of conversion random numbers derived from random numbers that are unknown to the adversary.

4.2.2 Protocol of TUS 2 Method

In this section, we explain the protocol for our MPC method. Instead of separating the protocol for addition and multiplication, we propose the protocol for computing product-sum operation $ab + c$. Thus, when the input a is replaced with $a = 1$, we can realize the computation of addition $b + c$; when the input c is replaced with $c = 0$, we can realize the computation for multiplication ab . Therefore, the protocol for $ab + c$ is more flexible as it can be used to compute all four arithmetic operations.

In sequence, we describe the protocol for the computation of $ab + c$. However, we assume that every server holds shares of secret inputs that were distributed by the player using Protocol 3.1 shown in Section 3.3. Moreover, all computations, including the distribution protocol, are performed in finite field $GF(p)$.

Notation:

- $[\alpha a]_i$: Share of αa held by server S_i .

Conditions:

1. Inputs in the multiplication protocol do not include the value 0.
2. There are sets of conversion random numbers derived from random numbers (excluding the value 0) unknown to the adversary. Here, we assume that each server S_i ($i = 0, \dots, n-1$) holds the following:

Set of conversion random number δ : $[\delta]_i = ([\delta]_i, [\delta_0]_i, \dots, [\delta_{k-1}]_i)$ ($i = 0, \dots, n-1$)

Set of conversion random number η : $[\eta]_i = ([\eta]_i, [\eta_0]_i, \dots, [\eta_{k-1}]_i)$ ($i = 0, \dots, n-1$)

Protocol 4.2: Product-sum operation $ab + c$

Input:

- Shares of secret input a from Protocol 3.1: $[\alpha a]_j, [\alpha_0]_j, \dots, [\alpha_{k-1}]_j$ ($j = 0, \dots, k-1$)
- Shares of secret input b from Protocol 3.1: $[\beta b]_j, [\beta_0]_j, \dots, [\beta_{k-1}]_j$ ($j = 0, \dots, k-1$)
- Shares of secret input c from Protocol 3.1: $[\lambda c]_j, [\lambda_0]_j, \dots, [\lambda_{k-1}]_j$ ($j = 0, \dots, k-1$)

Output: $[[\gamma(ab+c)]]_i, [[\gamma_0]]_i, \dots, [[\gamma_{k-1}]]_i$ ($i = 0, \dots, n-1$)

1. Server S_0 collects k shares of $[[\alpha a]]_j$ ($j = 0, \dots, k-1$) from k servers, restores αa , and sends αa to all servers S_i .
2. Each server S_i ($i = 0, \dots, n-1$) computes the following:

$$[[\alpha\beta ab]]_i = \alpha a \times [[\beta b]]_i$$

3. Server S_0 collects the following from k servers, restores $\alpha\beta ab, \lambda c$, and sends to all servers S_i :

$$[[\alpha\beta ab]]_j, [[\lambda c]]_j$$

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

$$[[\alpha\beta\delta ab]]_i = \alpha\beta ab \times [[\delta]]_i$$

$$[[\lambda\eta c]]_i = \lambda c \times [[\eta]]_i$$

5. Each server S_j ($i = 0, \dots, k-1$) collects k shares of the following and restores $\alpha_j, \beta_j, \lambda_j, \delta_j, \eta_j$:

$$[[\alpha_j]]_0, \dots, [[\alpha_j]]_{k-1},$$

$$[[\beta_j]]_0, \dots, [[\beta_j]]_{k-1},$$

$$[[\lambda_j]]_0, \dots, [[\lambda_j]]_{k-1},$$

$$[[\delta_j]]_0, \dots, [[\delta_j]]_{k-1},$$

$$[[\eta_j]]_0, \dots, [[\eta_j]]_{k-1}$$

6. Each server S_j ($i = 0, \dots, k-1$) generates a random number γ_j , computes the following, and sends to one of the servers (here, we assume server S_0):

$$\frac{\gamma_j}{\alpha_j\beta_j\delta_j}, \frac{\gamma_j}{\lambda_j\eta_j}$$

7. Server S_0 computes the following and sends it to all servers S_i ($i = 0, \dots, n-1$):

$$\frac{\gamma}{\alpha\beta\delta} = \prod_{j=0}^{k-1} \frac{\gamma_j}{\alpha_j\beta_j\delta_j}$$

$$\frac{\gamma}{\lambda\eta} = \prod_{j=0}^{k-1} \frac{\gamma_j}{\lambda_j\eta_j}$$

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

$$\llbracket \lambda(ab+c) \rrbracket_i = \left(\frac{\gamma}{\alpha\beta\delta} \times \llbracket \alpha\beta\delta ab \rrbracket_i \right) + \left(\frac{\gamma}{\lambda\eta} \times \llbracket \lambda\eta c \rrbracket_i \right)$$

9. Server S_j ($i = 0, \dots, k-1$) distributes random number γ_j (if the computation involves only multiplication of ab , $\gamma_j = \alpha_j\beta_j$; if the computation involves addition, $\gamma_j = \gamma_j$) to all servers S_i ($i = 0, \dots, n-1$) by using Shamir's (k, n) method.

10. Each server S_i ($i = 0, \dots, n-1$) holds the following as shares of the result $ab+c$.

$$\llbracket \gamma(ab+c) \rrbracket_i, \llbracket \gamma_0 \rrbracket_i, \dots, \llbracket \gamma_{k-1} \rrbracket_i$$

Based on Protocol 4.2 for the product-sum $ab+c$ shown above, Steps 1–2 are the same as those of Protocol 3.3 of multiplication, and Steps 6–9 are based on Protocol 3.5 for addition. Therefore, Steps 3–5 are new steps that require shares of random numbers unknown to the adversary, as stated in Condition (2).

As previously mentioned, in a product-sum operation of $ab+c$, if $c=0$, multiplication of ab can be realized, and if $a=1$, addition of $b+c$ can be realized. Therefore, in the product-sum operation protocol previously mentioned, if $c=0$, multiplication is achieved, and Steps 3, 4 and 6–8 can be skipped. However, in Step 5, only α_j, β_j is reconstructed without the need to generate γ_j . Moreover, if multiplication in Step 3 is replaced with division by αa , $\gamma_j = \alpha_j\beta_j$ in Step 9 is replaced with $\gamma_j = \beta_j/\alpha_j$, and division is realizable.

If $a=1$ (random number α is also equal to 1), addition is achieved, and Steps 1, 2 can be skipped. However, because multiplication in Step 2 is skipped, computation after Step 3, which involves $\llbracket \alpha\beta ab \rrbracket_j$ and $\alpha\beta ab$, becomes $\llbracket \beta b \rrbracket_i$ and βb , $\llbracket \alpha\beta\delta ab \rrbracket_i$ in Step 4 becomes $\llbracket \beta\delta b \rrbracket_i$, reconstruction of α_j in Step 5 is skipped, and α_j in Step 6 and onwards is set as $\alpha_j = \alpha = 1$.

In addition, subtraction is also realizable by changing the symbol of addition to subtraction. In conclusion, we can achieve all four basic arithmetic computations based on the aforementioned product-sum operation protocol, including division and subtraction. However, the evaluation of the effectiveness of our proposed method through the introduction of Steps 3–5 is discussed in a later section.

4.2.3 Security of TUS 2 method

The method proposed in Chapter 3 is a two-input-one-output operation and can be represented by Figure 4.1, where two inputs $\llbracket a \rrbracket_j, \llbracket b \rrbracket_j$ are inputted in the secure computation box, which contains the protocols shown in Section 3.3, to produce output $\llbracket c \rrbracket_i$. This method was proven to be secure against Adversary 1, which has access to information from $k-1$ servers, Adversary 2, which has information on secret input a or b in addition to information from

$k - 1$ servers, and Adversary 3, who has information on the output c in addition to information from $k - 1$ servers.

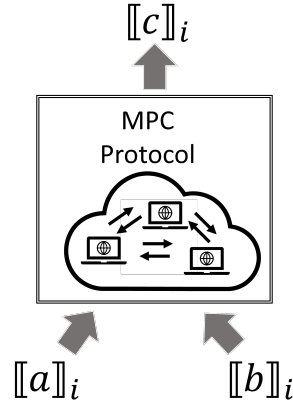


FIGURE 4.1: Basic computation of TUS 1 method.

As our proposed method in Protocol 4.2 is a three-input-one-output computation, it can be represented by Figure 4.2, where three inputs $[[a]]_j$, $[[b]]_j$, $[[c]]_j$ are inputted into the MPC box to produce an output $[[d]]_j$. However, if one of the outputs is known (for example, $a = 1$ or $c = 0$), we can realize a two-input-one-output computation of addition, subtraction, multiplication, and division, as in Figure 4.1. Here, when evaluating the security of $ab + c$, in addition to Adversaries 1–4 defined in Chapter 3, we include a new Adversary 4 and Adversary 5 as follows. The attack is considered a success if the adversary can learn the target information.

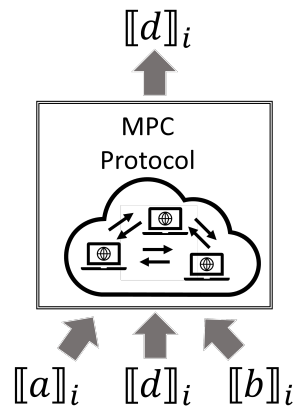


FIGURE 4.2: Basic computation of TUS 2 method.

Adversary 4: In the product-sum operation, one of the players who inputted the secret input and the player who reconstructed the output constitute the adversary. Adversary 4 has information of one of the inputs (and the random number used to encrypt it) and the information

needed to reconstruct the output. In addition, it also knows information from $k - 1$ servers. The adversary attempts to learn the remaining two inputs according to this information.

Adversary 5: In the product-sum operation, two players who inputted secret inputs constitute the adversary. Adversary 5 has information on two secret inputs (and the random numbers used to encrypt them). In addition, it also knows information from $k - 1$ servers. According to this information, the adversary attempts to learn the remaining input or the output of the computation.

Here, suppose that in the case of Adversary 4, but with the one known input treated as a constant number that does not contribute to the process of decoding other secret inputs, this type of adversary can be treated the same as Adversary 3, where the adversary knows only the output. In contrast, in the case of Adversary 5, if one of the known inputs is assumed to be constant and does not contribute to decoding other secret inputs, Adversary 5 can be treated the same as Adversary 2 that knows only one of the inputs.

Furthermore, Adversaries 4 and 5 have the information obtained by Adversary 1, which is information from $k - 1$ servers. Moreover, in a three-input-one-output computation, regardless of the security level of the method used, if two out of the three inputs and the output are leaked to the adversary, the remaining one input can also be leaked. Similarly, when all three inputs are known to the adversary, the output can also be leaked. Therefore, we do not consider these two types of adversaries. We can state that our proposed method is secure if it is also secure against Adversaries 4 and 5.

In the following, we evaluate the security of our proposed method against Adversaries 4 and 5.

Evaluation of security against Adversary 4

Assume that Adversary 4 controls the player who inputted secret input b . Adversary 4 also has information from $k - 1$ servers. Therefore, in the process of inputting secret input b through Protocol 3.1, Adversary 4 has information about b, β, β_j ($j = 0, \dots, k - 1$), and in Steps 1–2, he/she learns about αa , in Steps 3–4 about $\alpha\beta ab, \lambda c$, in Steps 5–6 about $\alpha_l, \beta_l, \lambda_l, \delta_l, \eta_l, \gamma_l$ ($l = 0, \dots, k - 2$), in Step 7 about $\gamma/\alpha\beta\delta, \gamma/\lambda\eta$, and finally in the reconstruction process using Protocol 3.2, about $\gamma, \gamma_j, ab + c$ ($j = 0, \dots, k - 1$).

Therefore, the evaluation of security against Adversary 4 can be translated to the problem of determining whether he/she can learn about the remaining secret inputs a, c from the following information:

$$b, \beta, \alpha\beta ab, \lambda c, \frac{\gamma}{\alpha\beta\delta}, \frac{\gamma}{\lambda\eta}, \gamma, ab + c, \beta_j, \gamma_j \quad (j = 0, \dots, k - 1)$$

$$\alpha_l, \beta_l, \lambda_l, \delta_l, \eta_l, \gamma_l \quad (l = 0, \dots, k - 2)$$

First, to simplify the problem, we redefined the parameters above to avoid any duplication

of a parameter. As a result, we can transform the problem into determining whether the adversary can learn about the remaining secret inputs a, c from the following information:

$$b, \beta, \alpha a, \lambda c, \gamma, \alpha \delta, \lambda \eta, ab + c, \alpha_l, \lambda_l, \delta_l, \eta_l \quad (l = 0, \dots, k-2)$$

To obtain information about secret input a from αa , the adversary must first obtain information of random number α . The information that is related to random number α is $\alpha a, \alpha \delta, \alpha_l, \delta_l$ ($l = 0, \dots, k-2$) ($b, \beta, c, \lambda, \eta$ are independent of α, a). However, even from this information, random number α and secret input a are not leaked. Therefore,

$$\begin{aligned} H(\alpha) &= H(\alpha | \alpha_0, \dots, \alpha_{k-2}) \\ H(\delta) &= H(\delta | \delta_0, \dots, \delta_{k-2}) \\ H(a) &= H(a | \alpha a, \alpha \delta, \alpha_0, \dots, \alpha_{k-2}, \delta_0, \dots, \delta_{k-2}) \end{aligned}$$

In the method proposed in Chapter 3, because there was no implementation of shares of random number δ unknown to the adversary, the value of $\alpha \delta$ becomes α and secret input a can be leaked. In contrast, in Protocol 4.2, through the implementation of shares of random number δ , we were able to prevent the leakage of secret input a to the adversary.

In addition, to obtain secret input c from λc , the adversary needs first to learn about random number λ ; therefore, the same can also be asserted about secret input c .

$$\begin{aligned} H(\lambda) &= H(\lambda | \lambda_0, \dots, \lambda_{k-2}) \\ H(\eta) &= H(\eta | \eta_0, \dots, \eta_{k-2}) \\ H(c) &= H(c | \lambda c, \lambda \eta, \lambda_0, \dots, \lambda_{k-2}, \eta_0, \dots, \eta_{k-2}) \end{aligned}$$

By implementing the shares of random number η , in contrast to the method in Chapter 3, Protocol 4.2 can prevent random number λ from being leaked, allowing our method to prevent secret input c from being known to the adversary.

Finally, because Adversary 4 also has information about output $ab + c$, it has information about $ab + c$ and b ; however, with no information about secret inputs a or c , it is not able to obtain any information about the remaining inputs. Therefore, the following can be stated:

$$\begin{aligned} H(a) &= H(a | ab + c, \beta, b, \alpha a, \alpha \delta, \lambda c, \lambda \eta, \alpha_l, \delta_l, \lambda_l, \eta_l) \quad (l = 0, \dots, k-2) \\ H(c) &= H(c | ab + c, \beta, b, \alpha a, \alpha \delta, \lambda c, \lambda \eta, \alpha_l, \delta_l, \lambda_l, \eta_l) \quad (l = 0, \dots, k-2) \end{aligned}$$

In addition, the evaluation above remains valid even if Adversary 4 controls the player who inputted secret inputs a or c . Therefore, our proposed method is secure against Adversary 4.

Evaluation of security against Adversary 5

Assume that Adversary 5 controls the player who inputted secret inputs b, c . It also has information from $k - 1$ servers. Therefore, in the process of inputting the secret inputs, Adversary 5 has information of $b, \beta, c, \lambda, \beta_j, \lambda_j$ ($j = 0, \dots, k - 1$), and in Steps 1–2 learns about αa , in Steps 3–4 about $\alpha\beta ab, \lambda c$, in Steps 5–6 about $\alpha_l, \beta_l, \lambda_l, \delta_l, \eta_l, \gamma_l$ ($l = 0, \dots, k - 2$), and finally, in Step 7, about $\gamma/\alpha\beta\delta, \gamma/\lambda\eta$.

Therefore, the evaluation of security against Adversary 5 can be translated into the problem of determining whether the adversary can learn about the remaining secret input a or output $ab + c$ from the following information:

$$b, \beta, c, \lambda, \alpha a, \alpha\beta ab, \lambda c, \frac{\gamma}{\alpha\beta\delta}, \frac{\gamma}{\lambda\eta}, \beta_j, \lambda_j \ (j = 0, \dots, k - 1)$$

$$\alpha_l, \beta_l, \lambda_l, \delta_l, \eta_l, \gamma_l \ (l = 0, \dots, k - 2)$$

First, to simplify the problem, we redefine the parameters above to avoid any duplication of a parameter. As a result, we can change the problem into determining whether the adversary can learn about the remaining secret input a or output $ab + c$ from the following information:

$$b, \beta, c, \lambda, \alpha a, \frac{\gamma}{\alpha\delta}, \frac{\gamma}{\eta}, \alpha_l, \delta_l, \eta_l, \gamma_l, \llbracket \gamma(ab + c) \rrbracket_l \ (l = 0, \dots, k - 2)$$

To obtain information of secret input a from αa , the adversary must first obtain information of random number α . The information related to random number α is $\alpha a, \gamma/\alpha\delta, \alpha_l, \delta_l, \gamma_l$ ($l = 0, \dots, k - 2$). Even from this information, random number α and secret input a cannot be leaked. Therefore, the following statements are true:

$$H(\alpha) = H(\alpha | \alpha_0, \dots, \alpha_{k-2})$$

$$H(\gamma) = H(\gamma | \gamma_0, \dots, \gamma_{k-2})$$

$$H(\delta) = H(\delta | \delta_0, \dots, \delta_{k-2})$$

$$H(a) = H\left(a \mid \frac{\gamma}{\alpha\delta}, \alpha_0, \dots, \alpha_{k-2}, \gamma_0, \dots, \gamma_{k-2}, \delta_0, \dots, \delta_{k-2}\right)$$

In the method in Chapter 3, because there was no implementation of shares of random numbers δ, η that are unknown to the adversary, $\gamma/\lambda\eta$ becomes γ/λ and $\gamma/\alpha\delta$ becomes γ/α . From information of γ/λ and λ , random number γ is leaked to the adversary. With information on γ/α and γ , the adversary can learn the value of random number α . Information of α and αa allows information of secret input a to be leaked. With the leaked information of secret input a and initial information of b, c , the adversary can learn about output $ab + c$. In contrast, our proposed method utilizes shares of random numbers δ, η , which are derived from random numbers that are not known to the adversary; thus, we can prevent secret input a from being leaked to the adversary.

In addition, the adversary has information about $\llbracket \gamma(ab+c) \rrbracket_l$ ($l = 0, \dots, k-2$), but it cannot learn about $\gamma(ab+c)$ from this. Therefore, the following can be stated:

$$H(\gamma(ab+c)) = H(\gamma(ab+c) | \llbracket \gamma(ab+c) \rrbracket_0, \dots, \llbracket \gamma(ab+c) \rrbracket_{k-2})$$

Next, we consider whether the output of the computation and random number γ can be leaked to the adversary. First, information related to random number γ are $\gamma/\alpha\delta$, γ/η , α_l , δ_l , γ_l , η_l ($l = 0, \dots, k-2$). However, even with this information, the adversary cannot learn about random number γ . Therefore:

$$\begin{aligned} H(\gamma) &= H(\gamma | \gamma_0, \dots, \gamma_{k-2}) \\ H(\alpha) &= H(\alpha | \alpha_0, \dots, \alpha_{k-2}) \\ H(\delta) &= H(\delta | \delta_0, \dots, \delta_{k-2}) \\ H(\eta) &= H(\eta | \eta_0, \dots, \eta_{k-2}) \\ H(\gamma) &= H\left(\gamma \left| \frac{\gamma}{\alpha\delta}, \frac{\gamma}{\eta}, \alpha_l, \delta_l, \gamma_l, \eta_l \right. \right) \quad (l = 0, \dots, k-2) \end{aligned}$$

In the method in Chapter 3, because there was no implementation of shares of random number η , $\gamma/\lambda\eta$ becomes γ/λ . With information of $\gamma/\lambda, \lambda$, the adversary can learn about random number γ . In contrast, Protocol 4.2 utilizes shares of random number η , which is not known to the adversary; thus, we can prevent the adversary from learning about random number γ .

Finally, Adversary 5 may know about secret inputs b, c by controlling the players, but with no information of output $ab+c$, it cannot learn about the remaining input. Therefore:

$$\begin{aligned} H(a) &= H\left(a \left| b, \beta, c, \lambda, \alpha a, \frac{\gamma}{\alpha\delta}, \frac{\gamma}{\eta}, \alpha_l, \delta_l, \eta_l, \gamma_l \right. \right) \quad (l = 0, \dots, k-2) \\ H(ab+c) &= H\left(ab+c \left| b, \beta, c, \lambda, \alpha a, \frac{\gamma}{\alpha\delta}, \frac{\gamma}{\eta}, \alpha_l, \delta_l, \eta_l, \gamma_l \right. \right) \quad (l = 0, \dots, k-2) \end{aligned}$$

In addition, the aforementioned evaluation remains valid, even if Adversary 5 controls the player who inputted secret inputs a, c or a, b . Therefore, we can state that our proposed method is secure against Adversary 5.

4.3 Extension of TUS 2 Method: Combination of Multiple Product-Sum Operation

4.3.1 Extended Method

In general, any computation that comprises the four basic operations (addition, subtraction, multiplication, and division) can be decomposed into a combination of multiple product-sum

operations.

For example, the computation of $a = f(a_1, a_2, \dots, a_{2m}, a_{2m+1}) = a_{2m+1}(a_1 a_2 + a_3 a_4 + \dots + a_{2m-1} a_{2m})$ can be divided into multiple combinations of product-sum operations:

$$\text{Product-sum operation 1: } f_1 = f(a_1, a_2, 0) = a_1 a_2$$

$$\text{Product-sum operation 2: } f_2 = f(a_3, a_4, f_1) = a_3 a_4 + a_1 a_2$$

\vdots

$$\text{Product-sum operation } m: f_m = f(a_{2m-1}, a_{2m}, f_{m-1}) = a_1 a_2 + a_3 a_4 + \dots + a_{2m-1} a_{2m}$$

$$\text{Product-sum operation } m+1: f_{m+1} = f(a_{2m+1}, f_m, 0) = a_{2m+1}(a_1 a_2 + a_3 a_4 + \dots + a_{2m-1} a_{2m}) = a$$

By combining the basic product-sum operation boxes shown in Figure 4.2, the above computation can be represented in Figure 4.3. However, because all the outputs of the boxes, except the last box, are not restored, in every connection between boxes, the output of each box is inputted into the next box in its encrypted state. Moreover, each computation for each box is performed by the same set of servers. In contrast, the computation of a can be generally represented as in Figure 4.4; however, if to decompose it into a basic product-sum operation, we could state that the secure computation box in Figure 4.4 is composed of the operations in Figure 4.3.

4.3.2 Security of Extended Method

Here, we consider the combination of product-sum operations shown in Section 4.2. For example, regardless of the security level of the computation method used in the box shown in Figure 4.4, if all the inputs except a_1 are known, the input a_1 is eventually leaked from $a_1 = f^{-1}(a, a_2, \dots, a_{2m}, a_{2m+1})$.

In contrast, in the computation of a , if two of the inputs (for example, a_1, a_2) are not leaked; we can state that these two inputs cannot be leaked. This is because $a_1 = f^{-1}(a, a_2, \dots, a_{2m}, a_{2m+1})$, and if the value of a_2 is not known, a_1 cannot be specified. The same applies to the opposite situation. Therefore, we define Adversary 6 as follows.

Adversary 6: In a t -input-1-output computation, $t - 1$ and below players are the adversary. Only the remaining two inputs or one input and one output are not known. According to the information known, the adversary attempts to learn about the remaining two unknown.

The evaluation of security against Adversary 6 is shown in the following section. However, because the same set of servers must execute the consecutive computation of multiple product-sum operations, we include the following Condition (3).

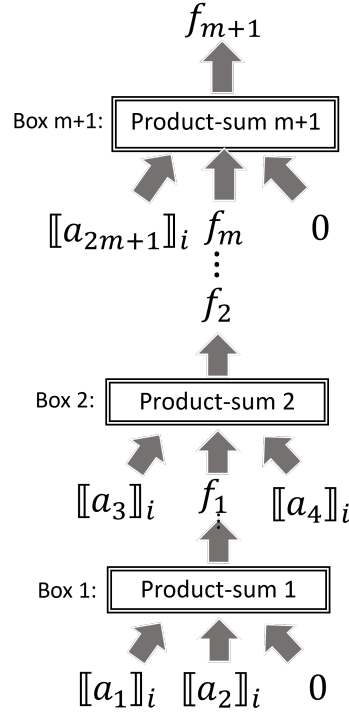


FIGURE 4.3: Computation of a by combining multiple product-sum operations.

3. In secure computation involving consecutive computation, the position of a share in a set of shares handled by each server is fixed.

In Shamir's (k, n) method, the computation of the product-sum operation is already defined and can be represented as in Figure 4.2, whereas the combination of multiple computations is as shown in Figure 4.3, where all computation boxes are independent of each other. In other words, a group of servers that performs a secure computation does not have to be the same group of servers that conducted the previous computation, and a combination of servers can be freely used to perform a computation.

In contrast, if servers have participated in computation in our proposed method, they retain fragments of information from that computation. Assuming the case where multiple computations are performed consecutively, if the same server handles random numbers that are different from those in the previous computation, the server has information of two different random numbers. This poses the risk of a situation where the adversary can obtain more than a k number of information from $k - 1$ computing servers, as one server may have more than one fragment of information of the random number.

Therefore, from Condition (3), when multiple computations are performed repeatedly, the random numbers handled by a server are limited to the same random numbers handled in a previous computation. In other words, for example, in multiplication, the server that

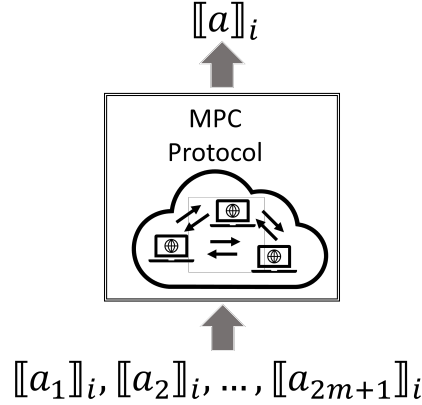


FIGURE 4.4: General computation of a .

reconstructs random numbers α_j and β_j and distributes the value of the random number $\alpha_j\beta_j$ also reconstructs the same random number $\alpha_j\beta_j$ even in the subsequent computation.

However, suppose a situation where $n > k$ and server S_j that handled random numbers α_j, β_j broke down and were replaced with a new server S_j , which did not previously participate in any computation. Here, there is a premise in (k, n) threshold secret sharing that the number of servers that the adversary can control among n servers is $k - 1$ or less. Therefore, suppose that $k - 1$ servers participating in the computation for a certain operation are dishonest servers that leak information to the adversary. It can be said that the new server that participates in the computation on behalf of the broken server is not a dishonest server. Therefore, adding condition (3) does not cause any new problems. Moreover, each server knows a fragment of the secret inputs, but it is safe because k pieces are not collected.

In the following, under the conditions previously mentioned, we explain the evaluation of our proposed method against Adversary 6 when multiple product-sum operations are performed consecutively, as shown in Figure 4.3.

Evaluation of security against Adversary 6

We consider three different situations in the security evaluation of our proposed method against Adversary 6.

1. Two out of three inputs in a product-sum operation box are unknown, whereas all the remaining inputs and final outputs are known.

Two inputs in a product-sum operation box are unknown. In other words, one input and one output of a product-sum operation box are known to the adversary (which is the same as Adversary 4). As our proposed method is secure against Adversary 4, the remaining two inputs will continue to be unknown to the adversary.

Next, as an example, the product-sum operation box mentioned previously is Box 2 shown in Figure 4.3, where all the inputs for the remaining boxes and the final output are leaked to the adversary. In this case, all information except for Box 2 are leaked; however, as computation in Box 2 is secure against Adversary 4, the remaining two inputs will not be leaked. This does not change, regardless of the position of the product-sum operation box in the computation.

2. Two of the unknown inputs are inputted in different boxes each, whereas the remaining inputs and the output for the last box are leaked.

If two of the unknown inputs are each inputted in different product-sum operation boxes, the remaining two out of three inputs of that box can be leaked to the adversary. For example, two unknown inputs are each inputted in Box 1 and Box 2, respectively in Figure 4.3. The remaining inputs for Box 1 and Box 2 are leaked to the adversary. In Box 1, even if two inputs are known to the adversary, it cannot learn information of the output without the remaining input, thus adding another unknown input to Box 2. Therefore, in Box 2, because two inputs are not leaked, and based on the previous case of evaluation against Adversary 4, we can state that the two inputs in Box 2 will not be leaked. This does not depend on how the boxes are combined.

3. The output for the last box and one input are not known, whereas all the remaining inputs are leaked.

The output will not be leaked in a box into which the unknown input is inputted (as previously shown). Therefore, inputs related to that output will also be unknown. Because of this, we can state that in the last box, one input and one output will be unknown. In this case, we can state that two inputs of the last box are known to the adversary. However, from the evaluation of security against Adversary 5, we can state that, even if two inputs of the last box are known to the adversary, the remaining input and output will not be leaked.

4.4 Discussion

4.4.1 Qualitative Comparison with TUS 1 and SPDZ Methods

As mentioned in Chapter 3, the SPDZ method uses SHE for the generation of multiplication triple. However, SHE, known to be computationally very expensive, is used in the offline phase of the SPDZ method, making it computationally secure against a dishonest majority. In addition, we also include the method in Chapter 3 in the comparison.

The SPDZ is limited to the setting $n = k$, where the owner of the secret input is one of the players involved in the secure computation. Provided that the owner protects his/her share, even if $n - 1$ players, excluding the owner, form a coalition, the protocol is secure and the

TABLE 4.1: Comparison with previous works

	Proposed Method	TUS Method	SPDZ Method
Parameters of n and k	$n \geq k$	$n \geq k$	$n = k$
Type of adversary	Semi-honest	Semi-honest	Active
Type of security	Information-theoretic	Information-theoretic	Computational
Combination of operation	Unlimited	Limited to same type of operation	Unlimited

secret input cannot be leaked, because insufficient shares are collected to restore the secret input. In particular, the SPDZ supposes an active adversary and is secure against a dishonest majority. Moreover, the SPDZ uses a SHE in the offline phase, making it computationally secure but exceptionally computationally expensive. Moreover, the SPDZ method is secure against computation that involves a combination of different types of operation, such as the combination of addition and multiplication. However, it cannot perform division directly from shares a and b inputted by the player (it is possible to compute the secrecy division using a method of secure multiplication if shares on $1/a$ and b are distributed by the player.).

In contrast, our proposed method and the method in Chapter 3 are not limited only to situations where $n = k$. Both methods can perform MPC even under the setting $n \geq k$; although our proposed method shows effectiveness under the setting $n < 2k - 1$, it is also usable in the setting $n \geq 2k - 1$. However, they both suppose a semi-honest adversary and include MPC protocols that are information-theoretic secure against a semi-honest adversary. In addition, the method in Chapter 3 faces issues when different types of operation are combined to perform a more complicated computation, whereas the method proposed in this chapter is secure even when different types of computations are combined. However, the method in Chapter 3 requires only one condition, whereas the improved method requires three conditions. Moreover, both methods can perform division directly from shares of a and b . These aspects are summarized in Table 4.1.

4.4.2 Quantitative Comparison with TUS 1 and SPDZ Methods

A comparison of the computational and communication costs of our proposed methods and the SPDZ method is shown in Tables 4.2, 4.3 and 4.4. The number of communications is evaluated as the number of rounds in proportion to the direction of the communication. We define the parameters used in the comparison as follows.

Definition of Parameters:

- d_1 : Size of share from secret sharing
- d_2 : Size of share from SHE

TABLE 4.2: Comparison with previous works (computation cost)

Process	Proposed Method	TUS 1 Method	SPDZ Method
Multiplication of ab	$2(3k+2)C_1$	$2(3k+2)C_1$	$8C_1 + 2C_2$
Addition of $a+b$	$(10k+7)C_1$	$3(2k+1)C_1$	$3C_1$
Product-sum of $ab+c$	$3(4k+3)C_1$	Not executable	$9C_1 + 2C_2$

- C_1 : Computational cost of secret sharing
- C_2 : Computational cost of SHE

Parameter d_1 is usually almost the same size as the original secret input, whereas d_2 is typically larger. Therefore, $d_2 > d_1$. Moreover, C_1 is considerably smaller than C_2 . Therefore, $C_1 \ll C_2$. In a secret sharing, the computational costs of the distribution and the reconstruction processes differs; however, as both are considerably smaller than C_2 , we consider that the computation costs of these processes are C_1 . As authentication processes, such as zero-knowledge proof and MAC, were not included in our proposed methods, corresponding processing costs are omitted.

In addition, the values in Table 4.2 include the costs of preprocessing, distribution, and reconstruction. Both methods proposed do not include the computational cost C_2 for SHE. Therefore, we can state that our methods are better in terms of this cost than the SPDZ method. In terms of communication cost, the merits and demerits of each method depend on n, k, d_1, d_2 . Our proposed method includes the processes of generating, restoring, and distributing random numbers in secret distribution and secure computation. Thus, a comparison of the number of rounds of each method shows that the total number of rounds of our method in this chapter is considerably higher than that of the method in Chapter 3 and the SPDZ method.

However, our proposed method is lighter than the SPDZ method in terms of overall processing cost because it does not contain the computational cost of SHE. The method in this chapter also enables the combination of different types of operation while remaining secure, unlike the method in Chapter 3.

4.4.3 Discussion about Conditions

This section discusses the realizability of our three proposed conditions.

Condition 1: Inputs and output in multiplication do not include value 0.

In multiplication, if the secret input inputted is $0, \alpha a$, which is restored in the protocol of multiplication, is $\alpha a = 0$. Thus, the adversary can know that secret input a is 0, as a random number does not contain the value 0. However, information that does not contain value 0 is abundant, such as medical data, such as pulse, blood pressure, and blood glucose level.

TABLE 4.3: Comparison with previous works (communication cost)

Process	Proposed Method	TUS 1 Method	SPDZ Method
Multiplication of ab	$(3(k+n)(k+1) - k)d_1$	$(3n(k+1) + k(3k+2))d_1$	$10nd_1 + 2nd_2$
Addition of $a+b$	$(5(k+n)(k+1) + 3n)$	$(n(3k+4) + 3k(k+1))d_1$	$3nd_2$
Product-sum of $ab+c$	$(6(k+n)(k+1) + 4n)d_1$	Not executable	$11nd_1 + 2nd_2$

TABLE 4.4: Comparison with previous works (round)

Process	Proposed Method	TUS 1 Method	SPDZ Method
Multiplication of ab	6	6	5
Addition of $a+b$	9	6	2
Product-sum of $ab+c$	11	Not executable	5

Therefore, when performing secure computation from data collected from patients admitted to or being treated at a hospital, excluding the value 0 in inputs does not raise a serious problem. In addition, the condition of exclusion of input 0 applies only to multiplication, because input 0 does not cause a problem when used in addition, subtraction, and division. Thus, a considerable amount of information does not require value 0 and our proposed method is an effective MPC protocol for this information.

Condition 2: There are shares of random numbers unknown to the adversary.

The simplest method to fulfill this condition is to obtain a set of shares from a TTP that is not involved in the MPC. This technique was included in methods such as those proposed in [49], where the assumption of a trustable server contributes to realizing a more effective process. Therefore, the establishment of a trustable server is effective in practical use. However, as shown in Section 4.2.1, the generation of shares of random numbers does not depend on the secret inputs and is easily realizable by producing k random numbers, multiplying all the random numbers, and distributing them using a secret sharing.

In addition, in this study, we assumed a semi-honest adversary. Therefore, if we add the process of producing shares of a random number from different random numbers into all servers and execute the “shuffle process”, the connection between the server that produced the set of shares is removed. Moreover, in a server set containing multiple servers, if they share no interest between each other, we can achieve a structure close to a TTP by utilizing these servers to exchange, mix, and remove while they shuffle the set of shares on conversion random numbers between each other. Therefore, this condition can be realized in several

manners it in practice.

Condition 3: In secure computation involving consecutive computation, the position of shares in a set of shares that are handled by each server is fixed.

Because our proposed method assumes a semi-honest adversary, we can realize this condition by setting regulations for servers that hold shares required for secure computation and servers involved in secure computation.

4.5 Chapter Summary

In this chapter, with three proposed conditions, we realized a secure MPC when $n < 2k - 1$ even when different types of computation are performed consecutively.

1. The value of secret inputs and output in multiplication does not include 0;
2. There are shares of random numbers that are constructed from random numbers unknown to the adversary.;
3. In secure computation involving consecutive computation, the position of shares in a set of shares handled by each server is fixed.

Chapter 5

Application of MPC: Searchable Encryption of Documents

In this chapter, we consider the applications of secure MPC into the SE of documents. This chapter mainly discusses how the MPC method proposed in Chapter 4 can be used to realize SE of documents.

Both methods proposed in Part I of this dissertation required the following three conditions to achieve security against semi-honest adversaries.

1. Inputs and output in multiplication do not include 0;
2. There are shares of random numbers that are constructed from random numbers unknown to the adversary;
3. The position of shares in a set of shares that are handled by each server is fixed.

In this chapter, when focusing on application into SE of documents, we also introduce methods of easing these three conditions. Moreover, we also perform a comparison with conventional methods of SE that use encryption-based approaches such as symmetric key encryption.

5.1 Introduction

In recent years, the computational capacity of computing machines and technology for communication has improved, and the spread of cloud computing has rapidly advanced. In cloud computing, data storage is outsourced to multiple servers, and the processing of information must be performed by the server. In addition, to protect the privacy and confidentiality of data, information must be encrypted before it is stored. However, searching encrypted data on a server is not possible without first decrypting the data. Therefore, several methods of SE have been proposed, where only a legitimate user or a user who was authorized can access information without decrypting the encrypted data [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69].

With the conventional method of SE, two of the main search functions realized are sought using one and using multiple search queries. Search methods that use one search query can be further divided into total matching search, partial matching search, range search, and similarity search. In addition, search methods that use multiple search queries can be further divided into conjunctive search, disjunctive search, and both combined. In a conjunctive search, only when registered information that corresponds to all the search queries is found, the encrypted file with that information is extracted. In a disjunctive search, if at least one of the defined search queries matches with the registered document, the encrypted document will be extracted.

SE using a single search query has been widely studied and researched [50] [51] [54] [55] [70] [60]. In contrast, few methods of SE using multiple search queries are available. In recent years, SE that can manage multiple search queries has been required. For example, instead of performing a search process on every single keyword when the searcher assigns multiple keywords in the system, a function that allows multiple keywords to be combined for a one-time search is more effective for obtaining an accurate search result from a large collection of files.

A conventional method of SE is SE using symmetric key encryption. Multiple-keyword search based on this was first proposed by Golle et al. [52]. However, it only performed a conjunctive search, and could not perform a disjunctive search. Hence, numerous methods have been proposed, such as those by Cash et al. [61] that allow for both conjunctive and disjunctive searches. Most of the SE methods proposed thus far utilize the concept of the search index, where an index is encrypted and the search function is performed by using the extracted index. Here, a search index is a body of structured data to which a search engine refers when looking for results that are relevant to a specific query. Index data is related to the keywords in a document, which are encrypted and registered in the search index. Therefore, because direct search over an encrypted sentence is not possible, information that is not registered in the index is not searchable. Additionally, when a document is added or removed from the system, the index that corresponds to the document must be added or removed. Consequently, the overall computational cost increases.

In this chapter, we realize direct search of a keyword in a document through conjunctive search and the search of multiple keywords by a single computation through the introduction of disjunctive search. In addition, to realize a faster search process, we use an approach to SE that applies a secret sharing. In particular, we utilize the method of secure MPC proposed in Chapter 4 to realize SE. Hitherto, no method of SE using the secret sharing has been proposed, and the method proposed by us in [66] can be regarded as the first method of its type that was implemented for image search. However, this method performs secure MPC only for searches using one pixel at a time. This implies that it only allows for one character of a keyword to be searched at a time and does not realize either conjunctive or disjunctive searches.

Hence, in this chapter, we propose a method for realizing conjunctive search of a query with multiple characters and disjunctive search that allows for multiple keywords or queries to be simultaneously searched.

New Protocols

We describe two new protocols as follows.

New Protocol 1

The conventional method for SE using a secret sharing, proposed by Kamal et al. [66], consisted of searching per pixel (implying that the search result was reconstructed for every pixel). However, when adapting this method to the search of documents (by changing the search target from the pixel of an image to the character of a document), it is not secure against brute force search attacks (refer to Section 5.3 for further details). Furthermore, most conventional methods that use symmetric key encryption or public-key encryption only realize index search, and cannot perform a direct search over an encrypted document. In New Protocol 1, we realize the direct search function over an encrypted document with the ability to perform a total matching search on a query with multiple characters simultaneously (conjunctive search). In this method, the search computation is performed per character; however, the search result for multiple characters is reconstructed simultaneously instead of per character, thereby achieving stronger security against brute force search attacks. Moreover, New Protocol 1 realizes the direct search over an encrypted document, solving the problems of conventional methods that rely on index search.

New Protocol 2

In New Protocol 2, aiming for implementation in a broader area of practical applications, we introduce the concept of SE with support for any Boolean functions by combining conjunctive search (New Protocol 1) and disjunctive search (New Protocol 2). For example, suppose that there are multiple search queries for performing a disjunctive search. Disjunctive search using multiple search queries only returns documents that contain a subset of the search queries that are to be searched. In New Protocol 2, conjunctive search (New Protocol 1) is first used to perform a total matching search on each search query. The results are used in a consecutive disjunctive search of multiple search queries. In other words, if at least one of the inputted search queries matches, the system returns the document to the searcher. Additionally, in our New Protocol 2, the search result is constructed once instead of for every search query, which was difficult to achieve using conventional methods.

System Model

Our proposed model of secure computation is based on a client/server model, where any number of clients (owners of the secret information) sends shares of their inputs to multiple servers (n number of servers). However, clients who wish to search for any information (searcher) send shares of their trapdoors to n number of servers, which perform the computation for the clients and return the results to them without learning any information. This

model is widely used and is the business model used in Sharemind [45]. In our protocol, the searcher can produce a valid trapdoor multiple times to perform searching. However, this can be changed according to the intended applications, as shown in Section 5.6. In addition, our proposed methods assume a semi-honest adversary, where the adversary follows the protocol specification but may attempt to learn more than the allowed by the protocol, with at most $k - 1$ corrupted servers. In addition, we also consider the following attacks: the adversary has information of the searcher in addition to information from $k - 1$ servers and attempts to learn the registered document. We also presume that secure communication exists between the owner of the secret input, the searcher, and the servers.

5.2 Building Block: Overview of Secure MPC

In Chapters 3 and 4, we proposed methods for secure MPC using secret sharing that are information-theoretic secure against semi-honest adversaries under certain conditions. In particular, the method proposed in Chapter 4 requires the following three conditions:

1. Inputs and output in multiplication do not include 0;
2. There are shares of random numbers that are constructed from random numbers unknown to the adversary;
3. The position of shares in a set of shares that are handled by each server is fixed.

In this chapter, we will use the MPC in Chapter 4 to perform secure computation in SE. However, here, we introduce a method of easing the aforementioned first condition into the following:

Condition 1’: Output in multiplication does not include value 0.

As explained in Chapter 4, in multiplication, if the secret inputted is 0, αa restored in the protocol of multiplication is $\alpha a = 0$. From this, the adversary can know that secret input a is 0, as a random number does not contain the value 0.

However, in this chapter, our focus is to realize SE in documents. To represent text (or numbers, punctuation, etc.) in a computer system, each character is given its own special number. This number is called code, and this code is stored in the computer using binary ones and zeros. One of the methods to represent data in a computer system is the American Standard Code for Information Interchange (ASCII) code. ASCII code allows computers to represent text and normally uses 8 bits (1 byte) to store each character. However, the 8th bit is used as a check digit, meaning that only 7 bits are available to store each character. This gives ASCII the ability to store a total of $2^7 = 128$ different values from 0–127. Table 5.1 shows some examples of ASCII code and the character it represents.

This means that for text/documents, only inputs between 0–127 need to be considered. However, the secure MPC method in Chapter 4 does not allow for input 0. Therefore, when

TABLE 5.1: Example of ASCII code and corresponding characters

Code	Character
0	Null
1	Start of header
⋮	⋮
65	A
66	B
67	C
⋮	⋮
127	DEL

implementing SE, we also introduce a solution for easing Condition 1 by adding value “1” to the input. In other words, assume that the secret input s is $0 \leq s < p - 1$; by adding the value “1” to the input, even if the secret input s is equal to 0, $s + 1 \neq 0$, allowing the secret input of 0 even in multiplication.

Furthermore, by setting the module p to be a very large number, as the secret input is less than $p - 2$, adding “1” will not result in “0”. Therefore, the improved method of secure MPC used in this chapter is secure even if the input of multiplication is 0. However, the condition that the output does not include 0 remains. To differentiate this from the original method in Chapter 4, we call this improved version of MPC the TUS 2’ method.

5.3 Related Work

5.3.1 SE Using Symmetric Key Encryption

In symmetric key encryption, only the user with the correct secret key can perform both encryption and decryption of data. Therefore, SE that is based on symmetric key encryption can only be used when the owner of the secret input and the searcher are the same person. If the owner and the searcher are not the same person, an additional process of key sharing between the owner and the searcher is required, which induces additional costs in both computation and communication.

In 2000, Song et al. [50] proposed a method for the SE of a single keyword by using symmetric encryption. In addition, an improved version was introduced by Curtmola et al. [55]. Golle et al. proposed in 2004 a symmetric SE (SSE) that allowed for multiple keywords to be searched [52]. In this method, a searcher can specify at most one keyword for each keyword field. For example, the keyword fields in emails consist of “to,” “from,” and “subject.” Therefore, to overcome this limitation, Wang et al. [57] proposed an improved version. However, both methods could only perform a conjunctive search and not disjunctive

search. Cash et al. [61] and Kurosawa et al. [60] proposed a method of multiple-keyword SSE that allowed for both disjunctive and conjunctive searches.

Moreover, Liu et al. [71] also proposed a method of SE using symmetric encryption that allows search over encrypted data that are shared among multiple users. There are also methods proposed by Li et al. [69] and Sun et al. [67] that provide forward and backward search privacy in SE to prevent information leakage during the update operation of the index. However, most of the methods proposed thus far utilize the concept of search index. Consequently, when adding or removing any registered information, an additional computation cost is required for updating the index. Moreover, because a direct search over an encrypted document is not possible, searching over information other than the registered index is not possible.

5.3.2 SE Using Public Key Encryption

In public key encryption, anyone can perform the encryption of data using a public key; however, only users with the correct secret key can perform the decryption of data. Therefore, SE using public key encryption is suitable for cases where there are multiple registered owners and only one searcher.

In 2004, Boneh et al. [51] proposed a method for SE with single keywords using public-key encryption, wherein the owner encrypts data using a public key, and stores the encrypted data in the cloud, whereas a user with the correct secret key produces a search tag for the cloud to perform a matching search. Many attempts have been made to propose a better method for SE using public key encryption.

For example, Park et al. [53] and Byun et al. [56] proposed an improved method of SE using public key encryption that allowed for the conjunctive search of multiple keywords through the use of keyword fields. However, only documents that matched the preset pattern of the keyword field could be searched. Zhang et al. [58] proposed another method for conjunctive search that did not depend on keyword fields. However, in this method, when the search process was performed, the server had to perform $n * m$ bilinear pairings, where n is the number of documents stored in the database and m is the largest number of keywords. This is very costly, making this method quite ineffective.

Xu et al. [68] proposed a method that supported both conjunctive and disjunctive searches. In this method, the server first finds documents that match with the first keyword and sets them to Set A. In sequence, the server confirms whether the remaining documents match the requirement of the keywords. By using this method, the range of documents to search can be minimized, reducing the search time. However, the amount of data stored on the server and the cost of encryption are quite large. In addition, the method developed by Xu et al. also uses the concept of index search, where searching for keywords that are not registered in the index is impossible.

5.3.3 SE Using Secret Sharing

One of the first methods for SE using secret sharing was presented by us in [66] for use in image searching. However, in this method, only a search that used a single pixel at a time was realized, where simultaneous conjunctive and disjunctive searches over multiple pixels could not be performed. Therefore, when changing the search target from pixel search to character search, the following problem will arise.

For example, suppose that each character is encoded using ASCII code. The number of possible ASCII codes for each character is 0–127. Because the adversary has information from at most $k - 1$ number of servers, if the adversary performs a search with a different character each time, a brute force search attack is possible after 128 searches. In contrast, if we add a restriction such as that the search must be performed with at least 18 characters each time, assuming that the adversary uses all values between 0–127, the number of possible candidates will be $(2^7)^{18} = 2^{126}$, meaning that the adversary must perform 2^{126} searches for a brute force search attack to be possible, thus solving this problem. However, in a normal search of a document, 18 characters may not be sufficient to achieve the 2^{126} security as a normal search query will only be composed of chars, numbers, and symbols. Therefore, a search with a longer query is needed. This can be counter-measured with the implementation of conjunctive search with a password [72] or secret keys (depending on the intended applications as shown in Section 5.6).

However, by using the method in [66], because the search is performed by shifting the characters one by one, each server needs to store $18 \times n$ number of shares for each character. In addition, if the searcher wishes to perform a search with 10 or 20 characters instead of 18, the previously registered shares would not be usable. Therefore, there is a need for a method that allows for search on a query of any length to be performed per character, but with the result reconstructed once instead of per character.

5.4 Proposed Method: Conjunctive Search

5.4.1 Overview of Conjunctive Search

In conjunctive search, searching is performed per character; however, the search results of multiple characters are reconstructed once instead of per character, realizing a total matching search on a search query with multiple characters. Suppose a document a with m number of characters, where the registered character string is represented by (a_1, \dots, a_m) .

In addition, suppose a search query b with g number of characters, where the registered character string is written as (b_1, \dots, b_g) . The difference between the characters of the registered document and the search query $(a_1 - b_1), \dots, (a_g - b_g)$ is written as c_1, \dots, c_g , where $c_y = a_y - b_y$ ($y = 1, \dots, g$). Here, the computation of function f , such that $f(c_1, \dots, c_g) \neq 0$ when any c_1, \dots, c_g are not equal to 0, is the computation of the logical conjunctive (AND)

and can be computed as $c_1 \cdots c_g$. Therefore, the computation of function f , such that $f(c_1, \dots, c_g) = 0$ when all differences c_1, \dots, c_g are equal to 0, can be written as $\overline{c_1 \cdots c_g}$. This can be further simplified in $\overline{c_1 \cdots c_g} = c_1 + \dots + c_g$ using the De Morgan's Law. Therefore, the total matching search of a search query with g number of characters can be computed as follows:

$$(a_1 - b_1)^2 + \dots + (a_g - b_g)^2 = c_1 + \dots + c_g$$

Here, we square the difference for each character to avoid any negative numbers. In addition, we can realize the direct search of a document by shifting the search query b character by character and searching through document a until the end.

Suppose that $x = 1, \dots, m, y = 1, \dots, g$, the character string of registered document a can be represented as $a_x (x = 1, \dots, m)$, and the character string of search query b can be represented as $b_y (y = 1, \dots, g)$. In other words, to perform a total matching search of search query b_y with g number of characters, suppose that the first search position is $h = 0$. Considering the character string $a_v (v = h + 1, \dots, h + g)$ where a_v consists of g continuous characters from document a , we need to perform only the computation shown below. In addition, if the reconstructed result is not equal to 0 (meaning that search query b_y and character string a_v do not match), h is set such that $h = h + 1$, and the search target a_{h+y} is shifted to the right. This process is repeated until $h = m - g$.

$$\sum_{y=1}^g (a_{h+y} - b_y)^2 = \sum_{y=1}^g c_y (h = 0, \dots, m - g)$$

5.4.2 Protocol of Conjunctive Search (when $n = k$)

The detailed protocol of the conjunctive search is presented as follows. The size of the registered character string a_x of the document, and the search query b_y is q ; however, both a_x and b_y are elements of finite field $GF(p)$, where the field p is set such that $gq^2 < p$. Here, the value g can be set to any number as long as it is in the range that can prevent a brute force search attack.

In addition, random numbers $\alpha_{x,j}, \gamma_{x,j}, c_x, \beta_{y,j}, \delta_{y,j}, d_y, \rho_{h,j}, \varphi_{h,j}, \kappa_{h,j}$ generated in our protocol are elements of $GF(p)$ (do not include the value 0). All computations shown below, including the process of secret sharing, are performed with field p . Furthermore, c_v, d_y in Protocols 5.2 and 5.3 are random numbers such that $c_v + d_y = 0$ does not occur. However, for ease of explanation, all explanations are written under the assumption that $n = k$, and let $x = 1, \dots, m, y = 1, \dots, g, i = 0, \dots, n - 1$, and $j = 0, \dots, k - 1$.

Notation:

- a : Document with m number of characters $a_x (x = 1, \dots, m)$

- $[a_x]'_i$: Set of encrypted values regarding character a_x held by each server S_i ($i = 0, \dots, n-1$)
- $[a_x]_i$: Set of encrypted values regarding character a_x and random number c_x held by each server S_i
- $[a]_i$: Set of encrypted values regarding document a held by each server S_i . For example, $[a]_i = [a_1]_i, \dots, [a_m]_i$
- b : Search query with g number of characters b_y ($y = 1, \dots, g$)
- $[b_y]'_i$: Set of encrypted values regarding character b_y held by each server S_i
- $[b_y]_i$: Set of encrypted values regarding character b_y and random number d_y held by each server S_i
- $[b]_i$: Set of encrypted values regarding search query b held by each server S_i . For example, $[b]_i = [b_1]_i, \dots, [b_g]_i$

In addition, each server is assumed to holds sets of conversion random numbers $[[\mathcal{E}_x^{(u)}]]_i, \mathcal{E}_{x,i}^{(u)}$, which can be generated through Protocol 5.1, shown below. However, random numbers $\mathcal{E}_{x,0}^{(u)}, \dots, \mathcal{E}_{x,k-1}^{(u)}$ generated in the algorithm below are elements of $GF(p)$ (do not include the value 0). Here, the subjects for executing Protocol 5.1 for generating the set of conversion random numbers are stated in Step 1 of Protocols 5.2 and 5.3.

Protocol 5.1: Generation of the set of conversion random numbers

1. Generate k random numbers $\mathcal{E}_{x,0}^{(u)}, \dots, \mathcal{E}_{x,k-1}^{(u)}$ ($x = 1, \dots, m, u = 1, \dots, \text{required number}$), and compute random number $\mathcal{E}_x^{(u)}$ as follows:

$$\mathcal{E}_x^{(u)} = \prod_{j=0}^{k-1} \mathcal{E}_{x,j}^{(u)}$$

2. Distribute random numbers $\mathcal{E}_x^{(u)}$ to n number of servers S_i ($i = 0, \dots, n-1$) by using Shamir's (k, n) method, and send $\mathcal{E}_{x,j}^{(u)}$ to server S_j . (if $n > k$, $\mathcal{E}_{x,j}^{(u)}$ are distributed to n servers using Shamir's (k, n) method as shown in Protocol 3.1)
3. Each server S_j hold the following as the set of conversion random numbers.

$$[\mathcal{E}_x^{(u)}]_j = \left([[\mathcal{E}_x^{(u)}]]_j, \mathcal{E}_{x,j}^{(u)} \right)$$

Protocol 5.2: Encryption of document

- **Input:** a_x ($x = 1, \dots, m$)

- **Output:** $[a]_j = [a_1]_j, \dots, [a_m]_j$ ($j = 0, \dots, k-1$)

1. The owner performs Protocol 5.1 beforehand, producing the following sets of conversion random numbers, and sends the set to k computing servers S_j . Each server S_j ($j = 0, \dots, k-1$) holds the following information:

$$\begin{aligned} [\varepsilon_x^{(2)}]_j &= \left(\llbracket \varepsilon_x^{(2)} \rrbracket_j, \varepsilon_{x,j}^{(2)} \right), \\ [\varepsilon_x^{(4)}]_j &= \left(\llbracket \varepsilon_x^{(4)} \rrbracket_j, \varepsilon_{x,j}^{(4)} \right), \\ [\varepsilon_x^{(5)}]_j &= \left(\llbracket \varepsilon_x^{(5)} \rrbracket_j, \varepsilon_{x,j}^{(5)} \right), \\ [\varepsilon_x^{(6)}]_j &= \left(\llbracket \varepsilon_x^{(6)} \rrbracket_j, \varepsilon_{x,j}^{(6)} \right), \\ [\varepsilon_x^{(7)}]_j &= \left(\llbracket \varepsilon_x^{(7)} \rrbracket_j, \varepsilon_{x,j}^{(7)} \right), \\ [\varepsilon_x^{(8)}]_j &= \left(\llbracket \varepsilon_x^{(8)} \rrbracket_j, \varepsilon_{x,j}^{(8)} \right) \end{aligned}$$

2. Regarding the secret information a_x ($x = 1, \dots, m$), the owner generates random numbers $\alpha_{x,j}, \gamma_{x,j}, c_x$ ($j = 0, \dots, k-1$), and computes the following.

$$\begin{aligned} \alpha_x &= \prod_{j=0}^{k-1} \alpha_{x,j}, \\ \gamma_x &= \prod_{j=0}^{k-1} \gamma_{x,j} \end{aligned}$$

3. The owner multiplies the secret information ($a_x + 1$) and random numbers c_x with the previously calculated random numbers α_x, γ_x , producing $\alpha_x(a_x + 1), \gamma_x c_x$, respectively. The owner sends $[a_x]'_j, [c_x]'_j$ shown below to k number of servers S_j ($j = 0, \dots, k-1$) that participate in the computation.

$$\begin{aligned} [a_x]'_j &= (\alpha_x(a_x + 1), \alpha_{x,j}), \\ [c_x]'_j &= (\gamma_x c_x, \gamma_{x,j}) \end{aligned}$$

4. Suppose that $[a_x]_j := ([a_x]'_j, [c_x]'_j)$. Each server S_j ($j = 0, \dots, k-1$) hold the following information:

$$[a]_j = [a_1]_j, \dots, [a_m]_j$$

Protocol 5.3: Generation of search query

- **Input:** b_y ($y = 1, \dots, g$)

- **Output:** $[b]_j = [b_1]_j, \dots, [b_g]_j$ ($j = 0, \dots, k-1$)
1. The searcher performs Protocol 5.1 beforehand, producing the following sets of conversion random numbers and sends the set to k computing servers S_j . Each server S_j ($j = 0, \dots, k-1$) holds the following information:

$$[\epsilon_x^{(1)}]_j = \left(\llbracket \epsilon_x^{(1)} \rrbracket_j, \epsilon_{x,j}^{(1)} \right),$$

$$[\epsilon_x^{(3)}]_j = \left(\llbracket \epsilon_x^{(3)} \rrbracket_j, \epsilon_{x,j}^{(3)} \right)$$

2. Regarding the character string b_y ($y = 1, \dots, g$) of the search query, the searcher generates random numbers $\beta_{y,j}, \delta_{y,j}, d_y$ ($j = 0, \dots, k-1$), and computes the following:

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

$$\delta_y = \prod_{j=0}^{k-1} \delta_{y,j}$$

3. The searcher multiplies the search query ($b_y + 1$) and random numbers d_y with the previously calculated random numbers β_y, δ_y , producing $\beta_y(b_y + 1), \delta_y d_y$, respectively. The searcher sends $[b_y]'_j, [d_y]'_j$ shown below to k number of servers S_j ($j = 0, \dots, k-1$) that participate in the computation:

$$[b_y]'_j = (\beta_y(b_y + 1), \beta_{y,j}),$$

$$[d_y]'_j = (\delta_y d_y, \delta_{y,j})$$

4. Suppose that $[b_y]_j := ([b_y]'_j, [d_y]'_j)$. Each server S_j ($j = 0, \dots, k-1$) holds the following information:

$$[b]_j = [b_1]_j, \dots, [b_g]_j$$

Protocol 5.4: Search process

- **Input:** $[a]_j, [b]_j$ ($j = 0, \dots, k-1$)
 - **Output:** a , or \perp
1. First, let $h = 0$, and let k number of servers S_j ($j = 0, \dots, k-1$) perform the following between a character string of the search query b and g number of consecutive characters from character string a . Suppose that $y = 1, \dots, g, v = h + y$.

- (a) Each server S_j ($j = 0, \dots, k-1$) generates random numbers $\rho_{h,j}, \varphi_{h,j}$, computes the following, and sends it to any one of the k servers (here, we assume server S_0):

$$\frac{\rho_{h,j}}{\gamma_{v,j}\epsilon_{v,j}^{(1)}}, \frac{\rho_{h,j}}{\delta_{y,j}\epsilon_{v,j}^{(2)}}, \frac{\varphi_{h,j}}{\alpha_{v,j}\epsilon_{v,j}^{(3)}}, \frac{\varphi_{h,j}}{\beta_{y,j}\epsilon_{v,j}^{(4)}}, \frac{\varphi_{h,j}}{\rho_{h,j}\epsilon_{v,j}^{(5)}}$$

- (b) Server S_0 computes the following and sends it to all servers S_j ($j = 0, \dots, k-1$):

$$\begin{aligned} \frac{\rho_h}{\gamma_v\epsilon_v^{(1)}} &= \prod_{j=0}^{k-1} \frac{\rho_{h,j}}{\gamma_{v,j}\epsilon_{v,j}^{(1)}}, \\ \frac{\rho_h}{\delta_y\epsilon_v^{(2)}} &= \prod_{j=0}^{k-1} \frac{\rho_{h,j}}{\delta_{y,j}\epsilon_{v,j}^{(2)}}, \\ \frac{\varphi_h}{\alpha_v\epsilon_v^{(3)}} &= \prod_{j=0}^{k-1} \frac{\varphi_{h,j}}{\alpha_{v,j}\epsilon_{v,j}^{(3)}}, \\ \frac{\varphi_h}{\beta_y\epsilon_v^{(4)}} &= \prod_{j=0}^{k-1} \frac{\varphi_{h,j}}{\beta_{y,j}\epsilon_{v,j}^{(4)}}, \\ \frac{\varphi_h}{\rho_h\epsilon_v^{(5)}} &= \prod_{j=0}^{k-1} \frac{\varphi_{h,j}}{\rho_{h,j}\epsilon_{v,j}^{(5)}} \end{aligned}$$

- (c) Each server S_j ($j = 0, \dots, k-1$) computes the following and sends it to server S_0 :

$$\llbracket \rho_h(c_v + d_y) \rrbracket_j = \left(\gamma_v c_v \times \frac{\rho_h}{\gamma_v \epsilon_v^{(1)}} \times \llbracket \epsilon_v^{(1)} \rrbracket_j \right) + \left(\delta_y d_y \times \frac{\rho_h}{\delta_y \epsilon_v^{(2)}} \times \llbracket \epsilon_v^{(2)} \rrbracket_j \right)$$

- (d) Server S_0 reconstructs $\rho_h(c_v + d_y)$ using Shamir's (k, n) method and sends it to all servers S_j (if the reconstructed value $\rho_h(c_v + d_y) = 0$, both the owner and the searcher will perform Steps 2 and 3 of Protocols 5.2 and 5.3 until $c_v + d_v \neq 0$; otherwise, in Step 2, the owner and searcher produce two different numbers for c_v, d_v in advance; by changing the combination in Step (c) of Protocol 5.4, $c_v + d_v \neq 0$ can be calculated).

- (e) Each server S_j ($j = 0, \dots, k-1$) computes the following and sends it to server S_0 :

$$\begin{aligned} \llbracket \varphi_h(a_v - b_y + c_v + d_y) \rrbracket_j &= \left(\alpha_v(a_v + 1) \times \frac{\varphi_h}{\alpha_v \varepsilon_v^{(3)}} \times \llbracket \varepsilon_v^{(3)} \rrbracket_j \right) \\ &\quad - \left(\beta_y(b_y + 1) \times \frac{\varphi_h}{\beta_y \varepsilon_v^{(4)}} \times \llbracket \varepsilon_v^{(4)} \rrbracket_j \right) \\ &\quad + \left(\rho_h(c_v + d_y) \times \frac{\varphi_h}{\rho_h \varepsilon_v^{(5)}} \times \llbracket \varepsilon_v^{(5)} \rrbracket_j \right) \end{aligned}$$

- (f) Server S_0 reconstructs $\varphi_h(a_v - b_y + c_v + d_y)$ using Shamir's (k, n) method and sends it to all servers S_j ($j = 0, \dots, k-1$).
- (g) Each server S_j ($j = 0, \dots, k-1$) generate random number $\kappa_{h,j}$, computes the following, and sends it to server S_0 .

$$\frac{\kappa_{h,j}}{(\varphi_{h,j})^2 \varepsilon_{v,j}^{(6)}}, \frac{\kappa_{h,j}}{\varphi_{h,j} \rho_{h,j} \varepsilon_{v,j}^{(7)}}, \frac{\kappa_{h,j}}{(\rho_{h,j})^2 \varepsilon_{v,j}^{(8)}}$$

- (h) Server S_0 computes the following and sends it to all servers S_j ($j = 0, \dots, k-1$):

$$\begin{aligned} \frac{\kappa_h}{(\varphi_h)^2 \varepsilon_v^{(6)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}}{(\varphi_{h,j})^2 \varepsilon_{v,j}^{(6)}}, \\ \frac{\kappa_h}{\varphi_h \rho_h \varepsilon_v^{(7)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}}{\varphi_{h,j} \rho_{h,j} \varepsilon_{v,j}^{(7)}}, \\ \frac{\kappa_h}{(\rho_h)^2 \varepsilon_v^{(8)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}}{(\rho_{h,j})^2 \varepsilon_{v,j}^{(8)}} \end{aligned}$$

- (i) Each server S_j ($j = 0, \dots, k-1$) computes the following:

$$\begin{aligned} \sum_{y=1}^g \llbracket \kappa_h(a_v - b_y)^2 \rrbracket_j &= \sum_{y=1}^g \left\{ \left[(\varphi_h)^2 (a_v - b_y + c_v + d_y)^2 \times \frac{\kappa_h}{(\varphi_h)^2 \varepsilon_v^{(6)}} \times \llbracket \varepsilon_v^{(6)} \rrbracket_j \right] \right. \\ &\quad - \left[2 \times \varphi_h(a_v - b_y + c_v + d_y) \times \rho_h(c_v + d_y) \times \frac{\kappa_h}{\varphi_h \rho_h \varepsilon_v^{(7)}} \times \llbracket \varepsilon_v^{(7)} \rrbracket_j \right] \\ &\quad \left. + \left[(\rho_h)^2 (c_v + d_y)^2 \times \frac{\kappa_h}{(\rho_h)^2 \varepsilon_v^{(8)}} \times \llbracket \varepsilon_v^{(8)} \rrbracket_j \right] \right\} \end{aligned}$$

- (j) k number of servers S_j ($j = 0, \dots, k-1$) cooperate to reconstruct $\sum_{y=1}^g \kappa_h(a_v - b_y)^2$ using Shamir's (k, n) method. If the reconstructed result is equal to 0, proceed to Step 2.

- (k) If the reconstructed result is not equal to 0, k servers S_j set $h = h + 1$ and repeat Steps (a)–(k) until $h = m - g$.
2. If search query b_1, \dots, b_g matches with a_h, \dots, a_{h+g-1} , servers S_j ($j = 0, \dots, k - 1$) send k number of $[a]'_j$ of document a to the searcher. If no matching document is found, the search process is stopped.
3. Using the received k number of $[a]'_j$, the searcher reconstructs document a .

5.4.3 Security of Conjunctive Search

In our conjunctive search, the final process of searching is performed in Step 1(j) of Protocol 5.4; however, $\sum_{y=1}^g \kappa_h(a_v - b_y)^2$ is equal to the sum of differences between g number of characters that had been multiplied with a random number. Here, because the prime number p is set such that $gq^2 < p$, even if one of $a_v - b_y \neq 0$, the reconstructed result is $\sum_{y=1}^g \kappa_h(a_v - b_y)^2 \neq 0 \pmod{p}$ because the random number κ_h does not include the value 0. In other words, if $\sum_{y=1}^g \kappa_h(a_v - b_y)^2 = 0$, we can state that all $a_v - b_y = 0$. This is equal to a simultaneous retrieval process of a string of g number of characters. This means that a total matching search is realized. As follows, we show the security analysis of our proposed conjunctive search. However, we assume the adversary to be a semi-honest adversary (implying that the adversary that pretended to be the authorized user or the corrupted servers follows the protocol specification).

Security of Protocols 5.2 and 5.3

The adversary has no information about the registered document a or search query b ; however, it has information from at most $t = k - 1$ number of servers. When the adversary can identify the unknown information of a, b , the attack is considered a success.

In Protocol 5.2, k number of servers S_j ($j = 0, \dots, k - 1$) each holds the information $[a_x]_j := ([a_x]'_j, [c_x]'_j)$ ($x = 1, \dots, m, j = 0, \dots, k - 1$). In $[a_x]'_j$ and $[c_x]'_j$, encrypted information $\alpha_x(a_x + 1)$ of original document a_x and encrypted information $\gamma_x c_x$ of random numbers c_x are not distributed using a secret sharing scheme; however, both a_x and c_x are encrypted by random numbers α_x and γ_x , respectively. In addition, all random numbers $\alpha_{x,0}, \dots, \alpha_{x,k-1}$ and $\gamma_{x,0}, \dots, \gamma_{x,k-1}$ compose α_x and γ_x , respectively, cannot be retrieved from the $k - 1$ number of servers. Therefore, even if t number of servers collude, the values of α_x and γ_x cannot be reconstructed, and the following statements are true. In addition, the same also remains true for random numbers c_x, γ_x .

$$H(a_x) = H(a_x | [a_x]_0, \dots, [a_x]_{t-1})$$

$$H(\alpha_x) = H(\alpha_x | [a_x]_0, \dots, [a_x]_{t-1})$$

Similarly, in Protocol 5.3, k number of servers S_j ($j = 0, \dots, k-1$) each holds the information $[b_y]_j := ([b_y]'_j, [d_y]'_j)$. However, search query b_y and random number d_y are encrypted with random numbers β_y, δ_y , respectively. In addition, random numbers $\beta_{y,0}, \dots, \beta_{y,k-1}$ and $\delta_{y,0}, \dots, \delta_{y,k-1}$ that compose β_y, δ_y , respectively, would not be leaked even if t number of servers collude. Therefore, the following statements are true. In addition, the same remains true for d_y, δ_y .

$$\begin{aligned} H(b_y) &= H(b_y | [b_y]_0, \dots, [b_y]_{t-1}) \\ H(\beta_y) &= H(\beta_y | [b_y]_0, \dots, [b_y]_{t-1}) \end{aligned}$$

Moreover, if there are k number of shares collected, the values of a_x, c_x ($x = 1, \dots, m$) inputted in Protocol 5.2 and b_y, d_y ($y = 1, \dots, g$) in Protocol 5.3 can be reconstructed correctly. Therefore, we can state that both Protocols 5.2 and 5.3 are secure.

Security of Protocol 5.4

Here, we study the security of our proposed Protocol 5.4 in detail. In an SE using symmetric key encryption, because the owner of the secret information (input) and the searcher are typically the same person, the evaluation of security is performed under the assumption that the adversary only controls the system, without considering the situation where either the owner or the searcher is the adversary.

In contrast to that, in SE using secret sharing, there is no such assumption. Therefore, it is necessary to consider the situation where the searcher is the adversary. Under the above presumption, we consider two types of adversaries. If the adversary can learn the target information, the attack is considered to be successful.

Adversary 1: The system constitutes the adversary. Adversary 1 has information from $k-1$ number of servers. According to this information, the adversary attempts to learn the character string a_x ($x = 1, \dots, m$) of the registered document a , or the character string b_y ($y = 1, \dots, g$) of the search query b .

Adversary 2: The searcher constitutes the adversary. Adversary 2 has all the information from Protocol 5.3 in addition to the information from $k-1$ number of servers. According to this information, the adversary attempts to learn the character string a_x ($x = 1, \dots, m$) of the registered document a .

Proof of security against Adversary 1

Assume that Adversary 1 has information from $t = k-1$ number of servers. In Step 1(b) of Protocol 5.4, Adversary 1 has information about $\rho_h/\gamma_v \varepsilon_v^{(1)}, \rho_h/\delta_y \varepsilon_v^{(2)}, \varphi_h/\alpha_v \varepsilon_v^{(3)}, \varphi_h/\beta_y \varepsilon_v^{(4)}, \varphi_h/\rho_h \varepsilon_v^{(5)}$; in Step 1(d), about $\rho_h(c_v + d_y), \gamma_v c_v, \delta_y d_y$; and in Step 1(f), about $\varphi_h(a_v - b_y + c_v + d_y), \alpha_v(a_v + 1), \beta_y(b_y + 1)$. To simplify the problem, we redefined the parameters above to avoid duplication. As a result, we know that Adversary 1 has the following

information:

$$\frac{\rho_h}{\gamma_v \varepsilon_v^{(1)}}, \frac{\rho_h}{\delta_y \varepsilon_v^{(2)}}, \frac{\varphi_h}{\alpha_v \varepsilon_v^{(3)}}, \frac{\varphi_h}{\beta_y \varepsilon_v^{(4)}}, \frac{\varphi_h}{\rho_h \varepsilon_v^{(5)}},$$

$$\rho_h(c_v + d_y), \varphi_h(a_v - b_y + c_v + d_y), \alpha_v(a_v + 1), \beta_y(b_y + 1), \gamma_v c_v, \delta_y d_y$$

First, from the reconstructed result of $\rho_h(c_v + d_y) \neq 0$, Adversary 1 learns that $c_v + d_y \neq 0$; however, it has no information about $c_v + d_y$. In addition, when $\varphi_h(a_v - b_y + c_v + d_y) = 0$, Adversary 1 learns about the relationship $b_y - a_v = c_v + d_y$. However, because Adversary 1 has no information about $c_v + d_y$, we can state that secret information a_v and search query b_y will not be leaked. Therefore, the following statements are true:

$$H(a_v) = H(a_v | \rho_h(c_v + d_y), \varphi_h(a_v - b_y + c_v + d_y))$$

$$H(b_y) = H(b_y | \rho_h(c_v + d_y), \varphi_h(a_v - b_y + c_v + d_y))$$

Moreover, in Step 1(d), server S_0 reconstructs $\rho_h(c_v + d_y)$ by collecting shares $\llbracket \rho_h(c_v + d_y) \rrbracket_j$ of servers S_j ($j = 0, \dots, k-1$). However, server S_0 will not be able to learn any information regarding $c_v + d_y$ as it is encrypted with the random number ρ_h . Therefore, there is no problem even if server S_0 can learn $\rho_h(c_v + d_y)$, and the following is true:

$$H(c_v + d_y) = H(c_v + d_y | \rho_h(c_v + d_y))$$

In addition, to learn random numbers c_v and secret information a_v of the owner, and random numbers d_y and search query b_y of the searcher from $\alpha_v(a_v + 1), \beta_y(b_y + 1), \gamma_v c_v, \delta_y d_y$, Adversary 1 first has to learn random numbers $\alpha_v, \beta_y, \gamma_v, \delta_y$, which requires random numbers ρ_h, φ_h and $\varepsilon_v^{(1)}, \varepsilon_v^{(2)}, \varepsilon_v^{(3)}, \varepsilon_v^{(4)}, \varepsilon_v^{(5)}$ that cannot be decomposed from the following information A:

$$A = \left(\frac{\rho_h}{\gamma_v \varepsilon_v^{(1)}}, \frac{\rho_h}{\delta_y \varepsilon_v^{(2)}}, \frac{\varphi_h}{\alpha_v \varepsilon_v^{(3)}}, \frac{\varphi_h}{\beta_y \varepsilon_v^{(4)}}, \frac{\varphi_h}{\rho_h \varepsilon_v^{(5)}} \right)$$

Thus, Adversary 1 cannot learn about random numbers $\rho_h, \varphi_h, \varepsilon_v^{(1)}, \varepsilon_v^{(2)}, \varepsilon_v^{(3)}, \varepsilon_v^{(4)}, \varepsilon_v^{(5)}$, and the following statements are true:

$$H(a_v) = H(a_v | \alpha_v(a_v + 1), A)$$

$$H(c_v) = H(c_v | \gamma_v c_v, A)$$

$$H(b_y) = H(b_y | \beta_y(b_y + 1), A)$$

$$H(d_y) = H(d_y | \delta_y d_y, A)$$

In addition, in Step 1(h), Adversary 1 has information about $\kappa_h / (\varphi_h)^2 \varepsilon_v^{(6)}, \kappa_h / \varphi_h \rho_h \varepsilon_v^{(7)}$, and $\kappa_h / (\rho_h)^2 \varepsilon_v^{(8)}$; in Step 1(j), about $\sum_{y=1}^g \kappa_h (a_v - b_y)^2$. However, because $\kappa_h / (\varphi_h)^2 \varepsilon_v^{(6)}$,

$\kappa_h/\varphi_h\rho_h\varepsilon_v^{(7)}$, and $\kappa_h/(\rho_h)^2\varepsilon_v^{(8)}$ cannot be decomposed, random number κ_h will not be leaked. Therefore, even if Adversary 1 has information about $\sum_{y=1}^g \kappa_h(a_v - b_y)^2$, it cannot learn about a_v and b_y . Thus, the following statements are true:

$$\begin{aligned} H(\kappa_h) &= H\left(\kappa_h \left| \frac{\kappa_h}{(\varphi_h)^2\varepsilon_v^{(6)}}, \frac{\kappa_h}{\varphi_h\rho_h\varepsilon_v^{(7)}}, \frac{\kappa_h}{(\rho_h)^2\varepsilon_v^{(8)}} \right.\right) \\ H(a_v) &= H\left(a_v \left| \sum_{y=1}^g \kappa_h(a_v - b_y)^2, \frac{\kappa_h}{(\varphi_h)^2\varepsilon_v^{(6)}}, \frac{\kappa_h}{\varphi_h\rho_h\varepsilon_v^{(7)}}, \frac{\kappa_h}{(\rho_h)^2\varepsilon_v^{(8)}} \right.\right) \\ H(b_y) &= H\left(b_y \left| \sum_{y=1}^g \kappa_h(a_v - b_y)^2, \frac{\kappa_h}{(\varphi_h)^2\varepsilon_v^{(6)}}, \frac{\kappa_h}{\varphi_h\rho_h\varepsilon_v^{(7)}}, \frac{\kappa_h}{(\rho_h)^2\varepsilon_v^{(8)}} \right.\right) \end{aligned}$$

From the arguments above, we can state that Adversary 1 cannot learn a_x ($x = 1, \dots, m$) and b_y ($y = 1, \dots, g$).

Proof of security against Adversary 2

From Protocol 5.3, Adversary 2 has information on the set of conversion random numbers $[\varepsilon_x^{(1)}]_j, [\varepsilon_x^{(3)}]_j$ ($j = 0, \dots, k-1$) in addition to the search query b_y ($y = 1, \dots, g$) and random numbers d_y ($y = 1, \dots, g$) inputted by the searcher. Therefore, Adversary 2 has the information known by Adversary 1 and also about $\varepsilon_x^{(1)}, \varepsilon_x^{(3)}, \beta_y, b_y, \delta_y, d_y$ from Protocol 5.3. To simplify the problem, we redefined the parameters above to avoid any duplication. As a result, we know that Adversary 2 has the following information from Protocols 5.3 and 5.4:

$$\begin{aligned} &\frac{\rho_h}{\gamma_v}, \frac{\rho_h}{\varepsilon_v^{(2)}}, \frac{\varphi_h}{\alpha_v}, \frac{\varphi_h}{\varepsilon_v^{(4)}}, \frac{\varphi_h}{\rho_h\varepsilon_v^{(5)}}, \frac{\kappa_h}{(\varphi_h)^2\varepsilon_v^{(6)}}, \frac{\kappa_h}{\varphi_h\rho_h\varepsilon_v^{(7)}}, \frac{\kappa_h}{(\rho_h)^2\varepsilon_v^{(8)}}, \\ &\varepsilon_v^{(1)}, \varepsilon_v^{(3)}, \rho_h(c_v + d_y), \varphi_h(a_v - b_y + c_v + d_y), \\ &\sum_{y=1}^g \kappa_h(a_v - b_y)^2, \alpha_v(a_v + 1), \gamma_v c_v, \beta_y, b_y, \delta_y, d_y \end{aligned}$$

Here, to learn the random numbers c_v and secret information a_v of the owner from $\rho_h(c_v + d_y), \varphi_h(a_v - b_y + c_v + d_y)$, Adversary 2 needs to learn random numbers ρ_h, φ_h . However, from $\rho_h/\gamma_v, \rho_h/\varepsilon_v^{(2)}, \varphi_h/\alpha_v, \varphi_h/\varepsilon_v^{(4)}$, and $\varphi_h/\rho_h\varepsilon_v^{(5)}$ known to the adversary, random numbers ρ_h, φ_h will not be leaked. In addition, even if Adversary 2 has information about random numbers $\varepsilon_v^{(1)}, \varepsilon_v^{(3)}$, because it has no information about γ_v, α_v or $\varepsilon_v^{(2)}, \varepsilon_v^{(4)}, \varepsilon_v^{(5)}$, random numbers ρ_h, φ_h will not be leaked. Therefore, Adversary 2 cannot learn random numbers c_v and secret information a_v from $\rho_h(c_v + d_y), \varphi_h(a_v - b_y + c_v + d_y)$, and the following

statements are true:

$$\begin{aligned}
H(\rho_h) &= H\left(\rho_h \left| \rho_h(c_v + d_y), \frac{\rho_h}{\gamma_v}, \frac{\rho_h}{\varepsilon_v^{(2)}}, \frac{\varphi_h}{\rho_h \varepsilon_v^{(5)}}, \varepsilon_v^{(1)}, \varepsilon_v^{(3)} \right.\right) \\
H(\varphi_h) &= H\left(\varphi_h \left| \varphi_h(a_v - b_y + c_v + d_y), \frac{\varphi_h}{\alpha_v}, \frac{\varphi_h}{\varepsilon_v^{(4)}}, \frac{\varphi_h}{\rho_h \varepsilon_v^{(5)}}, \varepsilon_v^{(1)}, \varepsilon_v^{(3)} \right.\right) \\
H(c_v) &= H\left(c_v \left| \rho_h(c_v + d_y), \frac{\rho_h}{\gamma_v}, \frac{\rho_h}{\varepsilon_v^{(2)}}, \frac{\varphi_h}{\rho_h \varepsilon_v^{(5)}}, \varepsilon_v^{(1)}, \varepsilon_v^{(3)} \right.\right) \\
H(a_v) &= H\left(a_v \left| \varphi_h(a_v - b_y + c_v + d_y), \frac{\varphi_h}{\alpha_v}, \frac{\varphi_h}{\varepsilon_v^{(4)}}, \frac{\varphi_h}{\rho_h \varepsilon_v^{(5)}}, \varepsilon_v^{(1)}, \varepsilon_v^{(3)} \right.\right)
\end{aligned}$$

In addition, in Step 1(h), Adversary 2 has information about $\kappa_h / (\varphi_h)^2 \varepsilon_v^{(6)}$, $\kappa_h / \varphi_h \rho_h \varepsilon_v^{(7)}$, and $\kappa_h / (\rho_h)^2 \varepsilon_v^{(8)}$; in Step 1(j), about $\sum_{y=1}^g \kappa_h (a_v - b_y)^2$. However, from the arguments above, because it cannot learn random numbers ρ_h, φ_h in addition to random numbers $\varepsilon_v^{(6)}$, $\varepsilon_v^{(7)}$, $\varepsilon_v^{(8)}$, random number κ_h will not be leaked from $\kappa_h / (\varphi_h)^2 \varepsilon_v^{(6)}$, $\kappa_h / \varphi_h \rho_h \varepsilon_v^{(7)}$, and $\kappa_h / (\rho_h)^2 \varepsilon_v^{(8)}$. Therefore, even if Adversary 2 learns about $\sum_{y=1}^g \kappa_h (a_v - b_y)^2$, it cannot learn secret information a_v , and the following statements are true:

$$\begin{aligned}
H(\kappa_h) &= H\left(\kappa_h \left| \frac{\kappa_h}{(\varphi_h)^2 \varepsilon_v^{(6)}}, \frac{\kappa_h}{\varphi_h \rho_h \varepsilon_v^{(7)}}, \frac{\kappa_h}{(\rho_h)^2 \varepsilon_v^{(8)}} \right.\right) \\
H(a_v) &= H\left(a_v \left| \sum_{y=1}^g \kappa_h (a_v - b_y)^2, \beta_y, \delta_y, d_y, \frac{\kappa_h}{(\varphi_h)^2 \varepsilon_v^{(6)}}, \frac{\kappa_h}{\varphi_h \rho_h \varepsilon_v^{(7)}}, \frac{\kappa_h}{(\rho_h)^2 \varepsilon_v^{(8)}} \right.\right)
\end{aligned}$$

From the above, we can state that Adversary 2 cannot learn a_x ($x = 1, \dots, m$).

However, because our proposed conjunctive search cannot count the length of character string g of the search query, the adversary could input a short character string, being able to perform a *brute force search attack*. This can be overcome by implementing a process that allows the system to count the length of a character string before the actual search process; this attack can also be counteracted by using a password to perform a conjunctive search on multiple character strings, as shown below.

Countermeasure: Set a password. Only a user with the correct password and correct search query can successfully perform a search.

For example, in addition to the encryption of document a , the owner encrypts character string p_1, \dots, p_s of the password in Protocol 5.2 (encryption of document); in addition to the encryption of search query b , the searcher encrypts character string q_1, \dots, q_s of the password

in Protocol 5.3 (generation of search query). The search process of Protocol 5.4 (searching process) is first performed on the password. If the password does not match, the search for a keyword is not performed. To perform searches simultaneously, without separating the search processes for the password and the query, a conjunctive search with two character strings (the character strings of the password and search query) can be performed. In other words, instead of computing the following:

$$\sum_{y=1}^g \llbracket \kappa_h(a_v - b_y)^2 \rrbracket_j$$

Each server S_j ($j = 0, \dots, k-1$) compute the following:

$$\sum_{y=1}^g \llbracket \kappa_h(a_v - b_y)^2 \rrbracket_j + \sum_{u=1}^s \llbracket \kappa_h(p_u - q_u)^2 \rrbracket_j$$

By introducing the use of a password, even if the length of the character string of the search query is short, if the total combined length $g + s$ of the password and search query is sufficiently long, a brute force search attack by the adversary can be avoided. In addition, if the adversary changes the inputted password each time, the entropy of the password will decrease; because the password can be easily changed/renewed, by renewing the password regularly, the entropy can be maintained at a constant secure size, enabling the protocol to be resistant to this type of attack.

5.4.4 Extension of Conjunctive Search (when $n > k$)

In our proposed conjunctive search (parameter $n = k$), the random numbers used in Protocol 5.4 are sent directly to all servers participating in the computation. However, if one of the servers is broken and cannot be used, the search computation can no longer continue because random numbers that are handled by that server will be lost. However, if parameter n is set such that $n > k$, we can counteract this problem as follows.

Random numbers used in the computation are not sent directly to all participating servers; instead, they are distributed by using the XOR method of secret sharing [48]. Thus, even if one of the servers is no longer functional, if the substitute server can reconstruct the random numbers used by that particular server, the process of computation can continue.

For example, in Protocol 5.2, if $\alpha_x(a_x + 1)$, $\gamma_x c_x$ are broadcast to all servers, and random numbers $\alpha_{x,j}$, $\gamma_{x,j}$ are distributed using the XOR method to all servers, even if one of the servers is broken, the replacement server can reconstruct the values of $\alpha_{x,j}$, $\gamma_{x,j}$ that correspond to the broken server, and continue to participate in the computation. In other words, by extending the protocol in section 5.4.2, the server loss-resistant characteristic of (k, n) threshold secret sharing can be maintained.

5.5 Proposed Method: Conjunctive and Disjunctive Searches

5.5.1 Overview of Conjunctive and Disjunctive Searches

In this section, we propose an algorithm for performing a disjunctive search; however, when we consider the most typically used case scenario in practical applications, we present a more functional method of SE that combines conjunctive and disjunctive functions (later referred to as functional SE). For example, suppose that there are w different types of search queries for performing a disjunctive search, each with g_l ($l = 1, \dots, w$) number of characters for the conjunctive search.

In addition, suppose a_x ($x = 1, \dots, m$) to be the character string of the registered document, and each $b_y^{(l)}$ ($l = 1, \dots, w, y = 1, \dots, g_l$) to be the character string of w number of search queries. Here, to perform a total matching search on a search query, we only need to compute $\sum (a_x - b_y^{(l)})^2 = \sum c_y^{(l)}$, as shown in Section 5.4.

Therefore, we first compute $\sum c_y^{(l)}$ regarding w number of search queries, and if at least one of the inputted numbers matches, we need to output the value 0. In other words, we need to realize disjunctive search as follows. Out of all $c_y^{(1)}, \dots, c_y^{(w)}$, if at least one does not equal 0, the computation of f such that $f(c_y^{(1)}, \dots, c_y^{(w)}) \neq 0$ becomes the computation of logical disjunction (OR) and can be represented by $c_y^{(1)} + \dots + c_y^{(w)}$. Therefore, out of all $c_y^{(1)}, \dots, c_y^{(w)}$, if one is equal to 0, the computation of f such that $f(c_y^{(1)}, \dots, c_y^{(w)}) = 0$ is as follows:

$$\overline{c_y^{(1)} + \dots + c_y^{(w)}} = c_y^{(1)} \dots c_y^{(w)}$$

Here, to perform a disjunctive search over multiple search queries, we only need to perform the computation of $\prod \sum (a_{h+y} - b_y^{(l)})^2 = \prod (\sum c_y^{(l)})$. (However, if only a disjunctive search is performed, we only need to complete the operation of $\prod (a_{h+y} - b_y^{(l)})$). Here, let h ($h = 0, \dots, m - g$) be the position of the first character for the search process.

In the proposed method, if $a_x, b_y^{(l)} < q$, field $GF(p)$ is set similarly as in our conjunctive search, where $gq^2 < p$. Because the disjunctive search is realized through the computation of multiplication, as long as the elements of multiplication do not include 0, the value of 0 will not appear simply from computing the disjunctive search.

5.5.2 Protocol of Conjunctive and Disjunctive Searches (when $n = k$)

We present our protocol for functional SE below; the process of encryption of a document and the generation of a search query were omitted because they are the same as in Protocols 5.2 and 5.3 of conjunctive search, respectively (however, when generating search queries, Protocol 5.3 is performed w times).

We only show the protocol for the disjunctive search process. For ease of understanding, we explain our protocol under the assumption that there are $w = 2$ number of search queries

(the first search query is $b_y^{(1)}$ ($y = 1, \dots, g_1$) and the second search query is $b_y^{(2)}$ ($y = 1, \dots, g_2$). Therefore, Steps 1–4 of Protocol 5.5 correspond to the total matching search (conjunctive search) for the first search query $b_y^{(1)}$, whereas Steps 5–8 correspond to the total matching search (conjunctive search) for the second search query $b_y^{(2)}$. Step 9 onward shows the disjunctive search of these two search queries $b_y^{(1)}, b_y^{(2)}$. However, if the reconstructed result of either of the following statements is 0, the random numbers $c_v, d_y^{(1)}$ or $c_v, d_y^{(2)}$ are changed and the processes are repeated; otherwise, multiple $c_v, d_y^{(1)}, d_y^{(2)}$ can be sent beforehand, and the computation is repeated by changing the combination.

$$\sum_{y=1}^{g_1} \left(\kappa_h^{(1)} \left((a_v - b_y^{(1)})^2 + c_v + d_y^{(1)} \right) \right) \text{ or } \sum_{y=1}^{g_2} \left(\kappa_h^{(2)} \left((a_v - b_y^{(2)})^2 + c_v + d_y^{(2)} \right) \right)$$

In addition, the proposed method of functional SE is explained under the assumption that $n = k$; however, it can also be made to realize resistance toward server loss by changing such that $n > k$, as shown in section 5.4.4. Note that $v = h + y$.

In addition, random numbers $\kappa_{h,j}^{(1)}, \kappa_{h,j}^{(2)}, \phi_{h,j}$ generated below are elements of $GF(p)$ (do not include 0). All computations including the secret sharing process are performed with finite field p . Sets of conversion random numbers $[\epsilon_x^{(1)}]_j, [\epsilon_x^{(3)}]_j$ for the conjunctive search of each query is generated by the searcher, and the remainder are generated by the owner in advance. Furthermore, in the protocol shown below, let $x = 1, \dots, m$, $y = 1, \dots, g_l$, $l = 1, \dots, w$, $i = 0, \dots, n - 1$, and $j = 0, \dots, k - 1$.

Protocol 5.5: Search Process

- **Input:** $[a]_j, [b1]_j, [b2]_j$ ($j = 0, \dots, k - 1$)

- **Output:** a , or \perp

1. Let $h = 0$. Each server S_j ($j = 0, \dots, k - 1$) performs Steps 1(a)–1(f) of Protocol 5.4 (conjunctive search) for the search query $b_y^{(1)}$ ($y = 1, \dots, g_1$), generates a random number $\kappa_{h,j}^{(1)}$, and computes the following. Then, server S_j ($j = 0, \dots, k - 1$) sends the following to server S_0 :

$$\frac{\kappa_{h,j}^{(1)}}{\left(\phi_{h,j}^{(1)}\right)^2 \epsilon_{v,j}^{(6)}}, \frac{\kappa_{h,j}^{(1)}}{\phi_{h,j}^{(1)} \rho_{h,j}^{(1)} \epsilon_{v,j}^{(7)}}, \frac{\kappa_{h,j}^{(1)}}{\left(\rho_{h,j}^{(1)}\right)^2 \epsilon_{v,j}^{(8)}}, \frac{\kappa_{h,j}^{(1)}}{\rho_{h,j}^{(1)} \epsilon_{v,j}^{(9)}}$$

2. Server S_0 computes the following and sends it to all servers S_j ($j = 0, \dots, k-1$):

$$\begin{aligned}\frac{\kappa_h^{(1)}}{\left(\varphi_h^{(1)}\right)^2 \varepsilon_v^{(6)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}^{(1)}}{\left(\varphi_{h,j}^{(1)}\right)^2 \varepsilon_{v,j}^{(6)}}, \\ \frac{\kappa_h^{(1)}}{\varphi_h^{(1)} \rho_h^{(1)} \varepsilon_v^{(7)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}^{(1)}}{\varphi_{h,j}^{(1)} \rho_{h,j}^{(1)} \varepsilon_{v,j}^{(7)}}, \\ \frac{\kappa_h^{(1)}}{\left(\rho_h^{(1)}\right)^2 \varepsilon_v^{(8)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}^{(1)}}{\left(\rho_{h,j}^{(1)}\right)^2 \varepsilon_{v,j}^{(8)}}, \\ \frac{\kappa_h^{(1)}}{\rho_h^{(1)} \varepsilon_v^{(9)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}^{(1)}}{\rho_{h,j}^{(1)} \varepsilon_{v,j}^{(9)}}\end{aligned}$$

3. Each server S_j ($j = 0, \dots, k-1$) computes the following and sends it to server S_0 :

$$\begin{aligned}\sum_{y=1}^{g_1} \left[\kappa_h^{(1)} \left(\left(a_v - b_y^{(1)} \right)^2 + c_v + d_y^{(1)} \right) \right]_j = \\ \sum_{y=1}^g \left\{ \left[\left(\varphi_h^{(1)} \right)^2 \left(a_v - b_y^{(1)} + c_v + d_y^{(1)} \right)^2 \times \frac{\kappa_h^{(1)}}{\left(\varphi_h^{(1)} \right)^2 \varepsilon_v^{(6)}} \times \llbracket \varepsilon_v^{(6)} \rrbracket_j \right] \right. \\ - \left[2 \times \varphi_h^{(1)} \left(a_v - b_y^{(1)} + c_v + d_y^{(1)} \right) \times \rho_h^{(1)} \left(c_v + d_y^{(1)} \right) \times \frac{\kappa_h^{(1)}}{\varphi_h^{(1)} \rho_h^{(1)} \varepsilon_v^{(7)}} \times \llbracket \varepsilon_v^{(7)} \rrbracket_j \right] \\ + \left[\left(\rho_h^{(1)} \right)^2 \left(c_v + d_y^{(1)} \right)^2 \times \frac{\kappa_h^{(1)}}{\left(\rho_h^{(1)} \right)^2 \varepsilon_v^{(8)}} \times \llbracket \varepsilon_v^{(8)} \rrbracket_j \right] \\ \left. + \left[\rho_h^{(1)} \left(c_v + d_y^{(1)} \right) \times \frac{\kappa_h^{(1)}}{\rho_h^{(1)} \varepsilon_v^{(9)}} \times \llbracket \varepsilon_v^{(9)} \rrbracket_j \right] \right\}\end{aligned}$$

4. Server S_0 reconstructs the following using Shamir's (k, n) method and sends it to all servers S_j ($j = 0, \dots, k-1$):

$$\begin{aligned}\sum_{y=1}^{g_1} \left(\kappa_h^{(1)} \left(\left(a_v - b_y^{(1)} \right)^2 + c_v + d_y^{(1)} \right) \right) &= \kappa_h^{(1)} \left[\sum_{y=1}^{g_1} \left(\left(a_v - b_y^{(1)} \right)^2 + c_v + d_y^{(1)} \right) \right] \\ &= \kappa_h^{(1)} f_h^{(1)}\end{aligned}$$

5. Each server S_j ($j = 0, \dots, k-1$) performs Steps 1(a)–1(f) of Protocol 5.4 (conjunctive search) for the search query $b_y^{(2)}$ ($y = 1, \dots, g_2$), generates a random number $\kappa_{h,j}^{(2)}$, and computes the following. Then, server S_j ($j = 0, \dots, k-1$) sends the following to server

S_0 :

$$\frac{\kappa_{h,j}^{(2)}}{\left(\varphi_{h,j}^{(2)}\right)^2 \varepsilon_{v,j}^{(10)}}, \frac{\kappa_{h,j}^{(2)}}{\varphi_{h,j}^{(2)} \rho_{h,j}^{(2)} \varepsilon_{v,j}^{(11)}}, \frac{\kappa_{h,j}^{(2)}}{\left(\rho_{h,j}^{(2)}\right)^2 \varepsilon_{v,j}^{(12)}}, \frac{\kappa_{h,j}^{(2)}}{\rho_{h,j}^{(2)} \varepsilon_{v,j}^{(13)}}$$

6. Server S_0 computes the following and sends it to all servers S_j ($j = 0, \dots, k-1$):

$$\begin{aligned} \frac{\kappa_h^{(2)}}{\left(\varphi_h^{(2)}\right)^2 \varepsilon_v^{(10)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}^{(2)}}{\left(\varphi_{h,j}^{(2)}\right)^2 \varepsilon_{v,j}^{(10)}}, \\ \frac{\kappa_h^{(2)}}{\varphi_h^{(2)} \rho_h^{(2)} \varepsilon_v^{(11)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}^{(2)}}{\varphi_{h,j}^{(2)} \rho_{h,j}^{(2)} \varepsilon_{v,j}^{(11)}}, \\ \frac{\kappa_h^{(2)}}{\left(\rho_h^{(2)}\right)^2 \varepsilon_v^{(12)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}^{(2)}}{\left(\rho_{h,j}^{(2)}\right)^2 \varepsilon_{v,j}^{(12)}}, \\ \frac{\kappa_h^{(2)}}{\rho_h^{(2)} \varepsilon_v^{(13)}} &= \prod_{j=0}^{k-1} \frac{\kappa_{h,j}^{(2)}}{\rho_{h,j}^{(2)} \varepsilon_{v,j}^{(13)}} \end{aligned}$$

7. Each server S_j ($j = 0, \dots, k-1$) computes the following and sends it to server S_0 :

$$\begin{aligned} &\sum_{y=1}^{g_2} \left[\left[\kappa_h^{(2)} \left((a_v - b_y^{(2)})^2 + c_v + d_y^{(2)} \right) \right]_j = \right. \\ &\sum_{y=1}^{g_2} \left\{ \left[\left(\varphi_h^{(2)} \right)^2 \left(a_v - b_y^{(2)} + c_v + d_y^{(2)} \right)^2 \times \frac{\kappa_h^{(2)}}{\left(\varphi_h^{(2)} \right)^2 \varepsilon_v^{(10)}} \times \left[\varepsilon_v^{(10)} \right]_j \right] \right. \\ &\quad - \left[2 \times \varphi_h^{(2)} \left(a_v - b_y^{(2)} + c_v + d_y^{(2)} \right) \times \rho_h^{(2)} \left(c_v + d_y^{(2)} \right) \times \frac{\kappa_h^{(2)}}{\varphi_h^{(2)} \rho_h^{(2)} \varepsilon_v^{(11)}} \times \left[\varepsilon_v^{(11)} \right]_j \right] \\ &\quad + \left[\left(\rho_h^{(2)} \right)^2 \left(c_v + d_y^{(2)} \right)^2 \times \frac{\kappa_h^{(2)}}{\left(\rho_h^{(2)} \right)^2 \varepsilon_v^{(12)}} \times \left[\varepsilon_v^{(12)} \right]_j \right] \\ &\quad \left. + \left[\rho_h^{(2)} \left(c_v + d_y^{(2)} \right) \times \frac{\kappa_h^{(2)}}{\rho_h^{(2)} \varepsilon_v^{(13)}} \times \left[\varepsilon_v^{(13)} \right]_j \right] \right\} \end{aligned}$$

8. Server S_0 reconstructs the following using Shamir's (k, n) method and sends it to all servers S_j ($j = 0, \dots, k-1$):

$$\begin{aligned} \sum_{y=1}^{g_2} \left(\kappa_h^{(2)} \left((a_v - b_y^{(2)})^2 + c_v + d_y^{(2)} \right) \right) &= \kappa_h^{(2)} \left[\sum_{y=1}^{g_2} \left((a_v - b_y^{(2)})^2 + c_v + d_y^{(2)} \right) \right] \\ &= \kappa_h^{(2)} f_h^{(2)} \end{aligned}$$

9. Each server S_j ($j = 0, \dots, k-1$) generates random number $\phi_{h,j}$, computes the following and sends it to server S_0 :

$$\frac{\phi_{h,j}}{\kappa_{h,j}^{(1)} \kappa_{h,j}^{(2)} \varepsilon_{h,j}^{(14)}}, \frac{\phi_{h,j}}{\rho_{h,j}^{(1)} \kappa_{h,j}^{(2)} \varepsilon_{h,j}^{(15)}}, \frac{\phi_{h,j}}{\rho_{h,j}^{(2)} \kappa_{h,j}^{(1)} \varepsilon_{h,j}^{(16)}}, \frac{\phi_{h,j}}{\rho_{h,j}^{(1)} \rho_{h,j}^{(2)} \varepsilon_{h,j}^{(17)}},$$

10. Server S_0 computes the following and sends it to all servers S_j ($j = 0, \dots, k-1$):

$$\begin{aligned} \frac{\phi_h}{\kappa_h^{(1)} \kappa_h^{(2)} \varepsilon_h^{(14)}} &= \prod_{j=0}^{k-1} \frac{\phi_{h,j}}{\kappa_{h,j}^{(1)} \kappa_{h,j}^{(2)} \varepsilon_{h,j}^{(14)}}, \\ \frac{\phi_h}{\rho_h^{(1)} \kappa_h^{(2)} \varepsilon_h^{(15)}} &= \prod_{j=0}^{k-1} \frac{\phi_{h,j}}{\rho_{h,j}^{(1)} \kappa_{h,j}^{(2)} \varepsilon_{h,j}^{(15)}}, \\ \frac{\phi_h}{\rho_h^{(2)} \kappa_h^{(1)} \varepsilon_h^{(16)}} &= \prod_{j=0}^{k-1} \frac{\phi_{h,j}}{\rho_{h,j}^{(2)} \kappa_{h,j}^{(1)} \varepsilon_{h,j}^{(16)}}, \\ \frac{\phi_h}{\rho_h^{(1)} \rho_h^{(2)} \varepsilon_h^{(17)}} &= \prod_{j=0}^{k-1} \frac{\phi_{h,j}}{\rho_{h,j}^{(1)} \rho_{h,j}^{(2)} \varepsilon_{h,j}^{(17)}} \end{aligned}$$

11. Each server S_j ($j = 0, \dots, k-1$) computes the following:

$$\begin{aligned} &\prod_{l=1}^2 \left[\left[\phi_h \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right) \right]_j = \right. \\ &\quad \left[\kappa_h^{(1)} f_h^{(1)} \times \kappa_h^{(2)} f_h^{(2)} \times \frac{\phi_h}{\kappa_h^{(1)} \kappa_h^{(2)} \varepsilon_h^{(14)}} \times \llbracket \varepsilon_h^{(14)} \rrbracket_j \right] \\ &\quad - \left[\kappa_h^{(2)} f_h^{(2)} \times \sum_{y=1}^{g_1} \rho_h^{(1)} (c_v + d_y^{(1)}) \times \frac{\phi_h}{\rho_h^{(1)} \kappa_h^{(2)} \varepsilon_h^{(15)}} \times \llbracket \varepsilon_h^{(15)} \rrbracket_j \right] \\ &\quad - \left[\kappa_h^{(1)} f_h^{(1)} \times \sum_{y=1}^{g_2} \rho_h^{(2)} (c_v + d_y^{(2)}) \times \frac{\phi_h}{\rho_h^{(2)} \kappa_h^{(1)} \varepsilon_h^{(16)}} \times \llbracket \varepsilon_h^{(16)} \rrbracket_j \right] \\ &\quad \left. + \left[\sum_{y=1}^{g_1} \rho_h^{(1)} (c_v + d_y^{(1)}) \times \sum_{y=1}^{g_2} \rho_h^{(2)} (c_v + d_y^{(2)}) \times \frac{\phi_h}{\rho_h^{(1)} \rho_h^{(2)} \varepsilon_h^{(17)}} \times \llbracket \varepsilon_h^{(17)} \rrbracket_j \right] \right] \end{aligned}$$

12. Servers S_j ($j = 0, \dots, k-1$) cooperate and reconstruct $\prod_{l=1}^2 \phi_h \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right)$ using Shamir's (k, n) method. If the reconstructed result is equal to 0, it is considered to be matched.
13. If the queries do not match, servers S_j ($j = 0, \dots, k-1$) set $h = h + 1$ and repeat Steps 1–12.

14. If either of the search queries $b_y^{(1)}$ or $b_y^{(2)}$ are matched with a_h, \dots, a_{h+g-1} of the registered document a , k number of servers S_j ($j = 0, \dots, k-1$) send $[a]_j'$ containing the matched document to the searcher. If no matching document is found, the search process is stopped.
15. Using the received k number of $[a]_j'$, the searcher reconstructs document a .

5.5.3 Security of Conjunctive and Disjunctive Searches

In our proposed method of functional SE, the final process of searching is performed in Step 12 of Protocol 5.5; however, $\prod_{l=1}^2 \phi_h \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right)$ is equal to the multiplication of the sum of the difference between $w = 2$ number of search queries and the registered document. If the result of $\prod_{l=1}^2 \phi_h \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right) = 0$, out of the inputted $w = 2$ number of search queries, one or more queries matched with the registered document.

However, the encryption of a document and the generation of a search query are the same as in Protocols 5.2 and 5.3 of our conjunctive search, where both protocols were proven to be secure in Section 5.4.3. Therefore, we omit the security analysis of these two protocols here. In addition, Steps 1–8 of Protocol 5.5 are equal to the total matching search (conjunctive search) of search queries 1 and 2, and the security of these steps is the same as in Section 5.4.3; therefore, we omit their security analysis here. In this section, we only show the security analysis for Protocol 5.5 from Step 9 onward.

Proof of security against Adversary 1

Adversary 1 has information from $t \leq k-1$ number of servers. Therefore, when performing a conjunctive search on the first search query, in Steps 1–4, Adversary 1 has the following information:

$$\frac{\kappa_h^{(1)}}{\left(\phi_h^{(1)}\right)^2 \varepsilon_v^{(6)}}, \frac{\kappa_h^{(1)}}{\phi_h^{(1)} \rho_h^{(1)} \varepsilon_v^{(7)}}, \frac{\kappa_h^{(1)}}{\left(\rho_h^{(1)}\right)^2 \varepsilon_v^{(8)}}, \frac{\kappa_h^{(1)}}{\rho_h^{(1)} \varepsilon_v^{(9)}} \\ \sum_{y=1}^{g_1} \left(\kappa_h^{(1)} \left((a_v - b_y^{(1)})^2 + c_v + d_y^{(1)} \right) \right)$$

However, Adversary 1 cannot learn the random number $\kappa_h^{(1)}$ from this information. In addition, as explained in Sections 5.3 and 5.4, because $c_v + d_y^{(1)} \neq 0$, the reconstructed value of $\sum_{y=1}^{g_1} \left(\kappa_h^{(1)} \left((a_v - b_y^{(1)})^2 + c_v + d_y^{(1)} \right) \right)$ will not be 0. Therefore, $\kappa_h^{(1)}$ will not be leaked from Steps 1–8, and we can state that it is secure against Adversary 1. The same can also be said regarding the second search query.

In addition, in Steps 9 and 10, Adversary 1 has the following information:

$$A = \left(\frac{\phi_h}{\kappa_h^{(1)} \kappa_h^{(2)} \varepsilon_h^{(14)}}, \frac{\phi_h}{\rho_h^{(1)} \kappa_h^{(2)} \varepsilon_h^{(15)}}, \frac{\phi_h}{\rho_h^{(2)} \kappa_h^{(1)} \varepsilon_h^{(16)}}, \frac{\phi_h}{\rho_h^{(1)} \rho_h^{(2)} \varepsilon_h^{(17)}} \right)$$

However, because Adversary 1 has no information about random numbers $\varepsilon_h^{(14)}$, $\varepsilon_h^{(15)}$, $\varepsilon_h^{(16)}$, $\varepsilon_h^{(17)}$, it cannot learn random number ϕ_h . Therefore, the following is true:

$$H(\phi_h) = H(\phi_h|A)$$

Furthermore, in Steps 11 and 12, Adversary 1 learns $\prod_{l=1}^2 \phi_h \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right)$. Here, if the reconstructed result is not equal to 0, to learn the result of $\prod_{l=1}^2 \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right)$, random number ϕ_h is required. However, as mentioned before, because Adversary 1 cannot learn random number ϕ_h , we can state that $\prod_{l=1}^2 \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right)$ will not be leaked. In addition, even if $\prod_{l=1}^2 \phi_h \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right) = 0$, because random number ϕ_h does not include value 0, Adversary 1 learns that $\prod_{l=1}^2 \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right) = 0$. From this information, the adversary cannot learn each individual $b_y^{(l)}$ and a_v . Therefore, the following statements are true.

$$H(b_y^{(l)}) = H\left(b_y^{(l)} \left| \prod_{l=1}^2 \phi_h \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right) \right.\right)$$

$$H(a_v) = H\left(a_v \left| \prod_{l=1}^2 \phi_h \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right) \right.\right)$$

From the arguments above, we can state that our proposed method of functional SE is secure against Adversary 1.

Proof of security against Adversary 2

Adversary 2 has information about search queries $b_y^{(1)}, b_y^{(2)}$ and random numbers $d_y^{(1)}, d_y^{(2)}$ inputted by the searcher. However, as argued previously, from Steps 1–8, Adversary 2 will not be able to learn random numbers $\kappa_h^{(1)}, \kappa_h^{(2)}$ and secret information a_v . In addition, in Step 4, because $c_v + d_y^{(1)} \neq 0$, even if $a_v = b_y^{(1)}$, the reconstructed result of the following will not be 0:

$$\sum_{y=1}^{g_1} \left(\kappa_h^{(1)} \left((a_v - b_y^{(1)})^2 + c_v + d_y^{(1)} \right) \right) \neq 0$$

Therefore, information regarding a_v will not be leaked. The same remains true for Step 8. Therefore, we can state that Steps 1–8 are secure against Adversary 2.

In addition, as Adversary 2 has no information about random numbers $\varepsilon_h^{(14)}$, $\varepsilon_h^{(15)}$, $\varepsilon_h^{(16)}$, $\varepsilon_h^{(17)}$, from the following information obtained in steps 9 and 10, the adversary will not be able to learn random number ϕ_h .

$$A = \left(\frac{\phi_h}{\kappa_h^{(1)} \kappa_h^{(2)} \varepsilon_h^{(14)}}, \frac{\phi_h}{\rho_h^{(1)} \kappa_h^{(2)} \varepsilon_h^{(15)}}, \frac{\phi_h}{\rho_h^{(2)} \kappa_h^{(1)} \varepsilon_h^{(16)}}, \frac{\phi_h}{\rho_h^{(1)} \rho_h^{(2)} \varepsilon_h^{(17)}} \right)$$

Therefore, the following is true:

$$H(\phi_h) = H(\phi_h|A)$$

Moreover, Adversary 2 has information regarding $\prod_{l=1}^2 \phi_h \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right)$ from Steps 11 and 12. However, as mentioned above, because Adversary 2 cannot learn random number ϕ_h , we can state that $\prod_{l=1}^2 \left(\sum_{y=1}^{g_l} (a_v - b_y^{(l)})^2 \right)$ will not be leaked.

If the owner and the searcher are not the same person, an adversary without the correct character string a_v can input any search queries $b_y^{(1)}$, $b_y^{(2)}$ into the system. If the adversary performs a multiple search and reconstruction process, as in the conjunctive search, the number of possible candidates for the character string will decrease one by one. Therefore, considering the security against a multiple search process, the attack and countermeasures explained in Section 5.4.3 can conceivably be used.

5.6 Discussion

5.6.1 Comparison with Conventional SEs

In our proposed method, we realized SE using secret sharing instead of the conventional approach using encryption-based techniques such as SE using symmetric key encryption and public key encryption. Secret sharing is considered a more efficient approach because it requires less computation cost compared to conventional encryption-based approaches [73]. This is because most encryption-based approaches such as public key encryption are based on computationally difficult problems, requiring more costly operations (such as modular exponential of large numbers). In contrast, secret sharing, such as Shamir's (k, n) method, requires very low computation cost. In Shamir's (k, n) method, the user only needs to choose a random polynomial and evaluate it in n different points. By implementing secret sharing in SE, we could realize a method with lower computation cost than conventional SE that uses a symmetric key or public key encryption. However, the implementation of secret sharing also means that information needs to be exchanged between at least two servers, which increases the number of communication rounds required compared to conventional SE that uses a symmetric key or public key encryption.

Furthermore, in our proposed methods of SE using secret sharing, we realized direct search over encrypted documents, which is difficult to achieve with the conventional SE. Most of the SEs that use a symmetric key or public key encryption proposed thus far utilize the concept of search index. Because direct search over an encrypted document is not possible, information that is not registered in the index is not searchable. In contrast, to achieve the same function of direct search as our method by using SE based on search index, the owner must first list all possible character string patterns in the sentence to be registered, and must register each of the possible character strings in the index. For example, suppose that an owner wants to register a document. The owner must list all possible character strings (for example, g number of characters starting from the first character, g number of characters starting from the second character, g number of characters starting from the third character, ..., final g number of characters), and register them as keywords in the index. To enable searches on any g number of character strings, the aforementioned process can be performed on $g = 1, 2, \dots, G$ (G is the largest number of characters to be searched) in parallel, but this process is extremely inefficient.

Direct searching over each document will cause the search time to increase linearly to the number of documents stored in the system. This is because every document will need to be queried even when there are only a few documents that matched the query. Here, we could also adapt our proposed method to realize the search index used in conventional methods of SE. To realize this, only the keyword of a document is secretly shared to the servers, and the search is performed directly on the index. In this case, the original document does not need to be secretly shared to multiple servers; instead, we only need to encrypt and store it. In addition, our method can also easily realize both conjunctive and disjunctive searches over an index. The computation cost will also be lower than that of conventional SE that uses a symmetric key or public key encryption.

Next, the differences between SE that uses a symmetric key or public key encryption and that using a secret sharing are the entity of the searcher and the owner of the document. In other words, in SE using secret sharing, anyone can register a document/data, and anyone can produce a valid search query and perform a search on the document/data. In the setting of searching on document/data encrypted with symmetric key encryption, the owner encrypts the document/data, such that only the owner or searcher with the correct secret key can access it. In the setting of searching on document/data encrypted with public key encryption, an owner publishes a public key and anyone with access to the public key can add keywords to the index, but only the owner or searcher with the correct secret key can generate valid search queries to test for the occurrence of a keyword. In other words, in a typical construction of SE that uses a symmetric key or public key encryption, to retrieve the documents that contain the keyword w , an authorized searcher computes a regular search query $Q(sk, w)$ using a secret key sk before sending it to the server. Note that the secret key sk currently used is only known by the owner and by the set of currently authorized searchers. This means that unauthorized

searchers cannot recover the correct secret key sk and, with overwhelming probability, their queries will not yield a valid search query.

5.6.2 Adaptation of the Proposed Methods

Based on the intended applications, our proposed methods could also be converted to realize the same limitation or functionality as in SE that uses a symmetric key or public key encryption, where only authorized searchers can produce a valid search query. In this section, we focus on the following two applications, which also support direct search over encrypted document/data:

1. **Converted Method 1:** Realizing the functionality of SE using symmetric key encryption

The *MAJOR* difference between SE using symmetric key encryption and our proposed method is the entity of the searcher and the owner of the document. In our proposed Protocols 5.2 and 5.3, anyone can register a document/data, and anyone can perform a search on the registered document/data. In contrast, in SE using symmetric key encryption, the owner of the registered document/data and the searcher are typically the same person and only the searcher with the correct secret key can produce a valid search query to perform the searching process. This is because symmetric key encryption generally uses one secret key that is kept secret and used for both encryption and searching of the document. To realize the same functionality as in SE using symmetric key encryption, in addition to searching on the character string, we need to perform an additional search on the secret key. However, when realizing this functionality using our proposed method, we also need to consider the case where the owner of the registered document/data and the searcher are not the same people. This can present a new problem of sharing the secret key between the owner and all the authorized searchers. Here, we eliminate this problem by assuming a secure key sharing protocol using public key encryption such as Diffie–Hellman key exchange method, Rivest–Shamir–Adleman (RSA), or Ephemeral Diffie–Hellman with RSA [74] [75] to securely exchange secret keys between all owners and authorized searchers. Therefore, in this section, we only concentrate on realizing the functionality of SE using symmetric key encryption with supports for direct search over encrypted document/data. In other words, conversion of our proposed method can be summarized as follows:

Protocol 5.6(a): Encryption of document and secret key

1. In addition to document a , the owner encrypts a secret key sk using Protocol 5.2 (encryption of document).

Protocol 5.6(b): Generation of search query and secret key

1. In addition to the search query b , the searcher also encrypts a secret key sk' using Protocol 5.3 (generation of search query).

Protocol 5.6(c): Search process

1. Servers S_j ($j = 0, \dots, k - 1$) first perform Protocol 5.4 (conjunctive searching process) between the registered secret keys sk and the secret key sk' of the searcher.
2. If the result of Step 1 is equal to 0 (meaning that the owner's secret key sk and the searcher's secret key sk' matched), servers S_j ($j = 0, \dots, k - 1$) perform Protocol 5.4 between the search query b and all registered documents. Moreover, a search with multiple queries is also possible by performing Protocol 5.5 (conjunctive and disjunctive searches) for the search queries. However, if the result of Step 1 is not equal to 0, the servers return the value of \perp , and stop the searching process.

Security of Protocols 5.6(a) and 5.6(b)

Here, we assume that the adversary has no information about the registered document a , registered secret key sk , search query b , and secret key sk' ; the adversary has information from at most $t = k - 1$ number of servers. When the adversary can identify the unknown information of a and sk of the owner or b and sk' of the searcher, the attack is considered a success.

As Protocols 5.6(a) and 5.6(b) make use of our proposed Protocols 5.2 and 5.3, which were proven to be secure in Section 5.4.3, we can state that the converted Protocol 5.6(a) and Protocol 5.6(b) are also secure. In other words, document a and secret key sk registered by the owner during Protocol 5.6(a) will not be leaked even if t number of servers collude. The same is true for the query b and secret key sk' inputted by the searcher. Therefore, we can state that Protocols 5.6(a) and 5.6(b) are secure against a semi-honest adversary with information from at most $t = k - 1$ number of servers.

Security of Protocol 5.6(c)

Here, we study the security of Protocol 5.6(c) in detail. In most conventional SE using symmetric key encryption, because the owner of the document and the searcher are typically the same person, the evaluation of security is performed under the assumption that the adversary only controls the system (Adversary 1), without considering the situation where the searcher is the adversary (Adversary 2). However, as Protocol 5.6(c) make use of our proposed Protocol 5.4 (and Protocol 5.5 for multiple queries searching) which was proven to be secure against Adversaries 1 and 2, we can state that Protocol 5.6(c) is secure even if the owner of the document and the searcher are not the same person.

As shown in the security analysis against Adversary 2 in Sections 5.4.3 and 5.5.3, both our proposed Protocols 5.4 and 5.5 are not secure against brute force search attacks if search query/queries with short character string is/are used by Adversary 2 for searching. However,

in this application, as Step 1 of searching is performed on the secret key: if the length of the secret key used is made to be almost the same as the length of the secret key used in SE using symmetric key encryption, we can achieve the same level of security as in typical SE using symmetric key encryption, allowing the converted proposed method to be also secure against brute force attacks by Adversary 2. Therefore, we can state that the converted proposed method is secure against both Adversaries 1 and 2.

2. **Converted Method 2:** Realizing the functionality of SE using public key encryption

To achieve the same restriction as in SE methods using public key encryption where there are multiple registered owners and only one searcher, no limitation is set or required for owners who want to register their document. However, we need to implement an extra process of pre-registration of the secret key to the servers, where searchers that had been authorized by the system can register his/her secret key to the system in advance and use that secret key to perform the searching process. Conversion of our proposed method to realize the functionality of SE using public key encryption can be summarized as follows:

Protocol 5.7(a): Pre-registration of secret key

1. The searcher first registers a secret key sk using Protocol 5.2 (encryption of document) to the servers.

Protocol 5.7(b): Encryption of document

1. The owner encrypts a document a using Protocol 5.2 (encryption of document).

Protocol 5.7(c): Generation of search query and secret key

1. In addition to the search query b , the searcher also encrypts a secret key sk' using Protocol 5.3 (Generation of search query).

Protocol 5.7(d): Search process

1. Servers S_j ($j = 0, \dots, k - 1$) first perform Protocol 5.4 (conjunctive searching process) between the pre-registered secret keys sk and the secret key sk' inputted by the searcher.
2. If the result of Step 1 is equal to 0 (meaning that the owner's secret key sk and the searcher's secret key sk' matched), servers S_j ($j = 0, \dots, k - 1$) perform Protocol 5.4 between the search query b and all registered documents. Moreover, a search with multiple queries is also possible by performing Protocol 5.5 (conjunctive and disjunctive searches) for the search queries. However, if the result of Step 1 is not equal to 0, the servers return the value of \perp , and stop the searching process.

Security of Protocols 5.7(a), 5.7(b), and 5.7(c)

In the Converted Method 2, Protocols 5.7(b) and 5.7(c) are the same as Protocol 5.2 of our conjunctive search and Protocol 5.6(b) of the Converted Method 1, respectively, where both protocols were proven to be secure against Adversaries 1 and 2. Therefore, we omit the security analysis of Protocols 5.7(b) and 5.7(c) here.

In the Converted Method 2, Protocol 5.7(a) is required to allow for all authorized searchers to pre-register their secret keys to the servers. However, as Protocol 5.7(a) also makes use of Protocol 5.2 (which was proven to be secure) to secret share the secret keys to the servers, we can state that Protocol 5.7(a) of the Converted Method 2 is also secure. Therefore, secret key sk registered by the searcher will not be leaked even if $t = k - 1$ number of servers collude together.

Security of Protocol 5.7(d)

Here, Protocol 5.7(d) of the Converted Method 2 is the same as Protocol 5.6(c) of the Converted Method 1, which was proven to be secure against both Adversaries 1 and 2. Therefore, we omit the security analysis here. Moreover, in the Converted Method 2, if the length of the secret key used is set to be the same as in SE using public key encryption, we can realize the same level of security as in conventional SE using public key encryption. In conclusion, we can state that by implementing the use of a pre-registered secret key, we can realize a secure many-to-one search, where multiple owners can register their documents freely and the search process can only be performed by searchers that had secret-shared their secret keys into the system in advance.

5.6.3 Acceptable Information Leakage of SE

When considering the security of SE using symmetric key encryption, there is some predefined information that will definitely be leaked, particularly when a document is added or a search is performed, and there are many cases where a method is considered to be secure as long as no more than the allowed information is leaked (meaning that the information leak is acceptable).

In this section, to perform a comparison between the leaked information of our proposed methods and conventional methods that use symmetric key encryption, we consider the information shown below, which was defined by Curtmola et al. [55] and Hori et al. [76].

Information 1: Length $|D_m|$ of Document D_m

When registering Ciphertext C_m of a Document D_m using symmetric key encryption on a server, because the server can predict the length of Document D_m from its Ciphertext C_m , the information of $|D_m|$ will definitely be leaked. In our proposed method, when performing Protocol 5.2 (Encryption of document), after $[a_x]_j'$, which corresponds to the character string a_x ($x = 1, \dots, m$) of document a (equal to Document D_m), is registered on the server, the server can estimate the length of the registered document from $[a_x]_j'$. Therefore, the length

of the document will be leaked. However, because the server cannot determine the character string a_x of a document from only $k - 1$ number of $[a_x]_j'$ registered, we can state that the actual document will not be leaked. In addition, the length of the search query will also be leaked, but the search query itself will not be leaked. However, because the aforementioned information will definitely be leaked for any SE method that registers the encrypted document and search query on the server, it is not a problem.

Information 2: The outputted result of searching $Did(K')$ when performing keyword matching process with a search query $Q(K')$

In conventional SE, when performing a search using search query $Q(K')$ generated from keyword K' , because the server can learn the result of the search for search query $Q(K')$, information regarding the identifier $Did(K')$ of documents that include keyword K' will be leaked. In our proposed method, when performing a matching or a search process regarding a search query, if the result matches, a value of 0 will be output; however, if the result does not match, random values will be output. In addition, when value 0 is output, k number of servers will send documents that correspond to that search query back to the searcher. Therefore, in our proposed method, when performing a matching process on a search query, information about the search result will be leaked; however, this is the same for most conventional methods for SE.

Information 3: If keyword K' used in i number of searches and keyword K'' used in j number of searches by the searcher is the same.

When the search queries used are definite, the value for $K' = K''$ can be learned by confirming $T(K') = T(K'')$. In our proposed method, even if the same search queries are used, when performing Protocol 5.3 (generation of search query), if different random numbers are used to distribute the search queries, we can state that the aforementioned Information 3 will not be leaked because the generated shares will also be different.

Information 4: Number of keywords correspond to document D_m

When using an index, because the index (equal to keywords) is set for each document, the number of keywords per document will be leaked. In our proposed methods, because any character string can be used as a keyword without the need for an index, we can say that any number of keywords that correspond to a document can exist, but this number cannot be known.

Information 5: Fulfillment of forward privacy

In SE, there is a concept of forward privacy, which is an important property of SE schemes that ensures that newly updated entries cannot be related to previous search results. For example, consider that the server performed a search for keyword u . When a document containing the keyword u is added, the question is whether the server can determine that the newly added document consists of keyword u by using the previous search query. When the

server cannot distinguish whether the document consists of keyword u , the method is said to fulfill the property of forward privacy. In our proposed method, even if a document with the same keyword is added, if the encryption is performed with different random numbers for each character, the result will take a different form even for the same keyword. In addition, as for the search query, even if the same search query is used, if different random numbers are used, the resulting search query will also be different. Therefore, we can state that our proposed method also fulfills the property of forward privacy.

From the above, we can conclude that our proposed methods do not leak any information other than the length of the document/search query and the identifier of the matching document. In particular, because our methods do not leak Information 3–Information 5, we can state that they realize a high level of security for a keyword/search query.

5.7 Chapter Summary

In this chapter, we proposed two SE methods using the secure MPC proposed in Chapter 4. In the first proposed method, we realized a total matching search function (or conjunctive search) with multiple characters by using the inverse computation of the logical conjunctive. Additionally, in the second method, we realized the disjunctive search function using the inverse computation of the logical disjunctive. By combining it with the previous conjunctive search function, we also realized the direct search of a document using multiple keywords.

In future studies, to perform a more detailed comparison with methods of SE using index search, we need to adapt our method to the index search concept, implement it, and compare the performance with that of conventional methods. Additionally, we also need to perform a detailed implementation and compare it with conventional methods of SE to evaluate whether our proposed SE is also fast in real-world use.

Chapter 6

Multiplication of Polynomials with $N < 2k - 1$ Servers

In Chapters 3 and 4, we proposed methods of secure MPC that realize secure computation including multiplication with $n < 2k - 1$ by using the approach of scalar value \times polynomial to perform multiplication ab . However, both methods require a certain condition, particularly on the input/output to achieve security against semi-honest adversaries (including Adversary 1 that only controls $t \leq k - 1$ of n computing servers). Therefore, the following question arises:

“Is there another manner to perform MPC with fewer conditions?”

In this chapter, aiming to find a method for easing the conditions required in the MPC when assuming $n < 2k - 1$, we study another approach of realizing multiplication in MPC. Instead of the scalar value \times polynomial approach shown in Chapters 3 and 4, we use the approach of multiplication by polynomial \times polynomial. We also implement the method of reducing the polynomial degree of the resulting polynomial back to $(k - 1)$. This enables the result of multiplication to be recovered from $n < 2k - 1$ shares, as in Sections 3 and 4. Moreover, as the encrypted secret input is not reconstructed as a scalar value, Condition (1), where the inputs do not include 0, is not required when assuming only Adversary 1. However, some conditions are still required when assuming other semi-honest adversaries.

6.1 Introduction

Secure MPC allows a set of servers to jointly compute an arbitrary function of their inputs without revealing these inputs to each other. Typically, two main techniques have been proposed to implement secure MPC: homomorphic encryption [33] [34] [19] [35] [17] [36] and threshold secret sharing [12] [15] [37][39] [77] [16] [23] [40][41] [42] [43]. Homomorphic encryption requires more computational resources, whereas threshold secret sharing has a relatively low computational cost. Therefore, threshold secret sharing is preferable in a cloud system.

The classical result of secure MPC using the (k, n) threshold secret sharing is that n servers can compute any function such that any subset of up to $k - 1 < n/2$ servers obtains no information about the inputs of other servers, except for what can be derived from public information [12] [78] [79] [80]. Conventional methods of secure MPC using Shamir's (k, n) threshold secret sharing perform addition by locally adding shares together. However, this is not the case for multiplication. For example, let secrets a and b be encoded by $f(x)$ and $g(x)$, two polynomials with degrees $(k - 1)$. The free coefficient of polynomial $h(x) = f(x)g(x)$ is ab .

However, when using $h(x)$ to encode the product of $a \times b$, the degree of $h(x)$ increases from $(k - 1)$ to $(2k - 2)$. In most conventional methods, this poses no problem for interpolating $h(x)$ from its n shares because it is assumed that $n \geq 2k - 1$. Each server holds only one share for each secret; hence, for each multiplication performed, the number of required servers increases from k to $2k - 1$. Therefore, the construction of information-theoretically secure multiplication under the dishonest majority setting is considered impossible, and approaches to avoid this impossibility result are as follows: (1) giving up information-theoretic security [19] [35] [36] (typically, generating a Beaver triple between computing parties via computationally secure primitives such as homomorphic encryption or oblivious transfer); and (2) assuming a trusted setup by other than computing parties [42].

To overcome this impossibility, we apply a different functionality of multiplication using encrypted shares instead of “normal” shares. This is different from most standard MPCs, where multiplication is performed on shares of secret inputs a, b to produce shares of ab . Instead, our functionality performs multiplication by using shares encrypted with random numbers (encrypted shares) and outputs encrypted shares of multiplication result ab . This is the same method as in Chapters 3 and 4, where the secret input is first encrypted with a random number. During multiplication, the encrypted secret is momentarily restored as a scalar value, and multiplication uses the scalar value \times polynomial approach to prevent an increase in the degree of the polynomial. However, these methods require some conditions, particularly regarding the reconstructed scalar value.

Watanabe et al.[43] also proposed a solution with a different approach by differentiating the relationship between the number of required servers N and parameter n of (k, n) threshold secret sharing. Thus, the authors used $N \leq 2k - 1$ servers to implement $n \geq 2k - 1$ multiplication. However, their method did not solve the problem of decreasing the degree of the polynomial from $(2k - 2)$ to $(k - 1)$. Therefore, although the multiplication was performed using only $N \leq 2k - 1$ servers, their method required $2k - 1$ instead of k shares to restore the multiplication result.

In this chapter, unlike the approach used in Chapters 3 and 4, we focus on solving the problem of multiplication through the conventional polynomial \times polynomial approach. Moreover, we apply the idea of Watanabe et al. of differentiating the parameters N and n , and introduce a new method of reducing the polynomial degree from $(2k - 2)$ to $(k - 1)$

such that the result of multiplication ab can be recovered from only $n < 2k - 1$ shares. The contributions of this chapter can be summarized as follows.

- We propose a new distribution protocol in which multiple encrypted shares of the same secret input can be sent to each computing server. This is implemented by encrypting each share with a different random number before sending it to the computing servers;
- We propose a new multiplication protocol and degree reduction method for multiplying $(k - 1)$ sharing of encrypted shares of two inputs a, b and reducing the degree of resulting shares from $(2k - 2)$ to $(k - 1)$ using the recombination vector to produce $(k - 1)$ sharing of encrypted shares of ab using only $N < 2k - 1$ servers;
- We present a clear evaluation of our method in terms of computation, communication, and round number. We also include the results of the implementation of our method in MATLAB. Finally, we compare the proposed method with the conventional methods of two-party multiplication of encrypted shares and show that our method can reduce the number of required servers (reduction of the total operating cost) and minimize the required communication between the client and N computing servers.

6.2 System Model and Adversary

In this chapter, we also assume a client-server model [15] [40] [41] [81] [82] for multiplication $c = ab$ of two inputs a, b . In addition, we assume a semi-honest adversary, where the adversary follows the protocol specification but may try to learn more than what is allowed by the protocol, with at most $k - 1$ corrupted servers. We also assume secure communication between clients and servers. However, we follow the definition of MPC security in [40] [41], where we only assume that there are at most $k - 1$ corrupted computing servers in the client-server computing model.

This is the same as the definition of Adversary 1 defined in Chapter 3. The main difference with the standard definitions of MPC is that in this setting, we do not assume that the adversary can simultaneously corrupt $k - 1$ servers and a client (non-server). This provides a weaker security guarantee than the standard security of MPC but is sufficient in many real-life cases, particularly in cloud computing (where the service provider or operator has no benefit to gain for colluding with any of the other clients). Furthermore, this scenario can be implemented by other means, such as physical means or the use of the law (we also show in Section 6.7 that the method is secure against the remaining Adversaries 2 and 3 with some conditions).

In this chapter, we state and prove all security guarantees by expressing entropy in the presence of a semi-honest adversary, as in Chapters 3 and 4, and we show that the computing

servers cannot learn the clients' inputs. In a future study, we also plan to evaluate security using the standard ideal/real-world definition of MPC proposed by Canetti et al.[83].

6.3 Related Work

Here, we introduce related work on two-party multiplication. Moreover, we also include the work by Watanabe et al. [43] that realizes two-party multiplication by multiplying encrypted shares based on (k, n) threshold secret sharing.

6.3.1 Two-party Multiplication Using Shamir's (k, n) Method

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

$$\begin{aligned} \llbracket a \rrbracket_i &= a + \alpha_1 x_i + \alpha_2 x_i^2 + \dots + \alpha_{k-1} x_i^{k-1} \\ \llbracket b \rrbracket_i &= b + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_{k-1} x_i^{k-1} \end{aligned}$$

Next, each server computes the multiplication between shares of a and b , obtaining $\llbracket ab \rrbracket_i$ as follows:

$$\llbracket ab \rrbracket_i = ab + \dots + (\alpha_{k-1} \beta_{k-1}) x_i^{2k-2}$$

Although secret inputs 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 the conventional method of multiplication in Shamir's (k, n) method is that the number of shares required to reconstruct ab increases from k to $2k - 1$. Thus, the following Theorem 1 was proposed for the passive model.

Theorem 1. In the passive model, a set $S = \{S_0, \dots, S_{n-1}\}$ of n servers can compute every specification securely if and only if the adversary corrupts at most $k - 1 < n/2$ of the servers.

6.3.2 Multiplication of Shares Using the Recombination Vector

As mentioned in the previous section, the result of the multiplication of two polynomials of degree $(k - 1)$ is a polynomial of degree $(2k - 2)$. Note that $n \geq 2k - 1$ implies that n product shares are sufficient for recovering ab . However, any further multiplication raises the degree, and once the degree passes n , there will not be sufficient points for interpolation. Hence, $(k - 1)$ sharing of ab can be achieved by using a recombination vector, as shown by Chaum et al [37].

To better understand this, let us assume that $k = 2, n = 2k - 1 = 3$, and the resulting multiplication is a quadratic polynomial $y(x_i) = \alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2$, where α_0 is the result of

the multiplication. As $n = 3$, the shares for each x_i are as follows:

$$\begin{aligned} y(1) &= \alpha_0 + \alpha_1 + \alpha_2 \\ y(2) &= \alpha_0 + 2\alpha_1 + 4\alpha_2 \\ y(3) &= \alpha_0 + 3\alpha_1 + 9\alpha_2 \end{aligned}$$

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

More precisely, each player first shares his computed value of the multiplication result $\llbracket ab \rrbracket_i$ using polynomials of $(k - 1)$ degrees to all players. Next, the player locally combines their shares by an inner product with the recombination vector. Thus, each player holds $(k - 1)$ sharing of ab . However, the problem with this method is that it requires $n > 2k - 1$ servers, thereby increasing the total operation cost of the system.

6.3.3 Watanabe et al.'s Method

Typically, in a (k, n) threshold secret sharing, a server only has one share. For multiplication of shares, the number of required servers increases from k to $2k - 1$.

Watanabe et al.[43] solved this problem by allowing a server to hold two shares. However, the problem of an increase in the polynomial's degree from $(k - 1)$ to $(2k - 2)$ remains. The number of shares required to reconstruct the result remains $2k - 1$ instead of k . Therefore, the communication cost between the client and the servers remains the same as in all conventional methods. Our method solves this problem by proposing a method of computing $(k - 1)$ sharing (instead of typical $(2k - 2)$ sharing) of ab using only $N \geq k$ servers.

We included the distribution, multiplication, and reconstruction protocols of Watanabe et al.'s method under the settings of $N \geq k$ and $n \geq 2k - 1$. Variables a, b and all generated random numbers are derived from finite field $GF(p)$, and all computations are performed under $GF(p)$.

Protocol 6.1(a): Distribution of a, b

1. Each player A and B generates $2n$ shares from secrets a and b and distributes the first n shares $\llbracket a \rrbracket_i, \llbracket b \rrbracket_i$ ($i = 0, \dots, n - 1$) to n servers S_i .
2. Player A generates a random number r_A and distributes $\llbracket r_A a \rrbracket_n, \dots, \llbracket r_A a \rrbracket_{2n-1}$ to n servers S_i . Then, distributes shares $\llbracket r_A \rrbracket_i$ of r_A to n servers S_i .
3. Player B generates a random number r_B and distributes $\llbracket r_B b \rrbracket_n, \dots, \llbracket r_B b \rrbracket_{2n-1}$ to n servers S_i . Then, distributes shares $\llbracket r_B \rrbracket_i$ of r_B to n servers S_i .

Protocol 6.1(b): Multiplication of ab

1. Each server S_i ($i = 0, \dots, n - 1$) calculates the following:

$$\begin{aligned} \llbracket ab \rrbracket_i &= \llbracket a \rrbracket_i \times \llbracket b \rrbracket_i \quad (i = 0, \dots, n - 1) \\ \llbracket a_A r_B ab \rrbracket_{i+k} &= \llbracket r_A a \rrbracket_{i+k} \times \llbracket r_B b \rrbracket_{i+k} \quad (i = 0, \dots, n - 1) \end{aligned}$$

Protocol 6.1(c): Reconstruction of ab

1. The player collects $\llbracket r_A \rrbracket_j, \llbracket r_B \rrbracket_j$ ($j = 0, \dots, k - 1$) from k servers S_j and reconstructs values r_A, r_B .
2. The player collects $\llbracket ab \rrbracket_j, \llbracket r_A r_B ab \rrbracket_{j+k}$ from k servers S_j ($j = 0, \dots, k - 1$) and computes the following:

$$\llbracket ab \rrbracket_{j+k} = \frac{\llbracket r_A r_B ab \rrbracket_{j+k}}{r_A r_B}$$

3. The player reconstructs multiplication result ab from $2k$ number of shares $\llbracket ab \rrbracket_j, \llbracket ab \rrbracket_{j+k}$.

6.4 Proposed Method: Multiplication of Two Polynomials (when $N = k$)

6.4.1 Overview of Proposed Method

In this section, we describe our first protocol that differentiates between parameter N , which is the number of servers that are actually needed, and parameter n of Shamir's (k, n) method. This protocol performs multiplication under the settings of $N = k, n \geq 2k - 1$.

To perform multiplication using only $N = k$ servers, we based our distribution protocol on the method by Watanabe et al. [43] by sending multiple shares of the same secret input to each server. However, instead of encrypting only one of the shares, our method encrypts both shares using different random numbers before sending the encrypted shares to each server. Moreover, to solve the problem of Watanabe et al.'s 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 degree of the polynomial of ab from $(2k - 2)$ to $(k - 1)$ by using a recombination vector with only $N = k$ servers.

In addition, because our method distributes encrypted shares, our multiplication protocol also produces an encrypted share (where shares of ab are encrypted with a random number). In contrast, most two-party multiplications using MPC take shares of a, b as the input and output shares of ab . However, this functionality requires an extra process of decrypting the output in addition to the reconstruction of the result using Shamir's (k, n) method.

6.4.2 Protocol of Proposed Method

In this section, we demonstrate our multiplication method for $N = k$ servers S_j ($j = 0, \dots, k - 1$) and with $n \geq 2k - 1$ as the number of required shares. For ease of understanding, we also include an example of multiplication of $a = 3, b = 5$ in Appendix B.

Notation:

- $\llbracket a \rrbracket_j$: Share of secret input a for server S_j ($j = 0, \dots, k - 1$), where the number of shares n required for reconstructing a is k .
- $\llbracket \alpha_1 \beta_1 ab \rrbracket_j^*$: Share of $\alpha_1 \beta_1 ab$ for server S_j ($j = 0, \dots, k - 1$), where the number of shares n required for reconstructing $\alpha_1 \beta_1 ab$ is $2k - 1$.

Protocol 6.2: Distribution of secret inputs a, b

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 $n = 2k$ shares of secret input a using Shamir's $(k, 2k)$ method and computes the following:

$$\llbracket \alpha_1 a \rrbracket_0 = \alpha_1 \times \llbracket a \rrbracket_0, \dots, \llbracket \alpha_1 a \rrbracket_{k-1} = \alpha_1 \times \llbracket a \rrbracket_{k-1}$$

$$\llbracket \alpha_2 a \rrbracket_k = \alpha_2 \times \llbracket a \rrbracket_k, \dots, \llbracket \alpha_2 a \rrbracket_{2k-1} = \alpha_2 \times \llbracket a \rrbracket_{2k-1}$$

3. Player A sends $\llbracket \alpha_1 a \rrbracket_j, \llbracket \alpha_2 a \rrbracket_{j+k}, \alpha_{1,j}, \alpha_{2,j}$ to server S_j ($j = 0, \dots, k - 1$).
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 $n = 2k$ shares of secret input b using Shamir's $(k, 2k)$ method and computes the following:

$$\begin{aligned} \llbracket \beta_1 b \rrbracket_0 &= \beta_1 \times \llbracket b \rrbracket_0, \dots, \llbracket \beta_1 b \rrbracket_{k-1} = \beta_1 \times \llbracket b \rrbracket_{k-1} \\ \llbracket \beta_2 b \rrbracket_k &= \beta_2 \times \llbracket b \rrbracket_k, \dots, \llbracket \beta_2 b \rrbracket_{2k-1} = \beta_2 \times \llbracket b \rrbracket_{2k-1} \end{aligned}$$

6. Player B sends $\llbracket \beta_1 b \rrbracket_j, \llbracket \beta_2 b \rrbracket_{j+k}, \beta_{1,j}, \beta_{2,j}$ to server S_j ($j = 0, \dots, k-1$).

Protocol 6.3: Multiplication of secret inputs a, b

1. Each server S_j ($j = 0, \dots, k-1$) computes the following:

$$\begin{aligned} \llbracket \alpha_1 \beta_1 ab \rrbracket_j^* &= \llbracket \alpha_1 a \rrbracket_j \times \llbracket \beta_1 b \rrbracket_j \\ \llbracket \alpha_2 \beta_2 ab \rrbracket_{j+k}^* &= \llbracket \alpha_2 a \rrbracket_{j+k} \times \llbracket \beta_2 b \rrbracket_{j+k} \\ \alpha_{1,j} \beta_{1,j} &= \alpha_{1,j} \times \beta_{1,j} \\ \alpha_{2,j} \beta_{2,j} &= \alpha_{2,j} \times \beta_{2,j} \end{aligned}$$

2. Each server S_j ($j = 0, \dots, k-1$) generates a random number γ_j , computes the following values, and sends them to one of the servers (here, we assume server S_0):

$$\frac{\gamma_j}{\alpha_{1,j} \beta_{1,j}}, \frac{\gamma_j}{\alpha_{2,j} \beta_{2,j}}$$

3. Server S_0 computes the following values and sends them to all servers S_j ($j = 0, \dots, k-1$):

$$\begin{aligned} \frac{\gamma}{\alpha_1 \beta_1} &= \prod_{j=0}^{k-1} \frac{\gamma_j}{\alpha_{1,j} \beta_{1,j}} \\ \frac{\gamma}{\alpha_2 \beta_2} &= \prod_{j=0}^{k-1} \frac{\gamma_j}{\alpha_{2,j} \beta_{2,j}} \end{aligned}$$

4. Each server S_j ($j = 0, \dots, k-1$) computes $[[\gamma ab]]_j^*, [[\gamma ab]]_{j+k}^*$ as follows and distributes the results using Shamir's (k, N) method to all servers S_i ($j = 0, \dots, k-1$):

$$\begin{aligned} [[\gamma ab]]_j^* &= \frac{\gamma}{\alpha_1 \beta_1} \times [[\alpha_1 \beta_1 ab]]_j^* \\ [[\gamma ab]]_{j+k}^* &= \frac{\gamma}{\alpha_2 \beta_2} \times [[\alpha_2 \beta_2 ab]]_{j+k}^* \\ [[\gamma ab]]_j^* &= \begin{cases} [[\gamma ab]]_{j,0} & \rightarrow \text{send to } S_0 \\ \vdots \\ [[\gamma ab]]_{j,k-1} & \rightarrow \text{send to } S_{k-1} \end{cases} \\ [[\gamma ab]]_{j+k}^* &= \begin{cases} [[\gamma ab]]_{j+k,0} & \rightarrow \text{send to } S_0 \\ \vdots \\ [[\gamma ab]]_{j+k,k-1} & \rightarrow \text{send to } S_{k-1} \end{cases} \end{aligned}$$

5. Each server S_j ($j = 0, \dots, k-1$) computes the following value (λ_i are the recombination vectors):

$$[[\gamma ab]]_j = \lambda_0 \times [[\gamma ab]]_{0,j} + \dots + \lambda_{k-1} \times [[\gamma ab]]_{2k-1,j}$$

Protocol 6.4: Reconstruction of output ab

1. The player collects $[[\gamma ab]]_j, \gamma_j$ from k servers S_j ($j = 0, \dots, k-1$), reconstructs γab , and computes γ as follows:

$$\gamma = \prod_{j=0}^{k-1} \gamma_j$$

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

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

6.4.3 Security of Proposed Method

In a two-party multiplication, when the adversary knows an input (e.g., input a) and output (e.g., output ab), the second input (e.g., input b) will be leaked. Furthermore, if all parties are corrupted, the adversary also knows all secret inputs in the system. Therefore, we only consider the following non-colluding semi-honest adversary, as described in Chapters 3 and 4. The attack is considered a success if the adversary can learn 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 adversary:

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

Next, we evaluate the security of our proposed method.

Evaluation of security of Protocol 6.2 (Distribution of secret inputs a, b)

In our proposed distribution protocol, two encrypted shares for each secret input are sent to each server. In the typical Shamir's (k, n) method, if multiple shares are distributed to one server, k shares are leaked from $k - 1$ servers (this immediately violates the security of (k, n) threshold secret sharing). For example, if the secret input a is distributed using Shamir's $(k, 2n)$ sharing ($2n$ shares of input a are computed using a $(k - 1)$ degree polynomial), and two shares are sent to each server, $2(k - 1)$ shares are leaked from $k - 1$ servers, and the adversary can reconstruct the secret input from these shares. Therefore, this approach violates the security requirement of (k, n) threshold secret sharing, where information from any $k - 1$ or fewer servers reveals no information about the original secret input.

To overcome this restriction, we send encrypted shares instead of "normal shares". Each share is encrypted with a different random number. In our proposed Protocol 6.2, Adversary 1 has the following information: D_A from Player A and D_B from Player B.

$$D_A = \llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k}, \alpha_{1,l}, \alpha_{2,l} \quad (l = 0, \dots, k - 2)$$

$$D_B = \llbracket \beta_1 b \rrbracket_l, \llbracket \beta_2 b \rrbracket_{l+k}, \beta_{1,l}, \beta_{2,l} \quad (l = 0, \dots, k - 2)$$

However, encrypted secrets $\alpha_1 a, \alpha_2 a, \beta_1 b, \beta_2 b$ do not leak from $k - 1$ encrypted shares $\llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k}, \llbracket \beta_1 b \rrbracket_l, \llbracket \beta_2 b \rrbracket_{l+k}$. Moreover, Adversary 1 cannot learn 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 expressions are true:

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

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

We can state that our proposed Protocol 6.2 is secure even if multiple encrypted shares are sent to each server, and Protocol 6.2 is secure against Adversary 1.

Evaluation of security of Protocol 6.3 (Multiplication of secret inputs a, b)

From Protocol 6.2, Adversary 1 has the following information D_A from Player A and D_B from Player B:

$$D_A = \llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k}, \alpha_{1,l}, \alpha_{2,l} \quad (l = 0, \dots, k - 2)$$

$$D_B = \llbracket \beta_1 b \rrbracket_l, \llbracket \beta_2 b \rrbracket_{l+k}, \beta_{1,l}, \beta_{2,l} \quad (l = 0, \dots, k - 2)$$

Moreover, in Step 1 of Protocol 6.3, Adversary 1 learns $\alpha_{1,l}\beta_{1,l}, \alpha_{2,l}\beta_{2,l}$ ($l = 0, \dots, k - 2$). In Step 2, it learns $\gamma_l, \gamma_l/\alpha_{1,l}\beta_{1,l}, \gamma_l/\alpha_{2,l}\beta_{2,l}$. In Step 3, $\gamma/\alpha_1\beta_1, \gamma/\alpha_2\beta_2$. In Step 4, $\llbracket \gamma ab \rrbracket_l^*, \llbracket \gamma ab \rrbracket_{l+k}^*$. In Step 5, $\llbracket \gamma ab \rrbracket_l$. As a result, we can transform security evaluation into

the problem of determining whether Adversary 1 can learn inputs a, b or output ab from the following information:

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

$$\llbracket \gamma ab \rrbracket_l^*, \llbracket \gamma ab \rrbracket_{l+k}^*, \llbracket \gamma ab \rrbracket_l \quad (l = 0, \dots, k-2)$$

Because $\llbracket \gamma ab \rrbracket_l^*$ is represented by a polynomial of $(2k-2)$ degree, $2k-1$ shares are required to reconstruct γab . However, Adversary 1 only has information about $2k-2$ shares; therefore, γab is not leaked, and the following statement is true:

$$H(\gamma ab) = H(\gamma ab | \llbracket \gamma ab \rrbracket_l^*, \llbracket \gamma ab \rrbracket_{l+k}^*) \quad (l = 0, \dots, k-2)$$

The same is true when Adversary 1 only has information about $k-2$ shares $\llbracket \gamma ab \rrbracket_l$. Therefore, γab is not leaked, and the following statement is true:

$$H(\gamma ab) = H(\gamma ab | \llbracket \gamma ab \rrbracket_l) \quad (l = 0, \dots, k-2)$$

Moreover, because Adversary 1 has no information about $\alpha_1, \alpha_2, \beta_1, \beta_2$, the random number γ used to encrypt output ab is not leaked. Thus, the following statements are true:

$$H(\alpha_1) = H\left(\alpha_1 | \alpha_{1,l}, \beta_{1,l}, \gamma, \frac{\gamma}{\alpha_1\beta_1}\right) \quad (l = 0, \dots, k-2)$$

$$H(\alpha_2) = H\left(\alpha_2 | \alpha_{2,l}, \beta_{2,l}, \gamma, \frac{\gamma}{\alpha_2\beta_2}\right) \quad (l = 0, \dots, k-2)$$

$$H(\beta_1) = H\left(\beta_1 | \alpha_{1,l}, \beta_{1,l}, \gamma, \frac{\gamma}{\alpha_1\beta_1}\right) \quad (l = 0, \dots, k-2)$$

$$H(\beta_2) = H\left(\beta_2 | \alpha_{2,l}, \beta_{2,l}, \gamma, \frac{\gamma}{\alpha_2\beta_2}\right) \quad (l = 0, \dots, k-2)$$

$$H(\gamma) = H\left(\gamma | \alpha_{1,l}, \alpha_{2,l}, \beta_{1,l}, \beta_{2,l}, \gamma, \frac{\gamma}{\alpha_1\beta_1}, \frac{\gamma}{\alpha_2\beta_2}\right) \quad (l = 0, \dots, k-2)$$

From the arguments above, we can state that our proposed method is secure against Adversary 1.

6.5 Extension of Proposed Method (when $N > k$)

6.5.1 Protocol of the Extended Method

In the previous protocol, parameter N is limited to $N = k$. Therefore, if a server breaks down, the multiplication process is no longer possible. To solve this problem, we can set $N > k$. For

example, if $N = 3$ and $k = 2$, and one server breaks down, the MPC remains possible with the remaining two servers.

Here, we show the protocol for multiplication of secret inputs a and b for $N > k$. Secret inputs a, b , all random numbers, and all computations are in finite field $GF(p)$.

Protocol 6.5: Distribution of secret inputs a, b

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 $n = 2N$ shares of secret input a using Shamir's $(k, 2N)$ method and computes the following. Next, Player A generates $n = N$ shares for each random number $\alpha_{1,j}, \alpha_{2,j}$ ($j = 0, \dots, k-1$) using Shamir's (k, N) method.

$$\llbracket \alpha_1 a \rrbracket_0 = \alpha_1 \times \llbracket a \rrbracket_0, \dots, \llbracket \alpha_1 a \rrbracket_{N-1} = \alpha_1 \times \llbracket a \rrbracket_{N-1}$$

$$\llbracket \alpha_2 a \rrbracket_N = \alpha_2 \times \llbracket a \rrbracket_N, \dots, \llbracket \alpha_2 a \rrbracket_{2N-1} = \alpha_2 \times \llbracket a \rrbracket_{2N-1}$$

3. Player A sends the following information to server S_i ($i = 0, \dots, N-1$):

$$\llbracket \alpha_1 a \rrbracket_i, \llbracket \alpha_2 a \rrbracket_{i+N},$$

$$\llbracket \alpha_{1,j} \rrbracket_i, \llbracket \alpha_{2,j} \rrbracket_i$$

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 $n = 2N$ shares of secret input b using Shamir's $(k, 2N)$ method and computes the following. Next, Player B generates $n = N$ shares for each random

number $\beta_{1,j}, \beta_{2,j}$ ($j = 0, \dots, k-1$) using Shamir's (k, N) method.

$$\begin{aligned} \llbracket \beta_1 b \rrbracket_0 &= \beta_1 \times \llbracket b \rrbracket_0, \dots, \llbracket \beta_1 b \rrbracket_{N-1} = \beta_1 \times \llbracket b \rrbracket_{N-1} \\ \llbracket \beta_2 b \rrbracket_N &= \beta_2 \times \llbracket b \rrbracket_N, \dots, \llbracket \beta_2 b \rrbracket_{2N-1} = \beta_2 \times \llbracket b \rrbracket_{2N-1} \end{aligned}$$

6. Player B sends the following information to server S_i ($i = 0, \dots, N-1$):

$$\begin{aligned} \llbracket \beta_1 b \rrbracket_i, \llbracket \beta_2 b \rrbracket_{i+N}, \\ \llbracket \beta_{1,j} \rrbracket_i, \llbracket \beta_{2,j} \rrbracket_i \end{aligned}$$

Protocol 6.6: Multiplication of secret inputs a, b

1. Each server S_j ($j = 0, \dots, k-1$) reconstructs random numbers $\alpha_{1,j}, \alpha_{2,j}, \beta_{1,j}, \beta_{2,j}$ by collecting the following shares from k servers S_j :

$$\begin{aligned} \llbracket \alpha_{1,j} \rrbracket_0, \dots, \llbracket \alpha_{1,j} \rrbracket_{k-1} \\ \llbracket \alpha_{2,j} \rrbracket_0, \dots, \llbracket \alpha_{2,j} \rrbracket_{k-1} \\ \llbracket \beta_{1,j} \rrbracket_0, \dots, \llbracket \beta_{1,j} \rrbracket_{k-1} \\ \llbracket \beta_{2,j} \rrbracket_0, \dots, \llbracket \beta_{2,j} \rrbracket_{k-1} \end{aligned}$$

2. Each server S_j ($j = 0, \dots, k-1$) performs Steps 1–4 of Protocol 6.3.

3. Each server S_i ($i = 0, \dots, N-1$) computes the following values (λ_i are the recombination vectors):

$$\llbracket \gamma ab \rrbracket_i = \lambda_0 \times \llbracket \gamma ab \rrbracket_{0,i} + \dots + \lambda_{2k-1} \times \llbracket \gamma ab \rrbracket_{2k-1,i}$$

4. Each server S_j ($j = 0, \dots, k-1$) distributes a random number γ_j generated in Step 2 to all servers S_i using Shamir's (k, N) method.

5. Each server S_i ($i = 0, \dots, N-1$) holds the following shares:

$$\llbracket \gamma ab \rrbracket_i, \llbracket \gamma_0 \rrbracket_i, \dots, \llbracket \gamma_{k-1} \rrbracket_i$$

Protocol 6.7: Reconstruction of ab

1. The player collects $\llbracket \gamma ab \rrbracket_j, \llbracket \gamma_0 \rrbracket_j, \dots, \llbracket \gamma_{k-1} \rrbracket_j$ from k servers S_j ($j = 0, \dots, k - 1$), reconstructs $\gamma ab, \gamma_0, \dots, \gamma_{k-1}$, and computes γ and multiplication result ab as follows:

$$\gamma = \prod_{j=0}^{k-1} \gamma_j$$

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

6.5.2 Security of the Extended Method

The difference between this extended protocol and the protocol shown in Section 6.4.2 is that the following random numbers are secret-shared using Shamir's (k, n) method to all servers instead of sending them directly to each server S_j ($j = 0, \dots, k - 1$):

$$\alpha_{1,j}, \alpha_{2,j}, \beta_{1,j}, \beta_{2,j}, \gamma_j \quad (j = 0, \dots, k - 1)$$

Thus, even if $N - k$ servers break down, the remaining k servers can still reconstruct the random numbers needed in Step 1 of Protocol 6.6, which provides resistance against loss of servers.

Furthermore, in Protocol 6.5, Players A and B distribute random numbers $\alpha_{1,j}, \alpha_{2,j}, \beta_{1,j}, \beta_{2,j}$ to all servers S_i ($i = 0, \dots, N - 1$). Because there are at most $k - 1$ corrupted servers, the adversary can learn the following $k - 1$ shares:

$$\llbracket \alpha_{1,j} \rrbracket_l, \llbracket \alpha_{2,j} \rrbracket_l, \llbracket \beta_{1,j} \rrbracket_l, \llbracket \beta_{2,j} \rrbracket_l \quad (l = 0, \dots, k - 2)$$

However, random numbers $\alpha_{1,j}, \alpha_{2,j}, \beta_{1,j}, \beta_{2,j}$ will not be leaked from the aforementioned $k - 1$ shares. Therefore:

$$H(\alpha_{1,j}) = H(\alpha_{1,j} | \llbracket \alpha_{1,j} \rrbracket_0, \dots, \llbracket \alpha_{1,j} \rrbracket_{k-2})$$

$$H(\alpha_{2,j}) = H(\alpha_{2,j} | \llbracket \alpha_{2,j} \rrbracket_0, \dots, \llbracket \alpha_{2,j} \rrbracket_{k-2})$$

$$H(\beta_{1,j}) = H(\beta_{1,j} | \llbracket \beta_{1,j} \rrbracket_0, \dots, \llbracket \beta_{1,j} \rrbracket_{k-2})$$

$$H(\beta_{2,j}) = H(\beta_{2,j} | \llbracket \beta_{2,j} \rrbracket_0, \dots, \llbracket \beta_{2,j} \rrbracket_{k-2})$$

In addition, in Step 4 of Protocol 6.6, each server S_j ($j = 0, \dots, k - 1$) distributes a random number γ_j to all servers S_i ($i = 0, \dots, N - 1$). The adversary can learn $k - 1$ shares of random numbers γ_j . However, random numbers γ_j will not be leaked. Therefore:

$$H(\gamma_j) = H(\gamma_j | \llbracket \gamma_j \rrbracket_0, \dots, \llbracket \gamma_j \rrbracket_{k-2})$$

Finally, the remaining steps of the extended protocol are identical to those described in Section 6.4.2. Therefore, we can state that the protocol for $N > k$ is also secure against

Adversary 1 as long as no more than $k - 1$ servers collude (we omit the detailed analysis here).

6.6 Limitation of the Proposed Method

This section discusses the limitation of the method proposed in Section 6.4. First, as mentioned in Section 6.2 regarding the adversary assumed in this chapter, we only assumed a semi-honest adversary that can corrupt up to $k - 1$ of the N computing servers in the client-server model setting. This means that the adversary will only have access to $k - 1$ servers and is equal to Adversary 1 defined in Chapter 3. In Section 6.4.2, we proved that the proposed method is secure against Adversary 1.

However, security against the remaining Adversaries 2 and 3 defined in Chapter 3 should be assessed. Here, we will discuss the security of the proposed method against these adversaries. For simplicity, we will only discuss the security of Protocol 6.3 against the following adversaries:

Adversary 2: When one of the players who inputted the secret input is the adversary, the adversary will be able to know one of the secret inputs. Moreover, the adversary also has information from $k - 1$ servers, and attempt to learn about the other input of the other player or the output of the computation.

Adversary 3: When the player who reconstructs the computation result is the adversary, the adversary will be able to learn about the information sent by k servers required to reconstruct the result. Moreover, the adversary also has information from $k - 1$ servers, and attempts to learn about the secret inputs.

Evaluation of security of Protocol 6.3 against Adversary 2

For ease of understanding, suppose that Adversary 2 controls the player who inputted secret b . In this case, this adversary will learn about random numbers β_1, β_2 (and $\beta_{1,0}, \dots, \beta_{1,k-1}, \beta_{2,0}, \dots, \beta_{2,k-1}$ that compose it) and secret input b , in addition to the following information regarding secret input a from Protocol 6.2:

$$D_A = \llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k}, \alpha_{1,l}, \alpha_{2,l} \quad (l = 0, \dots, k-2)$$

Moreover, in Step 1 of Protocol 6.3, Adversary 1 learns $\alpha_{1,l}\beta_{1,l}, \alpha_{2,l}\beta_{2,l}$ ($l = 0, \dots, k-2$). In Step 2, it learns $\gamma_l, \gamma_l/\alpha_{1,l}\beta_{1,l}, \gamma_l/\alpha_{2,l}\beta_{2,l}$. In Step 3, $\gamma/\alpha_1\beta_1, \gamma/\alpha_2\beta_2$. In Step 4, $\llbracket \gamma ab \rrbracket_l^*, \llbracket \gamma ab \rrbracket_{l+k}^*$. In Step 5, $\llbracket \gamma ab \rrbracket_l$. As a result, we can transform security evaluation into the problem of determining whether Adversary 2 can learn inputs a or output ab from the

following information:

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

$$\llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k}, \llbracket \gamma ab \rrbracket_l^*, \llbracket \gamma ab \rrbracket_{l+k}^*, \llbracket \gamma ab \rrbracket_l$$

$$(j = 0, \dots, k-1, l = 0, \dots, k-2)$$

From γ/α_1 and γ/α_2 , Adversary 2 will learn the ratio α_1/α_2 . Moreover, from α_1/α_2 and $\llbracket \alpha_2 a \rrbracket_{l+k}$, Adversary 2 will also learn $\llbracket \alpha_1 a \rrbracket_{l+k}$. As $\alpha_1 a$ is represented by a polynomial of $(k-1)$ degree, encrypted secret input $\alpha_1 a$ will be leaked from $2k-2$ shares of $\llbracket \alpha_1 a \rrbracket_l$, $\llbracket \alpha_1 a \rrbracket_{l+k}$. The same is true for encrypted secret input $\alpha_2 a$.

However, as the random numbers α_1, α_2 used to encrypt secret input a is not leaked, we can state that the proposed method is secure assuming the following Condition (1). This is the same as in the method in Chapters 3 and 4, where if the secret input $a = 0$, the reconstructed encrypted secret inputs $\alpha_1 a = \alpha_2 a = 0$, revealing that the secret input $a = 0$ to Adversary 2.

Condition 1. Secret inputs do not include 0.

Therefore, the following statements are true for random numbers α_1, α_2 and secret input a :

$$H(\alpha_1) = H\left(\alpha_1 \mid \alpha_{1,l}, \gamma, \frac{\gamma}{\alpha_1}, \frac{\alpha_1}{\alpha_2}\right) \quad (l = 0, \dots, k-2)$$

$$H(\alpha_2) = H\left(\alpha_2 \mid \alpha_{2,l}, \gamma, \frac{\gamma}{\alpha_2}, \frac{\alpha_1}{\alpha_2}\right) \quad (l = 0, \dots, k-2)$$

$$H(a) = H(a \mid b, \alpha_1 a, \alpha_2 a)$$

Next, we discuss the output ab . As Adversary 2 knows the values of $\alpha_1 a$ and $\alpha_2 a$ in addition to γ/α_1 and γ/α_2 , it will also be able to learn γa . With the information of secret input b , Adversary 2 will eventually be able to learn about the encrypted output γab . However, because this adversary cannot separate each random number α_1, α_2 from α_1/α_2 , the random number γ used to encrypt output ab is not leaked from γ/α_1 and γ/α_2 . Thus, the following statements are true assuming that Condition (1) is fulfilled:

$$H(\gamma) = H\left(\gamma \mid \alpha_{1,l}, \alpha_{2,l}, \gamma, \frac{\gamma}{\alpha_1}, \frac{\gamma}{\alpha_2}\right) \quad (l = 0, \dots, k-2)$$

$$H(ab) = H(ab \mid \gamma ab, b)$$

Therefore, from the arguments above, we can state that our proposed method is secure against Adversary 2 as long as the aforementioned condition (1) is followed.

Evaluation of security of Protocol 6.3 against Adversary 3

Here, suppose that Adversary 3 controls the player who reconstructed the multiplication

result ab . In this case, Adversary 3 will learn about random number γ (and $\gamma_0, \dots, \gamma_{k-1}$ that compose it), encrypted output γab and the multiplication result ab , in addition to the following information regarding secret inputs a and b from Protocol 6.2:

$$\begin{aligned} D_A &= \llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k}, \alpha_{1,l}, \alpha_{2,l} \quad (l = 0, \dots, k-2) \\ D_B &= \llbracket \beta_1 b \rrbracket_l, \llbracket \beta_2 b \rrbracket_{l+k}, \beta_{1,l}, \beta_{2,l} \quad (l = 0, \dots, k-2) \end{aligned}$$

Moreover, in Step 1 of Protocol 6.3, Adversary 1 learns $\alpha_{1,l}\beta_{1,l}, \alpha_{2,l}\beta_{2,l}$ ($l = 0, \dots, k-2$). In Step 2, it learns $\gamma_l, \gamma_l/\alpha_{1,l}\beta_{1,l}, \gamma_l/\alpha_{2,l}\beta_{2,l}$. In Step 3, $\gamma/\alpha_1\beta_1, \gamma/\alpha_2\beta_2$. In Step 4, $\llbracket \gamma ab \rrbracket_l^*$, $\llbracket \gamma ab \rrbracket_{l+k}^*$. In Step 5, $\llbracket \gamma ab \rrbracket_l$. As a result, we can transform security evaluation into a problem of determining whether Adversary 3 can learn secret inputs a and b from the following information:

$$\begin{aligned} &\alpha_{1,l}, \alpha_{2,l}, \beta_{1,l}, \beta_{2,l}, \gamma, \frac{\gamma}{\alpha_1\beta_1}, \frac{\gamma}{\alpha_2\beta_2}, \gamma, \gamma ab, ab \\ &\llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k}, \llbracket \beta_1 b \rrbracket_l, \llbracket \beta_2 b \rrbracket_{l+k}, \llbracket \gamma ab \rrbracket_l^*, \llbracket \gamma ab \rrbracket_{l+k}^*, \llbracket \gamma ab \rrbracket_l \quad (l = 0, \dots, k-2) \end{aligned}$$

From $\gamma/\alpha_1\beta_1, \gamma/\alpha_2\beta_2$ and γ , Adversary 3 will learn $\alpha_1\beta_1$ and $\alpha_2\beta_2$. However, it will not be able to separate each random number $\alpha_1, \alpha_2, \beta_1, \beta_2$ from $\alpha_1\beta_1$ and $\alpha_2\beta_2$. Therefore, the following statements are true for random numbers $\alpha_1, \alpha_2, \beta_1, \beta_2$:

$$\begin{aligned} H(\alpha_1) &= H(\alpha_1 | \alpha_{1,l}, \beta_{1,l}, \alpha_1\beta_1) \quad (l = 0, \dots, k-2) \\ H(\alpha_2) &= H(\alpha_2 | \alpha_{2,l}, \beta_{2,l}, \alpha_2\beta_2) \quad (l = 0, \dots, k-2) \\ H(\beta_1) &= H(\beta_1 | \alpha_{1,l}, \beta_{1,l}, \alpha_1\beta_1) \quad (l = 0, \dots, k-2) \\ H(\beta_2) &= H(\beta_2 | \alpha_{2,l}, \beta_{2,l}, \alpha_2\beta_2) \quad (l = 0, \dots, k-2) \end{aligned}$$

Moreover, all encrypted secret inputs $\alpha_1 a, \alpha_2 a, \beta_2 b$ and $\beta_1 b$ will not be leaked from only $k-1$ shares. Therefore, we can state that Adversary 3 will not be able to learn each secret input a, b from the aforementioned information. In addition, even if the Adversary 3 learns the multiplication result ab , the information of each secret input a and b will not be leaked. Therefore, the following statements are true:

$$\begin{aligned} H(a) &= H(a | \gamma, \alpha_1\beta_1, \alpha_2\beta_2, ab, \llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k}) \quad (l = 0, \dots, k-2) \\ H(b) &= H(b | \gamma, \alpha_1\beta_1, \alpha_2\beta_2, ab, \llbracket \beta_1 b \rrbracket_l, \llbracket \beta_2 b \rrbracket_{l+k}) \quad (l = 0, \dots, k-2) \end{aligned}$$

However, when one or both of the secret inputs are equal to 0, Adversary 3 will learn that $ab = 0$; as this is the same for any methods of secure computation and Adversary 3 cannot differentiate whether both $a = b = 0$ or only one is equal to 0, there is no problem.

Therefore, from the arguments above, we can state that our proposed method is secure against Adversary 3 without any conditions required.

TABLE 6.1: Communication and number of rounds of the proposed method (when $N = k$)

Process	Communication	Rounds
Distribution of a	$4kd_1$	1
Distribution of b	$4kd_1$	
Multiplication ab	Step 2 $2kd_1$	3
	Step 3 $2kd_1$	
	Step 4 $2k^2d_1$	
Reconstruction of ab	$2kd_1$	1

6.7 Discussion

6.7.1 Computational and Communication Costs

In this section, we evaluate our method proposed in Sections 6.7 and 6.8 in terms of computational and communication costs. First, we define the parameters used in our evaluation.

Definition of Parameters:

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

Note that in (k, n) threshold secret sharing, share d_1 usually has almost the same size as the original secret, and the computational costs are different for the distribution and the reconstruction processes. However, for ease of understanding, we consider that these computational costs are the same.

Tables 6.1 and 6.2 show the communication costs and the number of rounds for our proposed method. Tables 6.3 and 6.4 show the corresponding computational costs. From Tables 6.1 and 6.2, we learn that to extend our method to $N > k$, extra communications for the distribution and reconstruction of random numbers are required during the multiplication and reconstruction processes. The same is true for our extended protocol for $N > k$, and Table 6.4 shows that an extra computational cost is required compared with the case when $N = k$.

TABLE 6.2: Communication and number of rounds of the extended method (when $N > k$)

Process	Communication	Rounds
Distribution of a	$2Nd_1(k+1)$	1
Distribution of b	$2Nd_1(k+1)$	
Multiplication ab	Step 1 $4k^2d_1$	5
	Step 2 $2kd_1(N+2)$	
	Step 4 kNd_1	
Reconstruction of ab	$kd_1(k+1)$	1

TABLE 6.3: Computational cost of proposed method (when $N = k$)

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

TABLE 6.4: Computational cost of the extended method (when $N > k$)

Process	Computation Cost
Distribution of a, b	Step 1 $2(k-1)M$
	Step 2 $C_1 + 2kC_2 + 2NM$
	Step 4 $2(k-1)M$
	Step 5 $C_1 + 2kC_2 + 2NM$
	Step 1 $4kC_2$
Multiplication of ab	Step 2 $10kM + 2kD - 2M + 2kC_2$
	Step 3 $N(2kM + (2k-1)A)$
	Step 4 kC_2
	Reconstruction of ab

TABLE 6.5: Computational time of proposed method when $N = k = 2$ (for m multiplications)

Process		Computation time [s]			
		$m = 1,000$	$m = 10,000$	$m = 100,000$	$m = 1,000,000$
Distribution	Player A	0.000115	0.000871	0.008235	0.047846
	Player B	0.000106	0.000515	0.005008	0.079328
Multiplication		0.000718	0.006587	0.068876	0.653845
Reconstruction		0.000115	0.000815	0.008255	0.076631

6.7.2 Experimental Implementation of Proposed Method

In this section, we present the results of our implementation and simulation using MATLAB. Details of the implementation environment are presented below. However, this implementation was performed without any optimizations, such as parallel computation. In our implementation, we assume the minimum number of servers $N = 2$ and parameter $k = 2$ using our proposed method for $N = k$. In addition, the time shown in Table 6.5 is the time taken to complete each process m times.

Implementation Environment

- OS: Windows 10 Home 64 bit
- CPU: AMD Ryzen 5 3600 6-Core @ 3.6 GHz
- Memory: 16.00 GB
- Program: MATLAB R2019b

The results of our proposed method are presented in Table 6.5. However, for ease of implementation, a single computer played the role of both servers (S_0, S_1) and players. Therefore, the computational time for multiplication shown in Table 6.5 can be said to be approximately twice the time required for a single server (without considering the time of communication).

Specifically, Table 6.5 shows that each player requires less than 0.006 s to distribute 100,000 secrets to servers S_0 and S_1 . According to Table 6.5, the time taken to compute 1,000,000 multiplications was less than 1 s. Therefore, we can state that our proposed multiplication method is very efficient in terms of computational time.

6.7.3 Repetition of Multiplication

In Sections 6.4 and 6.5 of this chapter, we showed that the protocol for a single operation of two-party is secure against a non-colluding adversary that controls up to $k - 1$ computing

servers. We also showed that the proposed method is conditionally secure against colluding Adversaries 2 and 3. Our proposed method can also perform repetition of multiplication, where the result of the multiplication is used for consecutive multiplication. For example, the result of multiplication ab can be used to perform multiplication with a new input c to produce the result of abc without increasing the number of computing servers N needed.

To achieve this, the first multiplication protocol must also output $n = 2k$ number of shares of the multiplication result, instead of only $n = k$ shares as shown in Sections 6.4 and 6.5. In sequence, we show the multiplication protocol for computing $n = 2k$ shares of multiplication result ab . Here, we assume the scenario for $N = k$.

Protocol 6.8: Multiplication with $n = 2k$ shares of the multiplication result

1. Each server S_j ($j = 0, \dots, k-1$) computes the following:

$$\begin{aligned} \llbracket \alpha_1 \beta_1 ab \rrbracket_j^* &= \llbracket \alpha_1 a \rrbracket_j \times \llbracket \beta_1 b \rrbracket_j \\ \llbracket \alpha_2 \beta_2 ab \rrbracket_{j+k}^* &= \llbracket \alpha_2 a \rrbracket_{j+k} \times \llbracket \beta_2 b \rrbracket_{j+k} \\ \alpha_{1,j} \beta_{1,j} &= \alpha_{1,j} \times \beta_{1,j} \\ \alpha_{2,j} \beta_{2,j} &= \alpha_{2,j} \times \beta_{2,j} \end{aligned}$$

2. Each server S_j ($j = 0, \dots, k-1$) generates random numbers $\gamma_{1,j}$, $\gamma_{2,j}$, computes the following, and sends to one of the servers (here, we assume server S_0):

$$\frac{\gamma_{1,j}}{\alpha_{1,j} \beta_{1,j}}, \frac{\gamma_{1,j}}{\alpha_{2,j} \beta_{2,j}}, \frac{\gamma_{2,j}}{\alpha_{1,j} \beta_{1,j}}, \frac{\gamma_{2,j}}{\alpha_{2,j} \beta_{2,j}}$$

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

$$\begin{aligned} \frac{\gamma_1}{\alpha_1 \beta_1} &= \prod_{j=0}^{k-1} \frac{\gamma_{1,j}}{\alpha_{1,j} \beta_{1,j}}, \\ \frac{\gamma_1}{\alpha_2 \beta_2} &= \prod_{j=0}^{k-1} \frac{\gamma_{1,j}}{\alpha_{2,j} \beta_{2,j}}, \\ \frac{\gamma_2}{\alpha_1 \beta_1} &= \prod_{j=0}^{k-1} \frac{\gamma_{2,j}}{\alpha_{1,j} \beta_{1,j}}, \\ \frac{\gamma_2}{\alpha_2 \beta_2} &= \prod_{j=0}^{k-1} \frac{\gamma_{2,j}}{\alpha_{2,j} \beta_{2,j}} \end{aligned}$$

4. Each server S_j ($j = 0, \dots, k-1$) computes $n = 2k$ shares using Shamir's $(k, 2k)$ method:

$$\begin{aligned} \llbracket \alpha_1 \beta_1 ab \rrbracket_j^* &= \begin{cases} \llbracket \alpha_1 \beta_1 ab \rrbracket_{j,0} \\ \vdots \\ \llbracket \alpha_1 \beta_1 ab \rrbracket_{j,2k-1} \end{cases} \\ \llbracket \alpha_2 \beta_2 ab \rrbracket_{j+k}^* &= \begin{cases} \llbracket \alpha_2 \beta_2 ab \rrbracket_{j+k,0} \\ \vdots \\ \llbracket \alpha_2 \beta_2 ab \rrbracket_{j+k,2k-1} \end{cases} \end{aligned}$$

5. Each server S_j ($j = 0, \dots, k-1$) computes the following:

$$\begin{aligned} \llbracket \gamma_1 ab \rrbracket_{j,0} &= \frac{\gamma_1}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_{j,0}, \dots, \llbracket \gamma_1 ab \rrbracket_{j,k-1} = \frac{\gamma_1}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_{j,k-1} \\ \llbracket \gamma_2 ab \rrbracket_{j,k} &= \frac{\gamma_2}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_{j,k}, \dots, \llbracket \gamma_2 ab \rrbracket_{j,2k-1} = \frac{\gamma_2}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_{j,2k-1} \\ \llbracket \gamma_1 ab \rrbracket_{j+k,0} &= \frac{\gamma_1}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_{j+k,0}, \dots, \llbracket \gamma_1 ab \rrbracket_{j+k,k-1} = \frac{\gamma_1}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_{j+k,k-1} \\ \llbracket \gamma_2 ab \rrbracket_{j+k,k} &= \frac{\gamma_2}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_{j+k,k}, \dots, \llbracket \gamma_2 ab \rrbracket_{j+k,2k-1} = \frac{\gamma_2}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_{j+k,2k-1} \end{aligned}$$

For example, suppose $N = k = 3$, server S_0 computes the following:

$$\begin{aligned} \llbracket \gamma_1 ab \rrbracket_{0,0} &= \frac{\gamma_1}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_{0,0}, \llbracket \gamma_1 ab \rrbracket_{0,1} = \frac{\gamma_1}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_{0,1}, \\ \llbracket \gamma_1 ab \rrbracket_{0,2} &= \frac{\gamma_1}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_{0,2}, \llbracket \gamma_2 ab \rrbracket_{0,3} = \frac{\gamma_2}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_{0,3}, \\ \llbracket \gamma_2 ab \rrbracket_{0,4} &= \frac{\gamma_2}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_{0,4}, \llbracket \gamma_2 ab \rrbracket_{0,5} = \frac{\gamma_2}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_{0,5}, \\ \llbracket \gamma_1 ab \rrbracket_{3,0} &= \frac{\gamma_1}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_{3,0}, \llbracket \gamma_1 ab \rrbracket_{3,1} = \frac{\gamma_1}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_{3,1}, \\ \llbracket \gamma_1 ab \rrbracket_{3,2} &= \frac{\gamma_1}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_{3,2}, \llbracket \gamma_2 ab \rrbracket_{3,3} = \frac{\gamma_2}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_{3,3}, \\ \llbracket \gamma_2 ab \rrbracket_{3,4} &= \frac{\gamma_2}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_{3,4}, \llbracket \gamma_2 ab \rrbracket_{3,5} = \frac{\gamma_2}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_{3,5} \end{aligned}$$

6. Each server S_j ($j = 0, \dots, k-1$) sends $\llbracket \gamma_1 ab \rrbracket_{j,j'}$, $\llbracket \gamma_1 ab \rrbracket_{j+k,j'}$, $\llbracket \gamma_2 ab \rrbracket_{j,j'+k}$, $\llbracket \gamma_2 ab \rrbracket_{j+k,j'+k}$ to server $S_{j'}$ ($j' = 0, \dots, k-1$).

For example, suppose $N = k = 3$, server S_0 sends the following to server S_1 :

$$\llbracket \gamma_1 ab \rrbracket_{0,1}, \llbracket \gamma_1 ab \rrbracket_{3,1}, \llbracket \gamma_2 ab \rrbracket_{0,4}, \llbracket \gamma_2 ab \rrbracket_{3,4}$$

7. Each S_j ($j = 0, \dots, k-1$) computes the following using the recombination vector r :

$$\begin{aligned} \llbracket \gamma_1 ab \rrbracket_j &= \lambda_0 \times \llbracket \gamma_1 ab \rrbracket_{0,j} + \dots + \lambda_{2k-1} \times \llbracket \gamma_1 ab \rrbracket_{2k-1,j} \\ \llbracket \gamma_2 ab \rrbracket_{j+k} &= \lambda_0 \times \llbracket \gamma_2 ab \rrbracket_{0,j+k} + \dots + \lambda_{2k-1} \times \llbracket \gamma_2 ab \rrbracket_{2k-1,j+k} \end{aligned}$$

8. Steps 1–5 are repeated for multiplication with secret input c . However, if no more consecutive computation is required, the multiplication protocol in Protocol 6.3 is used instead to produce only $n = k$ shares of the multiplication result.

Evaluation of security against Adversary 1

The difference between the protocol shown above and the protocol shown in Section 6.7 is that each computing server computes an extra k shares of the result in steps 2–7. Therefore, in Steps 1–7, Adversary 3 learns the following. Note that, $l = 0, \dots, k-2$.

$$\begin{aligned} \alpha_{1,l} \beta_{1,l}, \alpha_{2,l} \beta_{2,l}, \gamma_{1,l}, \gamma_{2,l}, \frac{\gamma_{1,l}}{\alpha_{1,l} \beta_{1,l}}, \frac{\gamma_{1,l}}{\alpha_{2,l} \beta_{2,l}}, \frac{\gamma_{2,l}}{\alpha_{1,l} \beta_{1,l}}, \frac{\gamma_{2,l}}{\alpha_{2,l} \beta_{2,l}}, \frac{\gamma_1}{\alpha_1 \beta_1}, \frac{\gamma_1}{\alpha_2 \beta_2}, \frac{\gamma_2}{\alpha_1 \beta_1}, \frac{\gamma_2}{\alpha_2 \beta_2} \\ \llbracket \alpha_1 \beta_1 ab \rrbracket_l^*, \llbracket \alpha_2 \beta_2 ab \rrbracket_{l+k}^*, \llbracket \gamma_1 ab \rrbracket_l, \llbracket \gamma_2 ab \rrbracket_{l+k} \end{aligned}$$

Moreover, Adversary 1 learns the following from the distribution protocol 6.2 of secret inputs a and b :

$$\begin{aligned} D_A &= \llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k}, \alpha_{1,l}, \alpha_{2,l} \quad (l = 0, \dots, k-2) \\ D_B &= \llbracket \beta_1 b \rrbracket_l, \llbracket \beta_2 b \rrbracket_{l+k}, \beta_{1,l}, \beta_{2,l} \quad (l = 0, \dots, k-2) \end{aligned}$$

As a result, we can transform the security evaluation into the problem of determining whether Adversary 1 can learn secret inputs a, b and the output ab from the following information:

$$\begin{aligned} \alpha_{1,l}, \beta_{1,l}, \alpha_{2,l}, \beta_{2,l}, \gamma_{1,l}, \gamma_{2,l}, \frac{\gamma_1}{\alpha_1 \beta_1}, \frac{\gamma_1}{\alpha_2 \beta_2}, \frac{\gamma_2}{\alpha_1 \beta_1}, \frac{\gamma_2}{\alpha_2 \beta_2} \\ \llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k}, \llbracket \beta_1 b \rrbracket_l, \llbracket \beta_2 b \rrbracket_{l+k}, \llbracket \alpha_1 \beta_1 ab \rrbracket_l^*, \llbracket \alpha_2 \beta_2 ab \rrbracket_{l+k}^*, \llbracket \gamma_1 ab \rrbracket_l, \llbracket \gamma_2 ab \rrbracket_{l+k} \end{aligned}$$

As the encrypted secret inputs $\alpha_1 a, \alpha_2 a, \beta_1 b$ and $\beta_2 b$ cannot be reconstructed from $k-2$ shares, Adversary 1 will not be able to learn about each encrypted secret input. Moreover, Adversary 1 will not be able to separate each random number $\alpha_1, \alpha_2, \beta_1, \beta_2$ from $\alpha_2 \beta_2 / \alpha_1 \beta_1$; therefore, random numbers γ_1, γ_2 will not be leaked. Thus, we can state that secret inputs a, b will not be leaked and the following statements are true:

$$\begin{aligned} H(a) &= H \left(a | \alpha_{1,l}, \beta_{1,l}, \alpha_{2,l}, \beta_{2,l}, \frac{\alpha_2 \beta_2}{\alpha_1 \beta_1}, \llbracket \alpha_1 a \rrbracket_l, \llbracket \alpha_2 a \rrbracket_{l+k} \right) \\ H(b) &= H \left(b | \alpha_{1,l}, \beta_{1,l}, \alpha_{2,l}, \beta_{2,l}, \frac{\alpha_2 \beta_2}{\alpha_1 \beta_1}, \llbracket \beta_1 b \rrbracket_l, \llbracket \beta_2 b \rrbracket_{l+k} \right) \end{aligned}$$

Next, we will discuss the result ab . As Adversary 1 learns $\gamma_1/\alpha_1\beta_1, \gamma_2/\alpha_1\beta_1$ and $[[\gamma_1ab]]_l, [[\gamma_2ab]]_{l+k}$, it will also learn $[[\alpha_1\beta_1ab]]_l^*, [[\alpha_1\beta_1ab]]_{l+k}^*$, being able to recover the values of $\alpha_1\beta_1ab$. With $\gamma_1/\alpha_1\beta_1, \gamma_2/\alpha_1\beta_1, \alpha_1\beta_1ab$, Adversary 1 will also learn γ_1ab, γ_2ab . As the random numbers $\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2$ are not leaked, Adversary 1 will not be able to learn the output ab . If the recovered values $\gamma_1ab = \gamma_2ab = 0$, the adversary 1 will know that the input for the consecutive computation is equal to 0, but it will not be able to differentiate the value of each secret input a, b . To prevent this, we assume the following condition whose result is used for the consecutive computation.

Condition (1): Inputs and output do not include value 0.

The same is true when assuming the colluding Adversaries 2 and 3. However, we omit the detailed analysis here.

6.7.4 Comparison with Conventional Methods

Advantages and Disadvantages

This section compares our proposed method with the TUS 1 and TUS 2 methods in Chapters 3 and 4, and Watanabe et al.'s method that also realizes two-party multiplication of (k, n) threshold secret sharing. We explain in detail the advantages and disadvantages of our method compared to the solutions in Chapters 3, 4, and Watanabe et al.'s method.

- Comparison with the TUS 1, TUS 2 methods in Chapters 3, 4, respectively.

Advantage: Multiplication of shares can be performed without any preconditions/limitations on the input/output when assuming a non-colluding adversary (Adversary 1).

Disadvantage: $n \geq 2k$ encrypted shares are distributed for each input in the distribution process.

In the TUS 1, TUS 2 methods, $n \geq k$ encrypted shares are generated for each input. Moreover, multiplication under the setting of $n \geq k$ is performed by multiplying scalar values with a share. However, each method requires a certain condition on the input/output even when assuming only Adversary 1. In particular, the TUS 1 and TUS 2 methods require that the inputs and output in multiplication do not include 0. This limitation means that each method has limited applications.

In contrast, in the method proposed in this chapter, multiplication is realized under the setting of $n \geq 2k - 1$. The disadvantage of this approach is that $n \geq 2k$ shares are generated for each input. Hence, our protocol requires twice the amount of computational and communication costs for the distribution of the encrypted input. However, in our protocol, the shares of encrypted inputs are multiplied without the need to reconstruct any of the encrypted inputs as a scalar value. Therefore, no information about the input will be leaked even if one (e.g., a or b) or both inputs are equal to 0 when assuming Adversary 1. When considering Adversaries 2 and 3, the proposed method also requires a certain limitation on the input/output.

- Comparison with Watanabe et al.'s method.

Advantage: $(k - 1)$ sharing of multiplication result ab can be computed using only $N \geq k$ servers with $n \geq 2k - 1$. Moreover, the proposed method allows for the repetition of multiplication.

Disadvantage: Communication is required in the multiplication protocol to redistribute shares.

For all previously proposed multiplication methods with $n \geq 2k - 1$, at least $2k - 1$ shares must be collected to reconstruct the multiplication result. This condition holds even for the Watanabe et al.'s method, in which encrypted shares are multiplied with $N \geq k$. This problem means that the number of shares required to reconstruct the encrypted result remains $2k - 1$ instead of k .

In contrast, our proposed method is the only method that enables multiplication under the setting of $n \geq 2k - 1$, $N \geq k$ and produces $(k - 1)$ instead of $(2k - 2)$ shares of ab . Thus, the client only needs to collect k instead of $2k - 1$ shares to reconstruct the encrypted result ab . This significantly reduces the communication cost, minimizing long-distance communication between a client and computing servers.

Our method requires communication between the computing servers during multiplication to compute $(k - 1)$ shares of ab , whereas Watanabe et al.'s method requires no communication because all computations are performed locally. However, this is not a problem because the redistribution of shares occurs only once, and cloud servers are usually configured with a fast connection.

Computational Cost, Communication Cost, and Rounds

In this section, we compare computational cost, communication cost, and the number of rounds in our proposed method with the TUS 1 solutions in Chapter 3 and Watanabe et al.'s method (we omitted the comparison with TUS 2 method because this method focus on three-input computation, whereas the method in this chapter focuses on two-input computation). The parameters in Section 6.7.1 are used.

The computational cost of secret sharing C_1, C_2 are typically larger than the local computational costs of M, D , and A . Therefore, we omitted the costs of M, D , and A when either C_1 or C_2 was present in the computational cost.

Table 6.6 shows that when $N = k$, the computational cost for the distribution of a, b and the reconstruction of ab of our method are significantly lower than those of the method of Watanabe et al. and that of Chapter 3. This is because when $N = k$, all random numbers used are sent/collected directly from each computing server without the need for reconstruction using Shamir's (k, n) method as in the other two methods. However, when $N > k$, the computation costs for distribution and reconstruction of the proposed method are larger than

TABLE 6.6: Comparison with conventional methods (Computational cost)

Process	Proposed method	Watanabe et al's method	TUS 1 method
Distribution of a, b	$2C_1$ $\langle 2(C_1 + 2kC_2) \rangle$	$2(C_1 + C_2)$	$2(k + 1)C_2$
Multiplication of ab	$2kC_2$ $\langle 7kC_2 \rangle$	$2nM$	$(3k + 1)C_2$
Reconstruction of ab	C_2 $\langle (k + 1)C_2 \rangle$	$C_1 + 2C_2$	$(k + 1)C_2$

* $\langle \rangle$ shows the computation cost required for the extension to $N > k$

those of Watanabe et al.'s method. Moreover, when $N > k$, the computational cost of multiplying a, b in our method is higher than that in the other two methods. This is because our method requires the redistribution of the result of the local computation of multiplication to all servers, whereas Watanabe et al.'s method requires only local multiplication and TUS 1 method only needs to reconstruct the random numbers and encrypted secret input of a .

Table 6.7 compares communication costs with conventional methods. The performance of each method depends on d_1, n, k . However, because our proposed method includes the process of redistributing local shares to all servers during the multiplication process, it has a higher communication cost than that of Watanabe et al.'s method that only requires local computation without any communication. During the reconstruction of ab , when $N = k$, the communication cost for the reconstruction of ab on our proposed method is significantly lower than that of Watanabe et al.'s method as the polynomial degree of ab is $(k - 1)$ instead of $(2k - 2)$. In particular, the communication cost is exactly half that required by Watanabe et al.'s method. When extended to $N > k$, the communication cost required in our proposed method to reconstruct the multiplication result ab from the encrypted shares of ab increases and is the same as that of the method in Chapter 3. However, this trade-off means that our method can also provide tolerance for the loss of up to $N - k$ servers.

Finally, Table 6.8 compares the number of rounds required for each method. Because our method includes the process of redistributing and computing random numbers, Table 6.8 shows that the number of rounds in our method is higher compared to that of Watanabe et al.'s method but still lower than that of the method in Chapter 3 when $N = k$.

6.8 Chapter Summary

In this chapter, we proposed a method of multiplication of encrypted shares using only $N \geq k$ servers. Furthermore, using the recombination vector, we proposed a method of computing $(k - 1)$ shares of the encrypted multiplication result ab by using only k servers instead of

TABLE 6.7: Comparison with conventional methods (Communication cost)

Process	Proposed method	Watanabe et al's method	TUS 1 method
Distribution of a, b	$8kd_1$ $\langle 4Nd_1(k+1) \rangle$	$6nd_1$	$2nd_1(k+1)$
Multiplication of ab	$4kd_1 + 2k^2d_1$ $\langle (4k+4+3N)kd_1 \rangle$	0	$(k+n+2k^2+nk)d_1$
Reconstruction of ab	$2kd_1$ $\langle kd_1(k+1) \rangle$	$4kd_1$	$(k^2+k)d_1$

* $\langle \rangle$ shows the computation cost required for the extension to $N > k$

TABLE 6.8: Comparison with conventional methods (rounds)

Process	Proposed method	Watanabe et al's method	TUS 1 method
Distribution of a, b	1 $\langle 1 \rangle$	1	1
Multiplication of ab	3 $\langle 5 \rangle$	0	4
Reconstruction of ab	1 $\langle 1 \rangle$	1	1

* $\langle \rangle$ shows the computation cost required for the extension to $N > k$

previously used $2k - 1$ servers. We also implemented our method in MATLAB and showed that it can perform multiplications very quickly. In fact, our method performed 1,000,000 multiplications in less than 1 s.

By differentiating N (number of servers) and n (parameter of Shamir's (k, n) method), we showed that secure two-party computation of multiplication using Shamir's (k, n) method under the setting of $n \geq 2k - 1$ is possible with only $N < 2k - 1$ servers. The proposed method was proven to be unconditionally secure against Adversary 1; however, it still requires some conditions when assuming Adversary 2.

Chapter 7

Conclusion and Future Works

This chapter concludes our study on a secure MPC using (k, n) threshold secret sharing with $n < 2k - 1$ and its application into SE. First, we summarize the conclusion and contribution of each chapter. In sequence, potential future research directions are discussed.

7.1 Conclusions

Motivated by the need for a technology that could perform computation in an encrypted state to enable the analysis of personal information without infringing the individuals' privacy, we comprehensively investigated methods of secure MPC using a (k, n) threshold secret sharing with $n < 2k - 1$ that can realize minimal computational and operation costs. The conclusions and contributions of this dissertation are summarized below.

Chapter 1 provided the introduction and motivation of this dissertation. After the brief introduction of secure MPC and examples of its use, the main objective and list of contributions of this dissertation were summarized. Finally, the outline of this dissertation was presented.

Chapter 2 introduced the basic knowledge and building blocks required to understand this dissertation. We start by explaining in detail the formal definitions for MPC. In sequence, we showed in detail the definitions and examples for the concept of computation in a finite field. Finally, we introduced the technology of the secret sharing and its variants, including the detailed protocol for each variant.

Chapter 3 introduced our first approach to realizing a MPC using (k, n) threshold secret sharing with parameter $n < 2k - 1$ through the approach of scalar value \times polynomial. After describing the protocol, security evaluation was performed against three types of semi-honest adversaries. We also showed that repetition of the same types of computation, such as many-inputs multiplication and addition, is possible and secure against semi-honest adversaries. We also showed that computation involving the combination of different types of operations

is not secure against a semi-honest adversary with knowledge of one of the inputs, output, and information from $k - 1$ computing servers.

Chapter 4 introduced a method to solve the problem of the MPC method in Chapter 3 and also two new conditions. After describing the protocol, we evaluated and confirmed the security of a combination of multiple operations against semi-honest adversaries. Finally, we also performed an analysis of the proposed method and discussed each condition.

Chapter 5 proposed a method of SE using the MPC method proposed in Chapter 4. We proved that although the MPC method in Chapter 4 requires three conditions, it can still be used and the conditions can be easily fulfilled depending on the types of application assumed. We also introduced methods of conjunctive and disjunctive searches of multiple search queries. Finally, we compared our proposed methods with conventional methods of SE.

Chapter 6 proposed another method of MPC when $n < 2k - 1$ using the conventional approach of polynomial \times polynomial and introduced a new degree reduction method using only k computing servers. We proved that the proposed method could realize secure MPC when $n < 2k - 1$ without any conditions when assuming Adversary 1 that controls up to $k - 1$ computing servers. Moreover, we also showed that the proposed method is secure against the remaining semi-honest adversaries 2 and 3 when certain conditions are assumed.

The method of MPC discussed in this dissertation could realize the computation of sensitive information without the risk of information leakage. This is particularly important in our current and future society, where data involving an individual's personal information is critical to improving our life quality. For example, the aim of realizing the concept of Society 5.0 in Japan has created new opportunities, such as using the computational power of cloud computing to extract valuable statistics from big data. Society 5.0 is the concept of a super-smart society where various social challenges can be resolved by incorporating the innovations of the fourth industrial revolution (for example, big data, artificial intelligence, etc.) into every industry and social life.

According to Keidanren (Japan Business Federation), the most important key to realizing Society 5.0 is data utilization, where data such as medical information (e.g., health care information, data from medical checkups), business systems data (e.g., customer buying data, accounting data) and any other types of data can be utilized to improve competitiveness between businesses, improve quality of individual life, and resolve various social issues. As data collected may also include private information of individuals, the security of these data must be ensured to further encourage the sharing and utilization of data.

Therefore, this dissertation can change how data can be used in a secure manner in the future. We believe that the results obtained will be the cornerstone to secure data utilization in the future.

7.2 Future Works

This dissertation mainly aimed to realize a secure MPC using (k, n) threshold secret sharing when $n < 2k - 1$. However, there are several research directions for future works. We briefly discuss the remaining and potential future works as follows.

Computation of shares of random numbers unknown to the adversary for each computing server

In the proposed method shown in Chapter 4, conditions where each server holds shares of random numbers unknown to the adversary are required to realize a secure MPC when $n < 2k - 1$. The easiest manner of realizing this is to assume a TTP, which is difficult to realize. Therefore, in a future work, we need to also consider the method of realizing this condition with a different approach other than using a TTP.

Realizing security against a malicious adversary

When considering the application to big data, the existence of a malicious adversary must be considered. A malicious adversary can execute any computation for corrupting and modifying the data when executing the protocol. The secure MPC protocols proposed in this dissertation were proven to be secure against semi-honest adversaries. In a future work, we also need to study a mechanism to enable verification of data computed by the computing servers to ensure that all data transmitted and computed are correct and not altered.

Realizing nearest neighbor search (NSS)

In this dissertation, we proposed an SE method that enables encrypted information to be searched without decrypting the original information. In the future, we also need to study the nearest neighbor search method. As a method of proximity search, the nearest neighbor search is the optimization problem of finding the point in a given set that is closest (or most similar) to a given point. This is particularly important when considering implementation into areas such as machine learning.

Appendix A

List of Publications

A.1 Refereed Publication

Journal Papers

1. Shingu Takeshi, Ken Aoi, **Ahmad Akmal Aminuddin Mohd Kamal** and Keiichi Iwamura. “Secure computation without changing polynomial degree in (k, n) secret sharing scheme.” *Journal of Information Processing (JIP)*, vol. 59, no. 3, pp. 1038-1049, March 2018.
2. **Ahmad Akmal Aminuddin Mohd Kamal** and Keiichi Iwamura. “Conditionally secure multiparty computation when $n < 2k - 1$.” *Journal of Information Processing (JIP)*, vol. 59, no. 9, pp. 1581-1595, September 2018.
3. **Ahmad Akmal Aminuddin Mohd Kamal** and Keiichi Iwamura. “Searchable encryption using secret sharing that realizes direct search of encrypted documents and disjunctive search of multiple keywords.” *Journal of Information Security and Applications*, vol. 59, pp. 102828, June 2021.
4. **Ahmad Akmal Aminuddin Mohd Kamal** and Keiichi Iwamura. “(Server-aided) two-party multiplication of encrypted shares using (k, n) threshold secret sharing with $N \geq k$ servers.” *IEEE Access*, August 2021.
5. Keiichi Iwamura and **Ahmad Akmal Aminuddin Mohd Kamal**. “Efficient partial matching search using error correction code.” *Journal of Information Processing (JIP)*, vol. 63, no. 2. (accepted for publication)

International Proceedings

1. **Ahmad Akmal Aminuddin Mohd Kamal** and Keiichi Iwamura. “Conditionally secure multiparty computation using secret sharing scheme for $n < 2k - 1$ (short paper).” In Proceedings of 15th Annual Conference on Privacy, Security and Trust (PST 2017), August 2017.

2. **Ahmad Akmal Aminuddin Mohd Kamal** and Keiichi Iwamura. “Searchable encryption of image based on secret sharing scheme.” In Proceedings of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (AP-SIPA ASC 2017), December 2017.
3. **Ahmad Akmal Aminuddin Mohd Kamal** and Keiichi Iwamura. “Searchable encryption using secret sharing scheme for multiple keyword search using conjunctive and disjunctive searching.” In Proceedings of 2019 IEEE International Conference on Cyber Science and Technology Congress (CyberSciTech 2019), August 2019.
4. Shogo Ochiai, Keiichi Iwamura and **Ahmad Akmal Aminuddin Mohd Kamal**. “Secure pairwise key sharing using geometric group key sharing method.” In Proceedings of IEEE 17th Annual Consumer Communications & Networking Conference (CCNC 2020), January 2020.
5. **Ahmad Akmal Aminuddin Mohd Kamal** and Keiichi Iwamura. “Improvement of secure multi-party multiplication of (k, n) threshold secret sharing using only $N = k$ servers.” In Proceedings of 7th International Conference on Information Systems Security and Privacy (ICISSP 2021), February 2021.
6. Keiichi Iwamura and **Ahmad Akmal Aminuddin Mohd Kamal**. “Secure computation by secret sharing using input encrypted with random number.” In Proceedings of 18th International Conference on Security and Cryptography (SECRYPT 2021), July 2021.
7. Keiichi Iwamura and **Ahmad Akmal Aminuddin Mohd Kamal**. “Efficient secret sharing based partial matching search using error correction code.” In Proceedings of Future Technologies Conference 2021 (FTC 2021), October 2021.

Appendix B

Example of computation

B.1 Computation of protocols 6.2, 6.3 and 6.4

We demonstrate the computation of multiplication between secrets $a = 3$ and $b = 5$ of Players A and B, respectively, under the setting of $N = k = 2$ and $n = 2k = 4$. Since $k = 2$, multiplication of shares of a and b will produce a $(2k - 2) = 2$ degrees (quadratic) polynomials. The process of reducing the degree of polynomial from $(2k - 2) = 2$ to $(k - 1) = 1$ can be achieved by using the recombination vector $r = (3, -3, 1)$. All random numbers and all computations are performed with $p = 2971$.

Distribution of secret inputs a, b

1. Player A generates $2k = 4$ random numbers $\alpha_{1,0} = 2, \alpha_{1,1} = 6, \alpha_{2,0} = 3, \alpha_{2,1} = 3$ and computes the following:

$$\alpha_1 = \alpha_{1,0} \times \alpha_{1,1} = 2 \times 6 = 12$$

$$\alpha_2 = \alpha_{2,0} \times \alpha_{2,1} = 3 \times 3 = 9$$

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

$$[[\alpha_1 a]]_0 = \alpha_1 \times [[a]]_0 = 12 \times 4 = 48$$

$$[[\alpha_1 a]]_1 = \alpha_1 \times [[a]]_1 = 12 \times 5 = 60$$

$$[[\alpha_2 a]]_2 = \alpha_2 \times [[a]]_2 = 9 \times 6 = 54$$

$$[[\alpha_2 a]]_3 = \alpha_2 \times [[a]]_3 = 9 \times 7 = 63$$

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 $\beta_{1,0} = 1, \beta_{1,1} = 5, \beta_{2,0} = 8, \beta_{2,1} = 2$ and computes the following:

$$\beta_1 = \beta_{1,0} \times \beta_{1,1} = 1 \times 5 = 5$$

$$\beta_2 = \beta_{2,0} \times \beta_{2,1} = 8 \times 2 = 16$$

5. Player B generates $n = 2k = 4$ shares of secret input $b = 5$ using Shamir's $(2, 4)$ method and computes the following (here, let $[[b]]_j = 5 + 2x$):

$$[[\beta_1 b]]_0 = \beta_1 \times [[b]]_0 = 5 \times 7 = 35$$

$$[[\beta_1 b]]_1 = \beta_1 \times [[b]]_1 = 5 \times 9 = 45$$

$$[[\beta_2 b]]_2 = \beta_2 \times [[b]]_2 = 16 \times 11 = 176$$

$$[[\beta_2 b]]_3 = \beta_2 \times [[b]]_3 = 16 \times 13 = 208$$

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 .

Multiplication of secret inputs a, b

1. Each server S_j ($j = 0, 1$) computes the following:

- Server S_0 computes the following:

$$[[\alpha_1 \beta_1 ab]]_0^* = [[\alpha_1 a]]_0 \times [[\beta_1 b]]_0 = 48 \times 35 = 1680$$

$$[[\alpha_2 \beta_2 ab]]_2^* = [[\alpha_2 a]]_2 \times [[\beta_2 b]]_2 = 54 \times 176 = 591$$

$$\alpha_{1,0} \beta_{1,0} = \alpha_{1,0} \times \beta_{1,0} = 2 \times 1 = 2$$

$$\alpha_{2,0} \beta_{2,0} = \alpha_{2,0} \times \beta_{2,0} = 3 \times 8 = 24$$

- Server S_1 computes the following:

$$[[\alpha_1 \beta_1 ab]]_1^* = [[\alpha_1 a]]_1 \times [[\beta_1 b]]_1 = 60 \times 45 = 2700$$

$$[[\alpha_2 \beta_2 ab]]_3^* = [[\alpha_2 a]]_3 \times [[\beta_2 b]]_3 = 63 \times 208 = 1220$$

$$\alpha_{1,1} \beta_{1,1} = \alpha_{1,1} \times \beta_{1,1} = 6 \times 5 = 30$$

$$\alpha_{2,1} \beta_{2,1} = \alpha_{2,1} \times \beta_{2,1} = 3 \times 2 = 6$$

2. Each server S_j ($j = 0, 1$) generates a random number γ_j , computes the following values, and sends them to one of the servers (here, we assume server S_0):

- Server S_0 generates $\gamma_0 = 3$, computes the following and sends to server S_0 :

$$\frac{\gamma_0}{\alpha_{1,0}\beta_{1,0}} = \frac{3}{2} = 1487,$$

$$\frac{\gamma_0}{\alpha_{2,0}\beta_{2,0}} = \frac{3}{24} = 1857$$

- Server S_1 generates $\gamma_1 = 5$, computes the following and sends to server S_0 :

$$\frac{\gamma_1}{\alpha_{1,1}\beta_{1,1}} = \frac{5}{30} = 2476,$$

$$\frac{\gamma_1}{\alpha_{2,1}\beta_{2,1}} = \frac{5}{6} = 496$$

3. Server S_0 computes the following values and sends them to all servers S_j ($j = 0, 1$):

$$\frac{\gamma}{\alpha_1\beta_1} = \frac{\gamma_0}{\alpha_{1,0}\beta_{1,0}} \times \frac{\gamma_1}{\alpha_{1,1}\beta_{1,1}} = 1487 \times 2476 = 743$$

$$\frac{\gamma}{\alpha_2\beta_2} = \frac{\gamma_0}{\alpha_{2,0}\beta_{2,0}} \times \frac{\gamma_1}{\alpha_{2,1}\beta_{2,1}} = 1857 \times 496 = 62$$

4. Each server S_j ($j = 0, 1$) computes $\llbracket \gamma ab \rrbracket_j^*$, $\llbracket \gamma ab \rrbracket_{j+k}^*$ as follows and distributes the results using Shamir's (2,2) method to all servers S_i ($j = 0, 1$):

- Server S_0 performs the following:

$$\llbracket \gamma ab \rrbracket_0^* = \frac{\gamma}{\alpha_1\beta_1} \times \llbracket \alpha_1\beta_1 ab \rrbracket_0^* = 743 \times 1680 = 420$$

$$\llbracket \gamma ab \rrbracket_2^* = \frac{\gamma}{\alpha_2\beta_2} \times \llbracket \alpha_2\beta_2 ab \rrbracket_2^* = 62 \times 591 = 990$$

Let the polynomial be $\llbracket \gamma ab \rrbracket_0^* = 420 + 2x$

$$\begin{cases} \llbracket \gamma ab \rrbracket_{0,0} = 422 & \rightarrow \text{send to server } S_0 \\ \llbracket \gamma ab \rrbracket_{0,1} = 424 & \rightarrow \text{send to server } S_1 \end{cases}$$

Let the polynomial be $\llbracket \gamma ab \rrbracket_2^* = 990 + 3x$

$$\begin{cases} \llbracket \gamma ab \rrbracket_{2,0} = 993 & \rightarrow \text{send to server } S_0 \\ \llbracket \gamma ab \rrbracket_{2,1} = 996 & \rightarrow \text{send to server } S_1 \end{cases}$$

- Server S_1 performs the following:

$$\llbracket \gamma ab \rrbracket_1^* = \frac{\gamma}{\alpha_1 \beta_1} \times \llbracket \alpha_1 \beta_1 ab \rrbracket_1^* = 743 \times 2700 = 675$$

$$\llbracket \gamma ab \rrbracket_3^* = \frac{\gamma}{\alpha_2 \beta_2} \times \llbracket \alpha_2 \beta_2 ab \rrbracket_3^* = 62 \times 1220 = 1365$$

Let the polynomial be $\llbracket \gamma ab \rrbracket_1^* = 675 + x$

$$\begin{cases} \llbracket \gamma ab \rrbracket_{1,0} = 676 & \rightarrow \text{send to server } S_0 \\ \llbracket \gamma ab \rrbracket_{1,1} = 677 & \rightarrow \text{send to server } S_1 \end{cases}$$

Let the polynomial be $\llbracket \gamma ab \rrbracket_3^* = 1365 + 3x$

$$\begin{cases} \llbracket \gamma ab \rrbracket_{3,0} = 1368 & \rightarrow \text{send to server } S_0 \\ \llbracket \gamma ab \rrbracket_{3,1} = 1371 & \rightarrow \text{send to server } S_1 \end{cases}$$

5. Each server S_j ($j = 0, 1$) computes the following value using recombination vector $r = (3, -3, 1, 0)$:

- Server S_0 computes the following:

$$\begin{aligned} \llbracket \gamma ab \rrbracket_0 &= 3 \times \llbracket \gamma ab \rrbracket_{0,0} + (-3) \times \llbracket \gamma ab \rrbracket_{1,0} + 1 \times \llbracket \gamma ab \rrbracket_{2,0} + 0 \times \llbracket \gamma ab \rrbracket_{3,0} \\ &= (3 \times 422) - (3 \times 676) + (1 \times 993) + (0 \times 1368) \\ &= 231 \end{aligned}$$

- Server S_1 computes the following:

$$\begin{aligned} \llbracket \gamma ab \rrbracket_1 &= 3 \times \llbracket \gamma ab \rrbracket_{0,1} + (-3) \times \llbracket \gamma ab \rrbracket_{1,1} + 1 \times \llbracket \gamma ab \rrbracket_{2,1} + 0 \times \llbracket \gamma ab \rrbracket_{3,1} \\ &= (3 \times 424) - (3 \times 677) + (1 \times 996) + (0 \times 1371) \\ &= 237 \end{aligned}$$

Reconstruction of ab

1. The player collects $\llbracket \gamma ab \rrbracket_0$, $\llbracket \gamma ab \rrbracket_1, \gamma_0, \gamma_1$ from $k = 2$ servers S_0 and S_1 , reconstructs γab using Shamir's (2, 2) method, and computes random number γ as follows:

$$\gamma ab = 225$$

$$\gamma = \gamma_0 \times \gamma_1 = 3 \times 5 = 15$$

2. The player reconstructs the multiplication result ab as follows:

$$ab = \frac{\gamma ab}{\gamma} = \frac{225}{15} = 15$$

References

- [1] Andrei Lapets et al. *Web-based multi-party computation with application to anonymous aggregate compensation analytics*. Tech. rep. Computer Science Department, Boston University, 2015.
- [2] Prasan Kumar Sahoo, Suvendu Kumar Mohapatra, and Shih-Lin Wu. “Analyzing health-care big data with prediction for future health condition”. In: *IEEE Access* 4 (2016), pp. 9786–9799. DOI: 10.1109/ACCESS.2016.2647619.
- [3] Martin Moore and Damian Tambini. *Digital dominance: the power of Google, Amazon, Facebook, and Apple*. Oxford University Press, 2018.
- [4] Nadeem Ahmed et al. “A survey of COVID-19 contact tracing apps”. In: *IEEE Access* 8 (2020), pp. 134577–134601. DOI: 10.1109/ACCESS.2020.3010226.
- [5] Serge Vaudenay. *Centralized or decentralized? The contact tracing dilemma*. Cryptology ePrint Archive, Report 2020/531. <https://ia.cr/2020/531>. 2020.
- [6] Cristina Criddle and Leo Kelion. *Coronavirus contact-tracing: world split between two types of app*. URL: <https://www.bbc.com/news/technology-52355028>. (accessed: 01.01.2022).
- [7] Japan Personal Information Protection Commission. *Amended act on the protection of personal information*. URL: https://www.ppc.go.jp/files/pdf/APPI_english.pdf. (accessed: 01.01.2022).
- [8] David Evans, Vladimir Kolesnikov, and Mike Rosulek. “A pragmatic introduction to secure multi-party computation”. In: *Foundations and Trends® in Privacy and Security* 2.2–3 (2018), 70–246. ISSN: 2474-1558. DOI: 10.1561/33000000019.
- [9] Yehuda Lindell. *Secure multiparty computation (MPC)*. Cryptology ePrint Archive, Report 2020/300. <https://ia.cr/2020/300>. 2020.
- [10] Andrew C. Yao. “Protocols for secure computations”. In: *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.
- [11] Sophia Yakoubov. “A gentle introduction to Yao’s Garbled circuits”. In: *preprint on webpage at <https://web.mit.edu/sonka89/www/papers/2017ygc.pdf>* (2017).

- [12] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. “Completeness theorems for non-cryptographic fault-tolerant distributed computation”. In: STOC ’88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, 1–10. ISBN: 0897912640. DOI: 10.1145/62212.62213.
- [13] Adi Shamir. “How to share a secret”. In: *Communication of the ACM* 22.11 (1979), 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176.
- [14] Dahlia Malkhi et al. “Fairplay—A secure two-party computation system”. In: *13th USENIX Security Symposium (USENIX Security 04)*. San Diego, CA: USENIX Association, 2004.
- [15] Dan Bogdanov, Sven Laur, and Jan Willemson. “Sharemind: A framework for fast privacy-preserving computations”. In: *Computer Security - ESORICS 2008*. Ed. by Sushil Jajodia and Javier Lopez. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 192–206. ISBN: 978-3-540-88313-5.
- [16] Wenliang Du and Mikhail J. Atallah. “Protocols for secure remote database access with approximate matching”. In: *E-Commerce Security and Privacy*. Ed. by Anup K. Ghosh. Boston, MA: Springer US, 2001, pp. 87–111. ISBN: 978-1-4615-1467-1. DOI: 10.1007/978-1-4615-1467-1_6.
- [17] Craig Gentry. “A fully homomorphic encryption scheme”. AAI3382729. PhD thesis. Stanford, CA, USA, 2009. ISBN: 9781109444506.
- [18] NEC Corporation. *NEC and Osaka University demonstrate possibility of analyzing genome information from multiple institutions with reduced risk of privacy infringement using secure computation of encrypted data*. URL: https://www.nec.com/en/press/201907/global_20190723_03.html. (accessed: 01.01.2022).
- [19] Ivan Damgård et al. “Multiparty computation from somewhat homomorphic encryption”. In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 643–662. ISBN: 978-3-642-32009-5.
- [20] Donald Beaver. “Efficient multiparty protocols using circuit randomization”. In: *Advances in Cryptology — CRYPTO ’91*. Ed. by Joan Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 420–432. ISBN: 978-3-540-46766-3.
- [21] Monique Ogburn, Claude Turner, and Pushkar Dahal. “Homomorphic encryption”. In: *Procedia Computer Science* 20 (2013). Complex Adaptive Systems, pp. 502–509. ISSN: 1877-0509. DOI: 10.1016/j.procs.2013.09.310.
- [22] Koki Hamada et al. “MEVAL: A practically efficient system for secure multi-party statistical analysis”. In: *Workshop on Applied Multi-Party Computation*. 2014.

- [23] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. “Simplified VSS and fast-track multiparty computations with applications to threshold cryptography”. In: *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*. PODC ’98. Puerto Vallarta, Mexico: Association for Computing Machinery, 1998, 101–111. ISBN: 0897919777. DOI: 10.1145/277697.277716.
- [24] Toshinori Araki et al. “High-throughput semi-honest secure three-party computation with an honest majority”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: Association for Computing Machinery, 2016, 805–817. ISBN: 9781450341394. DOI: 10.1145/2976749.2978331.
- [25] Peter Bogetoft et al. “Secure multiparty computation goes live”. In: *Financial Cryptography and Data Security*. Ed. by Roger Dingledine and Philippe Golle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 325–343. ISBN: 978-3-642-03549-4.
- [26] Will Kenton. *Clearing price*. URL: <https://www.investopedia.com/terms/c/clearingprice.asp>. (accessed: 01.01.2022).
- [27] Partisia. URL: <https://partisia.com>. (accessed: 01.01.2022).
- [28] Boston women’s workforce council (BWWC). *Wage gap studies*. URL: <https://thebwbc.org/wage-gap-studies>. (accessed: 01.01.2022).
- [29] Andrei Lapets et al. “Secure MPC for analytics as a web application”. In: *2016 IEEE Cybersecurity Development (SecDev)*. 2016, pp. 73–74. DOI: 10.1109/SecDev.2016.027.
- [30] Unbound Security. *CORE key management*. URL: <https://www.unboundsecurity.com/solutions/key-management/>. (accessed: 01.01.2022).
- [31] Victor Shoup. “Practical threshold signatures”. In: *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT’00. Bruges, Belgium: Springer-Verlag, 2000, 207–220. ISBN: 3540675175.
- [32] Cinnamon S Bloss, Dilip V Jeste, and Nicholas J Schork. “Genomics for disease treatment and prevention”. In: *Psychiatric Clinics* 34.1 (2011), pp. 147–166.
- [33] Rikke Bendlin et al. “Semi-homomorphic encryption and multiparty computation”. In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 169–188. ISBN: 978-3-642-20465-4.
- [34] Zvika Brakerski and Vinod Vaikuntanathan. “Fully homomorphic encryption from ring-LWE and security for key dependent messages”. In: *Advances in Cryptology – CRYPTO 2011*. Ed. by Phillip Rogaway. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 505–524. ISBN: 978-3-642-22792-9.

- [35] Ivan Damgård et al. “Practical covertly secure MPC for dishonest majority – or: breaking the SPDZ limits”. In: *Computer Security – ESORICS 2013*. Ed. by Jason Cramp-ton, Sushil Jajodia, and Keith Mayes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–18. ISBN: 978-3-642-40203-6.
- [36] Marcel Keller, Valerio Pastro, and Dragos Rotaru. “Overdrive: making SPDZ great again”. In: *Advances in Cryptology – EUROCRYPT 2018*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Cham: Springer International Publishing, 2018, pp. 158–189. ISBN: 978-3-319-78372-7.
- [37] David Chaum, Claude Crépeau, and Ivan Damgard. “Multiparty unconditionally secure protocols”. In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC ’88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, 11–19. ISBN: 0897912640. DOI: 10.1145/62212.62214.
- [38] Koji Chida et al. “Fast large-scale honest-majority MPC for malicious adversaries”. In: *Advances in Cryptology – CRYPTO 2018*. Ed. by Hovav Shacham and Alexandra Boldyreva. Cham: Springer International Publishing, 2018, pp. 34–64. ISBN: 978-3-319-96878-0.
- [39] Ronald Cramer, Ivan Damgård, and Ueli Maurer. “General secure multi-party computation from any linear secret-sharing scheme”. In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Bart Preneel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 316–334. ISBN: 978-3-540-45539-4.
- [40] Seny Kamara, Payman Mohassel, and Mariana Raykova. *Outsourcing multi-party computation*. Cryptology ePrint Archive, Report 2011/272. <https://ia.cr/2011/272>. 2011.
- [41] Payman Mohassel, Ostap Orobets, and Ben Riva. “Efficient server-aided 2PC for mobile phones”. In: *Proceedings on Privacy Enhancing Technologies 2016.2* (2016), pp. 82–99.
- [42] Satsuya Ohata and Koji Nuida. “Communication-efficient (client-aided) secure two-party protocols and its application”. In: *Financial Cryptography and Data Security*. Ed. by Joseph Bonneau and Nadia Heninger. Cham: Springer International Publishing, 2020, pp. 369–385. ISBN: 978-3-030-51280-4.
- [43] Taihei Watanabe, Keiichi Iwamura, and Kitahiro Kaneda. “Secrecy multiplication based on a (k, n) -threshold secret-sharing scheme using only k servers”. In: *Computer Science and its Applications*. Ed. by James J. (Jong Hyuk) Park et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 107–112. ISBN: 978-3-662-45402-2.
- [44] Ryo Kikuchi and Dai Ikarashi. “Progress of secure computation: basic constructions and dedicated algorithms”. In: *IEICE ESS Fundamentals Review 12.1* (2018), pp. 12–20.

- [45] Cybernetica. *Sharemind*. URL: <https://sharemind.cyber.ee/>. (accessed: 01.01.2022).
- [46] Brilliant.org. *Lagrange interpolation*. URL: <https://brilliant.org/wiki/extended-euclidean-algorithm>. (accessed: 01.01.2022).
- [47] G. R. Blakley. “Safeguarding cryptographic keys”. In: *1979 International Workshop on Managing Requirements Knowledge (MARK)*. 1979, pp. 313–318. DOI: 10.1109/MARK.1979.8817296.
- [48] Jun Kurihara et al. “A new (k,n) -threshold secret sharing scheme and its extension”. In: *Information Security*. Ed. by Tzong-Chen Wu et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 455–470. ISBN: 978-3-540-85886-7.
- [49] Donald Beaver. “Commodity-based cryptography”. In: *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. 1997, pp. 446–455.
- [50] Dawn Xiaoding Song, D. Wagner, and A. Perrig. “Practical techniques for searches on encrypted data”. In: *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*. 2000, pp. 44–55. DOI: 10.1109/SECPRI.2000.848445.
- [51] Dan Boneh et al. “Public key encryption with keyword search”. In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by Christian Cachin and Jan L. Camenisch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 506–522. ISBN: 978-3-540-24676-3.
- [52] Philippe Golle, Jessica Staddon, and Brent Waters. “Secure conjunctive keyword search over encrypted data”. In: *Applied Cryptography and Network Security*. Ed. by Markus Jakobsson, Moti Yung, and Jianying Zhou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 31–45. ISBN: 978-3-540-24852-1.
- [53] Dong Jin Park, Kihyun Kim, and Pil Joong Lee. “Public key encryption with conjunctive field keyword search”. In: *Information Security Applications*. Ed. by Chae Hoon Lim and Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 73–86. ISBN: 978-3-540-31815-6.
- [54] Yan-Cheng Chang and Michael Mitzenmacher. “Privacy preserving keyword searches on remote encrypted data”. In: *Applied Cryptography and Network Security*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 442–455. ISBN: 978-3-540-31542-1.
- [55] Reza Curtmola et al. “Searchable symmetric encryption: improved definitions and efficient constructions”. In: *Journal of Computer Security* 19.5 (2011), pp. 895–934.
- [56] Jin Wook Byun, Dong Hoon Lee, and Jongin Lim. “Efficient conjunctive keyword search on encrypted data storage system”. In: *Public Key Infrastructure*. Ed. by Andrea S. Atzeni and Antonio Lioy. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 184–196. ISBN: 978-3-540-35152-8.

- [57] Peishun Wang, Huaxiong Wang, and Josef Pieprzyk. “Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups”. In: *Cryptology and Network Security*. Ed. by Matthew K. Franklin, Lucas Chi Kwong Hui, and Duncan S. Wong. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 178–195. ISBN: 978-3-540-89641-8.
- [58] Bo Zhang and Fangguo Zhang. “An efficient public key encryption with conjunctive-subset keywords search”. In: *Journal of Network and Computer Applications* 34.1 (2011), pp. 262–267. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2010.07.007.
- [59] Koji Nuida. “Privacy-preserving dataset search protocol for chemical compounds with additive-homomorphic encryption”. In: *Proceedings Computer Security Symposium 2012, Oct. 2012*.
- [60] Kaoru Kurosawa and Yasuhiro Ohtaki. “UC-secure searchable symmetric encryption”. In: *Financial Cryptography and Data Security*. Ed. by Angelos D. Keromytis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 285–298. ISBN: 978-3-642-32946-3.
- [61] David Cash et al. “Highly-scalable searchable symmetric encryption with support for Boolean queries”. In: *Advances in Cryptology – CRYPTO 2013*. Ed. by Ran Canetti and Juan A. Garay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 353–373. ISBN: 978-3-642-40041-4.
- [62] Kaoru Kurosawa. “Garbled searchable symmetric encryption”. In: *Financial Cryptography and Data Security*. Ed. by Nicolas Christin and Reihaneh Safavi-Naini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 234–251. ISBN: 978-3-662-45472-5.
- [63] Chenggen Song, Xin Liu, and Yalong Yan. “Efficient public key encryption with field-free conjunctive keywords search”. In: *Trusted Systems*. Ed. by Moti Yung, Liehuang Zhu, and Yanjiang Yang. Cham: Springer International Publishing, 2015, pp. 394–406. ISBN: 978-3-319-27998-5.
- [64] Yi Yang et al. “Secure dynamic searchable symmetric encryption with constant document update cost”. In: *2014 IEEE Global Communications Conference*. 2014, pp. 775–780. DOI: 10.1109/GLOCOM.2014.7036902.
- [65] Yang Yang and Maode Ma. “Conjunctive keyword search with designated tester and timing enabled proxy re-encryption function for e-health clouds”. In: *IEEE Transactions on Information Forensics and Security* 11.4 (2016), pp. 746–759. DOI: 10.1109/TIFS.2015.2509912.

- [66] Ahmad Akmal Aminuddin Mohd Kamal, Keiichi Iwamura, and Hyunho Kang. “Searchable encryption of image based on secret sharing scheme”. In: *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. 2017, pp. 1495–1503. DOI: 10.1109/APSIPA.2017.8282269.
- [67] Shi-Feng Sun et al. “Practical backward-secure searchable encryption from symmetric puncturable encryption”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’18. Toronto, Canada: Association for Computing Machinery, 2018, 763–780. ISBN: 9781450356930. DOI: 10.1145/3243734.3243782.
- [68] Peiming Xu et al. “Practical multi-keyword and Boolean search over encrypted e-mail in cloud server”. In: *IEEE Transactions on Services Computing* 14.6 (2021), pp. 1877–1889. DOI: 10.1109/TSC.2019.2903502.
- [69] Jin Li et al. “Searchable symmetric encryption with forward search privacy”. In: *IEEE Transactions on Dependable and Secure Computing* 18.1 (2021), pp. 460–474. DOI: 10.1109/TDSC.2019.2894411.
- [70] A Waseda and R Nojima. “Consideration for IT secure password protected secret sharing”. In: *IEICE Technical report, Information theory* 111.454 (2012), pp. 41–43.
- [71] Xueqiao Liu et al. “Multi-user verifiable searchable symmetric encryption for cloud storage”. In: *IEEE Transactions on Dependable and Secure Computing* 17.6 (2020), pp. 1322–1332. DOI: 10.1109/TDSC.2018.2876831.
- [72] Kentaro Tsujishita and Keiichi Iwamura. “Password-protected secret sharing scheme with the same threshold in distribution and restoration”. In: *2018 Fourth International Conference on Mobile and Secure Services (MobiSecServ)*. 2018, pp. 1–5. DOI: 10.1109/MOBISECSERV.2018.8311441.
- [73] Amos Beimel. “Secret-sharing schemes: a survey”. In: *Coding and Cryptology*. Ed. by Yeow Meng Chee et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 11–46. ISBN: 978-3-642-20901-7.
- [74] Whitfield Diffie and Martin Hellman. “New directions in cryptography”. In: *IEEE transactions on Information Theory* 22.6 (1976), pp. 644–654.
- [75] Hans Delfs and Helmut Knebl. “Symmetric-key encryption”. In: *Introduction to Cryptography*. Springer, 2007, pp. 11–31.
- [76] F Hori and W Kishimoto. “Public key encryption with delegated search expanded to a multi-user system”. In: *2016 Symposium on Cryptography and Information Security (SCIS 2016)*. (In Japanese). 2016.
- [77] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

- [78] Benny Chor and Eyal Kushilevitz. “A zero-one law for Boolean privacy”. In: *SIAM Journal on Discrete Mathematics* 4.1 (1991), pp. 36–47. DOI: 10.1137/0404004.
- [79] Gilad Asharov and Yehuda Lindell. “A full proof of the BGW protocol for perfectly secure multiparty computation”. In: *Journal of Cryptology* 30.1 (2017), pp. 58–151.
- [80] Martin Hirt. “Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting”. Reprint as vol. 3 of ETH Series in Information Security and Cryptography, ISBN 3-89649-747-2, Hartung-Gorre Verlag, Konstanz, 2001. PhD thesis. ETH Zurich, Sept. 2001.
- [81] Seny Kamara, Payman Mohassel, and Ben Riva. “Salus: a system for server-aided secure function evaluation”. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS ’12. Raleigh, North Carolina, USA: Association for Computing Machinery, 2012, 797–808. ISBN: 9781450316514. DOI: 10.1145/2382196.2382280.
- [82] Pille Pullonen, Dan Bogdanov, and Thomas Schneider. “The design and implementation of a two-party protocol suite for Sharemind 3”. In: *CYBERNETICA Institute of Information Security, Tech. Rep 4* (2012), p. 17.
- [83] R. Canetti. “Universally composable security: a new paradigm for cryptographic protocols”. In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.