Efficient Private Database Queries using Ring-LWE Somewhat Homomorphic Encryption[☆]

Tushar Kanti Saha[a], Mayank Rathee[b], Takeshi Koshiba[c]

[a]*Department of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.*
[b]*Microsoft Research India, Bangalore.*
[c]*Faculty of Education and Integrated Arts and Sciences, Waseda University, Tokyo, Japan.*

Abstract

We study the problem of private database queries over any outsourced encrypted database. In 2016, Kim et al. [IEEE Trans. on Dependable and Secure Comput.] showed three private query processing protocols for conjunctive, disjunctive, and threshold queries respectively over an encrypted database. First, we propose two more efficient protocols of processing conjunctive and disjunctive queries with a lower-depth equality circuit than Kim et al.'s one. To get the lower-depth circuit, we modify the packing methods of Saha and Koshiba [APWConCSE 2016] to support an efficient batch computation of our protocols within a few multiplications using binary encoded data. Secondly, we propose another efficient protocol engaging a batch technique for threshold query processing in which many inequality comparisons are indispensable. Finally, we propose the packing methods to support batch computation of many inequality circuits. In addition, our theoretical analysis along with practical experiments show that our protocols are more efficient than any other existing method. We further enhance the performance of conjunctive and disjunctive query protocols

[☆]This is an extended version of the paper presented at the 31st Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec 2017) [33]. This paper extends the contribution by adding a new packing method for batch inequality comparison (Section 3.3), a threshold query protocol (Section 4.3) and its performances (Section 8.5), and comparative performance analysis between binary encoding and base-N encoding for processing private conjunctive and disjunctive queries (Section 9).

*Corresponding author
Email address: tushar@jkkniu.edu.bd (Tushar Kanti Saha)

using a base-N fixed-length encoding.

Keywords: Private database queries, Conjunctive queries, Disjunctive queries, Threshold queries, Packing method, Homomorphic encryption, Batch technique

1. Introduction

Nowadays many people are using the Internet daily. Moreover, several organizations such as educational institutions, hospitals, clinics, research organizations, banks, and insurance companies have come forward to use and save their data to a remote server using the Internet so that they can access these data securely from anywhere in the world. At the same time, data are becoming gigantic rapidly due to uploading a lot of data daily by the users. Now managing big data is a big problem for their owners because it needs to be secure from the intruder, cheater, and other evil persons. Furthermore, cloud computing has proved itself as an effective web service to store the outsourced big data. Moreover, database owners are now interested in storing their data to a third party like the cloud so that they and their allowed users can access the data from everywhere through the Internet at a low cost. At the same time, the database owners are not interested in disclosing their whole database to their users as well as to the cloud. Besides, they do not like to keep their data in their personal computer or server because of high maintenance cost as well as avoiding storage failure or regular backup of local data. However, they can secure their data in the cloud using the encryption method of a cryptographic scheme. But the encrypted data needs to be decrypted by some trusted parties before utilizing it for some purposes, which raises another security problem. In reality, it is hard to find such a trusted party. In addition, users are not interested in disclosing their queries or results to the database owners or any other parties. Therefore, it is desirable to access those encrypted data without decryption using some private queries. Now private database queries (PDBQ) [4] is a solution to them for accessing these data privately to find the information of their interest.

Since Ajtai [1] introduced the lattice-based cryptography, it has become a vital candidate of post-quantum cryptography because of the security against the attacks by both classical and quantum computers. In 2005, Regev [29] introduced another efficient lattice-based public-key cryptography whose security depends on the hardness of learning with errors (LWE) problem. In the same year, Boneh et al. [5] proposed a homomorphic encryption scheme called somewhat homomorphic encryption (SwHE) which supports many additions and one multiplication. However, one multiplication technique in [5] is not enough for the applications where many multiplications are indispensable. In addition, the role of homomorphic encryption (HE) was limited to either addition [16, 26] or multiplication [12, 32] before introducing Gentry's revolutionary work in 2009 [14]. Generally, we classify the HE schemes into three types.

- Partial homomorphic encryption (PHE) allows either addition or multiplication but not both.
- Somewhat homomorphic encryption (SwHE) allows many additions and a few multiplications.
- Fully homomorphic encryption (FHE) allows any number of additions and multiplications.

Furthermore, Gentry [14] proposed the FHE scheme based on ideal lattices by applying bootstrapping technique to an SwHE scheme. But FHE scheme is far behind from any practical application due to its inefficiency [19]. For this reason, we consider the ring-LWE based SwHE scheme in [23, 42] which is a variant of SwHE by Brakerski and Vaikuntanathan [7] constructed based on ring version of LWE problem in [29]. The SwHE scheme is also believed to be resistant to the attacks by a quantum computer. Moreover, it is faster than FHE due to supporting a limited number of multiplications and many additions. Therefore, the SwHE scheme can be used to evaluate private queries in a private database.

1.1. Motivation

For processing the PDBQ, we consider three types of query such as conjunctive, disjunctive, and threshold (CDT) queries with the equality or inequality conditions. An example of the conjunctive query is that a managing staff of a hospital finds the patients who are suffering from "leukemia" along with "high blood sugar". In addition, a doctor finds the patients who suffer from fever or cold, which is an example of the disjunctive query. Moreover, a threshold query is the query which contains at least one inequality condition in its predicate. For instance, a health researcher finds the information about those middle-aged (31–50 years) patients who have high cholesterol (> 200 mg/dL). Furthermore, we consider the security of the attributes and values appeared in the predicate of a conjunctive, disjunctive, and threshold query. To get the answers of the above mentioned private queries from some private database, some efficient methods are indispensable.

In 2016, Kim et al. [21] took an approach to evaluate CDT queries using leveled FHE [6] employing single-instruction-multiple-data (SIMD) techniques. They showed only theoretical evaluation by providing security to the values appeared in the query and database. They evaluated the multiplicative depth of every query along with the equality or inequality circuit. Here equality (resp., inequality) circuit means the algorithm which decides the equality (resp., inequality) of two integers. Later on, they made an extended experimental in [22] for the mentioned queries by providing security to the operators and values appeared in the predicate of the queries. But none of the works in [21, 22] provide security to the attributes appeared in the predicate of a query and its corresponding database. In their implementation for conjunctive and disjunctive queries [22], it took about 80.784 sec. (seconds) to perform a query on 396 elements (0.204 sec. per record) including 11 attributes of 40-bit values with the 125-bit security level. Likewise, it took about 0.20 seconds per record for processing a 3-out-of-11 threshold query. To our knowledge, the theoretical solution in [21] and the practical solution in [22] provide the most efficient so-

lutions for the above mentioned PDBQ problems using HE in the semi-honest model. But the speed of processing a query is not satisfactory enough to handle big data stored in the cloud. Consequently, there should be an efficient method to improve the performances of private database queries with a better security level.

1.2. Problem Statement

In this paper, we consider the problem of processing private CDT queries with k equality or inequality conditions over the outsourced encrypted database. In addition, we consider the security of both attributes α_i and values v_i with $1 \leq i \leq k$ appeared in the predicate of a query. Also, consider the conventional approach of processing the CDT queries. For example, let us consider the queries with k equality conditions over any *Record* table as follows:

1. Conjunctive query - *select* V *from Record where* $\alpha_1 = v_1 \wedge \alpha_2 = v_2 \wedge \cdots \wedge \alpha_k = v_k$.
2. Disjunctive query - *select* V *from Record where* $\alpha_1 = v_1 \vee \alpha_2 = v_2 \vee \cdots \vee \alpha_k = v_k$.
3. Threshold query - *select* V *from Record where* $\alpha_1 > v_1 \wedge \alpha_2 < v_2 \wedge \cdots \wedge \alpha_k > v_k$.

The conventional solution of processing the above queries with k equality conditions is that a user needs to send k queries firstly to the database server. After that, the database server performs the matching of attributes α_i and check of the values v_i with its *Record* table for equality or inequality. Then it sends back the results to the client. Then the client needs to do the intersection or union of those k results from the database to get the actual results of queries mentioned above. In addition, most of the existing solutions with the HE [10, 11, 21, 22] used the equality (resp., inequality) circuits of depth $\lceil \log l \rceil$ (resp., $1 + \lceil \log l \rceil$) for comparing two integers of l bits for equality (resp., inequality), which makes their schemes inefficient.

1.3. Our Contribution

In this paper, we affirmatively solve the problems mentioned in Section 1.2. The contributions can be summarized as follows.

1. First, we propose three efficient protocols to improve the performances of private database queries (conjunctive, disjunctive, and threshold) respectively using the binary encoding. Here the security of the protocols is ensured by using SwHE based on ring learning with errors (ring-LWE) in the semi-honest model.
2. To gain the efficiency, we use an existing packing method [34] along with batch technique so that our equality circuit is reduced to a constant-depth circuit which performs many equality comparisons simultaneously. At the same time, we propose two packing methods to support batch computation of many inequalities required for processing threshold query.
3. Our practical implementation shows that our protocols for processing conjunctive, disjunctive, and threshold queries outperform existing works with a better security level.
4. From these results, we observe that the cost of lattice-based encryption scheme mostly depends on the lattice dimension. By engaging base-N fixed-length encoding, we achieve more reduction in lattice dimension which further reduces the processing time of conjunctive and disjunctive queries.

1.4. Organization

The rest of the paper is organized in the following order. We review the earlier works in Section 2. Section 3 discusses some preliminary terms used in this paper such as basic model, data encoding technique, attribute matching, and batch processing along with the underlying security scheme of our protocols and its correctness. Section 4 elaborates the protocols for conjunctive, disjunctive, and threshold query processing. Moreover, Section 5 reviews the existing packing methods and their weaknesses and discusses new packing methods used

in our protocols. Section 6 outlines the secure computation procedure of our protocols using HE. We outline an improvement technique of our conjunctive and disjunctive query protocols in Section 7. In addition, we show the evaluation of our protocols along with the performance comparisons in Section 8 and Section 9. Finally, we conclude this paper through Section 10.

2. Reviews of Previous Works

In this subsection, we review previous works of private database queries using hybrid encryption schemes, HE schemes, and garbled circuits. In 2011, Popa et al. [28] proposed a system named "CryptDB" to process general database queries utilizing the mixed encryption schemes namely deterministic encryption [15], order-preserving encryption [3], and additive HE [26]. Moreover, it needs to decrypt encrypted attributes if multiplication operations on them are required [11]. Furthermore, mixed encryption schemes will downgrade its security in the long run. In 2013, Tu et al. [39] proposed an upgraded scheme called "MONOMI" which is a variant of CryptDB. Their solution provides some extended techniques for achieving a more efficient performance by reusing ciphertext of the Paillier encryption scheme to save the amount of space wasted in encryption for CryptDB. However, MONOMI includes most of the security holes as in CryptDB.

In case of the schemes based on the HE schemes, in 2013, Boneh et al. [4] showed an efficient method of processing conjunctive queries only with SwHE. But the performance of their scheme is still far from practicality. In 2016, Cheon et al. [11] took an approach to private query processing on encrypted databases using leveled FHE in [6]. But their performances of query processing was highly time-consuming in a practical sense. Therefore, they declared a performance improvement challenge of their protocols for processing the private queries. At the same time, Kim et al. [21] used the FHE scheme in [6] to describe another protocol for processing conjunctive, disjunctive, and threshold queries over encrypted data in the cloud. They showed the practical implementation

of their protocols in another paper [22] with extended security to both values and operators appeared in the conditional part of the queries. But it took about 0.204 seconds to access each record with 11 attributes of 40-bit values for evaluating the conjunctive and disjunctive queries. Furthermore, it took 0.20 seconds to access each record using a threshold query including 3 conditions out of 11 attributes in the database. For all cases, they achieved a security level of 125 bits. Moreover, they used an equality circuit of depth $\lceil \log l \rceil$ to compare two l-bit integers. They also used another circuit of depth $1 + \lceil \log l \rceil$ to compare two numbers of l bits for inequality. Recently, Saha and Koshiba [35] showed a more efficient protocol than that in [11] for processing a conjunctive query. But their computation technique is useful for processing conjunctive queries only.

Finally, Pappas et al. [27] presented a system called "Blind Seer" to support a rich query set over private DBMS using Yao's garbled circuits and oblivious transfer in the semi-honest model in 2014. Then Fisch et al. [13] improved the security of "Blind Seer" by adding malicious-client security. However, both of them required a non-constant round of communication for searching data using Yao's garbled circuits. In addition, none of the above protocols was able to achieve a remarkable efficiency regarding practicality.

3. Preliminaries

In this section, we describe the basic model of processing private database queries, attribute matching technique, batch processing, binary versus base-N encoding, and security scheme used during the rest of the discussion.

3.1. Basic Model

To evaluate the PDBQ, we consider the security of both attributes and values in the predicate of a conjunctive, disjunctive, and threshold query. For the processing of every query, we skip the "group by" and "order by" clauses appeared in the queries to reduce the computation complexity. The basic model of processing PDBQ is shown in Fig.1. The client (Alice) uploads the encrypted

database D' to the cloud by using a symmetric or a public-key encryption. Later on, she sends the encrypted query Q' to the cloud after parsing. The cloud server (Bob) then homomorphically evaluates the queries with the database to get some encrypted result R'_H. Then Bob in the cloud sends the encrypted results R'_H back to Alice to provide some IDs depending on the query after decryption. Then Alice sends the IDs to Bob again, and Bob sends the corresponding encrypted results to Alice. Finally, Alice decrypts the result R'_D using her secret key to obtain the desired result R_D. Using the above scenario, we evaluate our protocols for CDT queries.

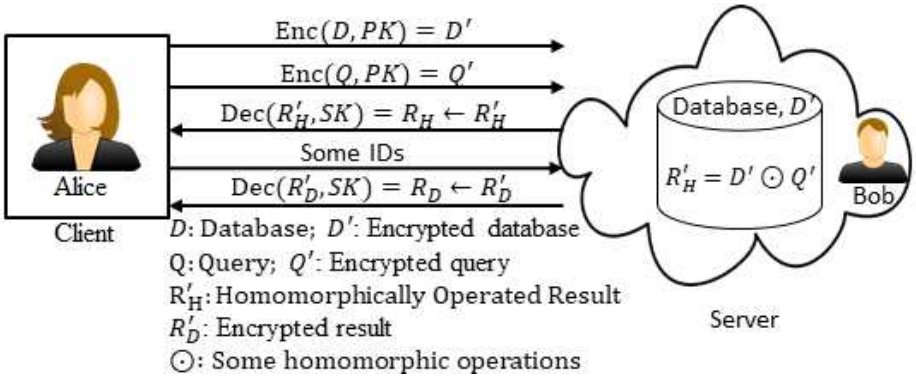

Figure 1: Basic model of private database queries.

Notations: Let $\mathbb{Z}$ denotes the ring of integers. Moreover, $\mathbb{Z}[x]$ denotes the polynomial ring with integers coefficients. For a prime number q, the ring of integers modulo q is denoted by $\mathbb{Z}_q$. Also, $\mathbb{Z}^n$ defines an n-dimensional lattice. For a vector $\mathbf{A} = (a_0, a_1, \ldots, a_{n-1})$, the maximum norm of $\mathbf{A}$ is denoted by $\|\mathbf{A}\|_\infty$. Let $\langle \mathbf{A}, \mathbf{B} \rangle$ denote the inner product between two vectors $\mathbf{A}$ and $\mathbf{B}$. In addition, the function $\mathrm{Enc}(m, pk) = \mathcal{C}$ defines the encryption of message m using the public key pk to produce the ciphertext $\mathcal{C}$. The ciphertexts $\mathcal{C}_{add}$ and $\mathcal{C}_{mul}$ denote homomorphic addition and multiplication of ciphertexts $\mathcal{C}_1 = \mathrm{Enc}(m_1, pk)$ and $\mathcal{C}_2 = \mathrm{Enc}(m_2, pk)$. The distribution $E_{\mathbb{Z}^n,\omega}$ indicates the n-dimensional discrete Gaussian distribution with the standard deviation ω. For the database

settings, k and λ represent the number of attributes in the query and *Record* table respectively. Furthermore, δ and l stand for the size of attribute and data respectively. Also, τ, $\hat{b}$, and η denote the total number of records in the *Record* table, the total number of blocks, and the total number of records in each block respectively in which a block consists of a number of records.

3.2. Binary versus Fixed Length Base-N Encoding

Encoding is a process of converting data into a certain presentation (such as binary, octal, decimal, and hexadecimal) for efficient digital transmission or storage [37]. This stored data can be processed further for getting some results. In lattice-based cryptography, the lattice dimension n of any computation depends on the length of the data to be processed. For instance, if we deal with many integers of l-bit, we can handle at most $\lfloor \frac{n}{l} \rfloor$ integers within lattice dimension n using binary encoding. As we know from existing works using SwHE based on ring-LWE [34, 35, 36, 40, 41], the cost of computation mostly depends on the lattice dimension n. If we are able to reduce the value of n, we will be able to cut down the computation cost. Recently, Saha and Koshiba [37] used base-N ($\mathbb{Z} \ni N > 2$) fixed-length encoding[1] to reduce the lattice dimension that improves the performance of the private batch equality test (PriBET) protocol in [34]. They also show that base-N encoding helps to achieve a lattice reduction factor of $\log_2(N)$ as compared to the binary encoding. Here base-N encoding constructs a number from the alphabet $\{0, 1, \ldots, N-1\}^d$, where d is the length of a number in the base-N encoded form. In this paper, we also use base-N encoding to show the reduction of computation cost for processing conjunctive and disjunctive queries.

3.3. Attribute Matching

Suppose that a medical research institute (MRI) stored its data of some patients in the cloud. Since patients' information is sensitive, MRI has uploaded

[1] In this paper, we will use "base-N encoding" to mean "base-N fixed-length encoding" for the rest of the discussion.

its database using a public-key encryption scheme. Consider that Bob has τ encrypted records $\{\mathcal{R}_1, \ldots, \mathcal{R}_\tau\}$ in its *Record* table of the MRI's database with λ attributes with $\lambda \geq k$. Here we require only the k attributes and their values from the predicate of a query to process that query. Furthermore, we denote each attribute name with a δ-bit binary vector $\alpha_i = (g_{i,0}, \ldots, g_{i,\delta-1})$, where $g_{i,\hat{c}}$ is the $\hat{c}+1$-th bit of the i-th attribute with $1 \leq i \leq k$ and $0 \leq \hat{c} \leq \delta - 1$. In addition, we need to consider k attributes among λ attributes in each record required for our conjunctive and disjunctive query processing. We also denote each attribute name in the *Record* table using a δ-bit binary vector $\beta_j = (h_{j,0}, \ldots, h_{j,\delta-1})$, where $h_{j,\hat{c}}$ is the $\hat{c}+1$-th bit of the j-th attribute for $1 \leq j \leq \lambda$. Since we consider the security of attributes, first the protocol matches encrypted attribute α_i with any attribute β_j in the *Record* table. In this case, Bob performs the matching by computing the Hamming distance between α_i and β_j as follows:

$$h_{i,j} = |\alpha_i - \beta_j| . \tag{1}$$

At this point, $h_{i,j}$ denotes the Hamming distance between attribute α_i and β_j. If $h_{i,j} = 0$, $\alpha_i = \beta_j$; otherwise, $\alpha_i \neq \beta_j$. According to our protocol, α_i must be matched with any β_j for some $1 \leq i \leq k$ and $1 \leq j \leq \lambda$.

3.4. Batch Processing

A batch processing is a method of executing a single instruction on multiple data. The performance of our protocols can be increased by using the batch processing within the lattice dimension n (see the lattice-based SwHE with ring-LWE in Section 3.5 for the details). Generally, a big database consists of many tables where each table contains numerous records. For our *Record* table, each record is represented as $\mathcal{R}_\mu = \{w_{\mu,1} \ldots, w_{\mu,\lambda}\}$ in which each value $w_{\mu,j} = (b_{\mu,j,0}, \ldots, b_{\mu,j,l-1})$ is considered as a binary vector of the same length l with $1 \leq \mu \leq \tau$ and $1 \leq j \leq \lambda$. We know that our *Record* table contains τ records. If we want to compute the Hamming distance of each v_i from each $w_{\mu,j}$ one by one then it is more time-consuming. In this case, we utilize the batch technique used in the private batch equality protocol in [34]. For processing our queries

with the batch technique, if we compare all the values of a certain attribute of a particular table using a single computation, then we will be required a higher lattice dimension n, which requires more memory for the comparison computation. This high requirement of memory may exceed the usual capacity of a machine in the cloud. For this reason, we divide all records of a table into blocks. For our given τ records, we divide the total records τ into $\hat{b}$ blocks as $\hat{b} = \lceil \frac{\tau}{\eta} \rceil$. Here each block consists of η records with λ attributes. If we access each record of our *Record* table one after another, then it requires $\tau \cdot k$ rounds communication between Alice and Bob in the cloud for accessing τ records. On the contrary, the batch processing allows us to access η values of any attribute β_j at a time. By utilizing the batch processing, we reduce the communication round between Alice and Bob in the cloud from $\tau \cdot k$ to $\lceil (\tau \cdot k)/\eta \rceil$. Now we can pack η values of the β_j-th attribute of each block in a single polynomial to support batch computation for $1 \leq j \leq \lambda$. Furthermore, we can say that the batch technique helps us to reduce not only communication rounds but also communication latency between Alice and Bob, which will reduce overall communication complexity between them.

3.5. Underlying Security Scheme

In this section, we review the ring-LWE based SwHE's scheme [23, 42] which is a variant of Brakerski and Vaikuntanathan's scheme [7]. Furthermore, we review the security and correctness of the scheme. Before reviewing the main security scheme, we give a brief overview of ring learning with errors (ring-LWE) problem.

3.5.1. Ring Learning with Errors (LWE) Problem

Learning with errors concept in cryptosystem was introduced by Regev in 2005 [29]. He showed a reduction of shortest vector problem (SVP) and shortest independent vector problem (SIVP) [30] in ideal lattices to learning problem which we call learning with errors (LWE) problem. Depending on the hardness, the LWE can be a basis of the future public key cryptography akin to the

integer factorization and the discrete logarithm problems. The LWE problem is defined by a set of random linear equations with error in which the goal is to distinguish those linear equations from truly uniform ones. For example, the random equations can be expressed as

$$\langle a_{\hat{i}}, \kappa \rangle + \epsilon_{\hat{i}} = b_{\hat{i}} \tag{2}$$

where $a_{\hat{i}}$ and secret κ are taken from $\in \mathbb{Z}_p^n$ uniformly, and the noise $\epsilon_{\hat{i}}$ is independently chosen from the distribution χ. In this example, the hardness of LWE problem is characterized by finding such secret κ with the input $(a_{\hat{i}}, b_{\hat{i}})$. Later on, Lyubashevsky et al. [24] proposed an efficient algebraic variant of LWE over ring called ring-LWE which is supported by a reduction from approximate shortest vector problem on ideal lattices to a ring-LWE problem. To define ring-LWE, let us consider a ring $R_q = \mathbb{Z}_q[x]/\phi(x)$, where q is a large prime and $\phi(x)$ is a cyclotomic polynomial. The ring-LWE problem is also defined by the random equations in (2) with the exception that $a_{\hat{i}}$ and κ are chosen from the ring R_q so that $(a_{\hat{i}}, a_{\hat{i}} \cdot \kappa \approx b_{\hat{i}}) \in (R_q)^2$.

3.5.2. Public-key Somewhat Homomorphic Encryption

Now we discuss the key generation, encryption, homomorphism, and decryption properties of SwHE scheme in [23] as follows:

PkSwHE.ParamGen. Let us consider the parameters as follows.

- n is an integer representing the lattice dimension which defines a degree n cyclotomic polynomial as $\phi(x) = x^n + 1$.
- R denotes a polynomial ring modulo $\phi(x)$ such that $R = \mathbb{Z}[x]/\phi(x)$. The ring R_q denotes the ciphertext space.
- q is a large odd prime such that $q \equiv 1 (\text{mod } 2n)$, which defines the ciphertext space R_q, a ring of polynomial modulo q and $\phi(x)$ is denoted by $R_q = \mathbb{Z}_q[x]/\phi(x)$.

- p ($p < q$) defines plaintext space as $R_p = \mathbb{Z}_p[x]/\phi(x)$ which is a ring of integer polynomials of degree $n-1$ with coefficient modulo p.
- χ denotes a Gaussian error distribution $\chi = E_{\mathbb{Z}^n,\omega}$, where $\mathbb{Z}^n$ is a ring of integer with dimension n with the standard deviation $\omega = 4 \sim 8$.

PkSwHE.KeyGen. Sample a ring element $\kappa \leftarrow \chi$ for our secret key $sk = \kappa \in R$. Now construct a secret key vector $\mathcal{K} = (1, \kappa, \kappa^2, \ldots, \kappa^\Gamma) \in R_q^{\Gamma+1}$ in which Γ defines the degree of allowed homomorphism. We then sample a uniformly random element $r_1 \in R_q$ and an error $R \ni \epsilon \leftarrow \chi$. Then public key can be defined as a pair of elements (r_0, r_1) with

$$r_0 = r_1\kappa + p\epsilon. \tag{3}$$

PkSwHE.Enc. Here ciphertext consists of two components akin to the public key. Furthermore, some errors are added during encryption of the message $m \in R_p$. First samples the errors $u, f, g \leftarrow \chi$ in the ring R. Then the encryption of m can be defined by a ciphertext pair $(\zeta_0, \zeta_1) = \mathcal{C} \in (R_q)^2$ using the public key $pk = (r_0, r_1)$ as follows:

$$\text{Enc}(m, pk) = (\zeta_0, \zeta_1) = (r_0u + pg + m, -(r_1u + pf)). \tag{4}$$

Here, the plaintext $m \in R_p$ is also in R_q because $p < q$.

PkSwHE.HomOp. Consider two ciphertexts $\mathcal{C}_1 = (\zeta_0, \ldots, \zeta_\psi) \in R_q^{\psi+1}$ and $\mathcal{C}_2 = (\zeta'_0, \ldots, \zeta'_\rho) \in R_q^{\rho+1}$. Since SwHE scheme supports many additions and a few multiplications, we show one addition and multiplication in the following way. Then the homomorphic addition ($\boxplus$) between two ciphertexts can be defined as

$$\mathcal{C}_{add} = \mathcal{C}_1 \boxplus \mathcal{C}_2 = \big(\zeta_0 + \zeta'_0, \ldots, \zeta_\gamma + \zeta'_\gamma\big) \tag{5}$$

such that $\gamma = max(\psi, \rho)$. Akin to component-wise addition, homomorphic subtraction can be defined.

Here the homomorphic multiplication ($\boxtimes$) between two same ciphertext vectors $\mathcal{C}_1$ and $\mathcal{C}_2$ can be expressed as

$$\mathcal{C}_{mul} = \mathcal{C}_1 \boxtimes \mathcal{C}_2 = \sum_{\boldsymbol{k}=0}^{\psi+\rho} \hat{\zeta}_{\boldsymbol{k}} z^{\boldsymbol{k}} = \left(\sum_{\boldsymbol{i}=0}^{\psi} \zeta_{\boldsymbol{i}} z^{\boldsymbol{i}} \right) \cdot \left(\sum_{\boldsymbol{j}=0}^{\rho} \zeta'_{\boldsymbol{j}} z^{\boldsymbol{j}} \right), \tag{6}$$

where z denotes the symbolic variable and $\hat{\zeta}_{\boldsymbol{k}}$ constitutes of the vector $(\hat{\zeta}_0, \ldots, \hat{\zeta}_{\psi+\rho}) \in R_q[z]$ for all $\boldsymbol{k}$.

PkSwHE.Dec. In this scheme, a fresh encrypted ciphertext contains two components initially. But the number of component increases due to multiplications. If a fresh or homomorphically operated ciphertext contains Γ components as $ct = (c_0, \ldots, c_\Gamma)$ and $p \in R_p$ then the decryption can be defined as

$$\mathrm{Dec}(\mathcal{C}, sk) = \left[\sum_{\boldsymbol{l}=0}^{\Gamma} c_{\boldsymbol{l}} \kappa^{\boldsymbol{l}} \right]_q \bmod \ p. \tag{7}$$

For the secret key vector $\mathcal{K} = (1, \kappa, \kappa^2, \ldots, \kappa^\Gamma)$, $\mathrm{Dec}(\mathcal{C}, sk) \equiv [\mathcal{C}, \mathcal{K}]_q \bmod p$. Now we give an example how two components ciphertext can be decrypted to obtain the plaintext in this scheme. Let us consider a fresh ciphertext $\mathcal{C} = (\zeta_0, \zeta_1)$ generated by Eq. (4) in which $\mathcal{C}$ can be decrypted by $\mathcal{K} = (1, \kappa)$ by the following ways:

$$\begin{aligned} \langle \mathcal{C}, \mathcal{K} \rangle &= [\zeta_1 + \zeta_2 \cdot \kappa]_p \\ &= (r_0 u + pg + m) - \kappa \cdot (r_1 u + pf) \bmod p \\ &= m + p \cdot (u\epsilon + g - f\kappa) \bmod p = m \end{aligned} \tag{8}$$

in the ring R_q since $r_0 - r_1\kappa = p\epsilon$ by Eq. (3). By considering adequately small values of ϵ, f, g, u, if the value $m + p \cdot (u\epsilon + g - f\kappa)$ does not wrap-around modular q then we have $[\langle \mathcal{C}, \mathcal{K} \rangle]_q = m + p \cdot (u\epsilon + g - f\kappa) = m$ in the base ring R. Using the scheme, we can recover the plaintext m by mod p operation. In addition, no wrap-around property in the encrypted results ensures the following homomorphic additions between two ciphertext $\mathcal{C}_1$ and $\mathcal{C}_2$.

$$\langle \mathcal{C}_1 \boxplus \mathcal{C}_2, \mathcal{K} \rangle = \langle \mathcal{C}_1, \mathcal{K} \rangle + \langle \mathcal{C}_2, \mathcal{K} \rangle \tag{9}$$

Furthermore, the property ensures the following homomorphic multiplication between the two same ciphertexts

$$\langle \mathcal{C}_1 \boxtimes \mathcal{C}_2, \mathcal{K} \rangle = \langle \mathcal{C}_1, \mathcal{K} \rangle \cdot \langle \mathcal{C}_2, \mathcal{K} \rangle \tag{10}$$

3.5.3. Security of the Scheme

For the plaintexts $m_0, m_1 \in R_p$ chosen by PPT adversary $\mathscr{A}$, an SwHE is indistinguishability under chosen-plaintext attack (IND-CPA) secure if the following condition holds:

$$|P[\mathscr{A}(pk, \mathrm{Enc}(m_0, pk)) = 1] - P[\mathscr{A}(pk, \mathrm{Enc}(m_1, pk)) = 1]| \leq neg(\Omega), \tag{11}$$

where $neg(\Omega)$ is negligible function in security parameter Ω. An SwHE is IND-CPA secure if it gains indistinguishability against chosen-plaintext attack. Moreover, the security of the SwHE scheme based on ring-LWE relies on the hardness of the ideal lattice problems and can be shown by the PLWE assumption as shown by Brakerski and Vaikuntanathan [7], which is a modified version of Lyubashevsky et al. [24] for the given parameters (n, q, p, ω).

PLWE Assumption. Let us consider the same polynomial ring R_q as mentioned in Section 3.5.2. Also, let $\kappa \leftarrow \chi$ be a uniformly random ring element. The assumption considers the following:

1. any polynomial number of samples of the form $(a_i, b_i = a_i \cdot \kappa + \epsilon_i) \in (R_q)^2$, where a_i is a uniformly random in R_q and ϵ_i is drawn from the error distribution χ.
2. a uniformly random pair $(a_i, u_i) \in (R_q)^2$ where the a_i's and the u_i's are uniformly random in R_q

Therefore, (a_i, b_i) is computationally indistinguishable from a uniformly random pair $(a_i, u_i) \in (R_q)^2$ where b_i's are also uniform in R_q. Besides, Lyubashevsky et al. [24] showed that the ring-LWE assumption is reducible to the worst-case hardness of the problems on ideal lattices that is believed to be secure against the cryptanalysis through a quantum computer. We can say that no

probabilistic polynomial-time (PPT) algorithms can solve PLWE problem with non-negligible advantage.

Remark 1. *Recently, Castryck et al. [8] showed provably weak instances of ring-LWE. But these kinds of weak instances do not affect the mentioned ring-LWE based SwHE scheme.*

3.5.4. Correctness of SwHE Scheme

The correctness of an encryption scheme relies on how does the decryption recover the original result from the ciphertext. In case of the HE scheme, the correctness depends on the recovery of the original result from the ciphertext through decryption after some homomorphic operations. For the ciphertexts $\mathcal{C}_1$ and $\mathcal{C}_2$ constructed from $m_1 \in R_q$ and $m_2 \in R_q$ respectively after encryption, we can write the decryption process after the homomorphic addition as

$$\mathrm{Dec}(\mathcal{C}_{add}, sk) = \mathrm{Dec}((\mathcal{C}_1 \boxplus \mathcal{C}_2), \mathcal{K}) = m_1 + m_2\,. \tag{12}$$

Similarly, we can write the decryption process after the multiplication as follows:

$$\mathrm{Dec}(\mathcal{C}_{mul}, sk) = \mathrm{Dec}((\mathcal{C}_1 \boxtimes \mathcal{C}_2), \mathcal{K}) = m_1 \cdot m_2\,. \tag{13}$$

The correctness of ring-LWE based SwHE is already described in Section 1.1 in [7]. The correctness of the scheme holds if the following lemma is satisfied mentioned in [41].

Lemma 1 (Correctness condition)**.** *For a ciphertext $\mathcal{C}$, the decryption $Dec(\mathcal{C}, sk)$ recovers the correct result if $\langle \mathcal{C}, \mathcal{K} \rangle \in R_q$ does not wrap around mod q, namely, if the condition $\|\langle \mathcal{C}, \mathcal{K} \rangle\|_\infty < \frac{q}{2}$ is satisfied in which $\|a\|_\infty = \max |a_i|$ for an element $a = \sum_{i=0}^{n-1} a_i x^i \in R_q$. Specifically, for a fresh ciphertext $\mathcal{C}$, the ∞-norm $\|\langle \mathcal{C}, \mathcal{K} \rangle\|_\infty$ is given by $\|m + p(ue + g - f\kappa)\|_\infty$. Moreover, for a homomorphically operated ciphertext, the ∞-norm can be computed by (9) and (10).*

4. Our Protocols

In this section, we describe the basic model of processing CDT queries and their corresponding protocols.

4.1. Protocol for Conjunctive Query

A conjunctive query is a query which contains many conditions in its predicate connected by "and"/"$\wedge$". For instance, a research staff tries to find the information about the patients who have from Leukemia and are 30 years old. This example is a conjunctive query request to the cloud (Bob). In this scenario, consider that Alice has a conjunctive query with k conditions in its predicate as "*select V from Record where* $\alpha_1 = v_1 \wedge \alpha_2 = v_2 \wedge \cdots \wedge \alpha_k = v_k$". For finding the answer to this query, we follow the conventional approach of processing a conjunctive query. Then it can be computed by intersecting "IDs" from the result of k sub-queries as $\bigcap_{i=1}^{k} Q(\alpha_i = v_i)$ such that $Q(\alpha_i = v_i) = \{\text{ID} \mid \text{the attribute } \alpha_i \text{ of ID takes } v_i \text{ as the value}\}$. Moreover, the values of k attributes $\{\alpha_1, \ldots, \alpha_k\}$ appeared in the predicate of the query are represented as a set $V = \{v_1, \ldots, v_k\}$ where $v_i = (a_{i,0}, \ldots, a_{i,l-1})$ is considered as a binary vector of the length l with $1 \leq i \leq k$. Here we consider the security of both attributes α_i and values v_i appeared in the predicate of the query. Firstly, Alice sends the encrypted attributes to find the required column in the *Record* table that is needed in the conjunctive query computation. Then she sends the encrypted values v_i to Bob to be matched with some $w_{\mu,j}$ using multiple Hamming distances [35] for $1 \leq \mu \leq \tau$ and $1 \leq j \leq \lambda$. To speed up the calculation using batch processing as discussed in Section 3.4, let us form a query vector $\mathbf{A}_i = (a_{i,0}, \ldots, a_{i,l-1})$ from the values of the i-th condition of the query. We also assume that the i-th attribute of query condition matches with β_j-th attribute of the *Record* table for $1 \leq j \leq \lambda$. Again we form another record vector from η values of the attribute β_j of the block σ as $\mathbf{B}_{j,\sigma} = (w_{j,\sigma,1}, \ldots, w_{j,\sigma,\eta})$ such that $w_{j,\sigma,d} = (b_{j,\sigma,d,0}, \ldots, b_{j,\sigma,d,l-1})$ with $1 \leq \sigma \leq \hat{b}$ and $1 \leq d \leq \eta$. Furthermore, $|\mathbf{A}_i| = l$ and $|\mathbf{B}_{j,\sigma}| = \eta \cdot l$. Now Bob computes multiple Hamming distances

between two vectors $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$ as

$$H_{i,\sigma,d} = \sum_{c=0}^{l-1} |a_{i,c} - b_{j,\sigma,d,c}| \tag{14}$$

for $1 \leq d \leq \eta$, $1 \leq \sigma \leq \hat{b}$, and $1 \leq j \leq \lambda$. Moreover, if $H_{i,\sigma,d}$ in Eq. (14) is 0 for some position d in the block σ then we can say that $v_i = w_{j,\sigma,d}$; otherwise, $v_i \neq w_{j,\sigma,d}$ in which $w_{j,\sigma,d}$ is the d-th sub-vector of $\mathbf{B}_{j,\sigma}$. Here multiple Hamming distances mean the distances between the vector $\mathbf{A}_i$ and each sub-vector in $\mathbf{B}_{j,\sigma}$. In this way, Alice gets some IDs for each value v_i appeared in the predicate of the query. Then she gets conjunctive query matched IDs after the intersection of all IDs for each v_i. Next, Alice sends the IDs to Bob in the cloud again and Bob returns the corresponding encrypted records to her. Finally, Alice decrypts the encrypted records using her secret key to obtain the desired result in response to her query. Now we explain our protocol for conjunctive query by the following steps.

Inputs: α_i and v_i for $1 \leq i \leq k$; β_j and $w_{j,\sigma,d}$ for $1 \leq j \leq \lambda$, $1 \leq \sigma \leq \hat{b}$ and $1 \leq d \leq \eta$

Output: $v_i = w_{j,\sigma,d}$ or $v_i \neq w_{j,\sigma,d}$

Conjunctive Query protocol:

1. Alice generates the public key and secret key by herself and encrypts each column of *Record* including attributes. Then she uploads the database to the cloud.
2. Then she also parses both attributes and values from the predicate part of her query. She encrypts α_i and v_i for $1 \leq i \leq k$ using her public key and sends them to Bob in the cloud.
3. For $1 \leq j \leq \lambda$, Bob finds out the β_j-th attribute of the *Record* table that matches α_i using the Hamming distance computation in Eq. (1) with every attribute in the database. Here Alice helps Bob to find the β_j-th attribute after decryption of the Hamming distance result of Bob.
4. For $1 \leq \sigma \leq \hat{b}$, Bob homomorphically calculates batch equality test se-

curely according to arithmetic in Eq. (14) and sends the encrypted result $ct(\mathbf{H}_{i,\sigma})$ to Alice to verify whether at least one of $H_{i,\sigma,d}$ is equal to 0.

5. For $1 \leq i \leq k$ and $1 \leq d \leq \eta$, Alice decrypts $ct(\mathbf{H}_{i,\sigma})$ using her secret key and checks each value $H_{i,\sigma,d}$ and gets the IDs for some position d such that $H_{i,\sigma,d} = 0$; In this way, she gets k sets of IDs for k conditions in the query.
6. Then Alice computes the intersection of k sets of IDs and sends the result to Bob to get her desired result.
7. Bob sends the encrypted data to Alice depending on those IDs given by Alice. Then Alice decrypts the data using her secret key to get the desired result.

4.2. Protocol for Disjunctive Query

A disjunctive query is a query which contains multiple conditions in the predicate of the query connected by "or"/"$\vee$" operator. As discussed in Section 4.1, we consider the same database settings for processing a disjunctive query. Let us consider a disjunctive query with k conditions in its predicate as "*select V from Record where* $\alpha_1 = v_1 \vee \alpha_2 = v_2 \vee \cdots \vee \alpha_k = v_k$". Here we follow the conventional approach of processing a disjunctive query. Moreover, we evaluate this query with the same technique mentioned in Eq. (14). Our protocol for processing the disjunctive query is similar to the conjunctive query protocol as discussed by 7 steps in Section 4.1 except step 6. In case of disjunctive query, Alice needs to calculate the union of k sets of IDs as $\bigcup_{i=1}^{k} Q(\alpha_i = v_i)$ instead of intersection (see step 6 in Section 4.1) required for conjunctive query protocol.

4.3. Protocol for Threshold Query

A threshold query is a query used for retrieving information from a database within some numeric limits, which contains one or more inequality condition in its predicate. It is also called range queries. As an illustration, consider that a health researcher (Alice) finds information about those middle-aged (31-50

years) patients who have high cholesterol (> 200 mg/dL) to detect chance of heart disease. In this scenario, suppose that Alice has a conjunctive query with k conditions in its predicate as "*select V from Record where* $\alpha_1 > v_1 \wedge \alpha_2 < v_2 \wedge \cdots \wedge \alpha_k > v_k$". To obtain the result of the threshold query, we follow the conventional approach for processing this query. Moreover, we calculate the answer of threshold query by taking the intersection of all "IDs" obtained from the result of the k sub-queries as $\bigcap_{i=1}^{k} Q(\alpha_i = v_i)$ such that $Q(\alpha_i = v_i) =$ {ID | the attribute α_i of ID takes v_i as the value}. Furthermore, each value of k attributes $\{\alpha_1, \ldots, \alpha_k\}$ appeared in the predicate of the query is represented by an l-bit binary vector $v_i = (a_0, \ldots, a_{l-1})$ with $1 \leq i \leq k$.

For comparing the values appeared in the predicate of a query with inequality condition, we need an inequality comparison technique. Moreover, we need to compare every single value with many values in the database record. In 2017, Saha and Koshiba proposed an efficient inequality comparison protocol for the blind online-auction [36]. Here they showed a single comparison. We modify their method to support many comparisons simultaneously. As mentioned in Section 3.4, we use batch processing to speed up the comparison calculation. At this point, we show the batch comparison between an l-bit value v_i of an attribute α_i in the predicate and a block of values $\{w_1, \ldots, w_\eta\}$ of the corresponding attribute β_j in the *Record* table. To illustrate this, consider the same query vector $\mathbf{A}_i$ and record vector $\mathbf{B}_{j,\sigma}$. Now we make the batch inequality comparison between two vectors $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$ by the following arithmetic equations as follows:

$$\begin{cases} \Delta_{i,\sigma,d,\theta} = \sum_{u=0}^{\theta-1} |a_{i,u} - b_{j,\sigma,d,u}|, \\ \Upsilon_{i,\sigma,d,c} = a_{i,c} - b_{j,\sigma,d,c} + 1, \\ \Lambda_{i,\sigma,d,c} = \Upsilon_{i,\sigma,d,c} + \Delta_{i,\sigma,d,c}, \end{cases} \tag{15}$$

where $1 \leq d \leq \eta$, $1 \leq \theta \leq l-1$, and $0 \leq c \leq l-1$. Here $\Delta_{i,\sigma,d,\theta}$ defines the Hamming distance between a binary sub-vector of $\mathbf{A}_i$ and that of $\mathbf{B}_{j,\sigma,d}$. Moreover, if $\Lambda_{i,\sigma,d,c}$ in Eq. (15) is 0 for some position d then we can say that $v_i < w_{j,\sigma,d}$; otherwise, $v_i \geq w_{j,\sigma,d}$, where $w_{j,\sigma,d}$ is the d-th sub-vector of $\mathbf{B}_{j,\sigma}$.

Furthermore, our protocol considers the security of both attributes α_i and values v_i appeared in the predicate of the query. Alice generates the public key and secret key by herself and encrypts the *Record* (row-wise and column-wise) using public key along with attributes. Now we explain our protocol for processing threshold query of Alice as follows.

Inputs: α_i and v_i for $1 \leq i \leq k$; β_j and $w_{j,\sigma,d}$ for $1 \leq j \leq \lambda$, $1 \leq \sigma \leq \hat{b}$ and $1 \leq d \leq \eta$

Output: $v_i < w_{j,\sigma,d}$ or $v_i \geq w_{j,\sigma,d}$

Threshold Query Protocol:

1. Alice parses both attributes and values from the predicate part of her threshold query. She encrypts attributes α_i and v_i using her public key and sends it to Bob in the cloud.
2. For $1 \leq i \leq k$ and $1 \leq j \leq \lambda$, Bob tries to find out the β_j-th attribute of the *Record* table that matches α_i using the Hamming distance computation in Eq. (1) with every attribute in the database. Here Alice helps Bob to find the β_j-th attribute after decryption of the Hamming distance result of Bob.
3. For each value w_d at the block σ, Bob performs the secure computation of batch inequality test as in Eq. (15) and sends the encrypted result to Alice to verify whether $\Lambda_{i,\sigma,d,c}$ is equal to 0.
4. For $1 \leq i \leq k$ and $1 \leq d \leq \eta$, Alice decrypts encrypted result using her secret key and checks each value $\Lambda_{i,\sigma,d,c}$ to decide $v_i < w_{j,\sigma,d}$ or $v_i \geq w_{j,\sigma,d}$ and gets the IDs for some position $\{d, c\}$ depending on whether $\Lambda_{i,\sigma,d,c}$ is 0 or not; In this way, she gets k sets of IDs for k conditions in the query.
5. Then Alice computes the intersection of k sets of IDs and sends the result to Bob to get her desired result.
6. Bob sends the encrypted data to Alice depending on those IDs given by Alice. Then Alice decrypts the data using her secret key and gets her desired result.

Remark 2. *Here we consider the conjunctive case in the condition for the description of threshold query protocol. Furthermore, we can process the threshold query with disjunctive conditions, which is similar to the conjunctive case achieved by changing only the intersection to union operation in step 5.*

4.4. Security of Our Protocols

In this section, we analyze the security of our protocols by proving the following theorem.

Theorem 1. *In the semi-honest model adversary model, no PPT algorithm can learn any information about the attributes and values in the query (conjunctive, disjunctive, and threshold) and database due to exploiting an IND-CPA secure SwHE in the protocols.*

Proof. Assume that Bob in our protocols is semi-honest (also known as honest-but-curious), i.e., he always follows the protocols but tries to learn information from the protocols. In this model, our protocols are IND-CPA secure if it achieves indistinguishability against chosen-plaintext attack by any PPT adversary.

Since we use the SwHE scheme mentioned in Section 3.5, our protocols generate a ciphertext pair (ζ_0, ζ_1) for every message $m \in R_p$. Moreover, any PPT adversary in CPA can choose the messages and have the access of encryption algorithm. For every attribute α_i^* with $1 \leq i \leq k$ in the simulated query, a PPT adversary generates the ciphertext pair $(\hat{\alpha}_{i_1}^*, \hat{\alpha}_{i_2}^*)$ using our encryption algorithm. On the contrary, our protocol generates the ciphertext pair $(\bar{\alpha}_{i_1}, \bar{\alpha}_{i_2})$ for each α_i in the actual query after encryption. According to PLWE assumption mentioned in Section 3.5.3,

$$(\hat{\alpha}_{i_1}^*, \hat{\alpha}_{i_2}^*) \overset{\text{c}}{\approx} (\bar{\alpha}_{i_1}, \bar{\alpha}_{i_2}),$$

where $\overset{\text{c}}{\approx}$ is computationally indistinguishable. Similarly, we can show that

$$(\hat{v}_{i_1}^*, \hat{v}_{i_2}^*) \overset{\text{c}}{\approx} (\bar{v}_{i_1}, \bar{v}_{i_2})$$

for every value $v_i^* \in R_p$ in the simulated query and $v_i \in R_p$ in the actual query. Now we can claim that our protocols achieve IND-CPA security under PLWE assumption in the semi-honest adversary model, which completes the proof. □

5. Packing Methods

In this section, we define the packing method and review some of the existing packing methods. Furthermore, we modify the packing method for private batch equality test (PriBET) protocol in [34] for our batch equality comparisons in the following subsection. Besides, we discuss a new packing method to support batch computation of many inequalities at the end of this section.

5.1. Definition of Packing Method

The method of encoding many bits into a single polynomial is called packing method. In 2011, Lauter et al. [23] used a packing method for efficient encoding of an integer $\mathbf{M} = (a_0, \ldots, a_{l-1})$ into a polynomial $Poly(\mathbf{M})$ to facilitate arithmetic operations as follows:

$$Poly(\mathbf{M}) = \sum_{i=0}^{l-1} a_i x^i \,.$$

Here each element a_i in $\mathbf{M}$ is appeared as a coefficient of $Poly(\mathbf{M})$ with different degrees of x. For instance, if $\mathbf{M} = (11001101)$ with $l = 8$, it can be encoded as $Poly(\mathbf{M}) = 1 + x^2 + x^3 + x^6 + x^7$ using the above packing method in which indexing started at least significant bit.

5.2. Some Existing Packing Methods

Recently, several packing methods were used in (see [40, 41, 34, 35, 36], and so on) for the efficient private computation in the cloud. Here we review some of the recent packing methods which were especially used for efficient single equality and inequality computation using ring-LWE based SwHE.

5.2.1. Packing Method for Single Equality Comparison

A single equality comparison helps to decide the equality between two l-bit integers a and b. Now we can realize the calculation of a single equality with the Hamming distance computation in Eq. (14) if $d = 1$. In 2016, Saha and Koshiba [34] used a packing method in [41] to support private computation of single equality of two integers. They employed the packing method to measure the Hamming distance between two l-bit integers with the help of inner product. To describe their packing method, let us consider two integer vectors $\mathbf{M} \in \mathbb{Z}^l$ and $\mathbf{N} \in \mathbb{Z}^l$ such that $\mathbf{M} = (a_0, \ldots, a_{l-1})$ and $\mathbf{N} = (b_0, \ldots, b_{l-1})$. They packed the vectors $\mathbf{M}$ and $\mathbf{N}$ in the ring R with $l \leq n$ as follows:

$$Pack_1(\mathbf{M}) = \sum_{c=0}^{l-1} a_c x^c, \quad Pack_2(\mathbf{N}) = \sum_{e=0}^{l-1} b_e x^{l-(e+1)}. \tag{16}$$

5.2.2. Packing Method for Single Inequality Comparison

We know that a single inequality comparison between two integers a and b helps to decide whether $a < b$ or $a \geq b$. The computation of a single inequality comparison is understood by the arithmetic computation in Eq. (15) if $d = 1$. In 2017, Saha and Koshiba [36] presented their packing methods to compute the single inequality of two integers efficiently. To review their packing methods, consider two same integer vectors $\mathbf{M}$ and $\mathbf{N}$ as mentioned in section 5.2.1. Here $\Delta_{i,\sigma,d,\theta}$ in Eq. (15) means multiple Hamming distances. To compute the distances, they form a special integer vector $\mathbf{U} = (b_0, b_0b_1, \ldots, b_0b_1 \cdots b_{l-2}) \in \mathbb{Z}$ from the vector $\mathbf{N}$ where $|\mathbf{U}_\theta| = \theta$ with $1 \leq \theta \leq (l-1)$. They compute multiple Hamming distances by measuring the distances between each sub-vector $\mathbf{M}_\theta = (a_0, \ldots, a_{\theta-1})$ in $\mathbf{M}$ and a sub-vector $\mathbf{U}_\theta = (b_0, \ldots, b_{\theta-1})$ in $\mathbf{U}$ of the same length as $\mathbf{U}_\theta$. Moreover, we know from [41] that the secure inner product $\langle \mathbf{M}_\theta, \mathbf{U}_\theta \rangle$ helps to compute the Hamming distances between $\mathbf{M}_\theta$ and $\mathbf{U}_\theta$. Yasuda et al. [41] pack these integer vectors by some polynomials with the highest degree$(x) = n$ in such a fashion so that the inner product $\langle \mathbf{M}_\theta, \mathbf{U}_\theta \rangle$ does not wrap-around a coefficient of x with any degree. For the integer vectors $\mathbf{M}$ and $\mathbf{U}$ with

$n > l(l-1)$, they defined their packing method in the same ring R as follows:

$$Pack_1(\mathbf{M}) = \sum_{c=0}^{l-1} a_c x^c, \quad Pack_3(\mathbf{U}) = \sum_{\theta=1}^{l-1} \sum_{\mathbf{j}=0}^{\theta-1} b_{\mathbf{j}} x^{l\cdot\theta-\mathbf{j}}. \tag{17}$$

Furthermore, the multiplication of the above two polynomials helps to compute the multiple Hamming distances between $\mathbf{M}_\theta$ and $\mathbf{U}_\theta$ in which each Hamming distance can be found as a coefficient of x with different degrees. Moreover, they need to compute $\Lambda_{i,\sigma,d,c}$ in Eq. (15). To compute $\Lambda_{i,\sigma,d,c}$, it is better to have both $\Delta_{i,\sigma,d,c}$ and $\Upsilon_{i,\sigma,d,c}$ as a coefficient of same degree polynomial to support component-wise homomorphic additions. Therefore, to obtain the $\Upsilon_{i,\sigma,d,c}$ as a coefficient of a polynomial with the same degree as $\Delta_{i,\sigma,d,c}$, they define another packing method in the same ring R for the integer vector $\mathbf{V} = (a_0, \ldots, a_{l-1})$ with $(l-1)\cdot l \leq n$ as

$$Pack_4(\mathbf{V}) = \sum_{c=0}^{l-1} a_c x^{c\cdot l}.$$

5.3. Limitations of Existing Packing Methods

In case of big database processing, the packing method for a single equality is not efficient enough to process any conjunctive and disjunctive query in which many comparisons are indispensable. Now we discuss the reasons behind the inefficiency of the packing method for a single equality. For the lattice dimension n, a single equality comparison of l ($l < n$) bits used an $n-1$ degree polynomial but uses l coefficients of that polynomial out of n coefficients, which leaves $n-l$ unused coefficients. As an illustration, for achieving more than 80-bit security level, we need to set the value of lattice dimension $n = 2048$ if we engage the packing method in Eq. (16). Here we can pack a vector of 2048 elements using the packing method. But we need at most 8 to 32 elements' vector for comparing two integers of 8 to 32 bits. As a result, there will be at least $n - l = 2048 - 32 = 2016$ unused coefficients in the polynomial for each comparison. Recall from Section 3.4, the batch technique can help us to reuse the unused space in this case. Therefore, we use the packing method for batch equality in [34].

Likewise, the packing method for a single inequality is not profitable for handling threshold query in which many inequality comparisons are indispensable. For the lattice dimension of n, the packing method can use at most $l(l-1)/2$ positions out of n for comparing two integers of l-bit integers. As mentioned above, the packing method in Eq. (17) also requires a lattice dimension of 2048 and can handle a vector of 2048. But we can use a vector of size 32(32-1)/2=496 out of 2048 for comparing two integers of 32-bit. Therefore, we will have many unused coefficients in the polynomials if we use the packing method in Eq. (17) for a single comparison, whereas we require many comparisons. Here we should have some new packing methods rather the packing methods mentioned in Section 5.2.2.

5.4. Our Packing Methods

In our protocols, we consider the security of both attributes and values appeared in the predicate of the CDT queries. Consequently, we need the packing methods for the secure computation of both attributes and values. In this subsection, we discuss the packing methods required for the batch equality and inequality computations of our protocols.

5.4.1. Packing Method for Batch Equality Comparison

We want to compute the multiple Hamming distances $H_{i,\sigma,d}$ in Eq. (14) with a few polynomial multiplications to accelerate the equality comparisons and reduce the computation cost of the conjunctive and disjunctive queries including value matching. As discussed in Section 4.1, for $1 \leq i \leq k$ and $1 \leq j \leq \lambda$, we consider two same integer vectors $\mathbf{A}_i = (a_{i,0}, \ldots, a_{i,l-1}) \in \mathbb{Z}^l$ and $\mathbf{B}_{j,\sigma} = (b_{j,\sigma,1,0}, \ldots, b_{j,\sigma,1,l-1}, \ldots, b_{j,\sigma,\eta,0}, \ldots, b_{j,\sigma,\eta,l-1}) \in \mathbb{Z}^{\eta \cdot l}$. Here we need to determine the Hamming distances between $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$.

Furthermore, we know from [41] that the secure inner product $\langle \mathbf{A}_i, \mathbf{B}_{j,\sigma} \rangle$ helps to compute the Hamming distances between $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$. Here we pack these integer vectors by some polynomials with the highest degree$(x) = n - 1$ in such a way so that the inner product $\langle \mathbf{A}_i, \mathbf{B}_{j,\sigma} \rangle$ does not wrap-around a

coefficient of x with any degree. *If we use the packing method in Section 5.2.1, then we have to use them $\hat{b} * \eta$ times, which results in wastage of $(n-l) \cdot \hat{b} \cdot \eta$ coefficients in the packed polynomials.* Therefore, we use the packing method for batch equality computation in [34]. Here we modify the coefficient part of the batch equality packing methods in [34] to support batch equality computation for each block. For the integer vectors $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$ with $n \geq \eta \cdot l$ and $1 \leq d \leq \eta$, the packing method of [34] in the same ring R can be rewritten as follows:

$$\begin{cases} Poly_1(\mathbf{A}_i) = \sum_{c=0}^{l-1} a_{i,c} x^c, \\ Poly_2(\mathbf{B}_{j,\sigma}) = \sum_{d=1}^{\eta} \sum_{e=0}^{l-1} b_{j,\sigma,d,e} x^{l \cdot d - (e+1)}. \end{cases} \tag{18}$$

If we multiply the above two polynomials, it will help us to find the inner product $\langle \mathbf{A}_i, \mathbf{B}_{j,\sigma} \rangle$, which in turn helps the multiple Hamming distances computation between the vectors $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$. Here each Hamming distance can be found as a coefficient the polynomial with different degrees of x. Now consider the above two vectors $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$ again. Furthermore, we represent the polynomial multiplications of $Poly_1(\mathbf{A}_i)$ and $Poly_2(\mathbf{B}_{j,\sigma})$ in the same base ring R as follows:

$$\begin{aligned} & \left(\sum_{c=0}^{l-1} a_{i,c} x^c \right) \times \left(\sum_{d=1}^{\eta} \sum_{e=0}^{l-1} b_{j,\sigma,d,e} x^{l \cdot d - (e+1)} \right) \\ = & \sum_{d=1}^{\eta} \sum_{c=0}^{l-1} \sum_{e=0}^{l-1} a_{i,c} b_{j,\sigma,d,e} x^{c + l \cdot d - (e+1)} \\ = & \sum_{d=1}^{\eta} \sum_{c=0}^{l-1} a_{i,c} b_{j,\sigma,d,c} x^{l \cdot d - 1} + \mathrm{HD} + \mathrm{LD} \\ = & \sum_{d=1}^{\eta} \langle \mathbf{A}_i, \mathbf{B}_{j,\sigma,d} \rangle x^{l \cdot d - 1} + \mathrm{HD} + \mathrm{LD}. \end{aligned} \tag{19}$$

Here $\mathbf{A}_i$ is the i-th query vector of the length l that appeared in the predicate of a conjunctive or disjunctive query for $1 \leq i \leq k$. In addition, $\mathbf{B}_{j,\sigma,d}$ is the d-th sub-vector of $\mathbf{B}_{j,\sigma}$ of the block σ with the j-th attribute of the *Record* table for $1 \leq \sigma \leq \hat{b}$, $1 \leq d \leq \eta$, and $1 \leq j \leq \lambda$. Moreover, the HD (terms of higher degree) defines the sub-polynomial with $deg(x) > l \cdot d - 1$ and the LD

(lower degrees) defines the sub-polynomial with $deg(x) < l \cdot d - 1$. The result in Eq. (19) shows that one polynomial multiplication includes η inner products $\langle \mathbf{A}_i, \mathbf{B}_{j,\sigma,d} \rangle$. According to the SwHE in Section 3.5.2, the packed ciphertexts for $Poly_\iota(A) \in R$ are defined for some $\iota \in \{1, 2, 3, 4\}$ as follows:

$$ct_\iota(A) = \mathrm{Enc}(Poly_\iota(A), pk) \in (R_q)^2 . \tag{20}$$

Depending on the packing method in Eq. (18) and the inner product in Eq. (19), the following proposition is needed to be held to get our result as the coefficients of a large polynomial using the inner product after homomorphic multiplication.

Proposition 1. *Consider the same integer vectors $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$. If the ciphertexts of $Poly_1(\mathbf{A}_i)$ and $Poly_2(\mathbf{B}_{j,\sigma})$ can be represented as $ct_1(\mathbf{A}_i)$ and $ct_2(\mathbf{B}_{j,\sigma})$ respectively by Eq. (20) then under the condition of Lemma 1, the decryption of homomorphic multiplication $ct_1(\mathbf{A}_i) \boxtimes ct_2(\mathbf{B}_{j,\sigma}) \in (R_q)^3$ will produce a polynomial of R_p with $x^{l \cdot d - 1}$ including coefficient $\langle \mathbf{A}_i, \mathbf{B}_{j,\sigma,d} \rangle = \sum_{c=0}^{l-1} a_{i,c} b_{j,\sigma,d,c}$ mod p. Alternatively, we can say that homomorphic multiplication of $ct_1(\mathbf{A}_i)$ and $ct_2(\mathbf{B}_{j,\sigma})$ simultaneously computes the multiple inner products $\langle \mathbf{A}_i, \mathbf{B}_{j,\sigma,d} \rangle$ for $1 \le i \le k$, $1 \le \sigma \le \hat{b}$, $1 \le d \le \eta$, and $1 \le j \le \lambda$.*

Proof. The vectors $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$ can be converted into polynomials $Poly_1(\mathbf{A}_i)$ and $Poly_2(\mathbf{B}_{j,\sigma})$ respectively using the packing method in Eq. (18). In addition, the ciphertexts $ct_1(\mathbf{A}_i)$ and $ct_2(\mathbf{B}_{j,\sigma})$ are produced from $Poly_1(\mathbf{A}_i)$ and $Poly_2(\mathbf{B}_{j,\sigma})$ respectively by Eq. (20). By the correctness in Eq. (13) and Lemma 1, we can say that the homomorphic multiplication of $ct_1(\mathbf{A}_i)$ and $ct_2(\mathbf{B}_{j,\sigma})$ corresponds to the polynomial multiplication of $Poly_1(\mathbf{A}_i)$ and $Poly_2(\mathbf{B}_{j,\sigma})$ in the ring R_p. Moreover, it is obvious from the inner product in Eq. (19), the polynomial multiplication of $Poly_1(\mathbf{A}_i)$ and $Poly_2(\mathbf{B}_{j,\sigma})$ produces the inner products $\langle \mathbf{A}_i, \mathbf{B}_{j,\sigma,d} \rangle$ as the coefficients of $x^{l \cdot d - 1}$ which equals $\sum_{c=0}^{l-1} a_{i,c} b_{j,\sigma,d,c}$ for $1 \le i \le k$, $1 \le \sigma \le \hat{b}$, $1 \le d \le \eta$, $0 \le c \le (l-1)$, and $1 \le j \le \lambda$. For this reason, the proposition 1 holds under the correctness in Eq. (13) and Lemma 1. □

Remark 3. *Here we use the same packing method in Section 5.4.1 for our*

attribute matching such that $\sigma = 1$, $0 \leq c \leq (\delta - 1)$, $0 \leq e \leq (\delta - 1)$, *and* $1 \leq d \leq \lambda$.

5.4.2. Packing Method for Batch Inequality Comparison

As mentioned in Section 5.2.2, Saha and Koshiba [36] used two packing methods for the efficient computations of a single inequality comparison. In our case, $\Lambda_{i,\sigma,d,c}$ in Eq. (15) represents many comparison results. To solve many inequalities problem, we need to use the packing method in Eq. (17) repeatedly. As mentioned in Section 5.3, it is not efficient to repeatedly use the packing method due to waste of the space in polynomials. To compute many comparisons result simultaneously and efficiently, we need to define another packing method instead of that in [41].

Here we want to calculate many inequality comparisons using a few polynomial multiplications. For this purpose, we need to compute incremental Hamming distances $\Delta_{i,\sigma,d,\theta}$ as a partial calculation according to Eq. (15). Consider the same integer vector $\mathbf{A}_i$. Moreover, let us consider the binary vector $w_{j,\sigma,d} = (b_{j,\sigma,d,0}, \ldots, b_{j,\sigma,d,l-1})$ for $1 \leq d \leq \eta$. Then we form another integer vector $\mathbf{P}_{j,\sigma} = (b_{j,\sigma,1,0}, b_{j,\sigma,1,0}b_{j,\sigma,1,1}, \ldots, b_{j,\sigma,1,0}b_{j,\sigma,1,1}\cdots b_{j,\sigma,1,l-2}, \ldots, b_{j,\sigma,\eta,0}, b_{j,\sigma,\eta,0}b_{j,\sigma,\eta,1}, \ldots, b_{j,\sigma,\eta,0}b_{j,\sigma,\eta,1}\cdots b_{j,\sigma,\eta,l-2}) \in \mathbb{Z}$ by taking sub-vectors sequentially from $w_{j,\sigma,d}$ of the length θ where $1 \leq \theta \leq l-1$. Now we pack $\mathbf{A}_i$ and $\mathbf{P}_{j,\sigma}$ in the ring R with $n > l^2 \cdot \eta$ as follows:

$$\begin{cases} Poly_1(\mathbf{A}_i) = \sum_{c=0}^{l-1} a_{i,c}x^c, \\ Poly_3(\mathbf{P}_{j,\sigma}) = \sum_{d=1}^{\eta}\sum_{\theta=1}^{l-1}\sum_{u=0}^{\theta-1} b_{j,\sigma,d,u}\, x^{l^2(d-1)-l(d-\theta)-u}. \end{cases} \tag{21}$$

We need to compute multiple inner products of the form $\langle \mathbf{A}_{i,\theta}, \mathbf{P}_{j,\sigma,d,\theta} \rangle$, where $\mathbf{A}_{i,\theta} = (a_{i,0}, \ldots, a_{i,\theta-1})$ and $\mathbf{P}_{j,\sigma,d,\theta} = (b_{j,\sigma,d,0}, \ldots, b_{j,\sigma,d,\theta-1})$. Now we represent the multiplication between $Poly_1(\mathbf{A}_i)$ and $Poly_3(\mathbf{P}_{j,\sigma})$ as follows:

$$\Big(\sum_{c=0}^{l-1} a_{i,c}x^c\Big) \times \Big(\sum_{d=1}^{\eta}\sum_{\theta=1}^{l-1}\sum_{u=0}^{\theta-1} b_{j,\sigma,d,u}\, x^{l^2(d-1)-l(d-\theta)-u}\Big)$$

$$
\begin{aligned}
&= \sum_{d=1}^{\eta}\sum_{\theta=1}^{l-1}\sum_{c=0}^{l-1}\sum_{u=0}^{\theta-1} a_{i,c}\cdot b_{j,\sigma,d,u}x^{l^2(d-1)-l(d-\theta)+(c-u)} \\
&= \sum_{d=1}^{\eta}\sum_{\theta=1}^{l-1}\sum_{u=0}^{\theta-1} a_{i,u}\cdot b_{j,\sigma,d,u}x^{l^2(d-1)-l(d-\theta)} + \mathrm{HD} + \mathrm{LD} \\
&= \sum_{d=1}^{\eta}\sum_{\theta=1}^{l-1}\langle \mathbf{A}_{i,\theta}, \mathbf{P}_{j,\sigma,d,\theta}\rangle x^{l^2(d-1)-l(d-\theta)} + \mathrm{HD} + \mathrm{LD},
\end{aligned}
$$

where $\mathbf{A}_{i,\theta}$ is the θ-th sub-vector of $\mathbf{A}_i$ of the length θ that appears in the predicate of a threshold query for $1 \leq \theta \leq l-1$. In addition, $\mathbf{P}_{j,\sigma,d,\theta}$ is the θ-th sub-vector of $\mathbf{P}_{j,\sigma}$ with $1 \leq d \leq \eta$. Moreover, the HD (higher degree) means the sub-polynomial with $deg(x) > l^2(d-1) - l(d-\theta)$ and the LD (lower degree) means the sub-polynomial with $deg(x) < l^2(d-1) - l(d-\theta)$. The result in Eq. (22) shows that one polynomial multiplication includes the multiple inner products of $\langle \mathbf{A}_{i,\theta}, \mathbf{P}_{j,\sigma,d,\theta}\rangle$. From the packing method in Eq. (21), the following proposition is needed to get our result as a coefficient of a large polynomial using the inner product in Eq. (22) after homomorphic multiplication.

Proposition 2. *Consider the same integers vectors $\mathbf{A}_i$ and $\mathbf{P}_{j,\sigma}$. If the ciphertexts of $Poly_1(\mathbf{A})$ and $Poly_3(\mathbf{P}_{j,\sigma})$ can be represented as $ct_1(\mathbf{A}_i)$ and $ct_3(\mathbf{P}_{j,\sigma})$ respectively by Eq. (20) then under the condition of Lemma 1, decryption of homomorphic multiplication $ct_1(\mathbf{A}_i) \boxtimes ct_3(\mathbf{P}_{j,\sigma}) \in (R_q)^3$ will produce a polynomial of R_p with $x^{l^2(d-1)-l(d-\theta)}$ including coefficients $\langle \mathbf{A}_{i,\theta}, \mathbf{P}_{j,\sigma,d,\theta}\rangle = \sum_{u=0}^{\theta-1} a_{i,u}\cdot b_{j,\sigma,d,u}$ mod p. Alternatively, we can say that homomorphic multiplication of $ct_1(\mathbf{A}_i)$ and $ct_3(\mathbf{P}_{j,\sigma})$ simultaneously computes many inner products $\langle \mathbf{A}_{i,\theta}, \mathbf{P}_{j,\sigma,d,\theta}\rangle$ for $1 \leq d \leq \eta$ and $1 \leq \theta \leq l-1$.*

Proof. The vectors $\mathbf{A}_i$ and $\mathbf{P}_{j,\sigma}$ can be converted into polynomials $Poly_1(\mathbf{A})$ and $Poly_3(\mathbf{P}_{j,\sigma})$ respectively using Eq. (21). In addition, the ciphertexts $ct_1(\mathbf{A}_i)$ and $ct_3(\mathbf{P}_{j,\sigma})$ are produced from $Poly_1(\mathbf{A})$ and $Poly_3(\mathbf{P}_{j,\sigma})$ respectively by Eq. (20). By the correctness in Eq. (13) and Lemma 1, we can say that the homomorphic multiplication of $ct_1(\mathbf{A}_i)$ and $ct_3(\mathbf{P}_{j,\sigma})$ corresponds to the polynomial multiplication of $Poly_1(\mathbf{A}_i)$ and $Poly_3(\mathbf{P}_{j,\sigma})$ in the ring R_p. Moreover, it is evident from

the inner product in Eq. (22), the polynomial multiplication of $Poly_1(\mathbf{A}_i)$ and $Poly_3(\mathbf{P}_{j,\sigma})$ produces the inner products $\langle \mathbf{A}_{i,\theta}, \mathbf{P}_{j,\sigma,d,\theta} \rangle$ as the coefficients of $x^{l^2(d-1)-l(d-\theta)}$ which equals $\sum_{u=0}^{\theta-1} a_{i,u} \cdot b_{j,\sigma,d,u}$ for $1 \leq d \leq \eta$ and $1 \leq \theta \leq l-1$. For this reason, the proposition 2 holds under the correctness in Eq. (13) and Lemma 1. □

Here the above packing methods is helpful for the computation of $\Delta_{i,\sigma,d,c}$ in Eq. (15). To calculate $\Lambda_{i,\sigma,d,c}$, we also need to compute $\Upsilon_{i,\sigma,d,c}$ in Eq. (15) using some packing methods so that the result of $\Delta_{i,\sigma,d,c}$ and $\Upsilon_{i,\sigma,d,c}$ are appeared as the coefficients of equal-degree polynomials. For $1 \leq d \leq \eta$ and $0 \leq c \leq l-1$, we define another packing method to compute $\Upsilon_{i,\sigma,d,c}$. Let us form another vector $\mathbf{Q} = (b_{j,\sigma,1,0}, \ldots, b_{j,\sigma,1,l-1}, \ldots, b_{j,\sigma,\eta,0}, \ldots, b_{j,\sigma,\eta,l-1})$ from the binary vector $w_{j,\sigma,d}$. Now we define another packing method with $l^2 \cdot \theta \leq n$ as follows:

$$Poly_4(\mathbf{Q}) = \sum_{d=1}^{\eta} \sum_{c=0}^{l-1} b_{d,c} x^{l^2(d-1)-l(d-c)}. \tag{22}$$

6. Secure Computation Procedure

We need to securely match both attribute and value as mentioned in our protocols in Section 4.1. Now we discuss the matching technique of both attributes and values of the CDT queries with the *Record* table in the following sub-sections.

6.1. Matching the Attributes

We match the attributes α_i with β_j for $1 \leq i \leq k$ and $1 \leq j \leq \lambda$ by the Hamming distance in Eq. (1) and the packing method in Eq. (18). In addition, according to Eq. (1), we need to find out the values of the multiple Hamming distances $h_{i,j}$. The multiple Hamming distances mean the distances between an attribute α_i and all β_j in the *Record* table. Furthermore, we consider two δ-bit vectors $\alpha_i = (g_{i,0}, \ldots, g_{i,\delta-1})$ and $\beta_j = (h_{j,0}, \ldots, h_{j,\delta-1})$. From these vectors, we form two integer vectors as $\mathbf{X}_i = (g_{i,0}, \ldots, g_{i,\delta-1}) \in \mathbb{Z}^{\delta}$ and $\mathbf{Y} = (h_{1,0}, \ldots, h_{1,\delta-1}, \ldots, h_{\lambda,0}, \ldots, h_{\lambda,\delta-1}) \in \mathbb{Z}^{\delta \cdot \lambda}$. Now we can find $h_{i,j}$ from

these vectors $\mathbf{X}_i$ and $\mathbf{Y}$. Here $h_{i,j}$ is computed by multiple Hamming distances between $\mathbf{X}_i$ and $\mathbf{Y}$ using the arithmetic equation as follows:

$$h_{i,j} = \sum_{\hat{c}=0}^{\delta-1} (g_{i,\hat{c}} + h_{j,\hat{c}} - 2g_{i,\hat{c}}h_{j,\hat{c}}) \,. \tag{23}$$

Since the inner product helps to obtain many Hamming distances [41], we can calculate λ Hamming distance using three inner products and two additions. According to Eq. (19), one inner product corresponds to one polynomial multiplication. For the above two integer vectors $\mathbf{X}_i$ and $\mathbf{Y}$, multiple Hamming distances $h_{i,j}$ in Eq. (23) can be computed by the packing method in Eq. (18) and the inner product property in Eq. (19). Then $h_{i,j}$ in Eq. (23) can be calculated by the equation as follows:

$$\begin{aligned} Poly(H_i) = Poly_1(\mathbf{X}_i) \times Poly_2(I_2) + Poly_2(\mathbf{Y}) \times Poly_1(I_1) \\ - 2Poly_1(\mathbf{X}_i) \times Poly_2(\mathbf{Y}) \,, \end{aligned} \tag{24}$$

where I_1 denotes an integer vector like $(1, \ldots, 1)$ of the length δ and I_2 denotes another integer vector $(1, \ldots, 1)$ of the length $\lambda \cdot \delta$.

Computation over packed ciphertext. The packed ciphertext is an encrypted polynomial in which the polynomial is generated from an integral vector using some packing methods. Moreover, we can calculate the packed ciphertexts of $Poly_1(\mathbf{X}_i)$, $Poly_2(\mathbf{Y})$, $Poly_1(I_1)$, and $Poly_2(I_2)$ as $ct_1(\mathbf{X}_i)$, $ct_2(\mathbf{Y})$, $ct_1(I_1)$, and $ct_2(I_2)$ respectively with the Eq. (20). Now $Poly(H_i)$ is computed by Proposition 1 and the packed ciphertexts in three homomorphic multiplications and two homomorphic additions such that $ct(H_i)$ equals

$$ct_1(\mathbf{X}_i) \boxtimes ct_2(I_2) \boxplus ct_2(\mathbf{Y}) \boxtimes ct_1(I_1) \boxplus (-2ct_1(\mathbf{X}_i) \boxtimes ct_2(\mathbf{Y})) \,, \tag{25}$$

where $\boxplus$ (resp., $\boxtimes$) stands for homomorphic addition (resp., multiplication). The above-encrypted polynomial $ct(H_i)$ includes λ Hamming distances between the vector $\mathbf{X}_i$ and sub-vectors of $\mathbf{Y}$. Here we need the Hamming distance $h_{i,j}$ in Eq. (23). Bob sends $ct(H_i)$ to Alice for decryption. According to Proposition 1 and our protocol, Alice decrypts $ct(H_i)$ in the ring R_q using her secret key and

extracts $h_{i,j}$ as a coefficient of $x^{\delta \cdot j-1}$ from the plaintext of $ct(H_i)$. Then Alice checks whether at least one of the $h_{i,j}$ equals 0 or not to help Bob to decide whether $\alpha_i = \beta_j$ or $\alpha_i \neq \beta_j$.

6.2. Comparing the Values

In this sub-section, we elaborate the procedure of comparing the values in the predicate of our queries with that of the database records for the equalities and inequalities.

6.2.1. Batch Equality Comparison

We need a secure batch equality comparison for the computation of both private conjunctive and disjunctive query protocols. Now we compute η Hamming distances in Eq. (14) using the packing method in Eq. (18) for matching the records. Here η Hamming distances $H_{i,\sigma,d}$ mean the distances between a value v_i of the i-th attribute in the predicate of a query and all the records in the block σ of the j-th attribute such that $\alpha_i = \beta_j$. Recall from Section 5.4.1, we consider two same integer vectors $\mathbf{A}_i \in \mathbb{Z}^l$ and $\mathbf{B}_{j,\sigma} \in \mathbb{Z}^{\eta \cdot l}$ from which $H_{i,\sigma,d}$ can be computed. For $1 \leq d \leq \eta$, $H_{i,\sigma,d}$ is computed by many Hamming distances between $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$ using the arithmetic equation as follows:

$$H_{i,\sigma,d} = \sum_{c=0}^{l-1} (a_{i,c} + b_{j,\sigma,d,c} - 2a_{i,c}b_{j,\sigma,d,c}) \,. \tag{26}$$

For two integer vectors $\mathbf{A}_i$ and $\mathbf{B}_{j,\sigma}$ mentioned above, many Hamming distances $H_{i,\sigma,d}$ in Eq. (26) can be computed by the packing method in Eq. (18) and the inner product property in Eq. (19) as follows:

$$\begin{aligned} Poly(\mathbf{H}_{i,\sigma}) = Poly_1(\mathbf{A}_i) \times Poly_2(I_4) + Poly_2(\mathbf{B}_{j,\sigma}) \\ \times Poly_1(I_3) - 2Poly_1(\mathbf{A}_i) \times Poly_2(\mathbf{B}_{j,\sigma}) \,, \end{aligned} \tag{27}$$

where I_3 is an integer vector like $(1, \ldots, 1)$ of the length l and I_4 is another integer vector like $(1, \ldots, 1)$ of the length $\eta \cdot l$.

Computation over packed ciphertext. The packed ciphertexts of $Poly_1(\mathbf{A}_i)$, $Poly_2(\mathbf{B}_{j,\sigma})$, $Poly_1(I_3)$, and $Poly_2(I_4)$ can be represented as $ct_1(\mathbf{A}_i)$, $ct_2(\mathbf{B}_{j,\sigma})$,

$ct_1(I_3)$, and $ct_2(I_4)$ respectively with the Eq. (20). Now $H_{i,\sigma,d}$ is computed from Proposition 1 and the packed ciphertexts in three homomorphic multiplications and two homomorphic additions as follows:

$$ct(\mathbf{H}_{i,\sigma}) = ct_1(\mathbf{A}_i) \boxtimes ct_2(I_4) \boxplus ct_2(\mathbf{B}_{j,\sigma}) \boxtimes ct_1(I_3) \boxplus (-2ct_1(\mathbf{A}_i) \boxtimes ct_2(\mathbf{B}_{j,\sigma})). \quad (28)$$

The above-encrypted polynomial $ct(\mathbf{H}_{i,\sigma})$ includes many Hamming distances between the sub-vectors of $\mathbf{A}_i$ and sub-vectors of $\mathbf{B}_{j,\sigma}$. Here we need η Hamming distances $H_{i,\sigma,d}$ in Eq. (26). Then Bob sends $ct(\mathbf{H}_{i,\sigma})$ to Alice for decryption. According to Proposition 1 and our PriBET protocol, Alice decrypts $ct(\mathbf{H}_{i,\sigma})$ in the ring R_q using her secret key and extracts $H_{i,\sigma,d}$ as a coefficient of $x^{l\cdot d-1}$ from the plaintext of $ct(\mathbf{H}_{i,\sigma})$. Then Alice checks whether at least one of the $H_{i,\sigma,d}$ equals 0 or not to decide whether $v_i = w_{j,\sigma,d}$ or $v_i \neq w_{j,\sigma,d}$.

6.2.2. Batch Inequality Comparison

As mentioned in Section 4.3, we need many inequality comparisons between a query value and a set of values in the record simultaneously for the efficient computation of threshold queries. Now we do these computations of secure batch inequality comparison using the packing methods of Section 5.4.2. According to Eq. (15), we need to find out the values of incremental Hamming distances $\Delta_{i,\sigma,d,\theta}$, inequality decision aider $\Upsilon_{i,\sigma,d,c}$, and final result $\Lambda_{i,\sigma,d,c}$.

	$\Delta_{i,\sigma,d,1}$	$\Delta_{i,\sigma,d,2}$	$\Delta_{i,\sigma,d,3}$	$\Delta_{i,\sigma,d,4}$	$\Delta_{i,\sigma,d,5}$	$\Delta_{i,\sigma,d,6}$	$\Delta_{i,\sigma,d,7}$
$\boldsymbol{A_{i,\theta}}$	a_0	a_0a_1	$a_0a_1a_2$	$a_0a_1a_2a_3$	$a_0a_1a_2a_3a_4$	$a_0a_1a_2a_3a_4a_5$	$a_0a_1a_2a_3a_4a_5a_6$
$\boldsymbol{P_{j,\sigma,d,\theta}}$	$b_{\theta,0}$	$b_{\theta,0}b_{\theta,1}$	$b_{\theta,0}a_{\theta,1}a_{\theta,2}$	$b_{\theta,0}b_{\theta,1}b_{\theta,2}b_{\theta,3}$	$b_{\theta,0}b_{\theta,1}b_{\theta,2}b_{\theta,3}b_{\theta,4}$	$b_{\theta,0}b_{\theta,1}b_{\theta,2}b_{\theta,3}b_{\theta,4}b_{\theta,5}$	$b_{\theta,0}b_{\theta,1}b_{\theta,2}b_{\theta,3}b_{\theta,4}b_{\theta,5}b_{\theta,6}$

Figure 2: Pictorial representation of the calculation of $\Delta_{i,\sigma,d,\theta}$ for $a_{i,u} = a_u$ and $b_{j,\sigma,\theta,u} = b_{\theta,u}$ for $0 \leq u \leq \theta - 1$

From Section 5.4.2, we consider the same vectors $\mathbf{A}_i \in \mathbb{Z}^l$ and $\mathbf{P}_{j,\sigma} \in \mathbb{Z}^{\eta\cdot l}$ to compute $\Delta_{i,\sigma,d,\theta}$. For $1 \leq \theta \leq (l-1)$, $\Delta_{i,\sigma,d,\theta}$ is computed by the incremental Hamming distances between sub-vectors of $\mathbf{A}_i$ and sub-vectors of $\mathbf{P}_{j,\sigma}$ as shown

in Fig.2 using the arithmetic computation as follows:

$$\Delta_{i,\sigma,d,c} = \sum_{u=0}^{c-1} (a_{i,u} + b_{i,\sigma,d,u} - 2a_i b_{i,\sigma,d,u})\,. \tag{29}$$

The above computation can be realized by engaging the packing method in Eq. (21) and the inner product property in Eq. (22) as follows:

$$\begin{aligned} Poly(\Delta_{i,\sigma}) = Poly_1(\mathbf{A}_i) \times Poly_3(\mathbf{I}) + Poly_1(\mathbf{W}) \\ \times Poly_3(\mathbf{P}_{j,\sigma}) - 2Poly_1(\mathbf{A}_i) \times Poly_3(\mathbf{P}_{j,\sigma})\,, \end{aligned} \tag{30}$$

where $\mathbf{I}$ denotes an integer vector $(1, \ldots, 1)$ of the same length as $\mathbf{P}_{j,\sigma}$ and $\mathbf{W}$ denotes another integer vector $(1, \ldots, 1)$ of the length l. To compute $\Upsilon_{i,\sigma,d,c}$, we construct the vectors $\hat{\mathbf{A}} = (a_0, \ldots, a_{l-1}, \ldots, a_0, \ldots, a_{l-1})$ by repeating vector $\mathbf{A}_i$ η times and $\hat{\mathbf{B}} = (b_{1,0}, b_{1,1}, \ldots, b_{1,l-1}, \ldots, b_{\eta,0}, b_{\eta,1}, \ldots, b_{\eta,l-1})$ by combining all the sub-vectors in $\mathbf{B}_{j,\sigma}$. Furthermore, the inequality decision aider $\Upsilon_{i,\sigma,d,c}$ and final result $\Lambda_{i,\sigma,d,c}$ can be computed for $0 \leq c \leq (l-1)$ and $1 \leq d \leq \eta$ as

$$\Upsilon_{i,\sigma,d,c} = a_{i,c} - b_{j,\sigma,d,c} + 1, \qquad \text{and} \tag{31}$$

$$\Lambda_{i,\sigma,d,c} = \Upsilon_{i,\sigma,d,c} + \Delta_{i,\sigma,d,c}\,. \tag{32}$$

Using the packing method in Eq. (22), $\Upsilon_{i,\sigma,d,c}$ in Eq. (31) can be calculated as follows:

$$Poly(\Upsilon_{i,\sigma}) = Poly_4(\hat{\mathbf{A}}) \boxplus (-Poly_4(\hat{\mathbf{B}})) \boxplus Poly_4(\hat{\mathbf{W}})\,, \tag{33}$$

where $\hat{\mathbf{W}}$ is a vector like $(1, \ldots, 1)$ of the length $\eta \cdot l$. Then $\Lambda_{i,\sigma,d,c}$ in Eq. (32) can be calculated from $Poly(\Delta_{i,\sigma})$ and $Poly(\Upsilon_{i,\sigma})$ as follows:

$$Poly(\Lambda_{i,\sigma}) = Poly(\Upsilon_{i,\sigma}) \boxplus Poly(\Delta_{i,\sigma})\,. \tag{34}$$

Computation over packed ciphertext. We show the process of computing all arithmetic equations in the inequality comparisons using homomorphic operations over packed ciphertexts. Moreover, the packed ciphertexts of $Poly_1(\mathbf{A}_i)$,

$Poly_3(\mathbf{I})$, $Poly_1(\mathbf{W})$, $Poly_3(\mathbf{P}_{j,\sigma})$, $Poly_4(\hat{\mathbf{A}})$, $Poly_4(\hat{\mathbf{B}})$, and $Poly_4(\hat{\mathbf{W}})$ are computed by the Eq. (20) as $ct_1(\mathbf{A}_i)$, $ct_3(\mathbf{I})$, $ct_1(\mathbf{W})$, $ct_3(\mathbf{P}_{j,\sigma})$, $ct_4(\hat{\mathbf{A}})$, $ct_4(\hat{\mathbf{B}})$), and $ct_4(\hat{\mathbf{W}})$ respectively. Now $\Delta_{i,\sigma,d,c}$ is computed from Proposition 2 and the packed ciphertexts in three homomorphic multiplications and two homomorphic additions as follows:

$$ct(\Delta_{i,\sigma}) = ct_1(\mathbf{A}_i) \boxtimes ct_3(\mathbf{I}) \boxplus ct_1(\mathbf{W}) \boxtimes ct_3(\mathbf{P}_{j,\sigma}) \boxplus (-2ct_1(\mathbf{A}_i) \boxtimes ct_3(\mathbf{P}_{j,\sigma}))\,. \quad (35)$$

According to the inner product property in Eq. (22), $ct(\Delta_{i,\sigma})$ includes many incremental Hamming distances $\Delta_{i,\sigma,d,c}$ for each block σ as the coefficients of $x^{l^2(d-1)-l(d-\theta)}$.

Now the inequality decision aider $\Upsilon_{i,\sigma,d,c}$ in Eq. (31) is computed from packed ciphertext vectors $ct_4(\hat{\mathbf{A}})$ and $ct_4(\hat{\mathbf{B}})$ in two homomorphic additions as

$$ct(\Upsilon_{i,\sigma}) = ct_4(\hat{\mathbf{A}}) \boxplus (-ct_4(\hat{\mathbf{B}})) \boxplus ct_4(\hat{\mathbf{W}})\,. \quad (36)$$

According to the packing method in Eq. (22), we will get $\Upsilon_{i,\sigma,d,c}$ as the coefficients of $x^{l^2(d-1)-l(d-\theta)}$.

Finally, $\Lambda_{i,\sigma,d,c}$ in Eq. (32) is computed from $ct(\Upsilon_{i,\sigma}) \in R_q$ and $ct(\Delta_{i,\sigma}) \in R_q$ by using one homomorphic addition over packed ciphertext which is presented as below:

$$ct(\Lambda_{i,\sigma}) = ct(\Upsilon_{i,\sigma}) \boxplus ct(\Delta_{i,\sigma})\,. \quad (37)$$

From $ct(\Lambda_{i,\sigma,d,c})$, we can get $\Lambda_{i,\sigma,d,c}$ as the coefficients of $x^{l^2(d-1)-l(d-\theta)}$. If $\Lambda_{i,\sigma,d,c} = 0$ for some $\{d, c\}$, $v_i < w_{\mathrm{J},\sigma,d}$; otherwise, $v_i \geq w_{\mathrm{J},\sigma,d}$.

6.3. Secure Computation of Our Protocols

We follow the same secure computation for the attribute matching of our all protocols. Here Alice sends the encrypted attributes from the predicate of the queries to Bob in the cloud. Bob first securely matches attributes according to Eq. (25). Then Alice processes the value appeared in the predicate of the query depending on the operators and types of query. For our conjunctive query, Bob

matches each query value v_i with the j-th column of *Record* table to find the equalities according to Eq. (28) and sends result $ct(\mathbf{H}_{i,\sigma})$ to Alice. Then Alice decrypts the results $ct(\mathbf{H}_{i,\sigma})$ and gets some IDs if $H_{i,\sigma,d}$=0 for some d of the σ-th block. In this way, Alice gets k sets of IDs for k values in the predicate of the query. After that, she performs the intersection of those sets of IDs to support conjunctive query computation and sends IDs to Bob. Later, Bob sends the corresponding encrypted records from the *Record* table depending on those IDs. Finally, Alice decrypts the encrypted records using her secret key to get her desired result. On the contrary, for evaluating our disjunctive query, Alice and Bob perform the same as required for conjunctive query except that Alice takes the union of those sets of IDs and sends those IDs to Bob.

Likewise, for threshold query, Alice gets k sets of IDs by the inequality comparison in Eq. (37) between v_i and values in the j-th attribute of the record. Now she takes the union (resp., intersection) of those IDs in case of disjunctive (resp., conjunctive) operator appeared in the predicate of the threshold query. Later on, she sends the IDs to Bob again to get the corresponding encrypted records. Finally, she decrypts the encrypted records given by Bob using her secret key.

In this way, we perform the secure computation for all of our protocols to evaluate the CDT queries.

7. Improving Protocols for Conjunctive and Disjunctive Queries

As mentioned in Section 6, we have used the binary encoding for the secure computation of the protocols for conjunctive and disjunctive queries. Recall from Section 3.2, we can use base-N encoding ($N > 2$) for the reduction of lattice dimension which can help to improve the cost of the computation. Moreover, we use the Hamming distance computation for the secure computation of equality circuit required for conjunctive and disjunctive protocols. Since the Hamming distance works only over binary vectors, we use another distance measurement technique instead of the Hamming distance. Here we can use squared

Euclidean distance (SED) in [37] for measuring the distance between two base-N encoded vectors. Furthermore, we name conjunctive or disjunctive query protocol using binary encoding (resp., base-N encoding) Method-1 (resp., Method-2). In addition, we follow the same protocols in Section 4.1 and 4.2 for calculating Method-2 except using SED instead of the Hamming distance. Let us review the SED technique used in [37] as follows.

Recall from Section 4.1, we consider two same integer vectors $v_i = (a_{i,0}, \ldots, a_{i,l-1})$ and $w_{j,\sigma,d} = (b_{j,\sigma,d,0}, \ldots, b_{j,\sigma,d,l-1})$ with $1 \leq \sigma \leq \hat{b}$ and $1 \leq d \leq \eta$, from which we form base-N encoded vectors as $\hat{\mathbf{X}}_i = (a_{i,0}, \ldots, a_{i,l_N-1}) \in \mathbb{Z}^{l_N}$ of the length l_N and $\hat{\mathbf{Y}}_{j,\sigma} = (b_{j,\sigma,1,0}, \ldots, b_{j,\sigma,1,l_N-1}, \ldots, b_{j,\sigma,\eta,0}, \ldots, b_{j,\sigma,\eta,l_N-1}) \in \mathbb{Z}^{\eta \cdot l_N}$ respectively. From these integer vectors, we calculate the squared Euclidean distance $\mathbb{E}_{i,\sigma,d}$ with the help of the arithmetic computation between $\hat{\mathbf{X}}_i$ and $\hat{\mathbf{Y}}_{j,\sigma}$ as follows:

$$\mathbb{E}_{i,\sigma,d} = \sum_{\bar{c}=0}^{l_N-1} (a_{i,\bar{c}}^2 + b_{j,\sigma,d,\bar{c}}^2 - 2a_{i,\bar{c}} \cdot b_{j,\sigma,d,\bar{c}}) \,. \tag{38}$$

We calculate $\mathbb{E}_{i,\sigma,d}$ securely the packing method in Eq. (18) and ciphertext pack in Eq. (20). Now we construct $Poly_1(\hat{\mathbf{X}}_i)$ and $Poly_1(\hat{\mathbf{X}}_i^2)$ (resp., $Poly_2(\hat{\mathbf{Y}}_{j,\sigma})$ and $Poly_2(\hat{\mathbf{Y}}_{j,\sigma}^2)$) from vector $\hat{\mathbf{X}}_i$ (resp., $\hat{\mathbf{Y}}_{j,\sigma}$) using the packing method in Eq. (18). With the help of the inner product property in Eq. (19), we compute $\mathbb{E}_{i,\sigma,d}$ in Eq. (38) as follows:

$$\begin{aligned} Poly(\mathbb{E}_{i,\sigma}) = Poly_1(\hat{\mathbf{X}}_i^2) \times Poly_2(\hat{\mathbf{I}}) + Poly_2(\hat{\mathbf{Y}}_{j,\sigma}^2) \\ \times Poly_1(\mathbf{I}_{l_N}) - 2Poly_1(\hat{\mathbf{X}}_i) \times Poly_2(\hat{\mathbf{Y}}_{j,\sigma})) \,, \end{aligned} \tag{39}$$

where $\hat{\mathbf{I}}$ denotes an integer vector like $(1, \ldots, 1)$ of the length $k \cdot l_N$ and $\mathbf{I}_{l_N}$ denotes another integer vector $(1, \ldots, 1)$ of the length l_N. Here $Poly(\mathbb{E}_{i,\sigma})$ contains many SEDs from which each $\mathbb{E}_{i,\sigma,d}$ can be obtained as the coefficient of $x^{l_N \cdot d-1}$ according to inner product property in Eq. (19). The packed ciphertexts $ct_1(\hat{\mathbf{X}}_i)$, $ct_1(\hat{\mathbf{X}}_i^2)$, $ct_2(\hat{\mathbf{Y}}_{j,\sigma})$, and $ct_2(\hat{\mathbf{Y}}_{j,\sigma}^2)$ are constructed from $Poly_1(\hat{\mathbf{X}}_i)$, $Poly_1(\hat{\mathbf{X}}_i^2)$, $Poly_2(\hat{\mathbf{Y}}_{j,\sigma})$, and $Poly_2(\hat{\mathbf{Y}}_{j,\sigma}^2))$ respectively by the Eq. (20). Moreover, we calculate $ct(\mathbb{E}_{i,\sigma})$ from Proposition 1 and the packed ciphertexts in

three homomorphic multiplications and two homomorphic additions as follows:

$$ct(\mathbb{E}_{i,\sigma}) = ct_1(\hat{\mathbf{X}}_i^2) \boxtimes ct_2(\hat{\mathbf{I}}) \boxplus ct_2(\hat{\mathbf{Y}}_{j,\sigma}^2) \boxtimes ct_1(\mathbf{I}_{l_N}) \boxplus (-2ct_1(\hat{\mathbf{X}}_i) \boxtimes ct_2(\hat{\mathbf{Y}}_{j,\sigma})) . \quad (40)$$

The encrypted polynomial $ct(\mathbb{E}_{i,\sigma})$ includes many SEDs as the coefficients of different degrees of x. Bob sends $ct(\mathbb{E}_{i,\sigma})$ to Alice for decryption. According to Proposition 1 and our protocol, Alice decrypts $ct(\mathbb{E}_{i,\sigma})$ in the ring R_q using her secret key and extracts $\mathbb{E}_{i,\sigma,d}$ as a coefficient of $x^{l_N \cdot d-1}$. Then Alice checks whether at least of one of the $\mathbb{E}_{i,\sigma,d}$ equals 0 or not to decide either $v_i = w_{j,\sigma,d}$ and $v_i \neq w_{j,\sigma,d}$ and return some IDs.

8. Performance Evaluation

In this section, we discuss the theoretical results of our protocols regarding the results in [21]. In addition, we show the practical performances of our protocols in comparison to protocols in [22]. Besides, we use the same scenario as Kim et al.'s protocol [22] along with the database and queries. For the performance comparison of every protocol, we consider their timing with the highest security level of 125-bit.

8.1. Theoretical Evaluation

First, we figure out the multiplicative depth of the circuits for homomorphic operations used in [21] and our protocols. To measure the equalities of attributes and values as discussed in Section 6, we require the Hamming distance computations in Eq. (23) and Eq. (26). For the inequalities computation, we use incremental Hamming distance in Eq. (29). Moreover, we use SED technique for improving the performance of conjunctive and disjunctive query protocols. In addition, the encrypted computation of the Hamming distances and SED required only three polynomial multiplications as in Eq. (25), Eq. (28), Eq. (35), and Eq. (40). Furthermore, Kim et al. [21] require a multiplicative depth of $\lceil \log l \rceil$ (resp., $1 + \lceil \log l \rceil$) for their equality (resp., inequality) circuit comparing two l-bit integers. On the contrary, both of our equality and inequality

circuits require only 3 (called constant-depth) due to using polynomial-based homomorphic computations.

Furthermore, Kim et al. [21] require a multiplicative depth of $\lceil \log l \rceil + \lceil \log(1+k) \rceil$ for processing conjunctive and disjunctive queries. Moreover, they require the depth of $\lceil \log l \rceil + 2\lceil \log(1+k) \rceil$ circuits for threshold query. In contrast, we require a multiplicative depth of 4 for processing the CDT queries because we use packing methods based on polynomial.

Finally, the communication cost of our protocols is $(k+\tau) \cdot l \log q$, whereas Kim et al. [21] involve a communication cost of $(\tau + 4\lambda + 1) \cdot l \log q$.

8.2. Correctness Side

We consider the correctness of our secure computation before showing the parameter settings for our experiments. As mentioned in Lemma 1, the ciphertext $ct(H_i)$ in Eq. (25) produces the correct result if it satisfies the condition $\|\langle ct(H_i), s\rangle\|_\infty < \frac{q}{2}$. Moreover, we take the upper bound Φ for the ∞-norm $\|\langle ct, s\rangle\|_\infty$ for any fresh ciphertext $ct \in (R_q)^2$. Besides, the value of the upper bound Φ can be set to $2t\omega^2\sqrt{n}$ (see theorem 3.3 in [23]). Now the ∞-norm size of $ct(H_i)$ in Eq. (25) is defined by the inequality as $\|\langle ct(H_i), s\rangle\|_\infty < 2n\Phi + 2n\Phi^2 \approx 8n^2t^2\omega^2$ (see [41] for the details). In addtion, we can get the correctness of ciphertext $ct(H_i)$ if the following equation is satisfied.

$$q > 16n^2t^2\omega^4 . \tag{41}$$

By using the same procedure as mentioned above, the correctness holds for the ciphertexts $ct(\mathbf{H}_{i,\sigma})$, $ct(\Delta_{d,i})$, $ct(\Upsilon_{d,i})$, $ct(\Lambda_{i,\sigma})$, and $ct(\mathbb{E}_{i,\sigma})$ mentioned in Section 6.3 and 7.

8.3. Parameter Settings

In our experiments, we use the same database settings as shown in [22]. We consider a database in which each record includes 11 attributes with $l = 40$ bits data. We also consider two cases of 100 and 1000 records in the *Record* table for processing the CDT query protocols. Moreover, we encode the name of each

attribute with $\delta = 8$ bits integer. Furthermore, we set the values of some other security parameters required for the SwHE in the experiments. In addition, we take the block size $\eta = 100$. We also consider appropriate values for the parameters (n, q, t, ω) of our security scheme as discussed in Section 3.5.2 for successful decryption to achieve a particular security level.

As mentioned in Section 5 of our protocols, we need the lattice dimension $n \geq \lambda \cdot \delta$ for attributes comparison and $n \geq \eta \cdot l$ for values comparison. For this reason, we set $n = 100 \cdot 40 = 4000$ for values matching. In addition, we set $n = 2048$ for attribute matching to provide better security in the computation. Since the value of ciphertext is usually large, we represent the value of q in bits rather than a decimal number. Furthermore, we set $p = 2048$ for our plaintext space R_p. According to the work in [23], the value of q we choose must be greater than 2^{60} since $16n^2t^2\omega^4 = 2^4 \cdot 2^{22} \cdot 2^{22} \cdot 2^{12} = 2^{60}$ for the ciphertext space R_q during attribute matching. As shown in Table 2, we fix our parameters as $(n, t, q, \omega) = (2048, 2048, 61 \text{ bits}, 8)$ for attribute matching. Similarly, the value of q must be greater than $16n^2t^2\omega^4 = 2^4 \cdot 2^{24} \cdot 2^{22} \cdot 2^{12} = 2^{62}$ for values matching. Then we set the security parameters $(n, t, q, \omega) = (4000, 2048, 63 \text{ bits}, 8)$ as shown in Table 1 for conjunctive and disjunctive query protocols. Similarly, we set the parameters $(n, t, q, \omega) = (16000, 2048, 67 \text{ bits}, 8)$ for processing our threshold query protocol (see Table 3). Besides, we estimate the paratmeters' values such that $(n, t, q, \omega) = (2700, 2^{19}, 79 \text{ bits}, 8)$ for conjunctive and disjunctive query protocols using base-N encoding as shown in Table 4.

8.4. Security Analysis

We propose CDT query protocols using SwHE based on ring-LWE in the semi-honest model by providing security to the attributes and values appearing in the predicate of the queries and the database, which is CPA secure even against quantum computers. Here readers may argue that why this paper does not consider malicious model instead of semi-honest model for their protocols' security. Our answer is due to getting efficiency of the protocols. Furthermore, the existing works [17, 20] support the conversion of a semi-honest model to the

malicious model using some compilers, which will degrade the efficiency of our protocols.

Moreover, NIST [2] showed different security levels for many security algorithms and their corresponding validity periods. Furthermore, they declared that a minimum strength of 112-bit level security has the security lifetime up to 2030. They also disclosed that a security algorithm including a minimum strength of 128-bit level security has a security lifetime ahead in 2030. Here, we consider the security of the encryption scheme against two attacks: distinguishing attack [25] and decoding attack [18]. As mentioned by Lindner and Peikert [18], we consider our parameter settings to assure more than 128-bit security level. The settings secure our protocols against some distinguishing attacks and more powerful decoding attacks with the advantage $\mathbb{A} = 2^{-64}$. In lattice-based cryptographic schemes, Chen and Nguyen [9] estimated that it is required to have the root Hermite factor $H < 1.0050$ to achieve a security level of 80-bit. As discussed in [23], the running time t_{adv} is defined as

$$\log(t_{adv}) = 1.8/\log(H) - 110 \tag{42}$$

where the root Hermite factor H is expressed as $c \cdot q/\sigma = 2^{2\sqrt{n. \cdot \log(q) \cdot \log(H)}}$. For our secure computation, our parameters settings provide 140-bit and 364-bit security level for attribute and value matching respectively to protect our protocols from some distinguishing attacks.

Remark 4. *Since Alice is the data owner and query owner in our protocols, leakage any information from the cloud to Alice regarding the database does not harm the security of the protocols.*

8.5. Implementation Details

In this sub-section, we discuss our experimental environment, results of CDT query protocols. Actually, Kim et al. [21] showed only theoretical performance of their protocols by providing security in the values appeared in the predicate of a query. Later on, they showed an extended experimental result in [22] by

Table 1: Performance of conjunctive and disjunctive query protocols for 40-bit data using binary encoding

τ (# of record)	k (# of conditions)	Parameters (n, t, q, ω)	Timing (seconds)			Security level	
			Kim et al. [22]	Our Protocols		Kim et al. [22]	Our protocols
				Conjunctive	Disjunctive		
100	10	(4000, 2048, 63 bits, 8)	20.4	2.948	3.058	125	364
1000	10		204	17.191	17.768	125	364

providing the security to both operators and constants appearing in the predicate of a query. Here, we experimented our three protocols and compared their performances with the results of CDT queries in [22]. To show the performance of the protocol in our all results, we use "Timing" which includes the time of query encryption, query evaluation, and result retrieval with decryption.

8.5.1. Experimental Environment

For our experiments, we implemented our protocols in C++ programming language with Pari C library (version 2.9.1) [38] and ran the programs on a single machine configured with 3.6 GHz Intel core-i5 processor and 8 GB RAM on Windows OS with Linux environment. Moreover, Kim et al.[22] used the same configuration except core-i7 processor with NTL library.

8.5.2. Database Generation

We have generated our database randomly using our program with the same settings as in [22].

Table 2: Performance of attribute matching for our all protocols

Protocol's name	Parameters (n, t, q, ω)	Data size	Timing (seconds)
Conj. & Disj.	(2000,2048,61 bits,8)	8 bits	0.608
Threshold		8 bits	0.280

8.5.3. Conjunctive and Disjunctive Query Protocols

We consider 10 equality conditions in the predicate of the conjunctive and disjunctive queries. Table 1 shows the performances of conjunctive and disjunctive query protocols compared to the performances of those protocols in [22]. For secure attribute matching, it took 0.608 sec. for both conjunctive and

disjunctive query protocols (see Table 2). For a database of 100 records (resp., 1000 records), our conjunctive query protocol took only 2.948 sec. (resp., 17.191 sec.). In addition, our disjunctive query protocol took only 3.058 sec. (resp., 17.768 sec.) for 100 records (resp., 1000 records). On the other hand, Kim et al. [22] required 20.4 sec. (resp., 204 sec.) for both conjunctive and disjunctive query processing over 100 records (resp., 1000 records). Besides, according to Eq. (42), we achieve 364-bit security level for both of our protocols, whereas Kim et al. achieved a security level of 125-bit.

8.5.4. Threshold Query Protocol

In case of threshold query, we take the same database settings as Kim et al. [22] with 3 conditions out of 11 attributes in the query. In addition, Kim et al. took the result for conjunctive case, whereas we measured both conjunctive and disjunctive case but compare only conjunctive case as shown in Table 3. Our protocol did the secure computation for both attribute and values. For attribute matching, it took 0.280 sec. for matching 3 attributes of the query out of 11 attributes (see Table 2). In this case, our protocol took 17.11 seconds (resp., 143.692 seconds), whereas Kim et al. took 20 seconds (resp., 200 seconds) for processing a 3-out-of-11 threshold query over 100 records (resp., 1000 records). Here it is obvious that our threshold query protocol shows more efficient performance when the record size is large. Furthermore, we achieved a more efficient performance along with the security level for both 100 and 1000 records. In addition, our threshold query protocol took 17.23 sec. (resp., 144.362 sec.) for processing 100 records (resp., 1000 records) if the disjunction condition exists in the predicate of that query.

9. Comparative Performance Analysis

In this section, we show the comparative performance analysis between conjunctive (resp., disjunctive) query protocol using binary encoding called Method-1 and conjunctive (resp., disjunctive) query protocol using base-N en-

Table 3: Performance of threshold query protocol for 40-bit data using binary encoding

τ (# of records)	k (# of conditions)	Parameters (n, t, q, ω)	Timing (seconds)			Security level	
			Kim et al. [22] (Conjunctive)	Our protocols		Kim et al. [22]	Our protocols
				Conjunctive	Disjunctive		
100	3	(16000, 2048, 67 bits, 8)	20	17.11	17.23	125	1616
1000	3		200	143.692	144.362	125	1616

coding called Method-2. We show this analysis regarding total computation time, ciphertext size, and security level as shown in Table 4.

9.1. Computational Time

The details of computation time required by Method-1 and Method-2 are shown in Table 4. Though our Method-1 performs more efficiently than the method in [22] as shown in Table 1, we use Method-2 to get more efficiency than Method-1 by employing the base-N encoding for lattice reduction. For our conjunctive query, Method-2 was able to process 100 records (resp.,1000 records) within 1.678 sec. (resp., 3.666 sec.), whereas Method-1 required 2.948 sec. (resp., 17.191 sec.) to process the same number of records. Additionally, for processing our disjunctive query of 100 records (resp.,1000 records), Method-2 took 1.872 sec. (resp., 3.294 sec.), whereas Method-1 took 3.058 sec. (resp., 17.768 sec.) to evaluate the same query processing with the same number of records. In agreement with Eq. (42), we achieved a security level of 143-bit for Method-2 while Method-1 achieved 364-bit level. Though the security level of Method-2 is less than that of Method-1, security lifetime of Method-2 still ahead in 2030 as mentioned in Section 8.4.

9.2. Ciphertext Size

Through base-N encoding, we have been able to reduce the packed ciphertext size in the database as shown in Table 4. Here the size of one packed ciphertext is $2n \cdot \log q$ bits. For 100 records (resp.,1000 records), Method-1 required 61.52 kilobytes (resp., 615.2 kilobytes) ciphertext, whereas Method-2 required 52.08 kilobytes (resp., 520.8 kilobytes) of ciphertext.

10. Conclusion

In this paper, we have shown three efficient protocols for processing private database queries over the encrypted outsourced database in which privacy is preserved using ring-LWE based SwHE in the semi-honest model. In addition, our experiments proved that our protocols achieved a remarkably more efficient

Table 4: Comparative performance between Method-1 and Method-2

τ (# of record)	k (# of cond.)	Parameters for base-N encoding (n, t, q, ω)	Timing (seconds)				Security level		Ciphertext size	
			Method-1		Method-2		Method-1	Method-2	Method-1	Method-2
			Conj.	Disj.	Conj.	Disj.				
100	10	(2700, 2^{19}, 79 bits, 8)	2.948	3.058	1.678	1.872	364	143	61.52 kB	52.08 kB
1000	10		17.191	17.768	3.666	3.294	364	143	615.2 kB	520.8 kB

Method-1: Conjunctive or disjunctive query protocol using binary encoding

Method-2: Conjunctive or disjunctive query protocol using base-N encoding

level for the conjunctive and disjunctive case than Kim et al. [22] with a better security level. Besides, we achieved a performance improvement in case 3-out-of-11 threshold query protocol than that in [22] with a better security level. In processing conjunctive and disjunctive queries, we attain further efficiency by employing base-N encoding based protocols as compared to binary encoding based protocols regarding timing and ciphertext size. We also believe that our protocols are practically usable. Furthermore, we have achieved the efficiency due to using low-cost equality circuits and batch technique along with the packing methods. Moreover, our protocols can support a larger data size for both query and database by increasing the lattice dimension n. Readers might also understand that our protocol can efficiently handle databases with a large number of records by dividing the database into blocks as we did for 1000 records case. Besides, we believe that the batch technique with packing methods will be used to process many new queries (e.g., insert, update, and delete) in future.

Acknowledgments.

This work is supported in part by JSPS Grant-in-Aids for Scientific Research (A) JP16H01705 and for Scientific Research (B) JP17H01695.

References

[1] M. Ajtai, "Generating hard instances of lattice problems." In Proc. 28th annu. ACM symp. on Theory of computing, ACM, 1996, pp. 99–108.

[2] E. Barker, "Recommendation for key management", NIST Special Publication 800-57 Part 1 Rev. 4, NIST, 2016.

[3] A. Boldyreva, N. Chenette, Y. Lee, and A. O' Neill, "Order-preserving symmetric encryption", in Joux A. (eds) Advances in Cryptology - EUROCRYPT 2009. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2009, vol. 5479. pp. 224–241.

[4] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," in Proc. ACNS-2013,

Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2013, vol. 7954. pp. 102–118.

[5] Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) Theory of Cryptography. TCC 2005. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2005, vol. 3378. pp. 325–341.

[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in Proc. 3rd ITCS, 2012, pp. 309–325.

[7] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in CRYPTO, Lecture Notes in Computer Science, Springer, Verlag, 2011, vol. 6841, pp. 505–524.

[8] W. Castryck, I. Iliashenko, and F. Vercauteren, "Provably weak instances of ring-LWE revisited," in Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2016, pp. 147–167.

[9] Y. Chen, P. Q. Nguyen, "BKZ 2.0: Better lattice security estimates," in Advances in Cryptology — ASIACRYPT 2011, LNCS, vol. 7073, Springer, 2011, pp. 1–20.

[10] J. H. Cheon, M. Kim, and M. Kim, "Search-and-compute on encrypted data," in Financial Cryptography Data Security (Lecture Notes in Computer Science), vol. 8976, M. Brenner, N. Christin, B. Johnson, and K. Rohloff, Eds. Berlin, Germany: Springer-Verlag, 2015, pp. 142–159.

[11] J. H. Cheon, M. Kim, and M. Kim, "Optimized search-and-compute circuits and their application to query evaluation on encrypted data," IEEE Trans. on Information Forensics and Security, vol. 11, no. 1, pp. 188–199, 2016.

[12] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in Advances in Cryptology (Lecture Notes in Computer

Science), vol. 196, G. R. Blakley and D. Chaum, Eds. Berlin, Germany: Springer-Verlag, 1984, pp. 10–18.

[13] B. Fisch, V. Kolesnikov, T. Malkin, F. Krell, A. Kumarasubramanian, B. Vo, and S. M. Bellovin, "Malicious-client security in Blind Seer: A scalable private DBMS", in 36th IEEE Symposium on Security and Privacy, IEEE 2015, pp. 395–410.

[14] C. Gentry, "Fully homomorphic encryption using ideal lattices," in Proc. of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31-June 2, 2009, pp. 169–178.

[15] O. Goldreich. Foundations of Cryptography: Volume I Basic Tools. Cambridge University Press, 2001.

[16] S. Goldwasser and S. Micali, "Probabilistic encryption & how to play mental poker keeping secret all partial information," in Proc. ACM Symp. Theory Comput., San Francisco, CA, USA, 1982, pp. 365–377.

[17] O. Goldreich, S. Micali, A. Wigderson, "How to play any mental game," in Proc. 19th annual ACM symposium on Theory of computing, ACM, 1987, pp. 218–229.

[18] R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption," in Topics in Cryptology — CT-RSA 2011, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2011, vol. 6558. pp. 319–339.

[19] Y. Hu, "Improving the efficiency of homomorphic encryption schemes," PhD diss., May 2013, Worcester Polytechnic Institute, Massachusetts.

[20] Y. Ishai, M. Prabhakaran, A. Sahai, "Founding cryptography on oblivious transfer – efficiently," In: D. Wagner (ed.) Advances in Cryptology – CRYPTO 2008. Lecture Notes in Computer Sciences, vol. 5157. Berlin, Germany: Springer-Verlag, 2008, pp. 572–591.

[21] M. Kim, H. T. Lee, S. Ling, and H. Wang, "On the efficiency of FHE-based private queries," IEEE Trans. on Dependable and Secure Comput. vol. 15(2), 2018. pp. 357–363.

[22] M. Kim, H. T. Lee, S. Ling, S. Q. Ren, B. H. M. Tan, H. Wang, "Better security for queries on encrypted databases," IACR Cryptology ePrint Archive, 2016/470, 2016.

[23] K. Lauter, M. Naehrig, and V. Vaikuntanathan, "Can homomorphic encryption be practical?," in Proc. of ACM Cloud Computing Security Workshop (CCSW), ACM Press, 2011, pp. 113–124.

[24] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," In: H. Gilbert (ed.) Proc. of Advances in Cryptology, Lecture Notes in Computer Science, vol. 6110, Berlin, Germany: Springer-Verlag, 2010, pp 1–23.

[25] D. Micciancio, and O. Regev, "Lattice-based cryptography," in Post-Quantum Cryptography, D.J. Bernstein, J. Buchmann, and E. Dahmen, Eds. Berlin, Germany: Springer-Verlag, 2009, pp. 147–191

[26] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in Advances in Cryptology — EUROCRYPT'99 (Lecture Notes in Computer Science), vol. 1592, H. Gilbert, Ed. Berlin, Germany: Springer-Verlag, 1999, pp. 223–238.

[27] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin, "Blind Seer: A Searchable Private DBMS," in Proc. of IEEE S&P'14, 2014.

[28] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. "CryptDB: Protecting confidentiality with encrypted query processing". In ACM SOSP, 2011, pp. 85–100.

[29] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography". In Symposium on Theory of Computing — STOC 2005, H.N. Gabow, R. Fagin, Eds. ACM, 2005, pp. 84–93.

[30] O. Regev, "The learning with errors problem (invited survey)". In IEEE Conference on Computational Complexity, IEEE Computer Society (2010), pp. 191–204.

[31] R. L. Rivest, L. Adleman, M. L. Dertouzos, "On data banks and privacy homomorphisms," Foundations of Secure Computation, Academia Press, 1978. pp. 169–179

[32] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems," Commun. ACM, vol. 21, no. 2, pp. 120–126, February 1978.

[33] T.K. Saha, Mayank, and T. Koshiba, "Efficient protocols for private database queries," in Data and Applications Security and Privacy XXXI. DBSec 2017 (Lecture Notes in Computer Science), vol. 10359. Springer, Cham 2017, pp. 337–348.

[34] T. K. Saha and T. Koshiba, "Private equality test using ring-LWE somewhat homomorphic encryption," in 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWConCSE), IEEE 2016, pp. 1–9.

[35] T. K. Saha and T. Koshiba, "Private conjunctive query over encrypted data. in Progress in Cryptology — AFRICACRYPT 2017 (Lecture Notes in Computer Science), vol. 10239, M. Joye and A. Nitaj, Eds. Berlin, Germany: Springer-Verlag, 2017, pp. 149–164.

[36] T. K. Saha and T. Koshiba, "An efficient privacy-preserving comparison protocol," in Advances in Network-Based Information Systems —NBiS 2017 (Lecture Notes on Data Engineering and Communications Technologies) vol. 7, L. Barolli, T. Enokido, and M. Takizawa, Eds. Springer-Cham, 2018, pp. 553–565.

[37] T. K. Saha and T. Koshiba, "Privacy-preserving Equality Test towards Big Data," in International Symposium on Foundations and Practice of Security—FPS 2016, Lecture Notes in Computer Science, vol. 10723. Springer, 2017, pp. 95–110.

[38] The PARI∼Group, PARI/GP version `2.7.5`, Bordeaux, 2014, `http://pari.math.u-bordeaux.fr/`

[39] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," in Proc. of the VLDB Endowment, vol. 6, 2013, pp. 289–300.

[40] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Secure pattern matching using somewhat homomorphic encryption," in Proc. of the 2014 ACM CCSW 2013. ACM, 2013, pp. 65–76.

[41] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Practical packing method in somewhat homomorphic encryption," in *Proc. Data Privacy Management*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2013, vol. 8247. pp. 34–50.

[42] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "New packing method in somewhat homomorphic encryption and its applications," Security and Communication Networks, vol. 8(13), pp. 2194–2213, 2015.