A novel secure scheme for supporting complex SQL queries over encrypted databases in cloud computing

Hindawi
Security and Communication Networks
Volume 2018, Article ID 7383514, 15 pages
https://doi.org/10.1155/2018/7383514

Research Article

A Novel Secure Scheme for Supporting Complex SQL Queries over Encrypted Databases in Cloud Computing

Guoxiu Liu,[1,2] Geng Yang,[1,3] Huaqun Wang,[1] Yang Xiang,[4] and Hua Dai[1,3]

[1]*Nanjing University of Posts and Telecommunications, Nanjing 210003, China*
[2]*School of Computer and Information Engineering, Chuzhou University, Chuzhou 239000, China*
[3]*Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, Nanjing 210003, China*
[4]*School of Information Technology, Deakin University, 221 Burwood Highway, Burwood, VIC 3125, Australia*

Correspondence should be addressed to Geng Yang; yangg@njupt.edu.cn

Received 6 January 2018; Revised 5 May 2018; Accepted 30 May 2018; Published 3 July 2018

Academic Editor: Emanuele Maiorana

With the advance of database-as-a-service (DaaS) and cloud computing, increasingly more data owners are motivated to outsource their data to cloud database for great convenience and economic savings. Many encryption schemes have been proposed to process SQL queries over encrypted data in the database. In order to obtain the desired data, the SQL queries contain some statements to describe the requirement, e.g., arithmetic and comparison operators (+, −, ×, <, >, and =). However, to support different operators (+, −, ×, <, >, and =) in SQL queries over encrypted data, multiple encryption schemes need to be combined and adjusted to work together. Moreover, repeated encryptions will reduce the efficiency of execution. This paper presents a practical and secure homomorphic order-preserving encryption (FHOPE) scheme, which allows cloud server to perform complex SQL queries that contain different operators (such as addition, multiplication, order comparison, and equality checks) over encrypted data without repeated encryption. These operators are data interoperable, so they can be combined to formulate complex SQL queries. We conduct security analysis and efficiency evaluation of the proposed scheme FHOPE. The experiment results show that, compared with the existing approaches, the FHOPE scheme incurs less overhead on computation and communication. It is suitable for large batch complex SQL queries over encrypted data in cloud environment.

1. Introduction

With the advance of cloud storage and computing, the business opportunity to offer a database as an outsourced service is gaining momentum. Today numerous enterprises and end users may outsource their data to those cloud service providers for lower cost and better performance [1, 2]. Outsourced databases can be applied to many scenarios. For example, one outsourced database application scenario is shown in Figure 1, and, in this example, the data owners, such as hospitals, may want to outsource the medical records to the cloud databases. Patients' medical records contain sensitive information (e.g., blood pressure, body mass index). Based on the assumption that service provider is honest-but-curious [3, 4], sensitive information needs to be encrypted before being uploaded to the cloud database. The data owners can query their data from cloud database. Then, the cloud database should execute SQL queries over the encrypted data. However, the encrypted data may also bring significant difficulty in executing standard SQL and computing over these data. For example, the encrypted data may lose the original order, without the set of primitive operators, such as equality checks, order comparisons, addition, multiplication, aggregates (sums), and joins.

To date, many fully homomorphic encryption (FHE) and order-preserving encryption (OPE) schemes were proposed [5–10]. The FHE schemes are not practical for either cloud database providers or users, because of high computational overhead [11, 12], and these schemes only support homomorphic addition and homomorphic multiplication over encrypted data. The OPE schemes reveal the order and expose some private information to the cloud service provider, which support SQL range queries. On the contrary, CryptDB uses onions to protect private data and support efficient SQL

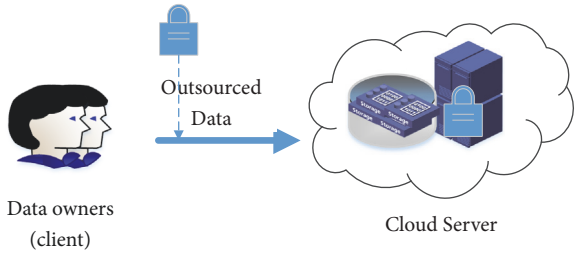

Figure 1: Data owners outsource their data (e.g., credit card details and patient's medical records) to the cloud database. The sensitive data need to be encrypted. The cloud server provides storage and query service.

queries over encrypted data. The onion encryption is a multilayered encryption scheme, and for processing different types of computations multiple onions are needed in practice, because the computations are supported by different encryption schemes. For example, the CryptDB can perform range queries while a column is encrypted with order-preserving encryption, and if it performs aggregate queries, such column is encrypted with homomorphic encryption. Furthermore, the functionalities of CryptDB lack some useful features (e.g., there is no support for queries containing multiplication, and there are also some other limitations). As a result, a query like SELECT $*$ FROM T1 WHERE $A = 100$ AND $A + C \times D > E$ cannot be executed by CryptDB. How to design a practical encrypted scheme that supports different operators (+,−,×,<,>,=) in complex SQL queries over encrypted data without privacy breaches remains a challenging and open problem.

Motivated by the aforementioned problem, we design a full homomorphic algorithm with the order-preserving feature to support complex SQL queries that contain different operators (such as addition, multiplication, order comparison, and equality checks) over encrypted data. Our proposed FHOPE scheme makes up for FHE's shortcoming, which supports order comparison, enables range queries to be executed in database operations, and reduces computational costs to increase efficiency. Because it has order-preserving function, the order of the plaintext values is inevitably revealed. The ideal security goal for an order-preserving scheme, IND-OCPA [13], is to reveal no additional information about the plaintext values besides their order. Our proposed scheme is an ideal security homomorphic order-preserving encryption scheme where the ciphertexts reveal nothing except for the order of the plaintext values. Regarding efficiency, we can see that security and efficiency are contradictory; the higher security, the lower efficiency. Practicality and efficiency are very important for database applications. The FHOPE is efficient and practical, satisfies the need of database applications, and solves the complex queries problems that need to be solved in the database. FHOPE can resist the homomorphic order-preserving chosen-plaintext attack. Here, we summarize our contributions as follows:

(i) In the cloud database environment, data is frequently queried by users. It is critical to determine whether an encryption scheme can provide complex SQL queries like the predicates containing different operators over encrypted data. We combine homomorphism with order-preserving and design a novel FHOPE scheme to support addition, multiplication, order comparison, and equality checks. These operators are data interoperable, so they can be combined to formulate complex SQL queries. Then, the FHOPE scheme enables a wide range of SQL queries over the encrypted data to be expressed. As a result, it does not require downloading the encrypted data to client. Therefore, it can improve the efficiency in dealing with data query and processing.

(ii) Furthermore, we optimize the FHOPE scheme by adding some random noise with a certain probability P and by specifying any sub-ciphertext with order-preserving property. Besides, we apply the FHOPE scheme to the cloud database application.

(iii) We evaluate the proposed FHOPE scheme in terms of security, efficiency, and complexity. The concrete FHOPE scheme is provably secure according to the formal security proof. The experiment results show that the FHOPE scheme incurs less overhead in computation and communication than the existing approaches. It is suitable for large batch of SQL queries over encrypted data in cloud environment.

The remainder of this paper is organized as follows. Section 2 discusses some related work. In Section 3 we describe the system model and attack model. Section 4 gives the basic idea of FHOPE scheme and its construction. Section 5 presents the correctness of FHOPE scheme. Section 6 describes the FHOPE's application in a cloud database. In Section 7 we give security analysis. Section 8 describes the evaluations. Section 9 concludes the paper.

2. Related Work

The security of data and processing of the encrypted data in a cloud database environment have caused much research concern recently [14, 15]. Many schemes have been designed with various techniques: fully homomorphic scheme (FHE) and order-preserving encryption (OPE). Gentry had described a FHE [7, 8] in 2009; the FHE supports various computations over ciphertexts. Since Gentry's result of research breakthrough, a great many improvements [12, 16–21] have been made; the performance was enhanced. However, since the current FHE schemes have low efficiency, they are not suitable for practical applications. Another encryption scheme is OPE; it is primarily used in databases for supporting order comparison on ciphertexts. OPE [5] solves the encrypted query problems in database systems, which was first proposed in 2004. Although a large number of researchers have made great efforts on the OPE schemes [9, 13, 22, 23], these schemes have failed to achieve ideal security. Until now, Popa et al. proposed the mutable order-preserving encoding (mOPE) scheme [24], which is an ideal security OPE scheme; it builds a balanced search tree, which contains the plaintext values

encrypted by the application. mOPE is an ideal security scheme, but it has the low efficiency due to the interaction and tree balancing. Moreover, their works only process order comparison on ciphertexts.

Some solutions were proposed for querying data over the encrypted database [25–27]. One of the most important fundamental schemes for processing queries on an encrypted database is proposed by H. Hacigümüş et al. in [25]. It encrypts the data at a tuple level, and then a predefined set of attributes can be used in queries. Following H. Hacigümüş's idea, some improvements were proposed [28, 29]. The scheme [28] stores redundant data for querying data over the encrypted database, and B. Hore et al. [29] extended the model of H. Hacigümüş et al. and added range queries over the encrypted database. To achieve the various computations over encrypted data, some Paillier-based improvements [8, 30–34] were presented. The schemes in [30, 31] can support homomorphic addition, homomorphic multiplication, and order comparison, but the order comparison is realized by converting to subtraction operation; they have high computation overhead. Yan et al. [33] can only support the addition and cannot support other computation operations. Peter et al. [34] proposed an efficient outsourcing multiparty computation framework under multiple keys, but the scheme only supports addition and multiplication and cannot support other operations.

CryptDB [35] and SDB [36] are well-known systems for processing queries over encrypted database. CryptDB uses onions to support SQL queries over encrypted databases, where range queries and equality condition queries rely on order-preserving encryption [24] and deterministic encryption, respectively. It performs specific operations with homomorphic encryption, to support aggregate queries; it implemented the Paillier cryptosystem [29], but it cannot support homomorphic multiplication. The CryptDB has the following limitations: (1) its queries are processed on the lowest-security level of data; (2) the same data needs to be reencrypted according to different types of computation. For example, it can perform range queries while a column is encrypted with order-preserving encryption, and if it performs aggregate queries, such column is encrypted with homomorphic encryption. SDB [36] can process queries that contain different kinds of operations; nevertheless, it requires massive computation resources and communication cost.

Thus, it is always necessary to establish an efficient scheme to process database queries without involving multiple incompatible encryption schemes.

3. System Model and Attack Model

In this section, we describe the system model and the attack model and give formal definition of the scheme. The prototype will be built based on the system model. The security of the proposed scheme will be analyzed in Section 7.

3.1. System Model. Figure 2 shows the overall architecture. The client receives queries from users, generates the private key and encrypts the sensitive data, sends the SQL queries to the cloud server, receives queries results, decrypts the results using the corresponding keys, and sends the decrypted result to the users.

A FHOPE scheme in this paper involves two different entities which are described below.

Client (CL). The client is data owner. For protecting data privacy, it uses the private key to encrypt the sensitive data and then outsources the encrypted data to a cloud server. The CL can also send the SQL queries to a CS and decrypt the queries results from the CS.

Cloud Server (CS). A CS is hosted by the service provider that stores the databases in cloud. It stores and manages the data of users. A CS also stores the encrypted intermediate and final results. Furthermore, a CS is able to perform homomorphic addition, homomorphic multiplication, order comparison, and equality checks over encrypted data and then process complex SQL queries on encrypted data.

To describe our scheme, we give the formal definition of FHOPE.

Definition 1 (FHOPE). A FHOPE scheme consists of four phases (key generation, encryption, decryption, and computation). The detailed phases are described below.

(1) Key generation: $sk \longleftarrow KeyGen(1^k)$. KeyGen runs at the CL, takes as input the security k, and outputs a private key sk. The CS cannot get access to the private key.

(2) Encryption: $c \longleftarrow Enc(sk, v)$. Enc runs at the CL. The inputs to the CL are sk and the sensitive data v in the SQL queries, and the CL obtains a ciphertext c and then sends the SQL queries to a CS.

(3) Decryption: $v \longleftarrow Dec(sk, c)$. The CL runs Dec on the private key and a ciphertext c and obtains a plaintext v.

(4) Computation: $res \longleftarrow HAMOE(c_1, \ldots, c_l)$. HAMOE runs at the server, takes as input ciphertext $c_1, \ldots, c_l$, and can perform addition, multiplication, order comparison, and equality checks over the ciphertext and then output the result of the computation.

To describe the correctness of our scheme, we define what it means for the scheme to be correct. Intuitively, the scheme should decrypt the correct values and correctly support homomorphic addition, homomorphic multiplication, and order comparison on the ciphertext. Suppose that we have a secret key vector $K(n)$ and that m integers $v_i \in V(1 \leq i \leq m)$ are encrypted into m vectors $C_1, C_2, \ldots, C_m$, where $C_i = (c_{i1}, \ldots, c_{in})$.

Definition 2 (correctness). A FHOPE scheme for plaintext domain Z is correct if, for all security parameters k, for all $K(n) \longleftarrow KeyGen(1^k)$,

(1) for all $v \in Z$ and for every C outcome of $FHOPE(v, K(n))$, $Dec(K(n), C) = v$;

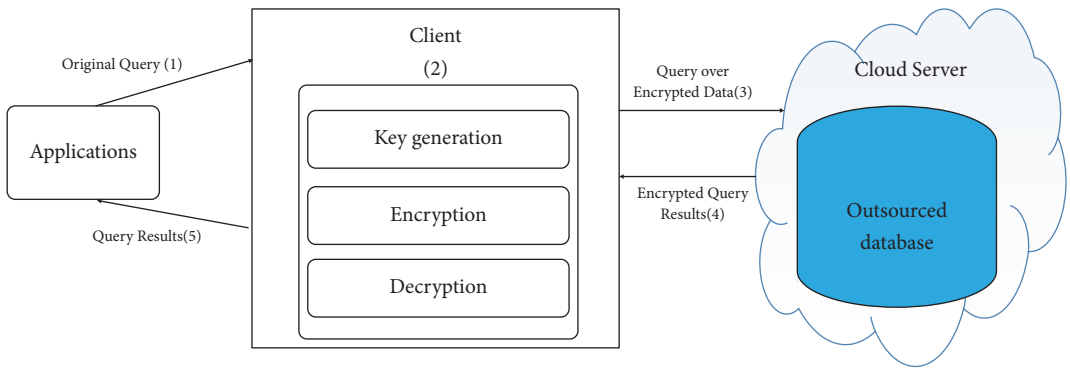

Figure 2: System model for outsourced databases.

(2) for all $v_i \in Z$ and for every C_i outcome of $FHOPE(v_i, K(n))$, $Dec(K(n), \sum_{i=1}^{m} C_i) = \sum_{i=1}^{m} v_i$;

(3) for all $v_i \in Z$ and for every C_i outcome of $FHOPE(v_i, K(n))$, $Dec(K(n), C_i \times C_j) = v_i \times v_j$;

(4) for all sequences $se = \{v_1, \ldots, v_m\} \in Z^m$, for all pairs $v_i, v_j \in se$, for all C_i, C_j obtained as above, we have $v_i < v_j \Longleftrightarrow C_i < C_j$;

(5) for all $v_i \in Z$ and for every C_i outcome of $FHOPE(v_i, K(n))$, we have $v_i + v_j \times v_k > v_l \Longleftrightarrow C_i + C_j \times C_k > C_l$.

3.2. Attack Model. In this section, we present the potential threats and the security requirements for database outsourcing in the cloud. In our scheme, we assume the same security model commonly adopted in related literatures in this field (e.g., [35]), where the CL is the data owner. Thus, the CL is trusted; the CS is honest-but-curious; that is, the computation provided by the CS is able to be executed correctly, and it does not change the data or query results, but the CS tries its best to obtain the privacy information of the processed data. Order-preserving encryption is primarily used in databases for supporting order comparison on ciphertexts, it exposes the order of data, and then the cloud can learn the statistical properties (like order) through repeated query requests. Therefore, we introduce an adversary A in our model, which aims to decrypt the ciphertexts of a challenge sent to applications with the following capabilities:

(1) A may try to obtain the private key and guess the plaintext values from ciphertexts outsourced from a CL.

(2) A may compromise the CS by guessing the plaintext values of the computation results received from the CS.

(3) A may compromise the CS to guess the plaintext values of the queries results based on statistical properties (like order).

For satisfying the security requirements of the FHOPE scheme, we formalize the security definition of a FHOPE scheme for IND-HOCPA (indistinguishability under a homomorphic order-preserving chosen-plaintext attack), which intuitively says that the scheme must not leak anything besides order. The homomorphic order-preserving chosen-plaintext attack is a restricted chosen-plaintext attack. We remark that the restricted chosen-plaintext attack is used in literature [10]. We adapt the security definition of literature [13] to the syntax of our proposed scheme.

Definition 3 (IND-HOCPA security). A FHOPE scheme is IND-HOCPA secure, if any probabilistic polynomial time (PPT) adversary A has only a negligible advantage $Adv_{FHOPE,A}^{IND-HOCPA}$ to win in the following game. The FHOPE game between the adversary A and the challenger CH is given below:

(1) For the secure parameter k, the challenger CH runs the key generation algorithm KeyGen and generates $sk \longleftarrow KeyGen(1^k)$.

(2) The challenger CH and the adversary A engage in a polynomial number of rounds of interaction. For round i,

  (1) the adversary A chooses two equal-length messages $v_i^0, v_i^1 \in Z$ and sends them to the challenger CH;

  (2) the challenger CH picks $b \in \{0, 1\}$ at random and leads the interaction for the Enc algorithm on inputs sk and v_i^b with the server CS, with the adversary A observing all the ciphertexts at CS.

(3) The adversary A outputs b', its guess for b.

We say that the adversary A wins the game if (1) its guess is correct ($b = b'$) and (2) the sequences $\{v_i^0\}_i$ and $\{v_i^1\}_i$ have the same order relations (namely, for all i, j,$v_i^0 < v_j^0 \Longleftrightarrow v_i^1 < v_j^1$). That is, A wins the above game if $Adv_{FHOPE,A}^{IND-HOCPA}(k)$ is nonnegligible, where the adversary's advantage $Adv_{FHOPE,A}^{IND-HOCPA}(k)$ in the above game is defined as

$$Adv_{FHOPE,A}^{IND-HOCPA}(k) = \left| \Pr\left[win^{A,k}\right] - \frac{1}{2} \right|, \quad (1)$$

where $win^{A,k}$ is the random variable indicating the success of the adversary in the above game.

Table 1: Notation.

Symbol	Meaning
V	the set of all input plaintexts
v	a plaintext
C	a ciphertext is comprised of two or more sub-ciphertexts
c_i	i-th sub-ciphertext
k	security parameter
$K(n)$	a secret key is comprised of a set of key components
k_i	i-th key component
$Enc()$	a function for encryption
$Dec()$	a function for decryption
$Enc_i()$	a strictly increasing function over $K(n)$ and v
$Noise_i()$	a function over $K(n)$ and R
S	the sensitivity of input values v

4. Fully Homomorphic Order-Preserving Encryption Scheme (FHOPE)

This section presents a novel fully homomorphic order-preserving encryption (FHOPE) scheme to realize various types of operations over encrypted data, such as addition, multiplication, order comparison, and equality checks. Firstly, we describe the notations employed in the remainder of the paper. Then, we construct the FHOPE scheme and prove the correctness of decryption. For clear description, Table 1 summarizes the notations employed in the paper.

4.1. Homomorphic Encryption Scheme. A practical homomorphic encryption scheme is presented by Liu in 2013 [37], which contains three steps and can be described as follows.

KeyGen. The secret key $K(n) = (k_1, k_2, \ldots, k_n)$ is chosen randomly from real number set R.

Encrypt. A message $m \in R$ is encrypted into $C = Enc(v, K(n)) = (c_1, \ldots, c_n)$; the encryption result is a tuple of n components, corresponding to n sub-ciphertexts.

Decrypt. Take as input the secret key $K(n)$ and a ciphertext $C = (c_1, \ldots, c_n)$; compute and output a message m:

$$m = Dec\left(K(n), (c_1, \ldots, c_n)\right). \tag{2}$$

Our proposed scheme differs from that of [37] in that we focus on designing an encryption scheme that supports complex expressions containing different operators (+, −, ×, <, >, and =) in SQL queries over encrypted data and data interoperable operators.

4.2. Construction of FHOPE Scheme. By using symmetric encryption, a full homomorphic order-preserving encryption is given as follows, which consists of three steps.

KeyGen(KG). Generate the secret key

$$\begin{aligned} K(n) &= \left[\overrightarrow{k_1}, \overrightarrow{k_2}, \ldots, \overrightarrow{k_n}\right] \\ &= \left[(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\right], \end{aligned} \tag{3}$$

where $(a_i, b_i)(1 \le i \le n)$ is a list of pairs of integers, which are large prime numbers, $a_i * b_i > 0$, $n > 1$, $a_i \neq 0$ for $1 \le i \le n, b_1 + \cdots + b_{n-1} \neq 0$, and $b_n \neq 0$. The number of key components in the key set is equal to the number of sub-ciphertexts.

Encrypt (Enc). Encrypt the plaintext $FHOPE(v, K(n)) = (c_1, \ldots, c_n)$, where v is a plaintext; the encryption result is a tuple of n components, corresponding to n sub-ciphertexts. The encryption algorithm uses the components $Enc_i(K(n), v)$, $Noise_i(K(n), R)$, and ξ_i to define each c_i, as shown below, where Enc_i is a strictly increasing function over $K(n)$ and v, in particular linear to v. $Noise_i$ is a function over $K(n)$ and R, which calculates a random number for randomizing c_i, ξ_i denote the random noise, which is randomly sampled from the range $[-\infty, +\infty]$, and a set R of n pairs of numbers $\{(r_1, p_1), \ldots, (r_n, p_n)\}$ is defined in a finite integer domain. We define the functions $Enc_i()$ and $Noise_i()$ by (5) and (6), respectively.

$$c_i = Enc_i\left(K(n), v\right) + Noise_i\left(K(n), R\right) + \xi_i, \tag{4}$$

$$Enc_i\left(a_i, b_i, v\right) = a_i * b_i * v, \tag{5}$$

$$\begin{aligned} &Noise_i\left(a_i, r_i, p_i\right) \\ &= \begin{cases} \dfrac{a_1 \times p_1}{a_2} - \dfrac{a_1 \times r_n}{a_n} + r_1 - p_n & i = 1 \\ \dfrac{a_i \times p_i}{a_{i+1}} - \dfrac{a_i \times r_{i-1}}{a_{i-1}} + r_i - p_{i-1} & 2 \le i \le n-1 \\ \dfrac{a_n \times p_n}{a_1} - \dfrac{a_n \times r_{n-1}}{a_{n-1}} + r_n - p_{n-1} & i = n \end{cases} \end{aligned} \tag{6}$$

The noise defined in (6) should satisfy condition (7).

$$\begin{aligned} &0 < Noise_i\left(K(n), R\right) \\ &\quad < \left(Enc_i\left(K(n), v + S\right) - Enc_i\left(K(n), v\right)\right). \end{aligned} \tag{7}$$

Decrypt (Dec). Decrypt a ciphertext $C = (c_1, \ldots, c_n)$, and get the plaintext v.

$$Dec\left(K(n), (c_1, \ldots, c_n)\right) = v, \tag{8}$$

where $K(n) = [(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)]$ is a secret key; v is a plaintext.

Then, the decryption algorithm is defined as

$$\sum_{i=1}^{n} Dec_i\left(a_i, b_i\right) * c_i = v, \tag{9}$$

$$Dec_i\left(a_i, b_i\right) = \frac{1}{a_i \times \sum_{i=1}^{n} b_i}, \tag{10}$$

where Dec_i is an i-th decryption function over the key vector, and it has a linear time complexity with respect to n. Based on the definition of c_i in (4), (8) is rewritten into (9), which is equal to

$$\begin{aligned} &\sum_{i=1}^{n} Dec_i\left(K(n)\right) \\ &\quad * \left(Enc_i\left(K(n), v\right) + Noise_i\left(K(n), R\right) + \xi_i\right) = v. \end{aligned} \tag{11}$$

In order to ensure the validity of decryption steps in (9), it has to satisfy the conditions

$$\sum_{i=1}^{n} Dec_i\left(K\left(n\right)\right) * \left(Noise_i\left(K\left(n\right), R\right) + \xi_i\right) = 0, \tag{12}$$

$$\sum_{i=1}^{n} Dec_i\left(K\left(n\right)\right) * Enc_i\left(K\left(n\right), v\right) = v. \tag{13}$$

Proof of Correctness for Decryption

Proof. To prove correctness of decryption, suppose that any $v_1 \in Z$ is encrypted into $C_1 = (c_{11}, \ldots, c_{1n})$ with the key $K(n)$, as shown below.

$$c_{1i} = Enc_i\left(K\left(n\right), v_1\right) + Noise_i\left(K\left(n\right), R_i^1\right) + \xi. \tag{14}$$

Suppose that the first sub-ciphertext (c_{11}) has order-preserving property, and random noise ξ is added to the first and second sub-ciphertext, respectively. Then, we have to prove

$$Dec(K\left(n\right), \left(c_{11}, \ldots, c_{1n}\right) = v_1. \tag{15}$$

That is,

$$\begin{aligned} &\sum_{i=1}^{n} Dec_i\left(K\left(n\right)\right) \\ &\quad * \left(Enc_i\left(K\left(n\right), v_1\right) + Noise_i\left(K\left(n\right), R\right) + \xi\right) \\ &\quad = v_1, \end{aligned} \tag{16}$$

where

$$Dec_i\left(a_i, b_i\right) = \frac{1}{a_i * \sum_{i=1}^{n} b_i}. \tag{17}$$

We have

$$\begin{aligned} &\sum_{i=1}^{n} Dec_i\left(K\left(n\right)\right) * Noise_i\left(K\left(n\right), R\right) \\ &\quad = \frac{1}{\sum_{i=1}^{n} b_i} * \left(\frac{p_1}{a_2} - \frac{r_n}{a_n} + \frac{r_1}{a_1} + \cdots - \frac{p_{n-1}}{a_n}\right) = 0. \end{aligned} \tag{18}$$

Because $a_2 = -a_1$ and $\xi \neq 0$, then

$$\sum_{i=1}^{n} Dec_i\left(K\left(n\right)\right) * \xi = \frac{1}{a_i * \sum_{i=1}^{n} b_i} * \xi = 0. \tag{19}$$

Then

$$\begin{aligned} &\sum_{i=1}^{n} Dec_i\left(K\left(n\right)\right) * Enc_i\left(K\left(n\right), v_1\right) \\ &\quad = \sum_{i=1}^{n} \frac{1}{a_i * \sum_{i=1}^{n} b_i} * a_i * b_i * v_1 = v_1. \end{aligned} \tag{20}$$

Therefore, the correctness of decryption is proved. □

To verify the correctness of operations supported by our scheme, suppose that we have a secret key vector $K(n)$ and that m integers $v_i \in V(1 \leq i \leq m)$ are encrypted into m vectors $C_1, C_2, \ldots, C_m$, where $C_i = (c_{i1}, \ldots, c_{in})$.

5. Correctness of the FHOPE

A query operation can request arbitrary data with a statement to describe the desired data. In order to obtain the desired data, the query contains some statements to describe the requirement, e.g., arithmetic and comparison operators (×, +, −, =, >, and <). These operators are data interoperable, so they can be combined to formulate complex queries, and we are concerned with executing queries that contain multiple different operations, such as WHERE $a + b \times c < d$. Our data model is column-based in a table. In this section, we prove the correctness of additive homomorphism, multiplicative homomorphism, order-preserving, and data interoperability and describe how these operators are implemented in our scheme.

5.1. Addition (AD)/Subtraction. Assuming two sensitive columns *col_A* and *col_B* of a table T, their values are integers. We use v_i and v_j to denote the values of *col_A* and *col_B* in a row t, respectively. Let C_i and C_j be the encrypted values of v_i and v_j, respectively, where $C_i = (c_{i1}, \ldots, c_{in})$ and $C_j = (c_{j1}, \ldots, c_{jn})$; they share the same secret key vector $K(n)$.

Given two sensitive columns *col_A* and *col_B*, if the application issues the query SELECT * FROM Table 1 WHERE *col_C* = *col_A* + *col_B*, the SQL query processing is as follows.

Step 1. The CL receives the SQL query, it uses the encryption algorithm Enc to encrypt the values of *col_A* and *col_B* with the private key $K(n)$; their ciphertexts are $C_i = (c_{i1}, \ldots, c_{in})$ and $C_j = (c_{j1}, \ldots, c_{jn})$, respectively.

Step 2. The CS executes the SQL query on the encrypted data just like on plaintext. Due to additive homomorphism, the CS can directly add encrypted data one by one as follows: $C_i + C_j = (c_{i1} + c_{j1}, \ldots, c_{in} + c_{jn})$, where the homomorphic addition of C_i and C_j is defined as a vector addition.

The FHOPE scheme guarantees homomorphic addition according to the following theorem.

Theorem 4. *The FHOPE scheme supports additive homomorphism; i.e.,* $Dec((\sum_{i=1}^{m} c_{i1}, \ldots, \sum_{i=1}^{m} c_{in}), K(n)) = \sum_{i=1}^{m} v_i$.

Proof.

$$\begin{aligned} &Dec\left(\left(\sum_{i=1}^{m} c_{i1}, \ldots, \sum_{i=1}^{m} c_{in}\right), K\left(n\right)\right) \\ &\quad = \sum_{j=1}^{n} Dec_j\left(K\left(n\right)\right) * \left(\sum_{i=1}^{m} c_{ij}\right) \\ &\quad = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} \left(Dec_j\left(K\left(n\right)\right) * c_{ji}\right)\right) \\ &\quad = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} Dec_j\left(K\left(n\right)\right) * c_{ji}\right) = \sum_{i=1}^{m} v_i. \end{aligned} \tag{21}$$

□

The correctness of homomorphic addition is proved.

Subtraction operation can be converted to addition operation for processing, so it is omitted.

5.2. Multiplication (MU). We describe the FHOPE scheme for "×". Given two sensitive columns *col_A* and *col_B* of a table T, let v_1 and v_2 denote the values of *col_A* and *col_B* in a row t, respectively. Let C_1 and C_2 be the encrypted values of v_1 and v_2, respectively, where $C_1 = (c_{11}, \ldots, c_{1n})$ and $C_2 = (c_{21}, \ldots, c_{2n})$; they share the same secret key vector $K(n)$.

If the application issues the query SELECT $*$ FROM Table 1 WHERE *col_C* = *col_A* × *col_B*, the SQL query processing is as follows.

Step 1. The CL receives the SQL query, it uses the encryption algorithm Enc to encrypt the values of *col_A* and *col_B* with the private key $K(n)$, and their ciphertexts are $C_1 = (c_{11}, \ldots, c_{1n})$ and $C_2 = (c_{21}, \ldots, c_{2n})$, respectively.

Step 2. The CS executes the SQL query on the encrypted data just like on plaintext. Due to multiplicative homomorphism, the CS can directly multiply encrypted data one by one as follows:

$$C_1 \times C_2 = \begin{pmatrix} c_{11} \times c_{21} & \cdots & c_{11} \times c_{2n} \\ \cdots & & \\ c_{1n} \times c_{21} & \cdots & c_{1n} \times c_{2n} \end{pmatrix}, \tag{22}$$

where the multiplication of two ciphertexts can be defined as an outer product.

Our objective is to perform multiplication operations on the encrypted data just like on plaintext. The FHOPE scheme guarantees homomorphic multiplication according to the following theorem.

Theorem 5. *The FHOPE scheme supports multiplicative homomorphism. That means* $Dec(K(n), C_1 \times C_2) = v_1 \times v_2$.

Proof. To prove this theorem, we first need to show that

$$Dec\left(K(n), C_1 \times C_2\right) = \begin{pmatrix} c_{11} \\ \cdots \\ c_{1n} \end{pmatrix} \times v_2. \tag{23}$$

Then, we prove $Dec(K(n), (c_{11}, c_{12}, \ldots, c_{1n}) \times v_2) = v_1 \times v_2$. The details are given below.

Step 1. Perform the following decryption for i, $1 \le i \le n$.

Because we have

$$\begin{aligned} &Dec\left(K(n), (c_{1i} \times c_{21}, \ldots, c_{1i} \times c_{2n})\right) \\ &= \sum_{i=1}^{n} Dec_i(K(n)) \times (c_{1i} \times c_{21}, \ldots, c_{1i} \times c_{2n}) \\ &= \sum_{i=1}^{n} Dec_i(K(n)) \times c_{1i} \times (c_{21}, \ldots, c_{2n})) \\ &= c_{1i} \times \sum_{i=1}^{n} Dec_i(K(n)) \times (c_{21}, \ldots, c_{2n}) = c_{1i} \times v_2, \end{aligned} \tag{24}$$

then it gives

$$Dec\left(K(n), C_1 \times C_2\right) = \begin{pmatrix} c_{11} \\ \cdots \\ c_{1n} \end{pmatrix} \times v_2. \tag{25}$$

Step 2. We have from Step 1:

$$\begin{aligned} &Dec\left(K(n), (c_{11} \times v_2, \ldots, c_{1n} \times v_2)\right) \\ &= \sum_{i=1}^{n} Dec_i(K(n)) \times (c_{11} \times v_2, \ldots, c_{1n} \times v_2) \\ &= \sum_{i=1}^{n} Dec_i(K(n)) \times v_2 \times (c_{11}, \ldots, c_{1n}) \\ &= v_2 \times \sum_{i=1}^{n} Dec_i(K(n)) \times (c_{11}, \ldots, c_{1n}) = v_2 \times v_1 \\ &= v_1 \times v_2. \end{aligned} \tag{26}$$

□

Hence, $Dec(K(n), C_1 \times C_2) = v_1 \times v_2$. The correctness of multiplicative homomorphism is proved.

5.3. Order Comparison (OC). We consider two comparison operators, namely, operator ">" and operator "<". They are mostly used in select queries. Given two sensitive columns *col_A* and *col_B* of table T. Let v_1 and v_2 denote the values of *col_A* and *col_B* in a row t, respectively. Let C_1 and C_2 be the encrypted values of v_1 and v_2, respectively; they share the same secret key vector $K(n)$. For privacy protection, we calculate $v_1 > v_2$ or $v_1 < v_2$, the plaintexts need to be encrypted, and we need to calculate $C_1 > C_2$ or $C_1 < C_2$. That is, the goal of FHOPE scheme is that the sort order of ciphertexts matches the sort order of the corresponding plaintexts. Here we prove that our scheme has order-preserving property.

Suppose that any two integers v_1 and v_2 are encrypted into $C_1 = (c_{11}, \ldots, c_{1n})$ and $C_2 = (c_{21}, \ldots, c_{2n})$ with the key $K(n)$, as shown below.

$$c_{1i} = Enc_i\left(K(n), v_1\right) + Noise_i\left(K(n), R_i^1\right) + \xi, \tag{27}$$

$$c_{2i} = Enc_i\left(K(n), v_2\right) + Noise_i\left(K(n), R_i^2\right) + \xi. \tag{28}$$

Definition 6. Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of all input plaintext values. The sensitivity of V is the minimum element in the set $\{|v_1 - v_2| \mid v_1 \in V, v_2 \in V, v_1 \ne v_2\}$. That is, the sensitivity S is defined as $S = \min_{\substack{v_1, v_2 \in V \\ v_1 \ne v_2}} |v_1 - v_2|$.

In fact, the sensitivity is the least gap, which was evaluated in different privacy protection [38]. And by its definition the sensitivity is always bigger than 0.

Theorem 7. *Given the sensitivity S of input value V, for all $v_1 \in V$, $v_2 \in V$, if $v_1 > v_2$, then $C_1 > C_2$.*

Proof. We have $C_1 > C_2$ if $c_{1i} > c_{2i}$, where c_{1i} and c_{2i} $(1 \le i \le n)$ are the sub-ciphertext of C_1 and C_2, respectively. Suppose that the sub-ciphertexts c_{1i} and c_{2i} have the same random noise ξ. To prove this theorem, we need to show $c_{1i} - c_{2i} > 0$; that is,

$$\begin{aligned} &Enc_i\left(K\left(n\right), v_1\right) + Noise_i\left(K\left(n\right), R_i^1\right) + \xi \\ &\quad - \left(Enc_i\left(K\left(n\right), v_2\right) + Noise_i\left(K\left(n\right), R_i^2\right) + \xi\right) \\ &\quad > 0. \end{aligned} \tag{29}$$

In other words, we have to prove

$$\begin{aligned} &Noise_i\left(K\left(n\right), R_i^2\right) - Noise_i\left(K\left(n\right), R_i^1\right) \\ &\quad < Enc_i\left(K\left(n\right), v_1\right) - Enc_i\left(K\left(n\right), v_2\right). \end{aligned} \tag{30}$$

Because the linear expression $Enc_i(K(n), v)$ is strictly increasing for any plaintext v, we have

$$\begin{aligned} &\min\left\{Enc_i\left(K\left(n\right), v_1\right) - Enc_i\left(K\left(n\right), v_2\right)\right\} \\ &\quad = Enc_i\left(K\left(n\right), v_2 + S\right) - Enc_i\left(K\left(n\right), v_2\right). \end{aligned} \tag{31}$$

Since $v_1 > v_2$, then the minimum v_1 is $v_2 + S$, which is bigger than v_2. Moreover, $Noise_i(K(n), R_i) > 0$; then,

$$\begin{aligned} &\max\left\{Noise_i\left(K\left(n\right), R_i^2\right) - Noise_i\left(K\left(n\right), R_i^1\right)\right\} \\ &\quad = Noise_i\left(K\left(n\right), R_i^2\right). \end{aligned} \tag{32}$$

Hence, the theorem holds if

$$\begin{aligned} Noise_i\left(K\left(n\right), R_i^2\right) &< Enc_i\left(K\left(n\right), v_2 + S\right) \\ &\quad - Enc_i\left(K\left(n\right), v_2\right). \end{aligned} \tag{33}$$

Because the noise for each sub-ciphertext satisfies condition (7), the theorem is proved. □

Therefore, the correctness of order-preserving property is proved.

5.4. Equality (EQ). Equality operator (=) is a common operator in SQL query, for example; a SQL operation is "SELECT name FROM table WHERE score = 90", which requires equality checks on ciphertext. The existing solution is to support equality checks by using deterministic encryption. The FHOPE scheme can also support equality checks even though some noise has been added, and it does not need to use deterministic encryption. We can employ two methods to implement the equality checks. A simple solution is to remove the random noise of existing ciphertexts in database. But the problem is that this solution needs to modify the ciphertexts in the database to increase the cost of computation; moreover, it makes the ciphertexts in an unsafe state. Hence, we take the second solution. Given a search keyword *score* = 100, to search a ciphertext whose plaintext value is 100 in encrypted database, the following steps need to be executed.

Step 1. The CL uses FHOPE to encrypt the search keyword *score* into a ciphertext $C = (c_1, \ldots, c_n)$ under the key $K(n)$; the random noise is ξ, which is stored in the CL. Then the CL computes the range of the sub-ciphertext c_i $(1 \le i \le n)$ as follows:

$$c_i = Enc_i\left(K\left(n\right), score\right) + Noise_i\left(K\left(n\right), R_i\right) + \xi, \tag{34}$$

and because

$$\begin{aligned} &0 < Noise_i\left(K\left(n\right), R_i\right) \\ &\quad < \left(Enc_i\left(K\left(n\right), score + S\right) - Enc_i\left(K\left(n\right), score\right)\right), \end{aligned} \tag{35}$$

we have

$$\begin{aligned} &Enc_i\left(K\left(n\right), score\right) + \xi < c_i \\ &\quad < Enc_i\left(K\left(n\right), score + S\right) + \xi, \end{aligned} \tag{36}$$

and the range $(Enc_i(K(n), score) + \xi, Enc_i(K(n), score + S) + \xi)$ of the sub-ciphertext c_i to the CS is sent.

Step 2. The CS compares the range of the sub-ciphertext c_i with the existing i-th sub-ciphertexts of ciphertexts in encrypted database, if an existing i-th sub-ciphertext falls within the range of the sub-ciphertext c_i, it has the same plaintext value 100 as the search keyword *score*, and so they are equal. The CS sends the search result to the CL.

Therefore, the second solution implements equality checks while ensuring the security. Because the equality check is based on the order comparison, the order comparison is correct; then the equality check is correct.

5.5. Data Interoperability (DI). As that shown below, the proposed scheme can also provide efficient operators with data interoperability. The data interoperability has the following two characteristics: (1) different operators share the same encryption scheme; (2) the output of an operator can be taken as input of another. With the data interoperability, these operators (×, +, −, =, >, and <) can be combined to formulate complex expressions in SQL queries (e.g., SELECT ∗ FROM Table 1 WHERE $col\_C + col\_A \times col\_B < 10000$). The FHOPE scheme guarantees data interoperability according to the following theorem.

Theorem 8. *The FHOPE scheme has the property of the data interoperability; i.e., if $v_1 + v_2 \times v_3 > v_4$, then $C_1 + C_2 \times C_3 > C_4$.*

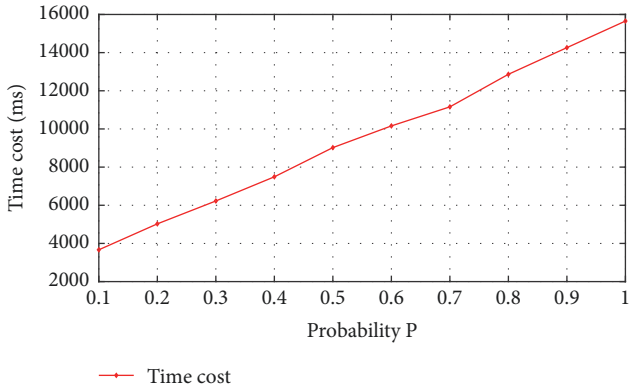

Figure 3: The relationship between time cost and probability.

Proof. To prove this theorem, we need to show $FHOPE(v_1 + v_2 \times v_3, K(n)) > FHOPE(v_4, K(n))$. According to Theorems 4, 5, and 7, we have

$$\begin{aligned} &FHOPE\left(v_1 + v_2 \times v_3, K\left(n\right)\right) > FHOPE\left(v_4, K\left(n\right)\right) \longrightarrow \\ &FHOPE\left(v_1, K\left(n\right)\right) + FHOPE\left(v_2 \times v_3, K\left(n\right)\right) \\ &\quad > FHOPE\left(v_4, K\left(n\right)\right) \longrightarrow \\ &FHOPE\left(v_1, K\left(n\right)\right) + FHOPE\left(v_2, K\left(n\right)\right) \\ &\quad \times FHOPE\left(v_3, K\left(n\right)\right) > FHOPE\left(v_4, K\left(n\right)\right) \longrightarrow \\ &C_1 + C_2 \times C_3 > C_4. \end{aligned} \tag{37}$$

□

The correctness of data interoperability is proved.

5.6. Improving Efficiency. In encryption algorithm Enc, some random noise ξ_i has been added in each sub-ciphertext to augment the security of FHOPE scheme. However, this process reduces the efficiency of the scheme. Here, we use two measures to improve efficiency. One approach is that two sub-ciphertexts in a ciphertext have order-preserving function. For example, the plaintext v_1 is encrypted into $C_1 = (c_1^1, \ldots, c_n^1)$ under the key $K(n)$, each sub-ciphertext is a ciphertext of the plaintext, and the sub-ciphertexts are independent of each other, so any sub-ciphertext of the n sub-ciphertexts has order-preserving property; it means that the ciphertext has order-preserving property. Then, we can specify that the first sub-ciphertext (c_i^1) of the n sub-ciphertexts has order-preserving property. Another approach is to add random noise ξ_i with a certain probability P. Figure 3 shows that, with the growth of probability P, the time cost increases while the length of the plaintext is fixed. We will analyze the relationship between probability P and the time cost of inserting a ciphertext.

The cost of inserting a ciphertext includes encrypting the plaintext to be inserted, inserting the ciphertext, and updating random noise. Assume that the length of plaintext v is m and the plaintext v is encrypted into C. Let t_e, t_i, and t_u denote the time of encrypting, the time of inserting operation, and the time of updating random noise, respectively. For

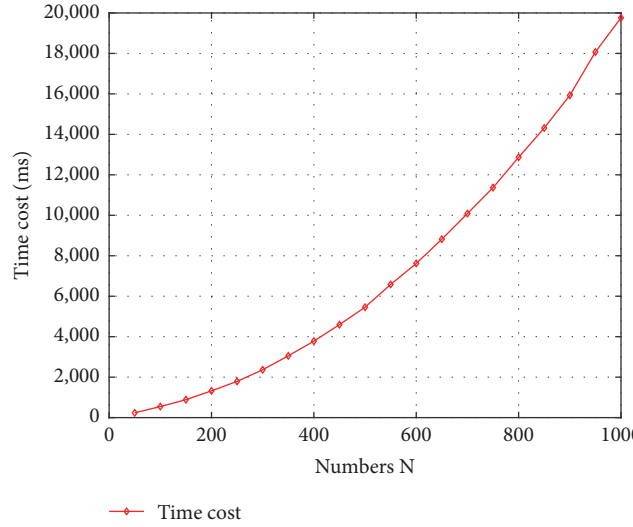

Figure 4: The relationship between time cost and numbers.

n ciphertexts of $attu$ stored in the database, the probability of adding random noise in the ciphertext is P. Then, the time for inserting a ciphertext C is T:

$$T = t_e + t_i + n \times t_u \times P, \tag{38}$$

where $n \times t_u \times P$ is generated by inserting the new random noise; it is the time of updating the existing random noise. When we insert N encrypted values into the database, we have

$$\begin{aligned} k &= 1, \\ T\left(1\right) &= t_e + t_i + n \times t_u \times P, \end{aligned} \tag{39}$$

$$\begin{aligned} k &= 2, \\ T\left(2\right) &= T\left(1\right) + t_e + t_i + (n+1) \times t_u \times P, \end{aligned} \tag{40}$$

$$\begin{aligned} k &= 3, \\ T\left(3\right) &= T\left(2\right) + t_e + t_i + (n+2) \times t_u \times P, \end{aligned} \tag{41}$$

$$\cdots \tag{42}$$

$$\begin{aligned} k &= N, \\ T\left(k\right) &= T\left(k-1\right) + t_e + t_i + (n+k-1) \times t_u \times P. \end{aligned} \tag{43}$$

Then,

$$\begin{aligned} T\left(N\right) = N \times \left(t_e + t_i\right) + \left(n \times N + 0.5 \times N \times (N-1)\right) \\ \times t_u \times P. \end{aligned} \tag{44}$$

According to (44), we can infer the approximate linear relationship between total time $T(N)$ and probability P when N is fixed, and it is consistent with Figure 3. Figure 4 shows that the total time $T(N)$ is exponentially related to N when probability P is fixed. For efficiency, we can conclude that security and efficiency are contradictory; the higher the security, the lower the efficiency.

6. Using FHOPE in a Database Application

The FHOPE is mainly used in the database; in this section, we describe how to use FHOPE in a database. As mentioned in Section 5, FHOPE allows efficient addition, multiplication, order comparison, and equality checks computations on an encrypted database in the same way as on unencrypted database, and the database server software does not need to be modified.

Setup. Using FHOPE in a database requires the following setup:

(i) A CL uses the FHOPE to encrypt the sensitive data, and the encrypted data is outsourced to be stored in a cloud database. The CL stores the private keys.

(ii) User-defined functions (UDFs) in the database server implement FHOPE's computation function.

Insert Queries. To understand how values in a query are encrypted, consider an application that wants to execute the query INSERT INTO student VALUES (10). The CL encrypts 10 using the FHOPE and issues the query INSERT INTO student VALUES (FHOPE(10)), where FHOPE() is a user-defined function that implements the encryption of the FHOPE scheme.

Select Queries. Consider a query: SELECT $*$ FROM T WHERE $col_1 \times col_2 + 1000 > 6800$. col_1 and col_2 denote the sensitive columns in a table T; their values are encrypted and stored in a cloud database. The CL encrypts 1000 and 6800 using the FHOPE, and the values of col_1 and col_2, 1000 and 6800, share the same private key and random noise ξ. col_1 and col_2 are encrypted as col_1c and col_2c, respectively. It delivers the query "SELECT $*$ FROM T WHERE $col_1c \times col_2c + FHOPE(1000) > FHOPE(6800)$" to a CS. The CS executes the query on encrypted data as if the data were not encrypted and returns the query results to CL. The CL decrypts the query results and returns them to the applications.

7. Security Analysis

The security analysis of the FHOPE scheme focuses on the security of the key $K(n)$, IND-HOCPA (indistinguishability under a homomorphic order-preserving chosen-plaintext attack) security and the security of FHOPE scheme. Assume that a CL sends the SQL query to a CS via a secure channel. First, we will prove that it is difficult to recover the secret component $\overrightarrow{k_i}$ in a key $K(n)$ from ciphertexts. Then, based on the difficulty of the key $K(n)$ recovery problem, we prove the IND-HOCPA security of the scheme. We present the privacy protection in queries. Finally, we demonstrate that the security of FHOPE's properties is guaranteed by the security of key $K(n)$ and IND-HOCPA security of FHOPE.

7.1. Security of the Key $K(n)$. The hardness of the key search problem is based on the approximate greatest common divisors (AGCD) problem. The AGCD problem was proposed by Howgrave-Graham [39]. Given any number of the approximate multiples $d_i = h*q_i+l_i$ of h, where h, q_i, and l_i are integers, the problem is to find the hidden common divisor h. Note that q_i and l_i change in each d_i. In particular, if l_i can be as large as h, it is impossible to reconstruct h from any number of approximate multiples d_i [40].

As we know, $K(n) = [\overrightarrow{k_1}, \overrightarrow{k_2}, \ldots, \overrightarrow{k_n}]$, where $\overrightarrow{k_i} = (a_i, b_i)$ is a secret vector. In the following, we prove that it is hard to recover the secret component $\overrightarrow{k_i}$ in a key $K(n)$ from any number of ciphertexts.

Theorem 9. *Given any number of ciphertexts from the FHOPE encryption with $K(n)$, it is difficult to recover $\overrightarrow{k_i}$ in a key $K(n)$.*

Proof. As shown in the FHOPE encryption, a ciphertext $C = (c_1, \ldots, c_n)$ is defined as

$$\begin{aligned} c_1 &= a_1 * b_1 * v + Noise_1\left(K(n), R_1\right) + \xi_1, \\ &\cdots \\ c_n &= a_n * b_n * v + Noise_n\left(K(n), R_n\right). \end{aligned} \tag{45}$$

In the first ciphertext element c_1, a_1 is the common divisor to be recovered. We are going to prove that it is difficult to find the secret value a_1 from the first element c_1 of any number of ciphertexts.

Let $N_1 = Noise_1(K(n), R_1) + \xi_1$. Then, we have $c_1 = a_1 * b_1 * v + N_1$. Since $b_1 * v$ is random number generated for each encryption, N_1 is a number that the adversary does not know, and it randomly changes for each encryption of the plaintext. Moreover, a_1 can be less than N_1. Hence, it is difficult to recover a_1 from the first element c_1 of any number of ciphertexts according to the hardness of the AGCD problem. The proofs for other secret values a_i and b_i in $K(n)$ are carried out similarly. □

7.2. IND-HOCPA Security. We analyze the semantic security of the FHOPE scheme by proving the indistinguishability of ciphertexts under a homomorphic order-preserving chosen-plaintext attack.

Theorem 10. *A FHOPE encryption scheme is IND-HOCPA secure.*

Proof. In the following game, the PPT adversary is denoted as A and the challenger is denoted as CH. Consider any adversary A and any two sequences of values A ask for in the security game: $v^0 = (v_1^0, \ldots, v_n^0)$ and $v^1 = (v_1^1, \ldots, v_n^1)$.

(1) The key generation algorithm generates the key $K(n) = [\overrightarrow{k_1}, \overrightarrow{k_2}, \ldots, \overrightarrow{k_n}] = [(a_1, b_1), (a_2, b_1), \ldots, (a_n, b_n)]$, where $(a_i, b_i)(1 \le i \le n)$ is a list of pairs of integers, which are large prime numbers, $a_i * b_i > 0$, $n > 1$, $a_i \neq 0$ for $1 \le i \le n, b_1 + \cdots + b_{n-1} \neq 0$, and $b_n \neq 0$.

(2) The adversary A chooses two equal-length sequences of values v^0 and v^1 and sends them to the challenger CH.

(3) The challenger CH randomly encrypts v_i^0 and v_i^1 with key $K(n)$ and outputs the ciphertext $C_0 = (c_{01}, \ldots, c_{0n})$

or $C_1 = (c_{11}, \ldots, c_{1n})$, where $c_{01} = a_1 * b_1 * v_i^0 + Noise_1(a_1, r_1, p_1) + \xi_0$, $c_{11} = a_1 * b_1 * v_i^1 + Noise_1(a_1, r_1, p_1) + \xi_1$, and ξ_b(b $\in$ {0, 1})$\in$ $[-\infty, +\infty]$. The ciphertext C_b (b $\in$ {0, 1}) is sent to the adversary A.

(4) If $v_i^0 \neq 0$ and $v_i^1 \neq 0$, then the expressions of $a_1 * b_1 * v_i^0 + Noise_1(a_1, r_1, p_1) + \xi_0$ and $a_1 * b_1 * v_i^1 + Noise_1(a_1, r_1, p_1) + \xi_1$ generate the same value from $-\infty$ to $+\infty$ with the same probability; since ξ_b is randomly sampled from the range $[-\infty, +\infty]$, v^0 and v^1 have the same order relation. Hence, the advantage $Adv_{FHOPE,A}^{IND-HOCPA}(k)$ of distinguishing is negligible.

In the following, we discuss the case where $v_i^0 = 0$ and $v_i^1 \neq 0$. The case where $v_i^0 \neq 0$ and $v_i^1 = 0$ is similar.

If $v_i^0 = 0$ and $v_i^1 \neq 0$, we have $c_{01} = Noise_1(a_1, r_1, p_1) + \xi_0$ or $c_{11} = a_1 * b_1 * v_i^1 + Noise_1(a_1, r_1, p_1) + \xi_1$; depending on whether v_i^0 or v_i^1 is encrypted, v^0 and v^1 have the same order relation. Then, the advantage $Adv_{FHOPE,A}^{IND-HOCPA}(k)$ is negligible.

The proofs for other sub-ciphertexts c_{0i} and c_{1i} $(2 \leq i \leq n)$ are carried out similarly; the advantage $Adv_{FHOPE,A}^{IND-HOCPA}(k)$ is negligible. □

Therefore, the adversary A cannot win the above game, and hence a FHOPE is IND-HOCPA secure.

7.3. Privacy Protection in Queries. The adversary can collect some useful statistical information after receiving query requests; it tries to guess the plaintext corresponding to the ciphertext based on statistical information. However, we will describe that the FHOPE scheme can reduce the privacy leakage greatly in this scenario.

To solve the mentioned problem, we add some random noise in each sub-ciphertext. Let ξ_i denote the random noise, which is randomly sampled from the range $[-\infty, +\infty]$, and *attu* denote the column attribute of the database table. Then, the sub-ciphertext c_i^1 $(1 \leq i \leq n)$ of v_1 is expressed as $Enc_i(K(n), v_1) + Noise_i(K(n), R_i^1) + \xi_1$; the range of noise is different for different input values. Suppose ξ (default value is 0 if there are no sub-ciphertexts of *attu* stored on cloud server) denotes the latest noise. To store c_i^1 in the cloud server, the following steps need to be executed (if no sub-ciphertexts of *attu* are stored, jump to Step 2).

Step 1. Update all the stored sub-ciphertexts (c_i^m) of *attu* by $c_i^m = c_i^m + \xi_1 - \xi$.

Step 2. Add random noise ξ_1 in the new sub-ciphertexts by $c_i^1 = c_i^1 + \xi_1$.

Step 3. Update the value of ξ by $\xi = \xi_1$.

Therefore, due to adding random noise, the ciphertext value is random. The same plaintexts are mapped to different ciphertexts. The random noise of the same attribute is continuously updated with the insertion of new data in the cloud database, and the adversary cannot guess the random noise.

Consider a query SELECT col_1 FROM T WHERE $col_1 >$ 100. The col_1 denotes the sensitive column in a table T; their values are encrypted and stored in a cloud database. The CL encrypts 100 using the FHOPE, and the values of col_1, 100, share the same private key and random noise ξ. And col_1 is encrypted as *col*1*c*. It delivers the query "SELECT *col*1*c* FROM T WHERE *col*1*c* > *FHOPE*(100)" to a CS. Since the data is in the encrypted form and the random noise of each ciphertext is different, the adversary cannot get any knowledge of the order information. The random noise of the same attribute is continuously updated with the insertion of new data in the cloud database (that is, the random noise of the same attribute is the same), and then the order of plaintexts remains in the ciphertexts in the cloud database. Then, the CS executes the query on encrypted data as if the data were not encrypted, and the adversary obtains the query results. Since the random noise is dynamically updated, the order of query results loses freshness. In other words, the repeated query is issued again, and the returned ciphertexts are different. Moreover, we used the restrictions of literature [10] for chosen-plaintext attack, even if the adversary can get the ciphertext of $\{v_1, v_2, \ldots, v_k\}$, where $\{v_1, v_2, \ldots, v_k\}$ is a dense one, but the ciphertexts are disordered because they are obtained at different time. Therefore, previous query requests will not help the adversary to learn the privacy information, and the adversary cannot gradually find out the order information and get some useful statistical information after many query requests.

7.4. The Security of FHOPE Properties. Our security model securely realizes ideal properties in the presence of noncolluding semihonest adversary. For the sake of simplicity, we do it for the specific scenario of our properties, which involves CL and CS. We need to construct simulator Sim_{CS} against adversary A_{CS} that corrupts CS.

Theorem 11. *The AD can securely perform addition operation on ciphertext in the presence of semihonest adversary* A_{CS}.

Proof. CL receives plaintexts v and v' as input and then generates ciphertexts C of v and C' of v'. Finally, C and C' are returned to Sim_{CS}.

Sim_{CS} simulates A_{CS} as follows: it receives C and C' as input and generates the sum of C and C' by performing addition operation. Sim_{CS} sends the sum of C and C' to A_{CS}.

The A_{CS}'s view contains encrypted data. In the real and ideal executions, the views of A_{CS} are indistinguishable, because CL is trusted and the FHOPE is IND-HOCPA secure. □

The security proofs of MU, OC, EQ, and DI are similar to that of AD under the semihonest adversary A_{CS}. We give only the theorems here.

Theorem 12. *The MU can securely perform multiplication operation on ciphertext in the presence of semihonest adversary* A_{CS}.

Theorem 13. *The OC can securely perform order comparison on ciphertext in the presence of semihonest adversary* A_{CS}.

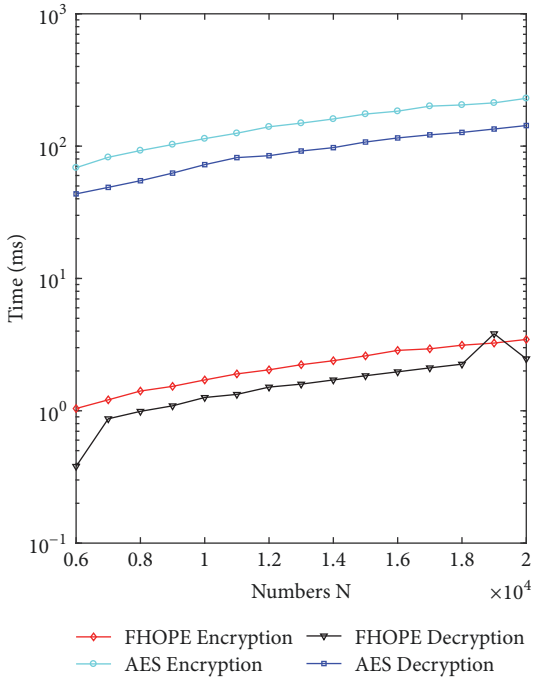

Figure 5: Performance of encryption and decryption.

Theorem 14. *The EQ can securely perform equality checks on ciphertext in the presence of semihonest adversary A_{CS}.*

Theorem 15. *The DI can securely perform complex operation on ciphertext in the presence of semihonest adversary A_{CS}.*

8. Evaluations

The section focuses on the testing of the FHOPE's performance. We design four experiments to test its performance. Simultaneously, the correctness of FHOPE's properties (such as additive homomorphism, multiplicative homomorphism, order-preserving, and data interoperability) is also checked in these experiments.

The experiments configuration is under CentOS Linux with an Intel Xeon CPU E3-1226 Processor (3.3GHz) and the 16.0GB RAM, which has 4 processor cores. The prototype is built based on the architecture shown in Figure 2. We implement the proposed prototype using Java language and MySQL 5.6. In our experiment, the secret key $K(n)$ is configured to have $n = 6$, with selection of a list of pairs of integers $[(a_1, b_1), (a_2, b_2),(a_3, b_3), (a_4, b_4), (a_5, b_5), (a_6, b_6)]$, $[(r_1, p_1), (r_2, p_2), (r_3, p_3), (r_4, p_4), (r_5, p_5), (r_6, p_6)]$, and ξ_i, where $a_i * b_i > 0$, $a_i \neq 0$ for $1 \leq i \leq 6$, $b_1 + \cdots + b_5 \neq 0$, $b_6 \neq 0$, $p_1 > -a_2 * b_1 * S$, $p_6 < a_1 * b_1 * S$, $r_1 < a_1 * b_1 * S$, and $r_6 < a_6 * b_1 * S$. And a simple synthetic dataset is a table *Tab* with three sensitive columns *A*, *B*, and *C*, which has 1 million records. The values in each column are randomly generated integers.

8.1. Performance of Encryption and Decryption. The experiment shows the performance of encryption and decryption by comparing FHOPE scheme with the AES algorithm. In our experiment, we randomly generate 20000 integers, each of which has 6 digits. Then, we compare our scheme with the AES algorithm by testing the time cost of encryption and decryption. According to Figure 5, the time cost of AES's encryption and decryption exhibits exponential growth with respect to the number of integers; it costs 230 milliseconds to encrypt 20000 integers. And the FHOPE scheme costs 4 milliseconds to encrypt 20000 integers, which is about 57 times faster than AES algorithm for encryption. We can see that FHOPE scheme is also faster than AES for decryption. Thus our scheme is practically efficient and suitable for large batch of data encryption and decryption.

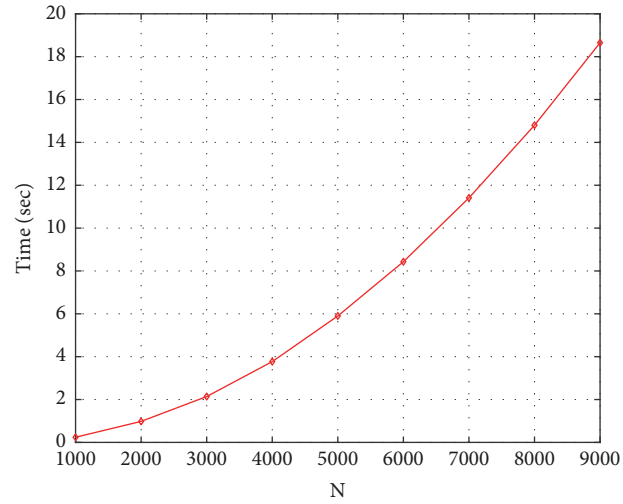

Figure 6: Time for running $\sum_{i=1}^{N} x^i$ over encrypted x.

8.2. Performance of Homomorphic Operations. The FHOPE scheme has additive homomorphism, multiplicative homomorphism, and data interoperability. The polynomial evaluation can demonstrate these properties of the FHOPE scheme. Then, we test the performance of addition and multiplication with high-degree polynomials over ciphertexts. The polynomial is $\sum_{i=1}^{N} x^i$, where x is the encryption of a randomly generated integers and has eight digits.

Figure 6 shows the cost of testing $\sum_{i=1}^{N} x^i$ from $N = 1000$ to $N = 9000$. The experiment result shows the efficiency of FHOPE scheme for performing many addition and multiplication operations. For instance, the addition and multiplication operations for calculating $\sum_{i=1}^{N} x^i$ take about 0.2 seconds while $N = 1000$ and about 18.7 seconds while $N = 9000$. The correctness of homomorphic addition, homomorphic multiplication, and data interoperability is also checked in the experiment.

8.3. Comparison with mOPE [24]. For evaluating the performance of the FHOPE's order-preserving, we compare the FHOPE scheme with mOPE scheme using a simple synthetic dataset on which data insertion is executed. In order to test the performance of data insertion, we generate N ($N \in [500, 6000]$) records and insert them into an encrypted database as shown in Figure 7.

From Figure 7, we can see that the mOPE scheme has the lowest performance. The FHOPE scheme is more efficient

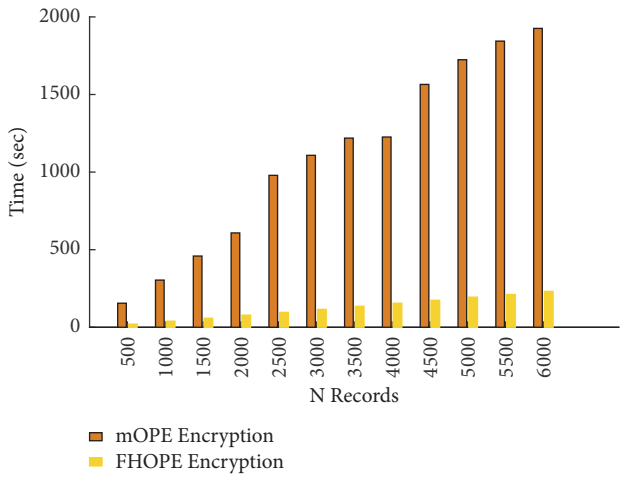

Figure 7: Comparison between FHOPE scheme and mOPE scheme.

than the mOPE scheme. In mOPE scheme, the client and the server side need to interact with each other when encrypting a message, and the server needs to adjust the encoding tree to achieve balance when adding new nodes. Our scheme is constructed by some linear mathematical functions without any interaction, and it has a higher efficiency.

8.4. Comparison with CryptDB [35]. For testing, we select 10000 records randomly from table *Tab*. Our scheme is compared with CryptDB by executing three queries.

[Range]: select A from *Tab* where $A < q$.

[Sum]: select $sum(A)$ from *Tab* where $A < q$.

[Avg]: select $avg(B)$ from *Tab* where $A < q$.

Let q control the queries' selectivity, which is randomly sampled from the range $[100, 10000]$. Figure 8 shows the time cost of FHOPE and CryptDB for performing the three queries, where the size of the table grows from $1K$ to $8K$ rows. The time cost is displayed as bar graph.

We can get some observations from the experimental result. (1) CryptDB takes more time to execute the range queries than FHOPE, because the CryptDB uses mOPE to implement the comparison operations. The efficiency of mOPE is lower than FHOPE as shown in Figure 7. (2) For the sum and avg queries, the execution time of FHOPE is lower than CryptDB, because CryptDB employs Paillier's homomorphic encryption scheme and UDFs (user-defined functions) to compute sum aggregates and averages. The low efficiency of Paillier's homomorphic encryption scheme leads to the low efficiency of CryptDB. (3) To perform range, sum, and avg queries, CryptDB employs various encryption schemes. FHOPE makes up for CryptDB's shortcoming, which can support homomorphic addition, homomorphic multiplication, order comparison, and equality checks.

Moreover, CryptDB cannot support some operations, such as "$A + B < q$" and homomorphic multiplication. Therefore, CryptDB cannot support some complex SQL queries, for example, (1) query "select $sum(A * B)$ from *Tab* where $A < q$", (2) query "select A from *Tab* where $A * B < 1000$", (3) query "select B from *Tab* where $A * (A + B) > 100$", and (4) query "select B from *Tab* where $A * B - 100 > 10$". The FHOPE can support the above complex SQL queries. Since FHOPE can support addition, multiplication, order comparison, and equality checks and the FHOPE scheme has the property of the data interoperability, with the data interoperability, these operators (+, −, ×, <, >, and =) can be combined to formulate complex expressions (e.g., "$A + B * C < q$") in SQL queries. Then, the FHOPE can support complex SQL queries.

The time cost of FHOPE and CryptDB for the range queries while we change the selectivity of the queries (by adjusting q) from 10% to 90% is shown in Figure 9. The queries time of FHOPE is less than that of CryptDB. From Figure 9, we can see that the queries overhead of FHOPE is approximate linear growth as the selectivity of the queries increases; it indicates that the FHOPE scheme has a good stability.

9. Conclusion

In this paper, we have presented a novel FHOPE scheme that can support direct homomorphic addition, homomorphic multiplication, order comparison, and equality checks on the ciphertext. The FHOPE scheme can be applied in a cloud database environment, which still uses standard SQL statements and allows the cloud server to perform complex SQL queries over the encrypted data without repeated encryption. We have proved the security of our FHOPE scheme from four aspects: the security of the key $K(n)$, IND-HOCPA security, the privacy protection in queries, and the security of FHOPE properties. The security of the key $K(n)$ is based on the AGCD problem. Moreover, we have implemented a prototype in Java and evaluated the performance of our scheme in terms of encryption, decryption, and homomorphic operations, and our scheme is compared with mOPE scheme and CryptDB. Through experiment, we prove that the FHOPE scheme incurs less overhead on computation. It is suitable for large batch of data encryption and decryption in cloud database systems.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61572263, Grant 61502251, Grant 61502243, and Grant 61602263, the Natural Science Foundation of Jiangsu Province under Grant BK20161516 and Grant BK20151511, the Natural Science Foundation of Anhui Province under Grant 1608085MF127, the Natural Science Foundation of Educational Commission of Anhui Province of China under Grant KJ2016B17,

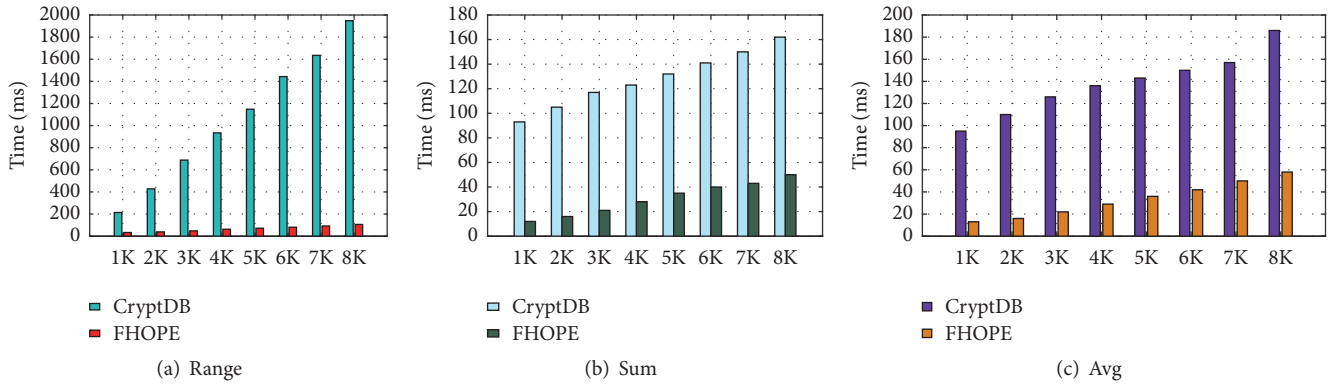

Figure 8: Execution times of FHOPE and CryptDB for the three sample queries.

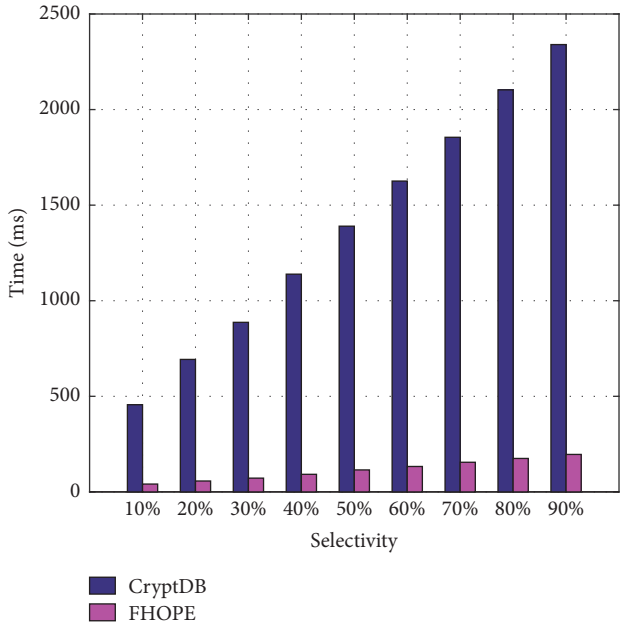

Figure 9: Execution times versus selectivity.

Grant KJ2015B19, and Grant KJ2017B15, China Postdoctoral Science Foundation under Grant 2016M601859 and Grant 2015M581794, Qing Lan Project of Jiangsu Province, 1311 Talent Plan Foundation of NUPT, NUPTSF, under Grant NY216001, and the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under Grant 14KJB520031 and Grant 15KJB520027.

References

[1] H. Wang, D. He, J. Yu, and Z. Wang, "Incentive and Unconditionally Anonymous Identity-Based Public Provable Data Possession," *IEEE Transactions on Services Computing*, pp. 1-1.

[2] H. Wang, D. He, and J. Han, "VOD-ADAC: Anonymous Distributed Fine-Grained Access Control Protocol with Verifiable Outsourced Decryption in Public Cloud," *IEEE Transactions on Services Computing*, pp. 1-1.

[3] W. Li, K. Xue, Y. Xue, and J. Hong, "TMACS: A Robust and Verifiable Threshold Multi-Authority Access Control System in Public Cloud Storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1484–1496, 2016.

[4] K. Xue, Y. Xue, J. Hong et al., "RAAC: Robust and Auditable Access Control with Multiple Attribute Authorities for Public Cloud Storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 953–967, 2017.

[5] R. Agrawal, J. Kiernan, R. Srikant, and Y. R. Xu, "Order preserving encryption for numeric data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*, pp. 563–574, ACM, Paris, France, June 2004.

[6] K. Li, W. Zhang, C. Yang, and N. Yu, "Security Analysis on One-to-Many Order Preserving Encryption-Based Cloud Data Search," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 9, pp. 1918–1926, 2015.

[7] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM symposium on Theory of Computing (STOC '09)*, vol. 9, pp. 169–178, ACM, Bethesda, Md, USA, 2009.

[8] X. Liu, K. R. Choo, R. H. Deng, R. Lu, and J. Weng, "Efficient and Privacy-Preserving Outsourced Calculation of Rational Numbers," *IEEE Transactions on Dependable and Secure Computing*, vol. 99, 2016.

[9] D. Liu and S. Wang, "Nonlinear order preserving index for encrypted database query in service cloud environments," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 13, pp. 1967–1984, 2013.

[10] Z. Liu, X. Chen, J. Yang, C. Jia, and I. You, "New order preserving encryption model for outsourced databases in cloud environments," *Journal of Network and Computer Applications*, vol. 59, pp. 198–207, 2016.

[11] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Proceedings of the 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 465–482, 2012.

[12] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Exploring the feasibility of fully homomorphic encryption," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 64, no. 3, pp. 698–706, 2015.

[13] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Advances in Cryptology-EUROCRYPT 2009*, vol. 5479, pp. 224–241, Springer, Berlin, Germany, 2009.

[14] H. Wang, D. He, and S. Tang, "Identity-Based Proxy-Oriented Data Uploading and Remote Data Integrity Checking in Public

Cloud," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1165–1176, 2016.

[15] H. Wang, "Identity-based distributed provable data possession in multi-cloud storage," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 328–340, 2015.

[16] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *Foundations of Computer Science IEEE*, vol. 2011, pp. 97–106, 2011.

[17] N. Smart and F. Vercauteren, "Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes," in *Proceedings of the International Conference on Practice and Theory in Public Key Cryptography Springer-Verlag*, vol. 6056, pp. 420–443, 2010.

[18] Y. n. Doröz, Y. Hu, and B. Sunar, "Homomorphic AES evaluation using the modified LTV scheme," *Designs, Codes and Cryptography. An International Journal*, vol. 80, no. 2, pp. 333–358, 2016.

[19] J. H. Cheon, J.-S. Coron, J. Kim et al., "Batch fully homomorphic encryption over the integers," in *Proceedings of the Advances in Cryptology-EUROCRYPT 2013*, vol. 7881, pp. 315–335, Springer, New York, NY, USA, 2013.

[20] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, and N. Hanley, "Optimised multiplication architectures for accelerating fully homomorphic encryption," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 65, no. 9, pp. 2794–2806, 2016.

[21] J. H. Cheon and D. Stehl'E, "Fully homomophic encryption over the integers revisited," in *Advances in Cryptology-EUROCRYPT 2015*, vol. 9056, pp. 513–536, Springer, New York, NY, USA, 2015.

[22] L. Xiao, I.-L. Yen, and D. T. Huynh, "A note for the ideal order-preserving encryption object and generalized order-preserving encryption," in *IACR Cryptology Eprint Archive*, 2012.

[23] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: improved security analysis and alternative solutions," in *Proceedings of the 31st annual conference on Advances in cryptology*, vol. 6841, pp. 578–595, Springer-Verlag, 2011.

[24] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *Proceedings of the 34th IEEE Symposium on Security and Privacy, SP 2013*, pp. 463–477, May 2013.

[25] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '02)*, pp. 216–227, New York, NY, USA, June 2002.

[26] R. Popa, N. Zeldovich, and H. Balakrishnan, "CryptDB: A Practical Encrypted Relational DBMS," Tech. Rep. MIT-CSAIL-TR-2011-005, MIT, 2011.

[27] J. Li, Z. Liu, X. Chen, F. Xhafa, X. Tan, and D. S. Wong, "L-EncDB: A lightweight framework for privacy-preserving data queries in cloud computing," *Knowledge-Based Systems*, vol. 79, pp. 18–26, 2015.

[28] Z. Yang, S. Zhong, and R. Wright, "Privacy-preserving queries on encrypted data," in *Proceedings of the Computer Security CESORICS*, pp. 479–495, 2006.

[29] B. Hore, S. Mehrotra, and G. Tsudik, "A privacypreserving index for range queries," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB'04*, vol. 30, pp. 720–731, 2004.

[30] K. Xue, S. Li, J. Hong, Y. Xue, N. Yu, and P. Hong, "Two-Cloud Secure Database for Numeric-Related SQL Range Queries with Privacy Preserving," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 7, pp. 1596–1608, 2017.

[31] W. Ding, Z. Yan, and R. H. Deng, "Encrypted data processing with Homomorphic Re-Encryption," *Information Sciences*, vol. 409-410, pp. 35–55, 2017.

[32] X. Liu, R. H. Deng, W. DIng, R. Lu, and B. Qin, "Privacy-preserving outsourced calculation on floating point numbers," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 11, pp. 2513–2527, 2017.

[33] Z. Yan, W. Ding, V. Niemi, and A. V. Vasilakos, "Two schemes of privacy-preserving trust evaluation," *Future Generation Computer Systems*, vol. 62, pp. 175–189, 2015.

[34] A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently outsourcing multiparty computation under multiple keys," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 12, pp. 2046–2058, 2013.

[35] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles, SOSP 2011*, pp. 85–100, prt, October 2011.

[36] W. K. Wong, B. Kao, D. W. L. Cheung, R. Li, and S. M. Yiu, "Secure query processing with data interoperability in a cloud database environment," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD 2014*, pp. 1395–1406, June 2014.

[37] D. Liu, *Homomorphic Encryption for Database Querying:Australian*, WO/2013/188929, 2013.

[38] F. D. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," in *Proceedings of the 35th SIGMOD international conference on Management of data, SIGMOD'09*, pp. 19–30, Providence, Rhode Island, USA, June 2009.

[39] J. H. Silverman, "Cryptography and lattices," in *Lecture Notes in Computer Science*, vol. 2146 of *chapter Approximate Integer Common Divisors*, pp. 51–66, 2001.

[40] H. Cohn and N. Heninger, "Approximate common divisors via lattices," in *IACR Cryptology Eprint Archive*, 2011.