J Inf Process Syst, Vol.13, No.3, pp.573~589, June 2017
https://doi.org/10.3745/JIPS.04.0034

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Query with SUM Aggregate Function on Encrypted Floating-Point Numbers in Cloud

Taipeng Zhu*, Xianxia Zou*, and Jiuhui Pan*

Abstract

Cloud computing is an attractive solution that can provide low cost storage and powerful processing capabilities for government agencies or enterprises of small and medium size. Yet the confidentiality of information should be considered by any organization migrating to cloud, which makes the research on relational database system based on encryption schemes to preserve the integrity and confidentiality of data in cloud be an interesting subject. So far there have been various solutions for realizing SQL queries on encrypted data in cloud without decryption in advance, where generally homomorphic encryption algorithm is applied to support queries with aggregate functions or numerical computation. But the existing homomorphic encryption algorithms cannot encrypt floating-point numbers. So in this paper, we present a mechanism to enable the trusted party to encrypt the floating-points by homomorphic encryption algorithm and partial trusty server to perform summation on their ciphertexts without revealing the data itself. In the first step, we encode floating-point numbers to hide the decimal points and the positive or negative signs. Then, the codes of floating-point numbers are encrypted by homomorphic encryption algorithm and stored as sequences in cloud. Finally, we use the data structure of DoubleListTree to implement the aggregate function of SUM and later do some extra processes to accomplish the summation.

Keywords

Coding Scheme, DoubleListTree, Encryption, Floating-Point Numbers, Summation

1. Introduction

Cloud database services, for example Amazon Relational Database Service and Microsoft SQL Azure, are attractive for enterprises to outsource their databases. Enterprises can get started without purchasing their expected future software and employing DBA, hence cloud can reduce the total cost of ownership [1]. However, the main problem is that parts of the data may be sensitive, such as credit card numbers or other personal information. Storing and processing sensitive data on infrastructure that provided by a third party increases the risk of unauthorized disclosure if the infrastructure is compromised by an adversary [2]. A straightforward approach to addressing the security and privacy problem is to encrypt data before they are sent to cloud. However, after being encrypted, a database might not be easily queried. When a database is large, it is not acceptable to decrypt the entire database

Manuscript received July 1, 2016; first revision November 11, 2016; accepted December 14, 2016.
Corresponding Author: Jiuhui Pan (jhpan_126@126.com)
* Dept. of Computer Science, Jinan University, Guangzhou, China (t.p.zhu@outlook.com, tzouxianxia@jnu.edu.cn, jhpan_126@126.com)

before performing each query because the decryption might be very slow. On the other hand, if the decryption is done on the cloud, the decrypted database is again at the risk of having its security and privacy breached. Ideally, a query should be executed directly over the encrypted database, producing encrypted query result, which can be decrypted by users. The database community, at least for the last decade, has been grappling with querying encrypted data. The CryptDB [3-7] is a system supporting SQL queries over encrypted database, which extends the existing DBMSs to support homomorphic operations like SUM and AVG. MONOMI [8] works by encrypting the entire database and running analytical queries over the encrypted data. A secure query processing system (SDB) on relational tables and a set of elementary operations on encrypted data that allow data interoperability are proposed [9,10]. TrustedDB [11] is a trusted relational database based on a trusted hardware. But these systems except TrustedDB do not query floating-point numbers. One primary cause is that existing encryption algorithms do not support floating-point numbers in which the decimal points are not fixed.

The goal of this paper is to enable database to query with SUM aggregate function on encrypted floating-point numbers without revealing the data itself in cloud. We apply different encrypting algorithms to different queries on floating-point data type. For queries which mainly involve comparison operators like equality checks, equality joins and so on, floating-point numbers could be as strings to be encrypted by a deterministic encryption algorithm, while order preserving encryption could be applied to range queries as well as the MAX, MIN queries [12]. When it comes to queries with arithmetic operators or SUM aggregate, the existing homomorphic encryption algorithms do not natively support. Hence, the focus of this paper is to provide a method of encoding floating-point numbers to hide their decimal points and positive or negative signs in order to meet the encryption algorithms. Then encrypt the codes and query the corresponding encrypted data with SUM aggregate function, which can't be performed in CryptDB [7]. The coding form of floating-point data type is represented by the data structure of DoubleListTree to facilitate implementation of the function of SUM. More concretely, we make the following contributions:

(1) Encoding floating-point numbers: We provide a coding scheme of floating point data type and the arithmetic rules for encoded floating points. After floating-point numbers are encoded to hide decimal points and positive or negative signs, the codes can be encrypted as sequences.
(2) The data structure of DoubleListTree: The DoubleListTree can help to sum the encoded floating-point numbers, and it would not lose the computational accuracy.
(3) The function of SUM: We present an algorithm to implement the function of SUM, which sums directly over the encrypted database. The function of SUM does not decrypt the input, and it can be run by an untrusted party without revealing their data and internal states. It will have great practical implications in the outsourcing of private computations, especially in the context of cloud computing.

The rest of this paper is structured as follows. In Section 2, we discuss the related work. Then, we describe the coding scheme of floating point data type and the arithmetic rules for encoded floating points in Section 3. The floating-point numbers after being encoded and encrypted are stored as sequences in cloud database, and in Section 4, we present the data structure of DoubleListTree, and the implementation of the function of SUM, as well as some extra processes (e.g., revision, adjustment) for summation. Next, we provide performance test in Section 5 and conclude our work in Section 6.

2. Related Work

The early research of executing SQL over encrypted data [13] proposes that SQL is executed over the encrypted data by rewriting a relational algebraic similarly to be executed over the unencrypted data. Decrypting data and complex queries are executed on the client. But there are some limitations on searching and querying on encrypted data, for example, certain queries with joining and sorting are not supported or highly inefficient. Moreover, in order to solve these problems, DBMS is required to be modified or some of queries are performed on the client.

In contrast, CryptDB proposes a database proxy layer to encrypt and decrypt data so that the internal structure of DBMS is not modified, namely CryptDB uses native DBMS [3]. It proposes three key ideas, the first is to execute SQL query on encrypted data, the second is to adopt adjustable encryption strategies for different queries, and the third is to chain encrypted keys to user passwords. More detail in the second idea, CryptDB encrypts data by an 'onion' in which different query is based on different encrypting algorithm. SQL queries, such as equality selection, equality join, order, range join, text searching, SUM of integer data, etc., can be performed. Processing a query in CryptDB involves the following four steps [3]. the application issues a query, which the proxy intercepts and rewrites: it anonymizes each table and column name, and using the master key MK, encrypts each constant in the query with an encryption scheme best suited for the desired operation; the proxy checks if the DBMS server should be given keys to adjust encryption layers before executing the query, and if so, issues an UPDATE query at the DBMS server that invokes a user defined function (UDF) to adjust the encryption layer of the appropriate columns; the proxy forwards the encrypted query to the DBMS server, which execute it using standard SQL (occasionally invoking UDFs for aggregation or keyword search); the DBMS server returns the encrypted query result, which the proxy decrypts and returns to the application.

Tu et al. [8] points out that CryptDB only supports queries including computation and hardly supports analytical query. Therefore, MONOMI is established based on the design of CryptDB, which can process the complex analytical query and large data set. As executing analytical load to encrypted data on the server is very difficult, MONOMI proposes an executing method of splitting client/server. Splitting executing allows MONOMI to execute parts of queries to encrypted data on untrusted server and performed through the scheme the same with CryptDB, while the rest of queries are executed after decrypting data on the trusted client.

Different from CryptDB and MONOMI in which different encrypting algorithms are used to different queries, a SDB in [9] is realized through a group of security operators (e.g., $\times$, $\pm$, π, $\otimes$, $\bowtie_S$) with data interoperation which can efficiently support a quantity of complex SQL query including all TPC-H benchmark queries on the server. In view of the limitations of fully and partially homomorphic encryption, SDB does not adopt homomorphic encryption algorithm adopted in CryptDB and MONIMI instead of using a secret-sharing scheme in SMC model [14] and the solution in ShareMind [15]. Encryption scheme adopted by SDB can support complex operations executed on the server, but it has some limitations. For example, SDB does not natively support operators which their output results are non-integer values, e.g., square root ($\surd$).

Hacigumus et al. [16] thinks that hardware encryption is better than software encryption. As a result, a security coprocessor unit (SCPU) hardware is introduced in TrustedDB. Bajaj and Sion [11] deems

any encryption methods based on software have an inherent defect that expression is limited, so it is best to guarantee the privacy of data through trusted hardware. Hence, TrustedDB provides more secure data protection by SCPU hardware, and supported query types are not limited. Nevertheless, there are some limitations as well, for example, query parser in TrustedDB cannot parse multi-level nested sub-queries and views defined by user [11]. The cost and the performance constraint of TrustedDB is higher than methods of software.

When it comes to floating point data type, CryptDB can encrypt floating-points values, but it cannot perform aggregations on encrypted floating-point values [7]. Neither does the MONOMI system, for it is adopted the same encryption mechanism with CryptDB. As for SDB, its operators are applicable only to data values of integer domains [9], so it's unavailable to floating point data type as well. Therefore, how to support encryption of floating point data type and simple arithmetic computation followed is an issue. If floating-point numbers are encrypted as strings, basic arithmetic calculations on strings are not supported. In addition, if using 3Kdec algorithm to realize numeric to numeric encryption for easily storing [17], there is no indication of enabling the arithmetic calculations on encrypted data as well. The method to expand the homomorphic encryption on the integer to the decimal by 'similar modular arithmetic' is proposed [18]. But there are some defects need to illustrate, for instance, one ciphertext needs to be larger than the other one if its corresponding plaintext is larger than that plaintext for additive homomorphic encryption. Moreover, floating-point numbers need to enlarge to be integers for multiplicative homomorphic encryption, which would make the computational accuracy lose.

3. Encoding and Encrypting Floating-Point Numbers

The existing researches on encryption of floating-point numbers either lose the precision of data or cannot query computation over encrypted data. So the processing mechanism in this paper is to encode floating-point numbers by a coding scheme to ensure the precision and to hide their decimal points and their positive or negative signs, so that they can meet the homomorphic encryption algorithm. Subsequently the encoded values perform summation according to addition rules.

3.1 Coding Scheme

Floating-point numbers can be represented in a variety of ways. Oracle uses the type of FLOAT, DOUBLE and NUMBER for typical floating-point numbers, where the precision of NUMBER type can reach 38 decimal digits. In order to implement the NUMBER data type, Oracle encodes a floating-point number to hide its decimal point and its positive or negative sign, and the floating-point calculations are performed by kernel drivers.

Inspired by this, we also encode floating-point numbers, and encrypt the codes which can be summed directly over the database instead of being decrypted.

Floating-point numbers are encoded as sequences, and a sequence has several components: *tag*, [*number 1*], [*number 2*], …, [*number k*], …, where each '*number n*' is a positive integer and 'tag' indicates whether the floating-point number is positive or negative and its decimal point. The coding rules are described as follows in detail.

(1) Segment: The decimal point of a floating-point number is taken as demarcation point. Starting from the right most side of the integer part, it is divided into groups of *S* decimal digits, and balancing the number of digits by putting zeros. Starting from the left most side of the decimal part, it is divided into groups of *S* decimal digits, and balancing the number of digits by putting zeros. Now we have certain numbers of groups of the floating-point number, and then write in the same order in which they used to be.

(2) The positive or negative sign: 'tag' reveals the positive or negative sign of a floating-point number. If the value of 'tag' is greater than or equal to 193, the sequence represents a positive floating-point number; if the value of 'tag' is less than or equal to 62, the sequence represents a negative floating-point number.

(3) The decimal point: 'tag' also indicates the decimal point. We set the value of 'tag' to *N*, then the number of groups of integer part in a positive floating-point number is *N-193+1*, and the number of groups of integer part in a negative floating-point number is *62-N+1*. The rest groups of the sequence belong to decimal part.

(4) Encode each group: If a floating-point number is positive, each group is encoded as the true value itself; otherwise each group is encoded as 10^S subtracts the true value.

Examples of encoding floating-point numbers in the instance of *S* equals to 2 are as follows:

53021.128: 195, 05, 30, 21, 12, 80

-123.128035: 061, 99, 77, 88, 20, 65

3.2 Addition Rules

The addition of two sequences includes 'Positive + Positive', 'Negative + Negative' and 'Positive + Negative'. Two sequences are added from the lowest order group to the highest order group. Due to a floating-point number is divided into groups of *S* decimal digits, the maximum number of each group is 10^S-1, and the minimum is 0.

(1) 'Positive + Positive'

The result of 'Positive + Positive' must be a positive value, and carry may be produced. The tag value of sum takes the larger tag of two addends. Suppose two characters, *a* and *b*, represent individually the value of one group place of two addends, and the corresponding true value and coding value are shown in Table 1.

Table 1. True value and coding value on type of 'Positive + Positive'

	True value	Coding value
Positive	a	a
Positive	b	b

1) If $a+b \geq 10^S$, it produces carry and the sum which is produced in one group place as result of addition is encoded into $a+b-10^S$, and the carry is propagated to the next high order group. If

the highest group produces the carry, a new group encoded into *1* is added to the sequence of the sum and the tag of the sum pluses *1*.

2) If $a+b<10^S$, it cannot produce carry and the sum which is produced in one group place as result of addition is encoded into $a+b$.

Proof. If $a+b\geq10^S$, the sum which is produced in one group place as result of addition is greater than or equal to 10^S, so it produces carry. The sum is $a+b-10^S$ and is encoded into $a+b-10^S$.

If $a+b<10^S$, the sum which is produced in one group place as result of addition is less than 10^S, so it cannot produce carry. The sum is $a+b$ and is encoded into $a+b$.

(2) 'Negative + Negative'

The result of 'Negative + Negative' must be a negative value, and carry may be produced. The tag value of sum takes the smaller tag of two addends. Suppose two characters, *a* and *b*, represent individually the value of one group place of two addends, and the corresponding true value and coding value are shown in Table 2.

Table 2. True value and coding value on type of 'Negative + Negative'

	True value	Coding value
Negative	a	10^S-a
Negative	b	10^S-b

1) If $(10^S-a)+(10^S-b)>10^S$, it cannot produce carry and the sum which is produced in one group place as result of addition is encoded into $(10^S-a)+(10^S-b)-10^S$.

2) If $(10^S-a)+(10^S-b)\leq10^S$, it produces carry and the sum which is produced in one group place as result of addition is encoded into $(10^S-a)+(10^S-b)$, and the carry is propagated to the next high order group. If the highest group produces the carry, a new group encoded into 10^S-1 is added to the sequence of the sum and the tag of the sum minuses *1*.

Proof. If $(10^S-a)+(10^S-b)>10^S$, namely $a+b<10^S$, the sum which is produced in one group place as result of addition is less than 10^S, so it cannot produce carry. The sum is $a+b$ and is encoded into $10^S-(a+b)$, namely $(10^S-a)+(10^S-b)-10^S$.

If $(10^S-a)+(10^S-b)\leq10^S$, namely $a+b\geq10^S$, the sum which is produced in one group place as result of addition is greater than or equal to 10^S, so it produces carry. The sum is $a+b-10^S$ and is encoded into $10^S-(a+b-10^S)$, namely $(10^S-a)+(10^S-b)$.

(3) 'Positive + Negative'

There are three possible results of 'Positive + Negative': a positive value, a negative value or 0. Carry is impossible to be yielded, but borrow may be produced. Suppose character *a* represents the value of one group place of the Positive, character *b* represents the value of one group place of the Negative, and t_1, t_2 is the value of the two tags. These are shown in Table 3.

Table 3. Tag value, true value and coding value on type of 'Positive + Negative'

	Tag value	True value	Coding value
Positive	t_1	a	a
Negative	t_2	b	10^S-b

1) If $t_1+t_2>255$, the result is a positive value, and the tag of the sum takes t_1.
   a) If $a+(10^S-b)\geq 10^S$, it does not require borrow and the sum which is produced in one group place as result of addition is encoded into $a+(10^S-b)-10^S$.
   b) If $a+(10^S-b)<10^S$, it requires borrow and the sum which is produced in one group place as result of addition is encoded into $a+(10^S-b)$, and the borrow is extorted from the next high order group. If the highest group equals to *0*, it would be removed from the sequence of the sum and the tag of the sum minuses *1*, then repeat the judgment.
2) If $t_1+t_2<255$, the result is a negative value, and the tag of the sum takes t_2.
   a) If $a+(10^S-b)\leq 10^S$, it does not require borrow and the sum which is produced in one group place as result of addition is encoded into $a+(10^S-b)$.
   b) If $a+(10^S-b)>10^S$, it requires borrow and the sum which is produced in one group place as result of addition is encoded into $a+(10^S-b)-10^S$, and the borrow is extorted from the next high order group. If the highest group equals to 10^S, it would be removed from the sequence of the sum and the tag of the sum pluses *1*, then repeat the judgment.
3) If $t_1+t_2=255$, the positive or negative sign of the result is uncertain, so the following judgment shall be made sequentially from the highest group to the lowest group.
   a) If $a+(10^S-b)>10^S$, the result is a positive value, the next process is the same as 1).
   b) If $a+(10^S-b)<10^S$, the result is a negative value, the next process is the same as 2).
   c) If $a+(10^S-b)=10^S$, then same judgment shall be continuously made to the next low order group. If this satisfies for all groups, two numbers represented by the two sequences are opposite numbers, and the result is *0*, corresponding coding adopts the coding of *+0*, namely *193, 0*.

Proof. If $t_1+t_2>255$, namely $t_1-193>62-t_2$, according to the signification of tag, the absolute value of the positive number is greater than that of the negative number, so the result of 'Positive + Negative' is a positive value. Then, if $a+(10^S-b)\geq 10^S$, namely $a-b\geq 0$, borrow is not required and the sum is $a-b$ and is encoded into $a-b$, namely $a+(10^S-b)-10^S$; if $a+(10^S-b)<10^S$, namely $a-b<0$, borrow is required and the sum is $(a+10^S)-b$ and is encoded into $(a+10^S)-b$, namely $a+(10^S-b)$.

Similarly, if $t_1+t_2<255$, the result of 'Positive + Negative' is a negative value. Then, if $a+(10^S-b)\leq 10^S$, namely $a-b\leq 0$, borrow is not produced and the sum is $b-a$ and encoded into $10^S-(b-a)$, namely $a+(10^S-b)$; if $a+(10^S-b)>10^S$, namely $a-b>0$, borrow is produced and the sum is $(b+10^S)-a$ and is encoded into $10^S-((b+10^S)-a)$, namely $a+(10^S-b)-10^S$.

If $t_1+t_2=255$, the length of integer part of the positive is the same with that of the negative, so judgment shall be made sequentially from the highest group to the lowest group. If $a+(10^S-b)>10^S$, namely $a-b>0$, it is shown that the absolute value of the positive value is greater than that of the negative, so the result is a positive value; if $a+(10^S-b)<10^S$, namely $a-b<0$, it is shown that the positive value is less than the absolute value of the negative, so the result is a negative value; otherwise, the judgment shall be made repeatedly to the right lower group.

3.3 Encryption of Each Group in Sequence

The decimal points of two floating-points are aligned if we add two floating-point numbers. Therefore, if we have wish to compute on encrypted floating-point numbers, the encryption algorithm must allow order relations between data items to be established based on their encrypted value without revealing the data itself and be additive homomorphism simultaneously. But there are not real to satisfy these requirements in encryption algorithms thus far. These systems, such as CryptDB, MONOMI, and SDB, cannot support floating point data type. For the above reasons, we encode floating-point numbers before encryption, and then we align the decimal points of floating-point numbers by comparing the value of the 'tag', so we do not encrypt the 'tag' in the codes of floating-point numbers or it is encrypted by order preserving encryption algorithm. In order to sum the values in one group place, we encrypt individually each group with homomorphic encryption.

Homomorphic encryption (HOM) is a secure probabilistic encryption scheme, allowing the server to perform computations on encrypted data with the final result decrypted at the client or the proxy. While fully homomorphic encryption [19-23] is prohibitively slow. Homomorphic encryption for specific operations is efficient. To support summation, we adopt the Paillier cryptosystem [24], which is a probabilistic asymmetric algorithm for public key cryptography with the security based on DCR assumption. It has the homomorphic addition of plaintexts and adopts the scheme of variable-length encryption. With Paillier, multiplying the encryptions of two values results is an encryption of the sum of the values, i.e. $HOMk(x).HOMk(y)=HOMk(x+y)$, where the multiplication is performed modulo some public-key value. To compute SUM aggregates, we replace SUM with calling a UDF that performs Paillier multiplication on a column encrypted with HOM. HOM can also be used for computing average by having DBMS server return the sum and the count separately.

In Paillier encryption algorithm, we choose two large prime numbers p and q randomly and both primes are of equal length but distinct at the beginning, where N decides decimal digits of n that equals to $p \times q$. The value to encrypt must be larger than 0 and less than n, so the maximum value that can be encrypted can reach at least $N-1$ decimal digits under conservative and is denoted by MAX_N. We segment a floating-point number into groups of S decimal digits, where we use MAX_S to denote the maximum value of one group being encrypted. Then the radio MAX_N / MAX_S captures the approximate amount that can be summed simultaneously. For example, MAX_N equals to 999,999,999 when set N to 10, MAX_S equals to 99 when set S to 2. In this case, MAX_N / MAX_S equals to 10,101,010, namely it meets approximately 10,101,010-row addition. The larger of the value N, the higher security it is, meanwhile, the more space for storing. When N is determined, the larger of the value S, the less amount that can be summed simultaneously. Consequently, we should balance the value of N and S based on specific case. In this paper, we set N to 10 and S to 2 for better presentation.

When it comes to encryption on each group, some issues appear when doing addition. According to the addition rules, we need to compare certain values to judge whether generating carry or borrow. But Paillier does not hold order preserving, neither do the other homomorphic encryptions. In addition, we need the sum to subtract 10^S when it yields carry or borrow, Paillier doesn't support subtraction and encryption of negative values as well. Consequently, we can do nothing but modify the coding scheme of the floating-points. From the above statement, we could grab that the key factor is requiring handling

carry or borrow while doing addition. Hence, we consider whether carry or borrow processing could be postponed. Being synthetically considered, when we encode floating-point numbers, the value of each group retains the true value itself. And floating point summation is resolved respectively by the positive and the negative in our new scheme, the corresponding groups are directly in accumulation without handling carry or borrow in the respective process. Besides, the two results of SUM are further processed at proxy, and we discuss it in Section 4.

According to our new coding scheme, the codes of floating-point numbers are encrypted as follows:

Tag#HOM(group 1)#HOM(group 2)#…#HOM(group k)#…

It is stored on database as sequence, where '#' is a separator of groups or tag. For example:
53021.128: 195, 05, 30, 21, 12, 80
195#75927073412870755 65#57164815257264946 02#21335261813427646 21#73500188781351727 58#2488746985469454098
-123.128035: 061, 01, 23, 12, 80, 35
061#691831179868185659 1#128020859825343251#481452169404525441 4#595150899742589063 0#610324281427975471

4. Summation on Encrypted Data

In this section, we describe the summation of floating-point numbers after encoded and encrypted. We must align the decimal points to add floating-point numbers. However, the native arithmetic addition could not be applied to encrypted sequences straightforward as the decimal points are hidden in them, so the codes of floating-point numbers will be converted to binary trees to assist summation. Meanwhile, the SUM aggregate operator must be replaced with an invocation of UDF that performs HOM addition of ciphertexts.

4.1 Data Structure

In order to align the decimal point, a code is converted to a binary tree. Root node stores the tag of a code which indicates the decimal point and the positive or negative sign. The left sub-tree denotes the integer part of a floating-point number, and the right sub-tree denotes the decimal part. All nodes of the tree except the root represent all groups. The group order of the integer part is from the leaf to upper in the left sub-tree, so the leaf of left sub-tree is the highest group of a code. The group order of the decimal part is from the first node of right sub-tree to the leaf, so the leaf of right sub-tree is the lowest group of a code. The nodes of two binary trees in the same position can be added while others cannot. We use double-headed points to link each node. The data structure is called a 'double linked list binary tree' structure (DoubleListTree), in which we can easily align the decimal points by the root node, decimal parts added would be done effortlessly by traversing the right sub-tree, integer parts added by traversing the left sub-tree, and the double linked list is better for reverse traversal. Examples are shown in Fig. 1.

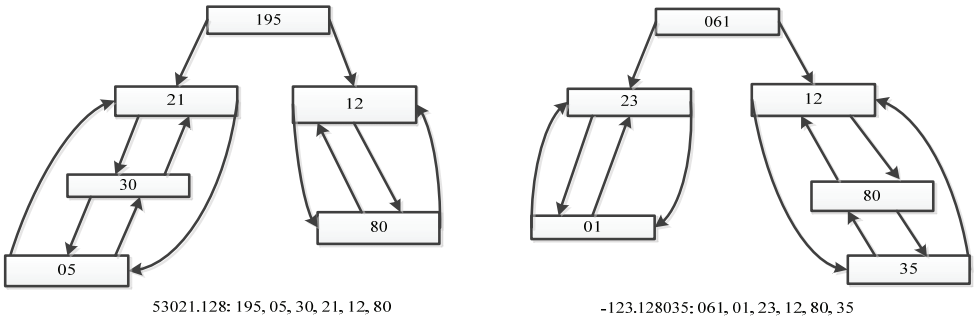

Fig. 1. DoubleListTree.

4.2 Implementing the Function of SUM

The summation on encrypted data is divided into three steps: at first all positive floating-point numbers and the negative among which all are encoded and encrypted are summed separately under encryption using DoubleListTree structure by calling UDF on server; then two summations will be decrypted and revised at proxy; subsequently, the summation of the negative is adjusted to do the final addition with the summation of the positive and the result of SUM aggregate is presented to the end user finally.

The floating-point numbers are encoded as sequences by the new coding scheme, then we encrypt each group in sequences respectively by Paillier. Transform sequences to DoubleListTree structures and traverse all nodes, and the nodes that are in the same position are added. Summation of all the positive floating-point numbers or all the negative is in the same binary tree structure, and we output the result in order of the code. Examples are shown in Fig. 2.

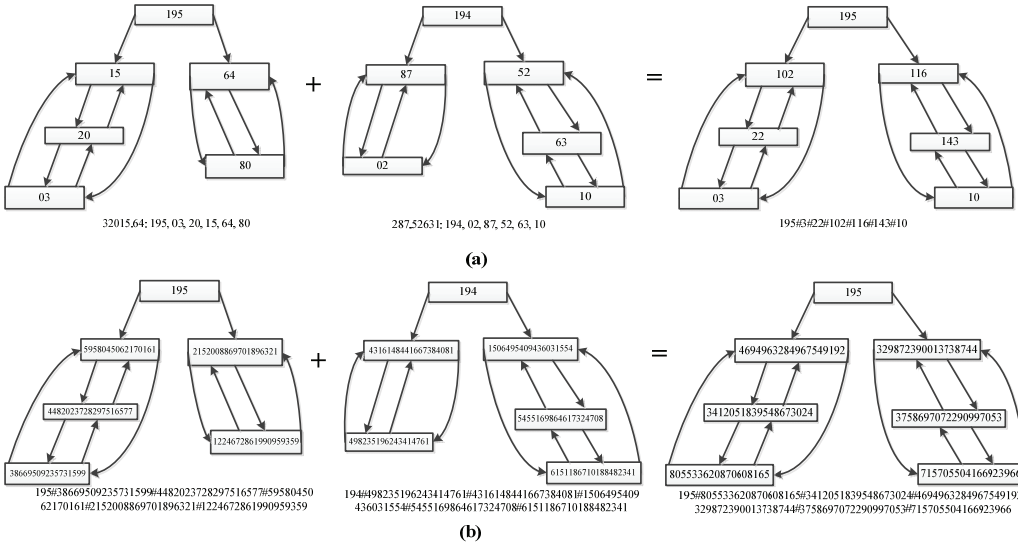

Fig. 2. (a) Addition of plaintexts and (b) addition of ciphertexts.

The first step is described in Algorithm 1.

Algorithm 1. The algorithm of summation on encrypted float-point numbers.

Input: *S*, a collection of sequences, and all sequences are encrypted.

Output: *PLUS*, the summation of all positive values in *S*; *MINUS*, the summation of all negative values in *S*. Both are sequences.

Begin

1. $T_{plus}=\varnothing$, $T_{minus}=\varnothing$; /* Both are *DoubleListTree* structure type. */
2. For each sequence $s \in S$:
   2.1 $T \leftarrow s$; /* Transform a sequence into a *DoubleLisTree* structure. */
   2.2 For every node $n \in T$ and every node $n_t \in T_{plus}$ *or* T_{minus}, do the following operations: /* If $s \geq 0$, added to T_{plus}, otherwise added to T_{minus} . */
   (i). $n_t \leftarrow n_t+n$; /* Two values of corresponding nodes are subjected to addition. */
3. $PLUS \leftarrow T_{plus}$, $MINUS \leftarrow T_{minus}$; /* Traverse the trees to sequences. */
   More detail about the traversal:
   a) Traverse the root value, and add a separator '#'.
   b) Traverse the left sub-tree from leaf node to root, and add a separator '#' between each node.
   c) Supplement a separator '#'.
   d) Traverse the right sub-tree from root node to leaf, and add a separator '#' between each node.
4. Return *PLUS*, *MINUS*;

End

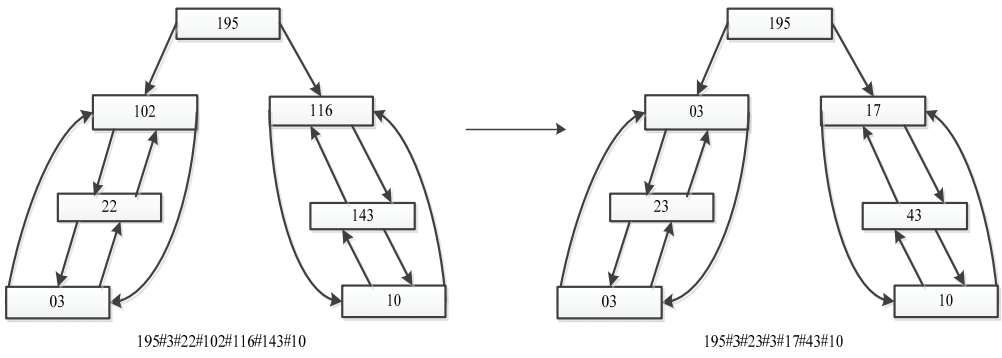

Fig. 3. Revision of sequence.

4.3 Remaining Work in Proxy

Two results in above section are generated. When the proxy receives these results, it decrypts and revises them at first. Each group is decrypted respectively by Paillier algorithm. Because the sum is produced in one group place of multiple floating-point numbers as result of addition, each group of the sum might be greater than or equals to 10^s, which is beyond the maximum of each group. If the sum is greater than or equals to 10^s, the carry is generated and is propagated to the next high order group.

Suppose the sum of one group is depicted in *GSUM*, it is depicted in *CGSUM* after being revised, and *C* denotes the carry the relationship among them is as follows:

$$CGSUM_i = (GSUM_i + C_{i-1}) \% 10^S.$$

$$C_i = (GSUM_i + C_{i-1}) / 10^S.$$

If the highest group has the carry, add a new group before the highest order group and increment the tag if the result is positive or decrement it if negative repeatedly, correspondingly a new node is hung at the leaf of the left sub-tree and increment the root value if the result is positive or decrement it if negative repeatedly. The example is shown in Fig. 3.

This step is described in Algorithm 2.

Algorithm 2. The algorithm of revising the decrypted sequences.

Input: *PLUS, MINUS*, both are sequences.

Output: *PLUS, MINUS*, two revised values, both are sequences.

Begin

1. For each group $g_{plus} \in PLUS$ and $g_{minus} \in MINUS$, decrypt g_{plus}, g_{minus}.
2. $T_{plus} \leftarrow PLUS$, $T_{minus} \leftarrow MINUS$; /* Transform sequences into *DoubleLisTree* structures. */
3. $C_{plus}=\varnothing$, $C_{minus}=\varnothing$; /* C_{plus} and C_{minus} represent the carry. */
4. For each node $n_{plus} \in T_{plus}$ and $n_{minus} \in T_{minus}$, do the following operations:
   (i). $n_{plus} \leftarrow n_{plus}+C_{plus}$, $n_{minus} \leftarrow n_{minus}+C_{minus}$; /* Get the revised values. */
   (ii). $C_{plus} \leftarrow n_{plus}/10^S$, $C_{minus} \leftarrow n_{minus}/10^S$; /* Set the carry. */
   (iii). $n_{plus} \leftarrow n_{plus}\%10^S$, $n_{minus} \leftarrow n_{minus}\%10^S$; /* Set the revised values of nodes. */
5. Loop C_{plus}, C_{minus}: if $C_{plus}>0$, $C_{minus}>0$, a new node n_{plus}, n_{minus}.
   (i). $n_{plus} \leftarrow C_{plus}\%10^S$, $n_{minus} \leftarrow C_{minus}\%10^S$; /* Set the value of new node. */
   (ii). $C_{plus} \leftarrow C_{plus}/10^S$, $C_{minus} \leftarrow C_{minus}/10^S$; /* Get the new value of carry. */
   (iii). $ROOT_{plus} \leftarrow ROOT_{plus}+1$, $ROOT_{minus} \leftarrow ROOT_{minus}-1$; /* Set the new value of root node. */
6. $PLUS \leftarrow T_{plus}$, $MINUS \leftarrow T_{minus}$; /* Traverse the trees to sequences. */
7. Return *PLUS*, *MINUS*;

End

Adding the summation of all positive floating-point numbers and the summation of all negative yields the final result of SUM aggregate. In order to keep the precision of floating points, the last addition step is executed by DoubleListTree and applied the coding scheme in Section 3.1 and addition rules in Section 3.2. The code of the negative sum is adjusted and later addition is completed meanwhile the group sum is revised. The detailed process is as Algorithm 3.

Algorithm 3. The algorithm of the final addition.

Input: A, a positive sequence; B, a negative sequence.
Output: C, the final result, a floating-point number.
Begin
1. $T_A \leftarrow A$, $T_B \leftarrow B$; /* Transform sequences into *DoubleLisTree* structures. */
2. $C=\varnothing$, $C_S=\varnothing$, $Borrow=\varnothing$; /* C_S is a sequence and *Borrow* represents the borrow. */
3. For each node $n_B \in T_B$, $n_B \leftarrow 10^S - n_B$; /* Adjust each node to satisfy the negative coding scheme. */
   3.1 Let root nodes r_A, r_B in T_A, T_B compare r_A-*193* with *62*-r_B to decide whether the result is positive or negative.
   3.2 For each node $n_A \in T_A$, $n_B \in T_B$ from the leaf to root of right subtrees and from root to leaf of left subtrees followed, do the following operations:
      3.2.1 If the result is positive,
         (i). $n_A \leftarrow n_A+n_B+Borrow$; /* Get the new value of node. */
         (ii). If $n_A \geq 10^S$, $n_A \leftarrow n_A-10^S$; /* Set the new value of node. */
         (iii). If $n_A < 10^S$, $1 \leftarrow Borrow$; /* Set the borrow. */
      3.2.2 If the result is negative,
         (i). $n_B \leftarrow n_A+n_B+Borrow$; /* Get the new value of node. */
         (ii). If $n_B > 10^S$, $n_B \leftarrow n_B-10^S$, $1 \leftarrow Borrow$; /* Set the new value of node and the borrow. */
4. If the result is positive, $C_S \leftarrow T_A$; /* Traverse the tree to sequence. */
5. If the result is negative, $C_S \leftarrow T_B$;
6. $C \leftarrow C_S$; /* Decode the sequence to a floating-point number. */
7. Return C.
End

5. Performance Analysis

For performance analysis, we mainly analyze time consumption here. The time is mainly consumed in summing the encrypted data in cloud and decrypting sequence and performing the final result in proxy. The coding scheme theoretically supports arbitrary decimal digits due to the infinite range of tag value.

We implemented encoding, encryption, decryption, revision, adjustment and decoding of the floating-points in proxy, and a UDF representing SUM aggregate function on MySQL 5.5 database. The proxy library and the server-side UDFs are implemented in approximately 2,600 lines of *C*. We use GMP for multiple precision numerical arithmetic.

The experimental proxy and the server are all running on a Pentium Dual-Core E5300 2.6 GHz processor, 2.50 GB memory and Ubuntu Server 14.04 LTS operation system. Here, we set N to 10 and S to 2 for experiment, we stored the corresponding plaintexts in database as an experimental comparison as well.

The original data are encoded and encrypted in proxy, and several examples are depicted in Table 4, and then Table 5 illustrates both the results in database and other data that are being decrypted, revised, adjusted and decoded in proxy.

We firstly execute the query with SUM aggregate directly on all columns stored as plaintext values. The time to execute under this scenario is denoted as T_{plain}. Then we execute the same query on

encrypted data, and use T_{cipher} to denote the execution time. The radio T_{cipher}/T_{plain} captures the degree to which the solution slows down the entire query processing time. We use a large number of data to record the time consumption in each case and the radio. The time consumption in different amount of participation are shown in Fig. 4 respectively and the trend of radio is presented in Fig. 5. We observe that it introduces a 11 times slowdown of the entire query processing time, which is affected by various factors, wherein the source code can be further optimized, and the improvement of experimental environment can reduce performance influence caused by processing the floating points in such a way.

Table 4. Several examples processed in proxy

Original	Encoded	Encrypted
8264.241	194#82#64#24#10	194#717355356642320590 3#4463318601723239731#425006031 8257300145#2830802455128342452
93.7	193#93#70	193#284408119179809809 3#88107619854274647
-30.82	062#30#82	062#9254491423672731 98#2927639092951954747
-5.334	062#05#33# 40	062#4272571384099659125#6458778131515620141#733276693 7278102523
1750.4092	194#17#50# 40#92	194#3818429545746992831#2676434521045999374#189091769 3032427186#7346519691564791194
-193.0382	061#01#93# 03#82	061#5360938982134324357#7807270477872491981#105927647 960703200#6131749293822456217

Table 5. Results processed in server and other corresponding data

	Sum	Decrypted	Revised	Adjusted	Decoded
Plus	194#29212981529002440 78#7803587025291817816#10221924887790 64550#2543484522585792288	194#99#207#134 #102	195#01#01#08 #35#02	195#01#01#08 #35#02	10108.3502
Minus	061#5360938982134324357#51883 84085972111174#50296295211382 0697#646768574441873443	061#01#128#118 #122	061#02#29#19 #22	061#98#71#81 #78	-229.1922
Total				194#98#79#15 #80	9879.1580

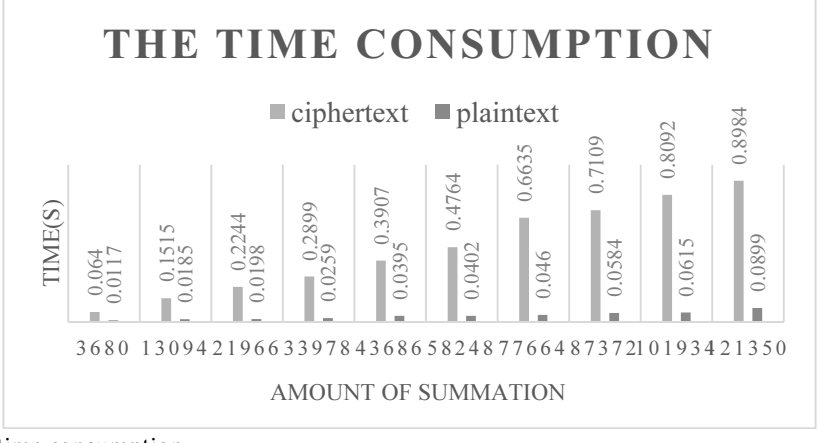

Fig. 4. The time consumption.

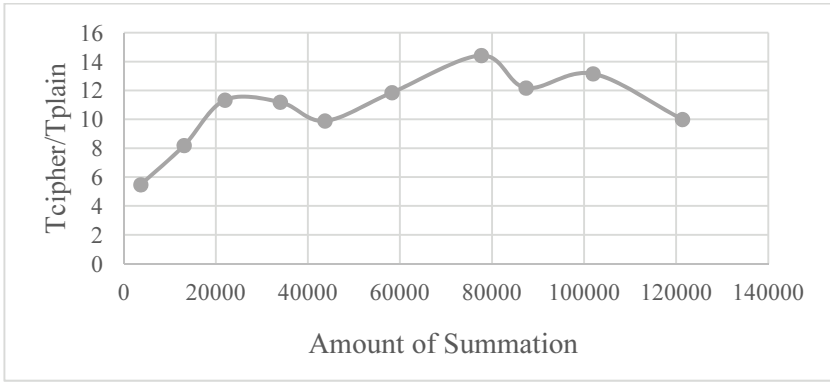

Fig. 5. T_{cipher}/T_{plain}.

6. Conclusions

In this paper, we mainly discussed query with SUM aggregate function on encrypted floating-point numbers in cloud without decryption in advance. In order to enable encrypting the floating points and summing followed, a coding scheme is proposed to hide the decimal points and the positive or negative signs. A floating-point number is divided into groups by preset length and each group is encrypted by homomorphic encryption algorithm Paillier in proxy, then all groups combine as a sequence to be stored in cloud database. We call a UDF to implement the SUM aggregate function on encrypted database without revealing the data itself, where we apply a data structure of DoubleListTree to accomplish addition. The summation results of the positive and the negative are sent to proxy, followed by decryption, revision, adjustment, and then do the final addition, later the result is decoded as the final result. Our research can serve as a supplement to CryptDB.

In our research, we mainly discuss query with SUM aggregate function over floating point data type, and it is inadequate to query floating-point numbers. More complex queries on encrypted database will be further explored in more detail.

Acknowledgement

This work was supported in part by Natural Science Foundation of Guangdong Province (Grant No. 2015A030310208), Technology Research Project of the Ministry of Public Security (Grant No. 2014JSYJB048), Science and Technology Project of Guangzhou (Grant No. 201604010037), and National Natural Science Foundation of China (Grant No. 6152163).

References

[1] C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362-375, 2013.

[2] N. H. Yu, Z. Hao, J. J. Xu, W. M. Zhang, and C. Zhang, "Review of cloud computing security," *Dianzi Xuebao (Acta Electronica Sinica)*, vol. 41, no. 2, pp. 371-381, 2013.

[3] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, Cascais, Portugal, 2011, pp. 85-100.

[4] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *Proceedings of 2013 IEEE Symposium on Security and Privacy (SP),* San Francisco, CA, 2013, pp. 463-477.

[5] R. A. Popa and N. Zeldovich, "Cryptographic treatment of CryptDB's adjustable join," Massachusetts Institute of Technology, Cambridge, MA, Technical Report No. MIT-CSAIL-TR-2012-006, 2012.

[6] C. Curino, E. P. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishna, and N. Zeldovich, "Relational cloud: a database-as-a-service for the cloud," in *Proceedings of 5th Biennial Conference on Innovation Data Systems Research (CIDR)*, Asilomar, CA, 2011, pp. 235-240.

[7] R. A. Popa, N. Zeldovich, and H. Balakrishnan, "CryptDB: a practical encrypted relational DBMS," Massachusetts Institute of Technology, Cambridge, MA, Technical Report No. MIT-CSAIL-TR-2011-005, 2011.

[8] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," *Proceedings of the VLDB Endowment*, vol. 6, no. 5, pp. 289-300, 2013.

[9] 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*, Snowbird, UT, 2014, pp. 1395-1406.

[10] Z. He, W. K. Wong, B. Kao, D. W. L. Cheung, R. Li, S. M. Yiu, and E. Lo, "SDB: a secure query processing system with data interoperability," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1876-1879, 2015.

[11] S. Bajaj and R. Sion, "Trusteddb: a trusted hardware-based database with privacy and data confidentiality," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 752-765, 2014.

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

[13] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, WI, 2002, pp. 216-227.

[14] A. C. Yao, "Protocols for secure computations," in *Proceedings of 23rd Annual Symposium on Foundations of Computer Science (SFCS)*, Chicago, IL, 1982, pp. 160-164.

[15] D. Bogdanov, R. Jagomagis, and S. Laur, "A universal toolkit for cryptographically secure privacy-preserving data mining," in *Pacific-Asia Workshop on Intelligence and Security Informatics*. Heidelberg: Springer, 2012, pp. 112-126.

[16] H. Hacigumus, B. Iyer, and S. Mehrotra, "Providing database as a service," in *Proceedings of 18th International Conference on Data Engineering*, San Jose, CA, 2002, pp. 29-38.

[17] K. Kaur, K. S. Dhindsa, and G. Singh, "Numeric to numeric encryption of databases: using 3Kdec algorithm," in *Proceedings of IEEE International Advance Computing Conference*, Patiala, India, 2009, pp. 1501-1505.

[18] G. L. Xiang, X. M. Chen, P. Zhu, and J. Ma, "A method of homomorphic encryption," *Wuhan University Journal of Natural Sciences*, vol. 11, no. 1, pp. 181-184, 2006.

[19] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, Bethesda, MD, 2009, pp. 169-178.

[20] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Heidelberg: Springer, 2010, pp. 24-43.

[21] Z. Brakerski, C. Gentry, and S. Halevi, "Packed ciphertexts in LWE-based homomorphic encryption," in *Public-Key Cryptography–PKC 2013*. Heidelberg: Springer, 2013, pp. 1-13.

[22] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831-871, 2014.

[23] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013*. Heidelberg: Springer, 2013, pp. 75-92.

[24] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology–EUROCRYPT '99*. Heidelberg: Springer, 1999, pp. 223-238.

Taipeng Zhu

He is a master's degree postgraduate of Jinan University. His major is Computer Software and Theory, and his research interests are in the area of data integration and cloud computing.

Xianxia Zou

She received the Ph.D. degree in School of Information Science and Engineering, Central South University. Her research interests are in the areas of data integration, data stream, distributed database and cloud computing.

Jiuhui Pan

He is a Professor of Jinan University. He was a visiting scholar at University of Waterloo in Canada. His research interests are in the areas of distributed database, data stream, information integration, etc. He has directed a number of doctoral and master students. He has published research articles in reputed international journals of computer sciences.