Private Comparison Protocol and Its Application to Range Queries

No Author Given

No Institute Given

Abstract. We review the problem of private comparison protocol and its application to private range queries for accessing a private database. Very recently, Saha and Koshiba (NBiS 2017) proposed an efficient privacy-preserving comparison protocol using ring-LWE based somewhat homomorphic encryption (SwHE) in the semi-honest model. The protocol took 124 milliseconds (resp., 125 milliseconds) for comparing two practical integers of 16-bit (resp., 32-bit). But this protocol is not efficient enough to process a range query to a large database where several thousand comparisons are required. In this paper, we propose an efficient private comparison protocol and show its application to private range queries. Here the security of the protocol is also ensured by ring-LWE based SwHE in the same semi-honest model. Our practical experiments show that our comparison protocol enables us to do a single comparison in 84 milliseconds (resp., 85 milliseconds) for 16-bit (resp., 32-bit) integers which are more efficient than Saha et al.'s protocol. Besides, it takes about 0.499 seconds (resp., 2.247 seconds) to process a 3-out-of-11 range query in a database of 100 records (resp., 1000 records) including 11 attributes which outperform state of the art.

Keywords: Comparison protocol, Range query, Somewhat homomorphic encryption, Batch technique

1 Introduction

Since the publication of Yao's two millionaires' problem in 1982 [22], the private comparison protocol plays a vital role in large cryptographic problems in data mining, machine learning, private database queries, and so on. Furthermore, private access to database records is now very important for maintaining the privacy of its users and their queries. In addition, a private range query is crucial when the users need to find some records within a certain range for some attributes in a database. For example, a health researcher of a research organization wants to count the number of middle aged ($31 \sim 50$ years) patients with high sugar level admitted in a hospital. Here neither the health researcher can reveal his query to the hospital nor the hospital can reveal its database records to the research organization. In this case, a third party like the cloud can help in this computation. But it is hard to find such a fully trusted cloud service provider.

On the contrary, cloud computation is being popular gradually all over the world since its evolution over the Internet. People are using the Internet not only for sending mail and documents and searching required information over the Internet but also outsourcing their data in the cloud. Besides, cloud service providers are now giving several services like IaaS, PaaS, and SaaS to their customers with a low cost [15]. Now business and research organizations, institutes, and IT companies are interested to store their data in the cloud. Here they want to secure their data by applying some encryptions. At the same time, they like to access data by engaging some queries and apply some statistics, machine learning, and data mining algorithms on encrypted outsourced data. Therefore, there should have some approaches which allow computation on encrypted data.

In this respect, homomorphic encryption (HE) can be a solution to the above problem which allows both addition and multiplication over encrypted data. Homomorphic encryption has come forward after the comprehensive work of Gentry in 2009 [8]. Moreover, there exists three types of homomorphic encryption namely: partial homomorphic, fully homomorphic, and somewhat homomorphic encryption. Among these types, somewhat homomorphic encryption (SwHE) is more acceptable due to supporting any number of additions and few multiplications with a comparatively high speed than others. In this paper, we use the SwHE scheme of Lauter et al. [13] which is a variant of SwHE scheme proposed by Brakerski and Vaikunthanathan [4]. In addition, from the above-mentioned applications, we concentrate on the problem of private access to database by employing range queries [2, 11, 12, 21]. For processing a range query, many comparisons are required between a given value in the predicate of an attribute and each value of the database records for the corresponding attribute. Here, an efficient private comparison protocol with SwHE is indispensable to evaluate the problem of private range queries for accessing records from a table in a large database.

1.1 Related Work

In this section, we review some recent problems related to private comparison protocol using homomorphic encryption. In 2008, Damgård et al. [7] proposed an efficient comparison protocol for on-line auction utilizing an additively homomorphic encryption scheme in the semi-honest model. But it took about 280 milliseconds for only one comparison between two 16-bit integers running on 6 otherwise idle machines including dual processors. Recently, Saha and Koshiba [19] improved their technique by proposing another efficient privacy-preserving comparison protocol engaging ring-LWE based somewhat homomorphic encryption along with some packing methods. But their protocol is not practical enough to address range queries where several thousand comparisons are required to access a large database. Here we also review the problems of private range queries for accessing a secure database. In 2013, Boneh et al. [2] proposed a technique of processing range queries only by applying SwHE scheme along with conjunctive, disjunctive, and update queries. But their performance of query processing

is time-consuming in a practical sense. In 2016, Kim et al. [12] engaged BGV's scheme in [3] to describe another protocol for processing conjunctive, disjunctive, and range queries over encrypted data in the cloud. They showed the implementation of their protocols in another paper [11] and it took about 160 milliseconds to access each record including 11 attributes of 40-bit values to achieve a security level of 125-bit. Furthermore, they required a multiplicative depth of $\lceil \log l \rceil + 2\lceil \log(1+\rho) \rceil$ for their range query on l-bit message with ρ attributes. Later on, to provide security to both attributes and data in the predicate of a query, Kim et al. [11] also showed another method for processing these same types of queries. By keeping the same database settings and security level, it took about 200 milliseconds per record to reply a threshold query with three conditions. Here they used a high depth multiplication circuit for computing the comparison for range queries. Very recently, Xue et al. [21] presented a two-cloud architecture for a secure access to database with Paillier cryptosystem [14] that provides privacy preservation to various numeric-related range queries. But their model stores the private key to one of the clouds. But it is difficult to find such a trusted cloud.

1.2 Motivation

As discussed in the previous section, existing comparison protocols are not efficient enough for the computation of a range query where many comparisons are required to process that query in a large database. We also observed that the cost of range query computation increases due to high depth multiplication circuit. The main motivation behind this work, if we would have a low depth circuit for comparison computation then we can reduce the computation cost. The another motivation is the vast applications of private comparison protocol including e-commerce [7], data mining [10], machine learning [9], database query processing [6, 11, 12, 21], and so on. Besides, outsourced private computation is now popular due to providing security to the queries and data of the individuals or some organizations.

1.3 Our Contribution

Our contribution in this paper is twofold.

- Firstly, Saha et al. [19] proposed an efficient privacy-preserving comparison protocol which took 124 milliseconds (resp., 125 milliseconds) for comparing two practical integers of 16-bit (resp., 32-bit). But the protocol is not efficient enough to address a range query to the large database where several thousand comparisons are required to be performed to process that query. Here they used binary encoding for comparing two integers. Therefore, we propose an efficient private comparison protocol by employing a base-β encoding which outperforms Saha et al.'s protocol of integer comparison. In addition, our practical implementation shows that our protocol consumes 84

ms. (milliseconds) (resp., 85 ms.) for a single comparison of 16-bit (resp., 32-bit) integer.

– Finally, Kim et al. [11] presented a protocol for processing conjunctive, disjunctive, and range queries over encrypted data in the cloud. But it took about 200 seconds to process a range query from a database 1000 records of 11 attributes with 40-bit values. Here we extend the comparison protocol to compute private range queries over encrypted data in the cloud where many comparisons are required in a single computation. To do this, we use the data packing method in [16] to support batch computation of the comparisons requited for range queries. Besides, it took 0.499 seconds (resp., 2.247 seconds) for a range query with three conditions in its predicate using the same database settings.

For the above two cases, we achieve a security level of at least 140-bit.

2 Preliminaries

In this section, we describe some basic notations and terms that we use for the rest of discussion of the paper.

2.1 Notations

R be a polynomial ring such that $R = \mathbb{Z}/(x^n + 1)$ where $\mathbb{Z}$ denotes the ring of integers and n denote the lattice dimension for the ring-LWE based SwHE scheme in [17]. In addition, $\mathbb{Z}^\theta$ defines a θ-dimensional integer vector space. For a prime number q the field of integer modulo q is denoted by $\mathbb{Z}_q$ called ciphertext space. Similarly, t is the largest prime less than $2^{\lceil \log_2 t \rceil}$ which defines plaintext space R_t. Furthermore, δ define standard deviation used to define discrete Gaussian error distribution in [17]. Moreover, the function $Enc_{pk}(m) = ct(m)$ defines the encryption of message m using the public key pk to produce the ciphertext $ct(m)$. Here $k \in \mathbb{Z}$ represents the block size of the comparison for the $(1, k)$-PET and (t, k)-PET protocols. Also, t represents the number of queries in the (t, k)-PET protocol. For the database, τ and α denote a total number of records and attributes respectively. Besides, k denotes the number of conditions in the predicate of a range query and σ denotes the batch size for a large set of records.

2.2 Private Comparison

The private comparison is a technique of comparing two numbers securely when their owners do not want to disclose their information to each other. Here the comparison result may be required by the data owners or any other third party who has interest in this computation. For example, a financial magazine wants to rank between two rich persons who do not want to disclose their assets either to one another or to the magazine company.

2.3 Private Range Queries

A threshold query is a query in which the predicate contains an attribute greater than or less than some threshold value. For instance, find those patients from a hospital database whose fasting plasma glucose $\geq$ 7.0 millimole/liter (126 milligram/deciliter). In addition, a range query in a database means a query containing predicate consisting comparison condition within some ranges ($<, >$, $\leq, \geq$). For example, find the patients in a hospital whose ages are between 31 to 50. Here we can say that range query is a conjunctive case of threshold query. Sometimes, range query and threshold query are used synonymously in database query processing. In this paper, we consider the private range queries to access a private database. Consider a patient table of a hospital database contains τ records with α attributes. Since a range query requires to compare the value of an attribute in the predicate with all the values appeared in the corresponding attribute of the patient table, many comparisons are needed here. Here, we can engage private comparison protocol for the range query computations.

2.4 Base-β Data Encoding

Data encoding is a technique of representing data (letters, numbers, punctuation, and certain symbols) into a specialized format (such as binary, decimal, octal, and so on) for efficient transmission or storage. This stored data can be processed further for getting some results. Moreover, the encoded data can be decoded (reverse of encoding) to get its original form. Here base-β ($\beta \in \mathbb{Z}$) encoding is kind of special encoding which represents the data using the alphabets $\{0, 1, 2, \ldots, \beta-1\}^d$ where d is the length of a number in the base-β form. Generally, data are represented in binary form ($\{0, 1\}^l$) for most of the digital storages. Saha et al. [19] used binary encoding to represent an l-bit number that we call base-2 encoding where alphabet set is $\{0, (2-1)\}^l = \{0, 1\}^l$. For representing a decimal number z with a binary encoding, it requires $l = \lfloor \log_\beta(z) \rfloor + 1$ digits where $\beta = 2$ for binary case. In addition, Saha et al. [19] used an l-bit binary conversion algorithm for any integer of $l = 16 \sim 32$-bit used in their comparison protocol. If we use base-β encoding rather than binary encoding, we can achieve vector size reduction of $d = \lceil l/\log_2(\beta) \rceil$ where l is the length of a number in binary form. For example, if we convert the decimal number $(248)_{10}$ into a binary vector then it can be represented as $(1, 1, 1, 1, 1, 0, 0, 0)_2$. But if we convert the same number to base-8 vector as $(3, 7, 0)_8$. In this paper, we use base-β encoding ($\beta > 2$) instead of binary encoding to reduce the vector size for handling range queries efficiently.

3 Our Protocol

Here we elaborate our private comparison protocol and shows its application to private range queries to a large database.

3.1 Private Comparison Protocol

To describe the private comparison protocol, let us consider two billionaires ranking problem which requires 4 parties namely: Alice, Bob, Charlie, and Dev. Only two parties (Alice and Dev) in our setting are needed to be online during the protocol execution. Consider that Alice is financial magazine owner who wants to rank between two billionaires named Bob and Charlie. Here Bob and Charlie do not want to disclose their wealth either to each other or to Alice. To do this rank, we apply a private comparison protocol. To facilitate the private comparison, we need an unbiased intermediary party like Dev in the cloud to perform all the computations needed to decide the comparison result. Here consider that the amount of wealth belonging to Bob and Charlie can be represented by the l-bit integers a and b which can be converted to base-β integer vectors as $a = (a_0, \ldots, a_{d-1})$ and $b = (b_0, \ldots, b_{d-1})$ such that a_i or b_i is the i-th digit in the base-β representation with $0 \leq i \leq d-1$. Now the computation of comparison between two vectors a and b can be realized by following arithmetic equation.

$$c = 2(a - b) \bmod t \tag{1}$$

where t is the largest prime less than $2^{\lceil \log_2 t \rceil}$. Here if the parity of first non-zero element of c is 1 then $a > b$; otherwise $a < b$. Besides, Alice and Dev also add some random masks to hide actual result from each other for security. Assume that Dev is an honest-but-curious party. Firstly, Alice generates a random number $r \in \mathbb{Z}_t^*$ and sends it to Dev in the cloud. She also generates a valid public key and secret key pair (pk, sk) and sends the public key pk to Bob, Charlie, and Dev via a secure channel. Here Bob encrypts his vector a using the public key pk and sends $Enc_{pk}(a)$ to Dev. In addition, Charlie encrypts his vector b by employing the same public key pk and sends $Enc_{pk}(b)$ to Dev. Now Bob and Charlie go offline for the rest of the protocol operating among Alice and Dev as follows.

1. First, Dev computes $2 \cdot \big(Enc_{pk}(a) - Enc_{pk}(b)\big)$ and a homomorphic multiplication $Enc_{pk}(r) \otimes 2 \cdot Enc_{pk}(a - b)$ is then performed.
2. To obfuscate the result from Alice, Dev generates a random integer vector $r_1 \in \mathbb{Z}_t^d$ and homomorphically adds this to $Enc_{pk}(2 \cdot r \cdot (a - b))$ to get $Enc_{pk}(2 \cdot r \cdot (a - b) + r_1)$, which is then sent to Alice for decryption.
3. Alice decrypts $Enc_{pk}(2 \cdot r \cdot (a - b) + r_1)$ to get $2 \cdot r \cdot (a - b) + r_1$ using her secret key sk and sends the decrypted result back to Dev.
4. Then Dev produces $2 \cdot r \cdot (a - b)$ by subtracting r_1. Since the result of $(a - b)$ is obfuscated here, Dev is unaware of the result of the computation.
5. Now Dev traverses through the vector and selects the first non-zero value ϕ found at position p with $\phi = 2 \cdot r \cdot (a_p - b_p)$ where a_p (resp., b_p) denotes the value at the p-th position in the vector a (resp., b).
6. Then Dev sends only $\phi \cdot r_2$ to Alice where r_2 is a random mask in $\mathbb{Z}^*_{\lfloor t/2\beta \rfloor}$. Here the random mask is multiplied to obfuscate the main result from Alice.
7. After receiving $\phi \cdot r_2$ from Dev, Alice computes $\gamma = \phi \cdot r_2 \cdot r^{-1}$ in the group $\mathbb{Z}_t^*$ (since t is prime, all the elements of $\mathbb{Z}_t - \{0\}$ are invertible).
8. Finally, Alice publishes the comparison result by checking the parity of γ. A parity bit of 1 implies that $a < b$; otherwise $a > b$.

Remark 1. Here we achieve a passive security for our protocol under the assumption that Dev is semi-honest. In other words, Bob follows the protocol but tries to learn information from the protocol. Here we use the same ring-LWE based SwHE scheme used in Saha et al [17] for the security of our protocol. Here we skip its review due to page limitation.

3.2 Use of Comparison Protocol to Private Range Queries

As discussed in Section 2.3, we can say that private comparison protocol can be employed to private range queries computation. Our comparison protocol can be used for only one comparison of two integers whereas a private range query requires many comparisons. To illustrate, consider the patient table in a hospital database containing τ records with α attributes. Here, a health researcher of a research organization wants to count the number of middle aged ($31 \sim 50$ years) patients admitted with type-2 diabetics from a hospital database. We also consider these computations among 4 parties where database expert (Bob) is sending a query to the cloud (Dev) on behalf of the health researcher (Alice). Here neither the Bob wants to reveal his query to the hospital (Charlie) nor Charlie wants to reveal its information to Bob. In addition, the corresponding SQL statement can be written as *select count(PID) from patient where age* > 30 *and age* < 51 *and fastingPlasmaGlucose* > 7. Here the predicate of the query contains $k = 3$ conditional statements. Now Alice can process this query by taking intersection three separate threshold queries with one condition as *select PID from patient where age* $> 30 \wedge$ *select PID from patient age* $< 51 \wedge$ *select PID from patient fastingPlasmaGlucose* > 7. Finally, she counts the number PIDs appeared in the intersection result. To evaluate the query, τ comparisons are required between the given value of an attribute in the predicate and each value of the records of the corresponding attribute. Here if we use our comparison protocol to process this query then it is required to run comparison protocol a total of 3τ times between Alice and Dev which is time-consuming. To process this range query efficiently, some other techniques are indispensable.

3.3 Batch Technique

The batch technique is a process of executing a single instruction on multiple data. That means, the batch technique allows us to perform a single instruction multiple data (SIMD) type operations on data. To process the range query mentioned in the previous section along with the same database settings, we need 3τ comparisons. Here if we use comparison protocol in Section 3.1, we will be required to run the protocol 3τ times which is time-consuming along with communication cost. Can we do the range query computation in an efficient way other than this? In 2016, Saha et al. [16] used the batch technique for the private batch equality test (PriBET) protocol to compare a single integer with a set of integers for finding equalities. Here we can also use the same batch technique for processing private range queries where many comparisons are needed to perform for getting the query results from the private database. To apply this

batch technique in our computation, we need to use some packing methods as in [13, 16, 17, 23].

3.4 Packing Methods

Packing method refers to the process of encoding many messages in a single polynomial. In [13], Lauter et al. used a packing method to efficiently encode the binary representation of an integer within a polynomial to support efficient arithmetic operations. Now we review Lauter et al.'s packing method for the arithmetic computation of our comparison protocol described in Section 3.1.

To describe the packing method, let a be an l-bit integer which can be presented by the base-β integer vectors as $\mathbf{A} = (a_0, a_1, \ldots, a_d) \in R_t$ of size $d = \lceil l/\log_2 \beta \rceil$. Here a_i represents the i-th digit of integer a in its base-β representation. By following the packing method in [13] for $d \leq n$, we encode the integer vector $\mathbf{A}$ in the base ring $R = \mathbb{Z}[x]/(x^n + 1)$ by the packing following method.

$$Poly_1(\mathbf{A}) = \sum_{i=0}^{d-1} a_i x^i \in R_t \tag{2}$$

Furthermore, we use polynomial ring-LWE based SwHE for our protocol's security where every computation is performed within lattice dimension n. From table I in [16], we observed that computation cost of our used encryption scheme mostly depends on lattice dimension n. Our private comparison protocol also requires a lattice dimension of n to compare two integers of length d in the base-β representation. For practical case, we need to set $n = 2048$ at least to get a security level of 140-bit [16]. On the other hand, to represent a practical integer of 40-bit, it requires a vector size of $d = 10$ in the base-16 representation. Here, d is very small as compared to n for a single comparison. Now there exist many unused spaces (for example, $n - d = 2048 - 10 = 2038$ in this case) for a single comparison. But our range query computation requires many comparisons. If we can encode many integers within this lattice dimension for many comparisons' computation at a time, then we will be able to reduce the computation cost. Now we encode $\sigma = \lfloor n/d \rfloor$ integers within the lattice dimension n to support batch computation of many comparisons. Specifically, we encode σ integers in a single polynomial. Let $\{c_1, \ldots, c_\sigma\}$ be a set of σ integers of l-bits. We generate another base-β integer vector $\mathbf{C} = (c_{1,0}, \ldots, c_{1,d-1}, \ldots, c_{\sigma,0}, \ldots, c_{\sigma,d-1}) \in R_t$ of length $d \cdot \sigma$ where $b_{i,j}$ represents the j-th digit of integer c_i in its base-β representation. For $d \cdot \sigma \leq n$, the vector $\mathbf{C}$ is encoded in the same base ring $R = \mathbb{Z}[x]/(x^n + 1)$ according to packing method in Saha et al. [16] as follows.

$$Poly_2(\mathbf{C}) = \sum_{i=1}^{\sigma} \sum_{j=0}^{d-1} b_{i,j} x^{(i-1)\cdot d + j} \in R_t \tag{3}$$

According to the SwHE in Section 2 of [17], the packed ciphertexts for $Poly_i(\mathbf{P}) \in$

R, for instance, $\mathbf{P}$ can be replaced by $\mathbf{A}$ and $\mathbf{C}$ as in Eq. (2) and Eq. (3) respectively, are defined for some $i = \{1, 2\}$ using the public key pk as

$$ct_i(\mathbf{P}) = Enc_{pk}(Poly_i(\mathbf{P})) \in (R_q)^2 . \tag{4}$$

In addition, the following proposition is needed to hold for the multiplication required for the comparison circuit.

Proposition 1. *Let $\mathbf{A} = (a_0, a_1, \ldots, a_{l-1}) \in R_t$ be an integer vector and $\mathbf{M} = (r, 0, \ldots, 0) \in R_t$ be another random integer vector where $|\mathbf{A}| = |\mathbf{M}| = d$. If the ciphertext of $\mathbf{A}$ and $\mathbf{M}$ can be represented by $ct_1(\mathbf{A})$ and $ct_1(\mathbf{M})$ respectively by Eq. (4) then under the condition of Lemma 1 (see Section 2.3 in [17] for details), the decryption of homomorphic multiplication $ct_1(\mathbf{A}) \otimes ct_1(\mathbf{M}) \in (R_q)^2$ will produce a polynomial of R_t without any change in the polynomial degree.*

Proposition 2. *Let $\mathbf{C} = (c_{1,0}, \ldots, c_{1,d-1}, \ldots, c_{\sigma,0}, \ldots, c_{\sigma,d-1}) \in R_t$ be an integer vector of size $d \cdot \sigma$ produced from set of σ integers $\{c_1, \ldots, c_\sigma\}$. In addition, $\mathbf{N} = (r, 0, \ldots, 0)$ be another random integer vector where $|\mathbf{N}| = d \cdot \sigma$. Whenever the ciphertext of $\mathbf{B}$ and $\mathbf{N}$ is represented by $ct_2(\mathbf{B})$ and $ct_2(\mathbf{N})$ respectively by Eq. (4), under the condition of Lemma 1 (see Section 2.3 in [17] for details), decryption of homomorphic multiplication $ct_2(\mathbf{B}) \otimes ct_2(\mathbf{N}) \in (R_q)^2$ will produce a polynomial of R_t without any change in the polynomial degree.*

4 Secure Computations

In this section, we talk about the homomorphic operations needed for our comparison protocol along with its extension to compute private range queries.

4.1 Private Comparison Protocol

The homomorphic computation required in our private comparison protocol can be done efficiently by applying our packing method along with the encryption scheme. Let us consider the same base-β integer vectors a and b as $\mathbf{A}$ and $\mathbf{B}$ respectively for the private comparison. As discussed the comparison protocol in Section 3.1, Dev homomorphically computes the arithmetic in Eq. (1) by employing the packing method in Eq. (2) as $ct_1(c) = 2 \cdot (ct_1(\mathbf{A}) \oplus (-ct_1(\mathbf{B})))$. According to Proposition 1, Dev also homomorphically multiplies a random mask $r \in \mathbb{Z}_t$ given by Alice as random integer vector $\mathbf{M} = (r, 0, \ldots, 0) \in R_t$ to $ct_1(c)$ to get $ct_1(c') = ct_1(c) \otimes ct_1(\mathbf{M})$. Besides, Dev generates another random mask $r_1 \in \mathbb{Z}_t^d$ that can be represented as another integer vector $\mathbf{M'} = (r_{1,0}, \ldots, r_{1,d-1}) \in R_t$ and adds the vector to $ct_1(c')$ to produce $ct_1(c'') = ct_1(c') \oplus ct_1(\mathbf{M'})$.

4.2 Private Range Queries Computation

Since we apply our comparison protocol using the batch technique for efficient computation of the private range queries, we need similar secure computations

as discussed in the previous section. Here we employ the packing method in Eq. (3) along with proposition 2. Let us consider an integer $a \in \mathbb{Z}_t^d$ of length d that is needed to compare with a set of τ integers $\{c_1, \cdots, c_\tau\}$ of the same length. Since we compute within the lattice dimension n, so we can process at best σ data at a time from the database records for batch comparison. Now we make a batch vector $\mathbf{C} = (c_{1,0}, \ldots, c_{1,d-1}, \ldots, c_{\sigma,0}, \ldots, c_{\sigma,d-1}) \in R_t$ of length $d \cdot \sigma$. By employing base-β encoding, we represent a as the base-β integer vector $\mathbf{A} = (a_0, \ldots, a_{d-1})$. We also form another base-β integer vector $\mathbf{D} = (a_0, \ldots, a_{d-1}, \ldots, a_0, \ldots, a_{d-1}) \in R_t$ by repeating integer a σ times where $|\mathbf{D}| = d \cdot \sigma$. According to comparison protocol in Section 3.1, Dev homomorphically computes the arithmetic in Eq. (1) by applying the packing method in Eq. (3) as $ct_2(g) = 2 \cdot (ct_2(\mathbf{C}) \oplus (-ct_2(\mathbf{D})))$. According to Proposition 2, Dev also homomorphically multiplies a random mask $r \in \mathbb{Z}_t$ given by Alice as random integer vector $\mathbf{N} = (r, 0, \ldots, 0) \in R_t$ of length $d \cdot \sigma$ to $ct_2(g)$ for getting $ct_2(g') = ct_2(g) \otimes ct_2(\mathbf{N})$. Besides, Dev generates another random mask $r_1 \in \mathbb{Z}_t^{d \cdot \sigma}$ that can be represented as another integer vector $\mathbf{N}' = (r_{1,0}, \ldots, r_{1,d-1}) \in R_t$ and adds the vector to $ct_2(g')$ to produce $ct_2(g'') = ct_2(g') \oplus ct_2(\mathbf{N}')$.

5 Evaluation

In this section, we show the implementation of private comparison protocol and its application to private range queries using the batch technique. Besides, we show the batch implementation of our comparison protocol separately to show its practicality towards big data processing. Here we show the parameter settings for our experiments and our experimental results. Also, we evaluate achieved security level at the end of this section.

Table 1. Performance comparison for private comparison protocol

Integer Size (bits)	Security level		Total time (milliseconds)	
	Saha et al. [19]	Our method	Saha et al. [19]	Our method
16	140	140	124	84
32	140	140	125	85

5.1 Parameter Settings

Now we describe the parameters for the underlying SwHE scheme in [17] that we used for our protocol security. As mentioned in [17], we chose the values of (n, q, t, δ) appropriately to ensure a correct decryption. To get a security level of minimum 128-bit, we need to set the lattice dimension n=2048, t=2048, and q=61-bit [16] where $q \geq 16n^2t^2\delta^4$ as discussed in Section 4.3 of [23]. Here we set

$(n, q, t, \delta) = (2048, 61\text{-bits}, 2039, 8)$ for our comparison protocol. In case of batch comparison, we set the batch size σ to 16384 (resp., 8192) for 16-bit (resp., 32-bit) integer which requires a lattice dimension of $n = 65536$. Here $q \geq 16n^2t^2\delta^4 = 2^4 \cdot 2^{32} \cdot 2^{22} \cdot 2^{12} = 2^{70}$. Therefore, we fix $(n, q, t, \delta) = (2048, 71\text{-bits}, 2039, 8)$ for our batch comparison. According to our packing method in Section 3.4, considering 40-bit values and using base-16 encoding, we get a vector of size 10 for each integer value. Now we can fit 200 values comfortably inside a lattice dimension of 2048. Following along, readers might argue using a smaller lattice dimension for the case of 100 records but we chose n=2048 to get an acceptable level of security ($\geq$ 128-bit). For the case of 1000 records, we required 5 blocks of $n = 2048$ each. We decide on q by fixing $n = 2048$, $t = 2039$ for records (and $t = 2048$ for attribute matching) and $\delta = 8$. Now $q \geq 2^4.2^{22}.2^{22}.2^{12} = 2^{60}$ for matching both attribute and value. Finally, we set the parameters $(n, q, t, \delta) = (2048, 61\text{-bits}, 2048, 8)$ and $(n, q, t, \delta) = (2048, 61\text{-bits}, 2039, 8)$ for matching both attribute and value respectively. For attribute name matching, we use the same procedure as used in [18].

Table 2. Performance of private comparison protocol with the batch technique

Integer size (bits)	Batch size (σ)	Parameters (n, t, q, δ)	Security level	Total time (Seconds)
16	16384	(65536, 2039, 71-bits, 8)	124	4.087
32	8192		125	4.057

5.2 Performance Analysis

We implemented our comparison protocol and private range query in C++ programming language along with PARI C library (2.9.1 version) [20] and ran the program on a single machine configured with 3.6 GHz Intel Core-i5 processor equipped with 8 GB of RAM inside Linux environment. As shown in Table 1, our comparison protocol took 84 milliseconds (ms.) (resp., 85 ms.) for a single comparison of both 16-bit (resp., 32-bit) integers. On the other hand, the protocol of Saha et al. [19] took 124 ms. (resp. 125 ms) for comparing two integers of 16-bit (resp., 32-bit). Besides, we experimented our protocol for batch computation for 16-bit (resp., 32-bit integers) which took 4.087 seconds (resp., 4.057 seconds) for 16384 (resp.,8192) comparisons as shown in Table 2. On an average, the batch comparison is able to handle 1 million comparisons of 16-bit integers within 4.16 minutes which can be further minimized in a real distributed cloud environment involving many PCs at a time. Now we show the performance of our protocol using the batch technique for processing the private range queries over an encrypted database. We created our two databases with 11 attributes with 40-bit integer values, with the first one having 100 records and the second one having 1000 records. Similar to [12], all the queries had 3 conditions in

Table 3. Performance of our protocol in application to 3-out-of-11 range query with 40-bit data

τ (# of Record)	k (# of conditions)	Timing (seconds)		Security Level	
		Kim et al. [12]	Our Protocol	Kim et al. [12]	Our Protocol
100	3	16	$(0.219 + 0.280) = 0.499$	125	140
1000	3	160	$(1.967 + 0.280) = 2.247$	125	140

the range query. All the values of the database record were encoded by base-16 meaning that all the digits in the encoded vector taken from the alphabet set {0,1,2,...,15}. We also encoded the attribute names using an 8-bit integer. As already mentioned in the previous section, the program segment for privately matching the attribute names was taken from the implementation of [18]. For the case of 100 records (resp., 1000 records), our 3-out-of-11 (3 range conditions out of 11 available attributes) range query took 0.219 seconds (resp., 1967 seconds) for value comparisons and 0.280 seconds for both cases of attribute matching as shown in Table 3. On the contrary, Kim et al. [12] needed 16 seconds for 100 records and 160 seconds for 1000 records case. It is obvious that our protocol performs a way faster than existing secure range query protocols as well as it is, in our knowledge, the fastest existing solution to private batch integer comparison. Moreover, Kim et al. required a multiplicative depth of $\lceil \log l \rceil + 2\lceil \log(1+\rho) \rceil$ for their range query on l-bit message with ρ attributes. On the contrary, our method required a depth of $\log 2$ due to using our packing method. Also, the communication complexity of our protocols is $\mathcal{O}(3k \cdot \tau \cdot l \log q)$.

5.3 Security Level

In 2016, NIST [1] defined an acceptable security level as more than 128-bit for any security scheme that is valid beyond 2030. In addition, Chen and Nguyen [5] estimated in lattice-based cryptographic schemes that it is required to have the root Hermite factor $\pi < 1.0050$ to achieve an 80-bit security level. As discussed in [13], the running time t_{adv} is defined as $\lg(t_{adv}) = 1.8/\lg(\pi) - 110$ where the root Hermite factor π is expressed as $c \cdot q/\sigma = 2^{2\sqrt{n \cdot \lg(q) \cdot \lg(\pi)}}$. According to above discussion, we achieve a security level of 140-bit for our comparison protocol which is equal to that of Saha et al. [19]. On the contrary, we also achieve 140-bit security level for our range query case whereas Kim et al. [12] achieved 125-bit of security level. Due to using a higher lattice dimension of 65536, we also achieve a higher security level of 6744-bit in our batch comparison.

6 Conclusions

Throughout this paper, we discussed an efficient private comparison protocol using base-β encoding by employing ring-LWE based somewhat homomorphic encryption in the semi-honest model. In addition, our private comparison protocol was able to compare two integers of 16-bit (resp., 32-bit) consuming 84 ms (85 ms). Therefore, our comparison protocol works faster than Saha et al's protocol [19] for a single comparison. Furthermore, database implementation of our comparison protocol to evaluate a private range query also outperformed Kim et al.'s implementation [12] for 100 and 1000 records case along with security level. Besides, we believe that our batch comparison enabled us to perform one million comparisons for 16-bit integer within a few minutes which a big step towards big data processing.

References

1. Barker, E.: Recommendation for key management. In: NIST Special Publication 800-57 Part 1 Rev. 4, NIST (2016).
2. Boneh, D., Gentry, C., Halevi, S., Wang, F., Wu, D.J.: Private database queries using somewhat homomorphic encryption. In: Jacobson M., Locasto M., Mohassel P., Safavi-Naini R. (eds) Applied Cryptography and Network Security. ACNS 2013. pp. 102–118. Springer Berlin Heidelberg (2013). doi:10.1007/978-3-642-38980-1_7
3. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, pp. 309–325. ACM (2012)
4. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011, LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011)
5. Chen, Y. and Nguyen, P. Q.: BKZ 2.0: Better lattice security estimates. In: Lee D.H., Wang X. (eds) Advances in Cryptology - ASIACRYPT 2011. ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer Berlin Heidelberg (2011)
6. Cheon, J.H., Kim, M., Kim, M.: Optimized search-and-compute circuits and their application to query evaluation on encrypted data. In: IEEE Transactions on Information Forensics and Security, 11(1), pp.188-199. IEEE Press, New York (2016).
7. Damgård, I., Geisler, M., Krøigård, M.: Homomorphic encryption and secure comparison. International Journal of Applied Cryptography, 1(1), pp. 22–31. (2008)
8. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Symposium on Theory of Computing - STOC 2009, pp. 169-178. ACM, New York (2009)
9. Graepel T., Lauter K., Naehrig M.: ML Confidential: Machine Learning on Encrypted Data. In: Kwon T., Lee MK., Kwon D. (eds) Information Security and Cryptology - ICISC 2012. ICISC 2012. LNCS, vol 7839. pp. 1-21. Springer, Berlin, Heidelberg (2013)
10. Kantarcioglu M., Nix R., Vaidya J.: An Efficient Approximate Protocol for Privacy-Preserving Association Rule Mining. In: Theeramunkong T., Kijsirikul B., Cercone N., Ho TB. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2009. LNCS, vol 5476. pp. 515-524. Springer, Berlin, Heidelberg (2009)
11. Kim, M., Lee, H. T., Ling, S., Ren, S. Q., Tan, B.H.M., Wang, H.: Better security for queries on encrypted databases. IACR Cryptology ePrint Archive, 2016/470, (2016)

12. Kim, M., Lee, H. T., Ling, S., Wang, H.: On the efficiency of FHE-based private queries. IEEE Transactions on Dependable and Secure Computing, vol. PP(99), pp.1-1. IEEE Press, New York (2016). doi: 10.1109/TDSC.2016.2568182
13. Lauter, K., Naehrig, M., Vaikuntanathan, V.: Can Homomorphic Encryption be Practical? In: ACM Workshop on Cloud Computing Security Workshop, CCSW 2011, pp. 113–124. ACM, New York (2011)
14. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern J. (eds) Advances in Cryptology - EUROCRYPT'99. EUROCRYPT 1999. LNCS 1592. pp. 223-238. Springer, Berlin, Heidelberg (1999)
15. Saha, T.K., Ali, A.B.M.: Storage cost minimizing in cloud- a proposed novel approach based on multiple key cryptography", In: 1st Asia-Pacific World Congress on Computer Science and Engineering (APWConCSE). pp. 1–9. IEEE (2014)
16. Saha, T. K., Koshiba, T.: Private equality test using ring-LWE somewhat homomorphic encryption, In: 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWConCSE), pp. 1–9. IEEE (2016). doi: 10.1109/APWC-onCSE.2016.013
17. Saha T. K., Koshiba, T.: Private conjunctive query over encrypted data. In: Joye M., Nitaj A. (eds) Progress in Cryptology - AFRICACRYPT 2017. AFRICACRYPT 2017, LNCS 10239, pp. 149-164. Springer (2017). doi: 10.1007/978-3-319-57339-7_9
18. Saha T. K., Koshiba, T.: Efficient protocols for private database queries. In: Livraga G., Zhu S. (eds) Data and Applications Security and Privacy XXXI. DBSec 2017. LNCS 10359, pp. 337-348. Springer, Cham (2017)
19. Saha, T. K., Koshiba, T.: An efficient privacy-preserving comparison protocol, In: Barolli L., Enokido T., Takizawa M. (eds) Advances in Network-Based Information Systems. NBiS 2017. Lecture Notes on Data Engineering and Communications Technologies, vol. 7. Springer, Cham (2018). doi: 10.1007/978-3-319-65521-5_48
20. The PARI∼Group, PARI/GP version `2.7.5`, Bordeaux, 2014, `http://pari.math.u-bordeaux.fr/`
21. Xue, K., Li, S., Hong, J., Xue, Y., Yu, N. and Hong, P.: Two-Cloud Secure Database for Numeric-Related SQL Range Queries With Privacy Preserving. IEEE Transactions on Information Forensics and Security, 12(7), pp.1596-1608. IEEE (2017)
22. Yao, A.C.: Protocols for secure computations. In 23rd Annual Symposium on Foundations of Computer Science, 1982. pp. 160–164. IEEE (1982)
23. Yasuda, M., Shimoyama, T., Kogure, J., Yokoyama, K., Koshiba, T.: Practical Packing Method in Somewhat Homomorphic Encryption. J. Garcia-Alfaro et al. (Eds.): DPM 2013 and SETOP 2013, LNCS, vol. 8247, pp. 34–50. Springer, Heidelberg (2014)