EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

A privacy protection-oriented parallel fully homomorphic encryption algorithm in cyber physical systems

Zhaoe Min[1], Geng Yang[1,2*], Arun Kumar Sangaiah[3], Shuangjie Bai[1] and Guoxiu Liu[1]

Abstract

Cyber physical system (CPS) is facing enormous security challenges because of open and interconnected network and the interaction between cyber components and physical components, the development of cyber physical systems is constrained by security and privacy threats. A feasible solution is to combine the fully homomorphic encryption (FHE) technique to realize the efficient operation of ciphertext without decryption. However, most current homomorphic encryption algorithms only support limited data types, making it difficult to be widely applied in actual environment. To address this limitation, we propose a parallel fully homomorphic encryption algorithm that supports floating-point numbers. The proposed algorithm not only expands the data types supported by the existing fully homomorphic encryption algorithms, but also utilizes the characteristics of multi-nodes in cloud environment to conduct parallel encryption through simultaneous group-wise ciphertext computations. The experimental results show that, in a 16-core 4-node cluster with MapReduce environment, the proposed encryption algorithm achieves the maximum speed-up exceeding 5, which not only solves the limited application problem of the existing fully homomorphic encryption algorithm, but also meets the requirements for the efficient homomorphic encryption of floating-point numbers in cloud computing environment.

Keywords: Privacy protection, Fully homomorphic encryption, Encryption of floating-point number, Parallel encryption, Cyber physical system

1 Introduction

The cyber physics system is a multi-dimensional complicated system that integrates computation, communication, and physical environments. The system emphasizes the interaction between cyber and the physical system, so secure information transmission between physical components and information system has become more important [1–3]. For the cyber physical system, a relatively complete secure service should provide privacy protection, data confidentiality, information integrity, ID certification and access control. Therefore, how to provide privacy and security protection to users in the cyber physics system in a secure and effective manner has become a hotspot in the current academic research [4, 5].

* Correspondence: yangg@njupt.edu.cn
[1]School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210046, China
[2]Jiangsu Key Laboratory of Big Data Security& Intelligent Processing, Nanjing 210046, China
Full list of author information is available at the end of the article

In recent years, various technologies have been broadly used for data privacy protection, such as private information retrieval [6–11], searchable encryption [12–17], and secure multi-party computation [18–23], but these technologies can only provide limited functions, such as keyword search, order search, range query, and subset search. However, for many application scenarios in the cloud environment, it requires various types operations of ciphertext data. For example, based on the medical data of thousands of patients, we could conduct analysis of drug effects, summarize frequently searched words by users in the search engine to release-related advertisement, and conduct statistical analysis of encrypted financial information of company. Most traditional encryption methods do not support ciphertext operation. According to the traditional method, these data are sent to the cloud after encryption, and when processing the data, the user needs to download data to a local system and uses the data after decryption. This approach tends to cause exposure of privacy, and in the meantime, when the user has frequent use of data to conduct communication with service provider and realize encryption and decryption of data, it will consume massive network bandwidth and user's time, which will significantly reduce the usability of cloud computing.

Another solution is to adopt the homomorphic encryption technique [24]. This technique supports ciphertext data management under privacy protection, which can be used to realize various operations such as direct search, computation, and statistics of ciphertext at cloud, and the result can be returned to the user in the form of ciphertext. Compared to the traditional encryption algorithms, this method does not require frequent encryption and decryption operations between the cloud and user, which can reduce the overhead of communication and computation resources. The user's private data are saved in the form of ciphertext at cloud, and the service provider cannot know the data content, which can prevent them from exploring user's privacy through illegal embezzling and tampering of user data. It has provided a security basis for the users to fully utilize the cloud computing resources to conduct massive data analysis and processing, and in particular, it can be combined with the secure multi-party computation protocol to well solve the privacy security issue when the user outsources the computation service.

Most current homomorphic encryption schemes support integer homomorphic operation, but do not support homomorphic operation of floating-point data, so they cannot satisfy the requirement of actual application. By combining the cloud computing environment, this paper proposes a fully homomorphic encryption algorithm that supports floating-point operation, and the objective is to expand encryption algorithm from integer to floating-point number. This scheme has combined the MapReduce framework to realize fully homomorphic encryption of parallel floating-point number. In the meantime, the cluster advantage is used to improve the execution efficiency of algorithm, realize efficient encryption and decryption operation, and effectively reduce the time of homomorphic operation. Both the theoretical analysis and experiment results show that the parallel homomorphic encryption algorithm supports floating-point operation, which can be used to conduct fast and efficient encryption and decryption operation of massive floating-point data. It has high security and practicality, and it is applicable to the cloud computing scenario.

The main contributions of our work are summarized as follows:

(1) We propose a fully homomorphic encryption algorithm supporting Floating-point operation (FFHE) in this paper, which has solved the problem that direct operation of many floating-point ciphertexts cannot be carried out in the real environment.
(2) We design a parallel homomorphic encryption scheme in order to address the low efficiency of homomorphic encryption algorithm. This scheme is based on the MapReduce environment, which can realize parallel performance of algorithm through data blocks. The experimental result shows that, in a 16-core 4-node cluster, this encryption algorithm can reach the maximum speed-up ratio exceeding 5.
(3) In addition to improve the security of algorithm, we add an operation to disrupt the ciphertext order in the proposed homomorphic encryption scheme that supports floating-point operation, which has eliminated the association between the child ciphertext and key pair.

The rest of this paper is organized as follows. Related work is summarized in Section 2. Section 3 introduces the background knowledge. The homomorphic encryption scheme that supports floating-point number is proposed in Section 4, and the homomorphic performance and security of algorithm are also proved. In Section 5, parallel design of algorithm is conducted, and specific realization method is provided. Section 6 consists of experiment and analysis, and the experiment results and discussion are presented in the form of a chart. Finally, we conclude this paper in Section 7.

2 Related work

In 1978, Rivest et al. proposed the concept of homomorphic encryption for the first time in Literature [25], which is also called the "privacy homomorphism," and in the same year, they also proposed that the RSA public key encryption algorithm has multiplication homomorphism [26], and the security of this scheme is based on integer factorization. Later, many homomorphic encryption schemes have been proposed, such as the ElGamal [27] encryption scheme with multiplication homomorphism and the Paillier [28] encryption scheme with the addition of homomorphism, but none of these methods have the feature of fully homomorphic encryption, and are called partial homomorphic encryption (PHE).

In 2009, Gentry proposed the fully homomorphic encryption(FHE) scheme based on the ideal lattice problem for the first time [29], and this scheme can be used to conduct any addition and multiplication operations of ciphertext. Later, the fully homomorphic encryption technique entered the period of fast development. Dijk et al. proposed the fully homomorphic encryption scheme DGHV within the integer field [30], and this scheme is based on the greatest common divisor problem. Brakerski et al. proposed a fully homomorphic encryption scheme based on the LWE (learning with errors) problem [31], its main idea is to address the defects of ideal lattice-based scheme through the re-linearization technique. Stehle et al. introduced the NTRU (number theory research unit) algorithm for the first time to improve the efficiency of initial FHE scheme [32]. Its security assumption is based on RLWE (ring learning with errors). Brakerski et al. proposed the BGV scheme in literature [33], which can support multi-bit operation, and the computation complexity is much lower than that of Gentry's initial scheme. From the initial scheme of Gentry to BGV scheme, the research on the homomorphic encryption scheme has made remarkable progress, but still far away from the actual application.

In recent years, some mature homomorphic encryption schemes are proposed in literatures [34–41], especially, Garg et al. proposed a fully homomorphic algorithm based on the LWE and RLWE problems, which utilizes the addition and multiplication operations of matrix to realize homomorphic computation of ciphertext, and it is believed as an ideal scheme at present. Based on the approximate greatest common divisor problem, Liu proposed a fully homomorphic encryption (LFHE) that supported integer operation [42], and a fully homomorphic encryption is realized through complicated algebraic equation, which has high execution efficiency. He also applied this scheme to cloud computing environment [43]. Liu et al. [44, 45] designed a computation framework and toolkit that support privacy protection, this scheme supports multi-key encryption, and it can be expanded to rational number computation.

In real scenarios, some homomorphic encryption techniques have been used in the cloud environment for privacy protection. According to the data privacy problem in cloud computing environment, Brenner et al. [46] adopted a fully homomorphic encryption technique to realize safe execution of confidential program at third-party server. For the multimedia information retrieval problem in cloud computing environment, Lu et al. [47] proposed SIFT (the security scale invariant feature transform) scheme based on Paillier encryption scheme. The feasibility and efficiency problems of existing homomorphic encryption schemes are discussed in literatures [48–56], and some application scenarios were combined to analyze the requirement for homomorphic algorithms. In accordance with the low efficiency problem of Paillier encryption algorithm, Min et al. [57] proposed a homomorphic encryption algorithm that can conduct parallel encryption in the cloud environment, but because most practical computations involve integer and floating-point number operation, this method still has its shortages.

Literatures [58–60] expanded the homomorphic encryption scheme of integer domain to the fixed--point and floating-point parts, which has extended the application scenarios of homomorphic algorithm. Literature [61] specifically analyzed the theoretical basis and characteristics of above homomorphic encryption schemes in theory, stipulated various terms, related concepts and definitions used in the homomorphic schemes, and made uniform description of above concepts based on mathematical knowledge.

As most current homomorphic encryption schemes support integer homomorphic operation, we propose a fully homomorphic encryption algorithm that supports floating-point operation. The proposed algorithm can not only solve the problem of limited application in the existing fully homomorphic encryption, but also conduct parallel encryption based on the characteristics of multi-nodes in cloud environment, and as a result, the efficiency can be improved.

3 Background

3.1 The LFHE algorithm

LFHE algorithm allows the ciphertext to contain huge noise, and the ciphertext generated after multiple homomorphic operations can still be accurately decrypted, no matter how big noise volume has been accumulated during this process. This scheme is mainly based on the approximate greatest common divisor problem, which depends on complicated algebraic operation, so it has higher efficiency than the homomorphic encryption scheme based on the ideal lattice problem. The specific encryption scheme is as follows:

1. Generation of key

Assume q is a prime number and Z_q be the set of integers modulo q, from $GF(q)^{n+1}$, select a random integer vector $K(n) = [\ k_1,\ \ldots,\ k_n\ ]$, $n\ \geq 3$; in $GF(q)^l$, select random vector $\Theta = [\theta_1,\ldots,\theta_l]$, select random ciphertexts encrypted by elements in Θ, which is $\Phi = [Enc_l(K(n),\theta_1),\ldots,Enc_l(K(n),\theta_l),\ Enc_l(K(n),1)]$. Then the private keys are $K(n)$ and Φ.

2. Encryption algorithm

LFHE algorithm generally consists of two parts: the lower level encryption algorithm and the upper level encryption algorithm. Given the secret key $K(n)$ and an integer $v \in Z_q$, the lower level encryption algorithm can be expressed as $Enc_l = (K(n),v) = (c_1,\ \ldots,\ c_{n+1})$; the specific algorithm is shown as Eq. (1).

$$c_{\Pi(i)} = \begin{cases} a * t_i * \left(v + \sum_{j=1}^{h-1} rv_j\right) + S(i) + t_i * (r_i - r_h) \bmod q & i = 1 \\ a * t_i * (-rv_{i-1}) + S(i) + t_i * (r_i - r_{i-1}) \bmod q & 2 \le i \le h \\ rs_u + \sum_{j=u+1}^{m} s_{ij} * rs_j + t_i * rr \bmod q & h+1 \le i \le n-1, u = i-h \\ rs_m + t_i * rr \bmod q & i = n \\ rr \bmod q & i = n+1 \end{cases} \tag{1}$$

where $r_1,\ldots,r_h$, $rs_1,\ldots,rs_m$, $rv_1,\ldots,rv_{h\text{-}1}$ and rr are random integers uniformly sampled from Z_q. For correctness, we require $a \neq 0$ $t_i \neq 0$ for $1 \le i \le h$, and$S(i)$is defined as $S(i) = \sum_{j=1}^{m} s_{ij} * rs_j$.

The lower level encryption algorithm is only used to generate the key element in the fourth part Φ of key. Assume in key $K(n)$, $\Theta = [\theta_1,\ldots, \theta_l]$ has been defined, and it satisfies $l \le n\text{-}2$. For each element θ in Θ, $n + 1$ ciphertexts $C_{\theta 1}, \ldots, C_{\theta(n+1)}$ can be obtained according to Eq.(1), which is element ϕ in key Φ. Therefore, Φ can be defined as Eq. (2).

$$\begin{aligned} \Phi &= [\phi_1, \cdots, \phi_{l+1}] \\ &= [Enc_l(K(n), \theta_1), \cdots, Enc_l(K(n), \theta_l), Enc_l(K(n), 1)] \end{aligned} \tag{2}$$

And the specific definition of ϕ_i is:

$$\phi_i = \begin{cases} Enc_l(K(n), \theta_i) & i \le l \\ Enc_l(K(n), 1) & i = l + 1 \end{cases} \tag{3}$$

where ϕ_{l+1} is the ciphertext result obtained through encryption of integer 1. If it satisfies the requirement of low-order encryption algorithm, it can use maximal $n -1$ constraints.

Assume $ru_1,\ldots,ru_{l\text{-}1}$, and ru_l are all random integers samples from Z_q, ru_{l+1} and plaintext v are satisfied $ru_{l+1} = v - \sum_{i=1}^{l} ru_i * \theta_i \bmod q$. In key $K(n)$, the third and fourth parts are $\Theta = [\theta_1,\ldots\theta_l]$ and $\Phi = [\phi_1, \cdots, \phi_{l+1}]$ respectively, every plaintext can be encrypted into $l + 1$ ciphertexts. And then, the upper level encryption algorithm can be expressed as Eq. (4).

$$Enc(K(n), v) = (c_1, \ldots, c_{n+1}) \tag{4}$$

where $c_i = ru_1 \times c_{\theta 1i} + \ldots + ru_{l+1} \times c_{\theta(l+1)i}$.

3. Decryption algorithm

The decryption algorithm uses key $K(n)$ to decrypt ciphertext $(c_1,\ldots,c_{n+1})$ into plaintext v, and it mainly involves the following steps:

- $RR = c_{\Pi(n+1)} \bmod q$;
- $RS_m = c_{\Pi(n)} - t_n * RR \bmod q$;
- $RS_u = c_{\Pi(i)} - t_i * RR - \sum_{j=u+1}^{m} s_{ij} * RS_j \bmod q$

where u ranges from $m - 1$ to $1, i = u + h$;

- $F = \sum_{i=1}^{h}((c_{\prod(i)} - \sum_{j=1}^{m} * RS_j)/t_i) \bmod q$;
- $v = F/a \bmod q$.

In the above definition, the decryption algorithm is described in five steps by using intermediate variables, such as RS_u and F. Actually, we can fuse these steps and then we can get a linear form of the decryption algorithm as Eq. (5).

$$v = dk_1{}^* c_{\Pi(1)} + \ldots + dk_{n+1}{}^* c_{\Pi(n+1)} \bmod q \tag{5}$$

Compared to the common fully homomorphic encryption algorithm, LFHE can provide good execution efficiency, which has certain practical value and realistic significance. The defects mainly consist of two aspects: (1) the algorithm can only support integer homomorphic operation, but do not support homomorphic operation of floating-point data, so they cannot satisfy the requirement of actual application; (2) it has certain limitation on the aspect of security, which has the risk of leaking the key. The reason why the attacker is able to decode all ciphertexts information based on the PEK is that the relative location of key elements k_i is maintained the same when the LFHE scheme uses the key to encrypt plaintext data, so the attacker is still able to decode the key information by solving the equations.

In this chapter, a new fully homomorphic encryption algorithm is proposed to support floating-point operation, which increase its application scene range, and makes further improvements in the areas of security flaws. In the meantime, in order to improve the execution efficiency of algorithm, we combine the MapReduce framework to realize fully homomorphic encryption of parallel floating-point number.

3.2 MapReduce model

The MapReduce parallel computation framework is a parallel program execution system, and it provides the parallel processing model and process that consists of the two stages of Map and Reduce. The Map function and Reduce function provide two high-level abstract models and interfaces for parallel programming, and the programmer only needs to realize these two interfaces to quickly complete parallel programming.

The basic processing procedure of MapReduce parallel programming model is as follows:

- Various Map nodes conduct parallel processing of divided data, generating corresponding intermediate results from different input data and output the results;

– Various Reduce nodes also conduct parallel computation, and they are responsible of processing the datasets of different intermediate results;
– The processing of all Map nodes must be completed before the Reduce processing, so it requires a synchronous barrier (Barrier) before the Reduce processing.
– By summarizing the output results of Reduce nodes, the final result can be obtained.

4 The proposed full homomorphic encryption algorithm supporting floating-point operation

4.1 The proposed FFHE algorithm

The FFHE encryption scheme proposed in this paper supports both integer and floating-point number operations (in the following part, it will be illustrated with floating -point number as example), which simultaneously has the characteristics of addition homomorphism and multiplication homomorphism. This algorithm mainly consists of three parts: generation of key, encryption algorithm, and decryption algorithm.

Generation of key: assume k and s are n-dimension key vectors, set $K(n) = [(k_1,s_1),\ldots, (k_n,s_n)]$, where $k_i \in$R, $s_i \in$R and $n > 3$, and they satisfy Eq. (6).

$$\begin{cases} k_i \neq 0, \forall 1 \leq i \leq n \\ s_1 + \cdots s_{n-1} \neq 0, s_n \neq 0 \end{cases} \tag{6}$$

Encryption algorithm: assume v is the floating-point number that needs to be encrypted. The specific encryption process mainly consists of the following steps:

1. Randomly generate $n - 1$ pairs of floating-point number sets $P = [(r_1, p_1),\ldots,(r_{n-1}, p_{n-1})]$ as the encrypted noise.
2. Compute the order ciphertext C' which includes $n$ciphertexts, and the specific computation Equation is:

$$c_i = \begin{cases} k_i \times (s_i \times v + p_i) + r_i, 1 \leq i \leq n-1 \\ k_n \times s_n \times \sum_{i=1}^{n-1} (p_i + r_i/k_i), i = n \end{cases} \tag{7}$$

in which $v \in$R.

3. Define the mapping function Π:

$$\Pi(i) = j \quad 1 \leq i, j \leq n \tag{8}$$

Rearrange the ciphertext fragments according to the output result of mapping function Π, and generate out-of-order ciphertext C''; according to the mapping result of function Π, map the ith child ciphertext c_i of order ciphertext C' into the jth child ciphertext of out-of-order ciphertext C'', denoted as c_{dj}, so $d_j = i$. In which, the subscript j indicates that c_{dj} is at the jth position of ciphertext C''. For $i \in [1,2,\ldots,n]$, define the set of all mapping results as J, so the child ciphertexts c_i and c_{dj} satisfy:

$$\forall c_{d_j} \in C'', \exists c_i \in C', c_{d_j} = C''[j] = C'[i] = c_i \tag{9}$$

According to Eq. (9), function Π maintains the relationship among the child ciphertexts of C' and C''. The mapping results of function Πare random, and the mapping results for different ciphertext C' are independent from each other. Therefore, the child ciphertexts order of different ciphertexts do not influence each other, and they are all random arrangement.

4. Use the AES (Advanced Encryption Standard) encryption algorithm to encrypt mapping array J and generate child ciphertext c_{n+1}, i.e., $c_{n+1} = Enc(J)$, and ciphertext C' and child ciphertext c_{n+1} are the final encryption result $C = [c_{d_1}, \cdots c_{d_n}, c_{n+1}]$ of plaintext v.

Decryption algorithm: the process to decrypt ciphertext$C$into plaintext v mainly consists of the following three steps:

(1) Use the AES algorithm to decrypt ciphertext c_{n+1} and obtain array J, determine the child ciphertext c_i according to Eq. (9), and build corresponding relationship between c_i and key elements k_i and s_i;
(2) Compute S:

$$S = \sum_{i=1}^{n-1} s_i \tag{10}$$

(3) Compute plaintext v:

$$v = \sum_{i=1}^{n-1} c_i/(k_i \times S) - c_n/(k_n \times s_n \times S) \tag{11}$$

Specifically, based on Eqs. (7) and (11), the derivation process of Eq. (11) can be described as follows:

$$\begin{aligned}&Dec(C)\\&=\sum\nolimits_{i=1}^{n-1}c_i/(k_i\times S)-c_n/(k_n\times s_n\times S)\\&=(k_1\times s_1\times v+k_1\times p_1+r_1)/(k_1\times S)\\&\quad+(k_2\times s_2\times v+k_2\times p_2+r_2)/(k_2\times S)+\cdots\\&\quad+(k_{n-1}\times s_{n-1}\times v+k_{n-1}\times p_{n-1}+r_{n-1})/(k_{n-1}\times S)\\&\quad-k_n\times s_n\times\sum\nolimits_{i=1}^{n-1}(p_i+r_i/k_i)/(k_n\times s_n\times S)\\&=v\end{aligned}\tag{12}$$

4.2 Homomorphism proof

For this scheme, the security parameter is n. During the homomorphic addition and multiplication operations, the n child ciphertexts of ciphertext C'' are believed as participating in computation by default. Unless it is pointed out otherwise, child ciphertext c_{n+1} is only used as the ciphertext to reflect the mapping relation, which does not participate into the addition and multiplication operations of child ciphertext. In the improved FFHE scheme, the encryption and decryption operations can be reflected by the following expressions:

$$\begin{cases}Enc(K(n),v)=[c_{d_1},\cdots,c_{d_n},c_{n+1}]\\Dec(K(n),[c_{d_1},\cdots,c_{d_n},c_{n+1}]=v\end{cases}\tag{13}$$

4.2.1 Addition homomorphism

For plaintexts v_1 and v_2, assume their ciphertext data after encryption are C_1 and C_2, respectively, then

$$\begin{cases}C_1=\left[c_{1d_1},\cdots,c_{1d_n},c_{1(n+1)}\right]=Enc(K(n),v_1)\\C_2=\left[c_{2d_1},\cdots,c_{2d_n},c_{2(n+1)}\right]=Enc(K(n),v_2)\end{cases}\tag{14}$$

In this scheme, the homomorphic addition operation of ciphertexts C_1 and C_2 is defined as vector addition. However, because the child ciphertexts have been randomly shuffled, the child ciphertexts c_{1dj} and c_{2dj} at corresponding locations of C_1 and C_2 are not necessarily encrypted from the same key pair k_i and s_i, so the child ciphertexts at corresponding locations cannot be directly added.

If ciphertext C is decrypted as original ciphertext C', and addition homomorphism operation is completed by adding the child ciphertexts at corresponding locations of C_1' and C_2', the attacker might obtain corresponding location relation between child ciphertext c_i and keys k_i and s_i, and crack the keys.

This paper utilized the mapping function to regenerate a group of new mapping relationships, denoted as $J_{\text{-}adj}$. Based on $J_{\text{-}adj}$, adjust the arrangement of the child ciphertexts of out-of-order ciphertexts C_1'' and C_2'' into new out-of-order arrangement. Assume the ciphertexts are $C''_{1\_adj}=[c_{1d_1},\cdots,c_{1d_n}]$ and $C''_{2\_adj}=[c_{2d_1},\cdots,c_{2d_n}]$ after adjustment, and the adjustment method is:

$$C''_{adj}[J\_adj[i]]=C''[J[i]]\tag{15}$$

Adjust the child ciphertext order of C_1'' and C_2'', generate new ciphertexts $C''_{1\_adj}=[c_{1d_1}\cdots c_{1d_n}]$ and $C''_{2\_adj}=[c_{2d_1}\cdots c_{2d_n}]$, add corresponding terms of ciphertexts C''_{1_adj} and C''_{2_adj}, and use the new mapping relation $J_{\text{-}adj}$ as child ciphertext c_{n+1}, i.e.,

$$C_1\oplus C_2=[c_{1d_1}+c_{2d_1},\cdots c_{1d_n}+c_{2d_n},c_{n+1}]\tag{16}$$

where $\oplus$ denotes the addition operation of ciphertext vectors. Using the Eq. (12), we decrypt the ciphertext of homomorphic addition:

$$\begin{aligned}&Dec(K(n),C_1\oplus C_2)\\&=\sum\nolimits_{i=1}^{n-1}c_{(1d+2d)_i}/(k_i\times S)-c_{(1d+2d)_n}/(k_n\times s_n\times S)\\&=v_1+v_2=Dec(K(n),C_1)+Dec(K(n),C_2)\end{aligned}\tag{17}$$

In other words, they are the corresponding results of plaintext addition. In summary, it can be inferred that the FFHE scheme has additive homomorphism.

4.2.2 Multiplication homomorphism

Assume C_1 and C_2 are the generated ciphertexts of plaintexts v_1 and v_2 after using key $K(n)$ for encryption ($K(n)$ can be different). The child ciphertext c_{n+1} has mapping relation, which does not participate in the operation, then we can obtain a $n*n$ ciphertext matrix:

$$C_1*C_2=\begin{bmatrix}c_{1d_{11}}*c_{2d_{21}},\cdots,c_{1d_{11}}*c_{2d_{2n}}\\\cdots\\c_{1d_{1n}}*c_{2d_{21}},\cdots,c_{1d_{1n}}*c_{2d_{2n}}\end{bmatrix}\tag{18}$$

Using Eq. (13), we conduct decryption operation of ciphertext matrix according to rows or lines (here, we conduct decryption based on lines), i.e.:

$$\begin{aligned}c^*_{2d_{2i}}&=Dec\left(K(n),\left[c_{1d_{11}}*c_{2d_{2i}},\cdots,c_{1d_{1n}}*c_{2d_{2i}},c_{1(n+1)}\right]\right)\\&=c_{2d_{2i}}{}^*Dec\left(K(n),\left[c_{1d_{11}},\cdots,c_{1d_{1n}},c_{1(n+1)}\right]\right)=c_{2d_{2i}}{}^*v_1\end{aligned}\tag{19}$$

LFHE scheme supports the multiplication operation between plaintext constant and ciphertext vector. Assume $d\in Z_n$, then $d\odot C=(d^*c_1 \bmod q,\ldots,d^*c_{n+1} \bmod q)$ in which $\odot$ represents the multiplication operation between plaintext constant and ciphertext vector. Then, according to the property of homomorphic addition, this multiplication operation also satisfies the homomorphic decryption algorithm, i.e., $Dec(K(n),d\odot C)=d^*Dec(K(n),C) \bmod q$. So, we can obtain:

$$\begin{aligned}C^*&=\left[c^*{}_{2d_{21}},\cdots,c^*{}_{2d_{2n}},c_{2(n+1)}\right]\\&=\left[v_1{}^*c_{2d_{21}},\cdots,v_1{}^*c_{2d_{2n}},c_{2(n+1)}\right]\\&=v_1\odot C_2\end{aligned}\tag{20}$$

Ciphertext C^* is the result of ciphertext multiplication, and its ciphertext dimension and child ciphertext order are the same as that of ciphertext C_2. Similarly, if the decryption is conducted based on rows, the dimension of ciphertext $C^*$and the order of child ciphertext should be maintained the same as ciphertext C_1, and by decrypting ciphertext C^*, we can obtain the product of $v_1{}^*v_2$.

$$\begin{aligned} Dec(C_1 \times C_2) &= Dec(C^*) = Dec(v_1 \odot C_2) \\ &= v_1{}^* Dec(C_2) = v_1{}^* v_2 = Dec(C_1) \times Dec(C_2) \end{aligned} \tag{21}$$

In conclusion, this scheme has multiplication homomorphism. After executing the homomorphic multiplication operation, the number of child ciphertexts will be maintained the same, which will not cause the expansion of ciphertext data.

4.3 Security analysis

In order to avoid the problem of cracking the key through the linear equations in the LFHE algorithm that may occur in this algorithm, this paper introduces a new mapping function Π, and through the mapping function Π, it can turn the order ciphertext after reach encryption to randomly generate out-of-order ciphertext. The mapping function can ensure that the orders of ciphertext fragments obtained from plaintext data v and v' through encryption algorithm are independent and irrelevant, and each mapping is random with no rules. For random and independent mapping relationship Π, during execution of certain encryption operation, it requires using a convenient and effective method to save the relative order of result ciphertext (i.e., the specific mapping relationship of this mapping Π into the result ciphertext for subsequent homomorphic operation or decryption operation). Without the assistance of mapping relationship, even the legitimate user cannot accurately match the corresponding relationship between key k_i and ciphertext fragment c_i and accurately decrypt the original plaintext.

This paper encrypts each specific corresponding relationship of mapping relationship Π as critical data and adds it to the end of result ciphertext C as additional ciphertext fragment c_{n+1}, which can be used as the baseline to localize ciphertext fragment during subsequent operation. Therefore, in this paper, the final ciphertext after encryption is $C_{v,\ \Pi} = (c_{\Pi(0)}, \cdots, c_{\Pi(n)})$, and for different plaintext v, the child ciphertext arrangement $[d_1,\ldots,d_n]$ of different ciphertext C is independent and random, without any relationship between them. Based on the above analysis, we can draw the following conclusion.

Theorem 1 *Adopting the Chosen-ciphertext attack model, the probability of obtaining key from known ciphertexts is* $1/n!^n$.

Proof We assume that the order of ciphertext C' is randomly disrupted, which contains n child ciphertexts, and there are $n!$ different arrangements. If the n child ciphertexts are chosen to crack the key, the accurate coefficients $1/(k_i{}^*S)$ and $1/(k_n{}^*s_n{}^*S)$ of decryption algorithm can only be obtained when the arrangements of n child ciphertexts are completely consistent. Because each ciphertext C' has $n!$ different arrangements, n groups of ciphertexts have $n!^n$ combinations and the probability of accurately obtaining the coefficients is $n!/n!^n$. Even after obtaining the accurate coefficients $1/(k_i{}^*S)$ and $1/(k_n{}^*s_n{}^*S)$, there are still $n!$ possible arrangements, so the probability of obtaining accurate coefficients and recovering the original relative order is $1/n!^n$, and the time complexity is $O(n!^n)$. That is, the probability of obtaining the key is $1/n!^n$. According to Lemma 1, we have that this scheme cannot be cracked within linear time, and related information of key cannot be obtained.

5 Design of parallel algorithm based on MapReduce

This paper proposes a parallel floating-point number encryption scheme based on MapReduce, which combines the parallel characteristics of cloud computing with the floating-point encryption algorithm to realize parallel encryption through plaintext blocking, and it has significantly increased the encryption efficiency.

5.1 Algorithm procedure

In the MapReduce programming model, the Split function is used to split the input data into data blocks with fixed size according to the user's requirement, and then, these blocks will be distributed to different slave nodes by the master node based on

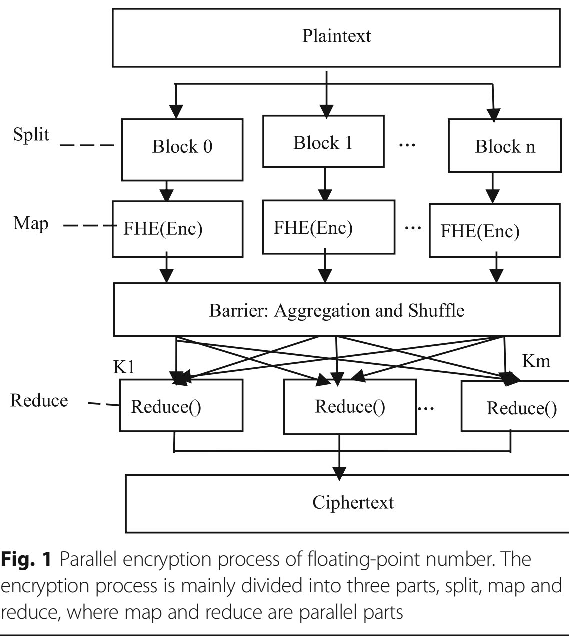

Fig. 1 Parallel encryption process of floating-point number. The encryption process is mainly divided into three parts, split, map and reduce, where map and reduce are parallel parts

corresponding scheduling mechanism. The Map function conducts corresponding operation of each data block after splitting based on the user-defined encryption algorithm. Each Map completes one part of final result, and each Reduce is responsible of integrating all partial results completed by Maps. Each encryption computation of parallel encryption scheme is independent, so it can be distributed to multiple Maps for simultaneous encryption. It can be defined as a cubic polynomial time algorithm $\prod = (Split, Map, Reduce)$, and the specific process is as shown in Fig. 1.

5.2 Split algorithm

Assume the plaintext file size is L(MB), the number of cluster processing nodes is P, and the original file is split into t data blocks ($t \geq 1$). We give the specific split algorithm as algorithm 1. In which, *pos* refers to the location of currently processed data in the original file; i represents the ith data block; l_i represents the size of the ith block ($i \leq t$). The specific process is as follows: Open the big-data file to be encrypted, use the *pos* variable to save the offset of first byte in the file, and when current data under processing has not reached the end of file, use Eq. (22) to calculate the length l_i of the ith fragment. Filebuffer refers to the content saved in the ith block; Key refers to the offset of the start of ith block in the file; value is the value saved in each filebuffer. *pos* points to the next shard The size of each data block l_i can be calculated based on Eq. (22).

Algorithm 1. Spliting algorithm

```
Input : plaintext file
Output : key,value
1: Readfile()
2: pos←start;
3: i←1;
4: WHILE pos<end DO
5:     Compute $l_i$
6:     filebuffer.readFull($l_i$);
7:     Key.set(pos);
8:     Value.set(filebuffer);
9:     pos←pos+ $l_i$;
10:    i=i+1;
11: END DO
```

$$l_i = \begin{cases} 2^{20} \times \lceil L/t \rceil, i < n \\ 2^{20} \times [L - \lceil L/t \rceil \times (t-1)], i = n \end{cases} \tag{22}$$

5.3 "Map" function and "Reduce" function

After the splitting stage, each Mapper will independently compute part of ciphertext. Each Mapper will conduct the encryption process of steps 1 ~ 4 in the encryption algorithm of Section 4.1, before each encryption of data, generate $n - 1$ pairs of encrypted noise; then, compute the order child ciphertext c_i according to Eq. (7); finally, generate out-of-order ciphertext with the mapping function.

The definition of the specific interface of Map ( ) function is as follows: public void map (Object key, Text value, Context context) throws IOExecption, Interrupted Exception. In which, the parameter key is the key value passed into map; value is the value of corresponding key value; and context is the context object parameter, which is the context object of Hadoop to accessed by the program. For each floating-point number in each data block, repeat executing the Map algorithm in Algorithm 2.

Reduce function waits for the partial ciphertext computation by all Map functions to be completed and then conducts sorting according to the key value. Because the key value is the offset of text, the sorting result is the read-in order of file. When writing in the file, only the value part is output, and the final file splices partial ciphertexts based on the order and forms splices ciphertext for output.

5.4 Performance analysis

The encryption process of floating-point number FFHE scheme can be divided into two stages: the preparation stage and encryption stage. The first stage mainly involves the generation and check of key, the second stage mainly involves the data encryption operation, and they are the main parts of algorithm performance analysis. In the improved FHE scheme, the operation granularity is floating-point number. In the computer, the addition operation, shift operation and assignment operation have close complexity. An X operation is defined in this paper to uniformly express the above three operations.

Assume the file with the size of L(MB), contains N floating-point numbers, the total encryption time of plaintext file is T_{seq}, the generation and check time of key is T_{key}, the encryption time is T_{Enc}, and T_{seq} can be expressed by Eq. (23).

$$T_{seq} = T_{key} + T_{Enc} \tag{23}$$

The preparation stage mainly involves the generation of key $K(n)$, including the generation of two n-dimension vectors, i.e., keys k and s. By adding subsidiary conditions, the preparation stage consists of *2n* fixed assignment operations, so T_{key} consists of *2n X* operations.

The data encryption stage mainly involves the addition, multiplication, division, and assignment operations. The part to generate random noise involves $2(n-1)$ assignment operations. Execute encryption algorithm to real number v and obtain n-dimension ciphertext array. In which, the first $n-1$ child ciphertexts correspond to $2(n-1)$ multiplication operations and $2(n-1)$ addition operations, and there are $4(n-1)$ X operations in total. The child ciphertext c_n mainly consists of two multiplication operations, $n-1$ addition operations and $n-1$ division operations, which can be expressed as $2(n-1)+2$ X operations. The random sorting operation involves n mapping and n assignment operations of mapping function Π, as well as deterministic encryption operation that includes m X operations, which can be expressed as $2n+m$ X operations, and it require the following number of X operations in order to encrypt N floating-point number plaintexts:

Algorithm 2. Map function

```
Input: $v$
Output: $C$
1: Generate noise $P$ , generateNonce ( );
2: For $i$=1 to $n$-1
3:     Compute child ciphertext $c_i$ ;
4: End for
5: Compute child ciphertext $c_n$ ;
6: Obtain order ciphertext $C'=(c_1,c_2,\ldots,c_n)$;
7 : Generate out-of-order ciphertext $C''$;
8: Compute child ciphertext $c_{n+1}$ ;
9: Generate ciphertext $C=(c_{d1}, c_{d2},\ldots, c_{dn}, c_{n+1})$.
```

$$[2(n-1)+4(n-1)+2(n-1)+2+2n+m] \times N \tag{24}$$

Assume one X operation takes time of T_{fc}, then, the FHE parallel encryption algorithm that includes N floating-point number takes time of T_{seq}:

$$\begin{aligned} T_{seq} &= T_{key}+T_{Enc} \\ &= [2n+N\times(10n-6+m)]T_{fc} \end{aligned} \tag{25}$$

When $N>>n$, the value of $2n$ (i.e., T_{key}) can be ignored; however, with the increase of N, the encryption time generally presents linear increase.

Table 1 Software and hardware configuration

Product name	The parameter and model
Cash	3.2 GHz/8 M
Memory bank	16 GB(2 × 8 GB)1333 MHz
Dual ranked	RDIM
Hard disk	1 TB 3.5-in. 7200 RPM SATA II
Operating system	CentOS Linux Server6.6
JAVA VM	JAVA 1.7.0
Hadoop	Hadoop-2.5.2

During the parallel encryption process, assume there is no overlapping during the operation process, and the execution time of parallel encryption algorithm consists of four parts, i.e.:

$$T_n = T_{comm}+T_{key}+T_{Map}+T_{reduce} \tag{26}$$

In which, T_{comm} is the communication time, T_{key} is the generation and check time of key, T_{Map} is the parallel encryption time of Map, and T_{reduce} is the time to merge encrypted ciphertexts according to the key value.

During the parallel encryption stage of Map, each slave node would have communication with host during the start and ending stages of task, and the plaintext of N floating-point numbers are divided into t data blocks, so it requires overhead for at least $2\,t$ data communications, and we can set $T_{comm}=\xi_1 t T_{fc}$.

Assume each data block contains x floating-point numbers, then $x=N/t$. T_i represents the time required to encrypt the ith data block, then $T_i = x\times(10n\text{-}6+m)\times T_{fc}$. Set S_T as the speed-up ratio of each Map during the parallel encryption stage, then

$$\begin{aligned} S_T &= \frac{T_{seq}}{T_{comm}+T_{key}+T_i} \\ &= \frac{[2n+N\times(10n-6+m)]T_{fc}}{[\xi_1 t+2n+x\times(10n-6+m)]T_{fc}} \\ &= \frac{2n+N\times(10n-6+m)}{\xi_1 t+2n+x\times(10n-6+m)} \end{aligned} \tag{27}$$

In the actual application scenario, both the file partition number t and the number of child ciphertexts n are significantly smaller than the floating-point number M in the plaintext, i.e., t, $n<<M$. In addition, the communication time can be ignored, so we can know that the speed-up ratio S_T is close to N/x, i.e., the block number t of plaintext.

For plaintext data with the same size, the generated ciphertexts also have the same size. Assume it requires reduce time of T_{ric} for each floating-point number to generate ciphertext, then the plaintext that contains N floating-point numbers requires the time of $T_{reduce} = N\times T_{ric}$, and it can be seen that T_{reduce} is proportional to the size of generated ciphertext.

Table 2 The test results of different size file

File size (MB)	Serial time (seconds)	Parallel time (seconds)	Max Map time (seconds)	Reduce time (seconds)	SP
256	61	33	16	13	1.8
512	123	44	17	22	2.8
768	186	51	18	30	3.6
1024	254	58	17	36	4.4
1280	312	83	18	45	3.8
1536	378	88	17	51	4.3
1792	443	97	17	58	4.6
2048	516	104	18	63	5.0

Considering Reduce takes a lot of time, and the experiment involves 16 nodes, so in this experiment, the number of Reduce is 15, and the time required by parallel encryption can be expressed as:

$$S_T = \frac{T_{seq}}{T_{comm} + T_{key} + T_i} = \frac{[2n + N \times (10n–6 + m)]T_{fc}}{[\xi_1 t + 2n + x \times (10n–6 + m)]T_{fc}} = \frac{2n + N \times (10n–6 + m)}{\xi_1 t + 2n + x \times (10n–6 + m)} \tag{28}$$

when the value of N is high, T_{key} and T_{comm} can be ignored. Therefore, it can be seen that during parallel encryption, if N stays the same, with the increase of t, the time consumed by Reduce also stays the same, the time consumed by Map gradually declines, and the time consumed by Reduce gradually becomes dominant.

If S_P is used to represent the overall speed-up ratio, S_P can be expressed as:

$$S_P = \frac{T_{seq}}{T_n} = \frac{T_{key} + T_{Enc}}{T_{comm} + T_{key} + T_{Map} + T_{reduce}} = \frac{[2n + N \times (10n–6 + m)] \times T_{fc}}{\xi_1 t \times T_{fc} + 2n \times T_{fc} t + T_{fc} \times [(10n–6 + m) \times N/t] \times \lceil t/p \rceil + N/15 \times T_{rci}} \approx \frac{}{\lceil t/p \rceil + t/15 \times T_{rci}/T_{fc} \times 1/(10n–6 + m)} \tag{29}$$

It can be seen that under fixed core number p, when $t \in (kp, kp + p]$ (in which k is a natural number), the speed-up ratio η_T presents growth trend, and it will not exceed p.

6 Experimental results and analysis

The hardware platform of experiment includes 1 Master node and 3 Slave nodes. The Master node is responsible of the monitoring and scheduling of tasks, and the Slave nodes are responsible of the distributive storage data file and computation task, see "Table 1-experiment cluster node configuration" for the specific hardware configuration and software environment for each node.

In this experiment, data test was conducted from two main different perspectives: in the first scenario, plaintext data with different sizes were chosen to compare their encryption speed and speed-up ratio in different serial and parallel environment; in the second situation,

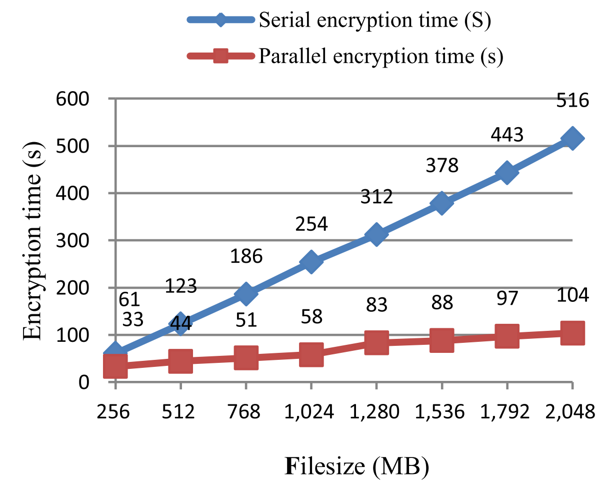

Fig. 2 The encryption time of different size file. The figure compares the encryption time under different file sizes in serial and parallel environments

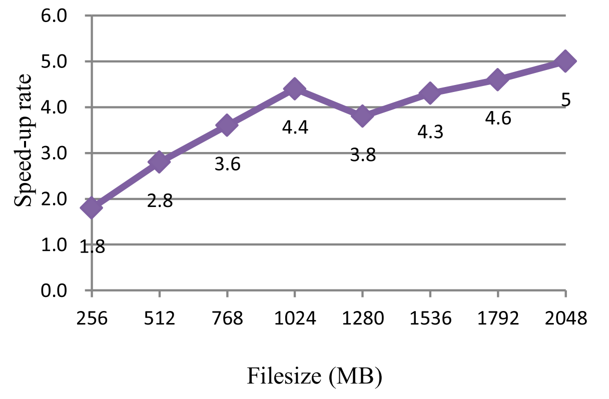

Fig. 3 The speed-up rate of different size file. The figure shows the overall speed-up rate under different file sizes in parallel environments

Table 3 The test results of a 2-GB file on different cores

No. P	Max Map time (seconds)	Reduce time (seconds)	General time (seconds)	Map SP	General SP
1	456	65	526	1.0	1.0
4	129	62	198	3.5	2.6
8	61	64	131	7.4	4.0
12	47	64	117	9.7	4.5
16	33	63	105	13.8	5.0
24	25	64	114	18.2	4.6
32	17	63	107	26.8	4.9

plaintext data with fixed size were chosen to compare their encryption speed and speed-up ratio under different block sizes. In the first situation, the chosen plaintext data had the sizes of 256 MB, 512 MB, 768 MB, 1024 MB, 1280 MB, 1536 MB, 1792 MB, and 2048 MB, the default data block size was 64 MB in the parallel environment, and the encryption test was conducted in both serial and parallel environment. For the second situation, the plaintext data with the sizes of 2G and 4G were chosen in the experiment, the data fragment numbers were 1, 4, 8, 12, 16, 20, 24, 28, and 32, respectively, and their encryption speeds were tested. In the experiment, the size of float-point numbers is 32 bit, and the dimension of the security parameter n is 128 bit.

In the experiment, 4 computation nodes were used, and each node had 4-core CPU, so the CPU had total 16 cores. In the parallel experiment, we found that with the increase of plaintext data volume, the time occupied by Reduce would keep growing, and in order to increase the efficiency, the number of Reduce was all set as 15 in parallel experiment.

In this paper, the file encryption time and overall speed-up ratio under different file sizes are summarized in the serial and parallel environment, and the results are shown in Table 2, Figs. 2 and 3. Tables 3 and 4 have recorded the overall encryption time of file, the execution time of Map process, and the overall speed-up ratio and the speed-up ratio of Map process when the plaintext size is 2G and 4G, respectively.

According to Table 2 and Figs. 2 and 3, it can be seen that under fixed number of nodes: (1) the time required by serial encryption is basically proportional to the plaintext size; (2) the time required by parallel encryption will increase with the increase of plaintext; (3) when $t < p$, the time consumed by Reduce function will gradually increase with the increase of t, and with the time consumed by Map function stayed the same, the proportion of time consumed by Reduce function in the overall parallel encryption process will gradually increase; (4) when $t < p$, the increase of speed-up ratio S_P is fast, and it will reach the highest value when $t = p$. When $t > p$, for each t∈$(kp, kp + p]$, the speed-up ratio presents the trend of slow growth, and it will reach the highest value when $t = (k + 1) \times p$.

Figures 4 and 5 show the encryption time of 2-GB and 4-GB files allocated to different cores in parallel environments. According to Figs. 4 and 5, we can see that with the increase of usable cores and file partitions in the cluster: (1) for plaintext big-data file with certain length, with the increase of Map quantity, the file encryption time presents a general trend of decline, and the time consumption will be the lowest when the Map quantity equals the node number; (2) the time consumed by max Map will gradually decline, because with the increase of Map quantity, the size of each Map data block will decline, and the time-consuming of Map mainly concentrates on the encryption operation, so the time consumed by this part will be low. The time consumed by Reduce is basically the same, because no matter how

Table 4 The test results of a 4-GB file on different cores

No. P	Max Map time (seconds)	Reduce time (seconds)	General time (seconds)	Map SP	General SP
1	901	120	1025	1.0	1.0
4	266	119	389	3.5	2.6
8	125	120	251	7.2	4.1
12	92	119	216	9.8	4.8
16	65	120	194	13.9	5.3
24	47	118	211	19.2	4.9
32	33	119	196	27.3	5.2

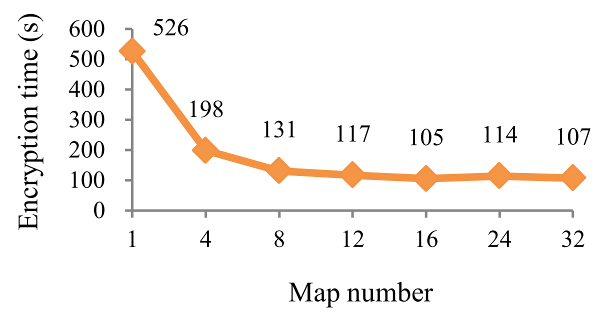

Fig. 4 The encryption time of 2 GB on different cores. The figure shows the encryption time of a 2-GB file allocates to different cores in parallel environments

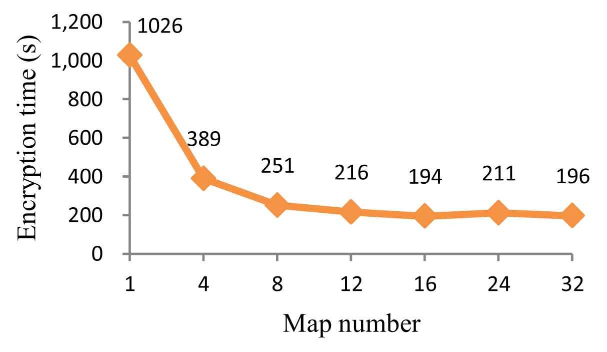

Fig. 5 The encryption time of 4 GB on different cores. The figure shows the encryption time of a 4-GB file allocates to different cores in parallel environments

high the Map quantity is, the number of Reduce is always 15.

Figures 6 and 7 compares the Map speed-up and General speed-up of 2-GB and 4-GB files under different cores in parallel environments. According to Figs. 4 and 5, we can see that with the increase of usable cores in the cluster: (1) the speed-up ratio of Map will increase with the increase of Map quantity, and it will always be lower than the Map quantity, which is consistent with the theoretical analysis in previous section; (2) the speed-up ratio of general also increase with the increase of Map quantity, but the acceleration ratio tends to be stable when the Map quantity equals the node number. During the early stage, with the increase of usable cores, the file encryption time presents significant decline, and the cluster performance can be effectively carried out; when all cores of cluster are used in the computation equation, the increase of Map quantity will have little influence on the improvement of cluster performance, and the file encryption time will become stable.

7 Conclusions

With the rapid development of cyber physical systems technology, the privacy protection problem of data in cyber physical systems has become more and more important. Most of the existing fully homomorphic encryption algorithms are limited to process the integer type. In order to expand the practical application range of the existing fully homomorphic encryption algorithm, we propose a parallel fully homomorphic encryption scheme that supports floating-point operation. The proposed scheme can enhance the algorithm security by using out-of-order ciphertexts operations. In addition, we also design and implement an efficient algorithm performed on the MapReduce platform based on the proposed scheme. Specifically, during the encryption process, a file is divided into different number of data blocks, and the algorithm's parallelism can be controlled by specifying the usable cores and the number of partitions. Meanwhile, the multiple Reduce functions can be parallel carried out to alleviate the high real-time cost of Reduce operation. The experimental results show that, compared to the traditional linear encryption algorithm, the proposed algorithm obtains the greater speed-up ratio when processing big data files in MapReduce cluster.

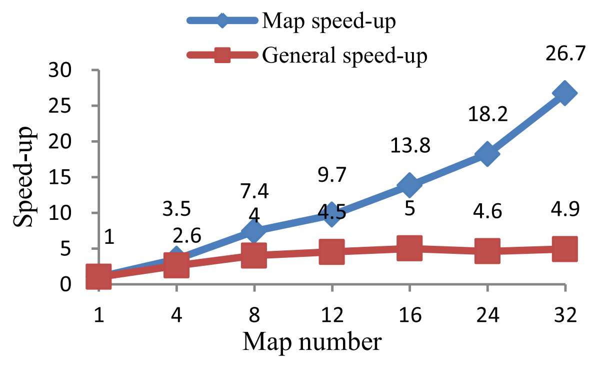

Fig. 6 The speed-up rate of 2 GB on different cores. The figure compares the Map speed-up and General speed-up of a 2-GB file under different cores in parallel environments

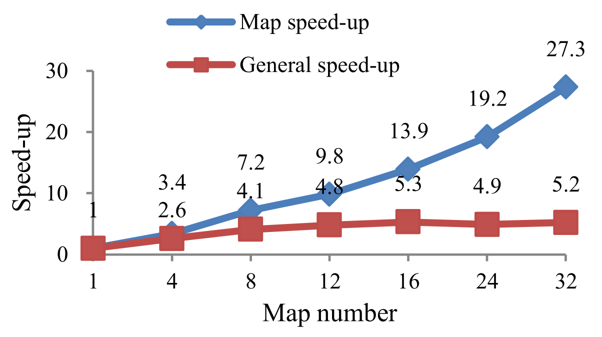

Fig. 7 The speed-up rate of 4 GB on different cores. The figure compares the Map speed-up and General speed-up of a 4-GB file under different cores in parallel environments

Funding
This work was supported in part by the National Natural Science Foundation of China under the Grant 61872197, Grant 61572263, Grant 61502251 and Grant 61502243, in part by the Colleges and Universities in Jiangsu Province plans to graduate research and innovation under Grant KYLX_0816, and in part by the Natural Science Foundation of Jiangsu Province under the Grant BK20151511 and Grant BK2016151.

Availability of data and materials
We declared that materials described in the manuscript will be freely available to any scientist wishing to use them for non-commercial purposes, without breaching participant confidentiality.

Authors' contributions
ZM and GY designed the study, performed the research, analyzed the data, and wrote the paper. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note
Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details
[1]School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210046, China. [2]Jiangsu Key Laboratory of Big Data Security& Intelligent Processing, Nanjing 210046, China. [3]School of Computing Science and Engineering, Vellore Institute of Technology (VIT), Vellore, Tamil Nadu, India.

Received: 16 October 2018 Accepted: 11 December 2018
Published online: 21 January 2019

References
1. F. Pasqualetti, F. Dörfler, F. Bullo, Attack detection and identification in cyber-physical systems. IEEE Trans. Autom. Control **58**(11), 2715–2272 (2013)
2. R. Rajkumar, I. Lee, L. Sha, et al., 44.1 Cyber-physical systems: The next computing. Theol Rev. **14**(6), 731–736 (2010)
3. J. Wang, R. Zhu, S. Liu, A differentially private unscented Kalman filter for streaming data in IoT. IEEE Access **6**(99), 6487–6495 (2018)
4. R. Zhu, X. Zhang, X. Liu, et al., ERDT: Energy-efficient reliable decision transmission for intelligent cooperative spectrum sensing in industrial IoT. IEEE Access. **3**(28), 2366–2378 (2015)
5. K. Zhu, R. Zhu, H. Nii, et al., PaperIO: a 3D interface towards the internet of embedded paper-craft. IEICE Trans inf System. **97**(10), 2597–2605 (2014)
6. B. Dan, E. Kushilevitz, R. Ostrovsky, W.E. Skeith, *Public key encryption that allows PIR queries, Advances in Cryptology CRYPTO* (2007), pp. 50–67
7. H. Avni, S. Dolev, N. Gilboa, X. Li, in *Proc. of ALGOCLOUD, Patras, Greece*. SSSDB: Database with private information search (2015), pp. 49–61
8. Q. Liu, G. Wang, J. Wu, Secure and privacy preserving keyword searching for cloud storage services. J. Netw. Comput. Appl. **35**(3), 927–933 (2012)
9. K. Gu, W.J. Jia, J.M. Zhang, Identity-based multi-proxy signature scheme in the standard model. Fund. Inform. **150**(2), 179–210 (2017)
10. X.B. Shen, W. Liu, I.W. Tsang, et al., Multilabel prediction via cross-view search. IEEE Trans. Neural Netw. Learn. Syst. **29**(9), 4324–4338 (2018)
11. X.B. Shen, F.M. Shen, Q.S. Sun, et al., Semi-paired discrete hashing: Learning latent hash codes for semi-paired cross-view retrieval. IEEE Trans. Cybern. **47**(12), 4275–4288 (2017)
12. C.Y. Yin, J.W. Xi, R.X. Sun, J. Wang, Location privacy protection based on differential privacy strategy for big data in industrial internet of things. IEEE Trans. Ind. Inf. **14**(8), 3628–3636 (2018)
13. S.K. Pasupuleti, S. Ramalingam, R. Buyya, An efficient and secure privacy-preserving approach for outsourced data of resource constrained mobile devices in cloud computing. J. Netw. Comp. Appl. **64**(C), 12–22 (2016)
14. S. Gajek, *Symmetric Searchable Encryption from Constrained Functional Encryption, in Cryptographers' Track at the RSA Conference* (Springer, Cham, 2016), pp. 75–89
15. M. Long, F. Peng, H.Y. Li, Separable reversible data hiding and encryption for HEVC video. J. Real-Time Image Proc. **14**(1), 171–182 (2018)
16. J. Wang, C.W. Ju, Y. Gao, A.K. Sangaiah, G.J. Kim, A PSO based energy efficient coverage control algorithm for wireless sensor networks. Comp. Mater. Continua **56**(3), 433–446 (2018)
17. J. Wang, Y. Cao, B. Li, H.J. Kim, S.Y. Lee, Particle swarm optimization based clustering algorithm with mobile sink for WSNs. Futur. Gener. Comput. Syst. **76**, 452–457 (2017)
18. Y. Lindell, B. Pinkas, Secure multiparty computation for privacy -preserving data mining. J. Priv. Confid. **25**(2), 761–766 (2009)
19. I. Damgård, A. Polychroniadou, V. Rao, in *Proc. of PKC, New York*. Adaptively secure multi-party computation from LWE via equivocal FHE (2016), pp. 208–233
20. X.B. Shen, F.M. Shen, L. Li, et al. Multiview discrete hashing for scalable multimedia search, in Proc. of ACM TIST, 2018
21. Q. Zhou, G. Yang, S. Li, L. Chen, An integrity-checking private data aggregation algorithm. J. Electron. Inform. Technol, **35**(6), 1277–1283 (2013)
22. J. Wang, Z.Q. Zhang, B. Li, S.Y. Lee, R.S. Sherratt, An enhanced fall detection system for elderly person monitoring using consumer home networks. IEEE Trans. Consum. Electron. **60**(1), 23–29 (2014)
23. E.B. Tirkolaee, A.A.R. Hosseinabadi, M. Soltani, A.K. Sangaiah, J. Wang, A hybrid genetic algorithm for multi-trip green capacitated arc routing problem in the scope of urban services. Sustain **10**, 5 (2018)
24. D. Micciancio, A first glimpse of cryptography's holy grail. Commun. ACM **53**(3), 96–96 (2010)
25. R.L. Rivest, L. Adleman, M.L. Dertouzos, in *Foundations of Secure Computation, London*. On data banks and privacy homomorphisms (1978), pp. 169–179
26. R.L. Rivest, L. Adleman, M.L. Dertouzos, A method for obtaining digital signatures and public key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
27. T. Elgamal, A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inform. Theor. **31**(4), 469–472 (1985)
28. P. Paillier, in *Proc. of Eurocrypt'99*. Public-key cryptosystems based on composite degree residuosity classes (1999), pp. 223–238
29. C. Gentry, in *Proc. of the Annual ACM Symposium on Theory of Computing , Bethesda*. Fully homomorphic encryption using ideal lattices (2009), pp. 169–178
30. M. Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan, in *Proc. of EUROCRYPT'2010*. Full homomorphic encryption over the integers (Springer, Berlin, 2010), pp. 24–43
31. Z. Brakerski, V. Vaikuntanathan, in *Foundations of Computer Science. IEEE*. Efficient fully homomorphic encryption from (standard) LWE (2011), pp. 97–106
32. D. Stehlé, R. Steinfeld, *Making NTRU as Secure as Worst-Case Problems over Ideal Lattices, in Proc. of EUROCRYPT'2011, LNCS* (2011), pp. 27–47
33. Z. Brakerski, C. Gentry, V. Vaikuntanatha, in *Proc. of the 3rd Innovations in Theoretical Computer Science Conf, New York*. (Leveled) fully homomorphic encryption without bootstrapping (2012), pp. 309–325
34. Z. Brakerski, V. Vaikuntanathan, in *Proc. of CRYPTO, Berlin*. Fully homomorphic encryption from ring-LWE and security for key dependent messages (2011), pp. 505–524
35. A. López-Alt, E. Tromer, V. Vaikuntanathan, in *Proc. of the Annual ACM Symposium on Theory of Computing, New York*. On-the-fly multiparty computation on the cloud via multikey full homomorphic encryption (2012), pp. 1219–1234
36. C. Gentry, A. Sahai, B. Waters, in *Proc. of the 33rd Annual International Cryptology Conference, Berlin*. Homomorphic encryption from learning with errors: Conceptually- simpler, asymptotically-faster, attribute -based (2013), pp. 75–92
37. J.H. Cheon, J.S. Coron, J. Kim, M.S. Lee, T. Lepoint, in *Proc. of CRYPTO 2013, Berlin*. Batch full homomorphic encryption over the integes (2013), pp. 315–335
38. J.N. Gaithuru, M. Bakhtiari, *Insight Into the operation of NTRU and a Comparative Study of NTRU, RSA and ECC public key cryptosystems, in Software Engineering Conference* (2014), pp. 273–278
39. H. Chen, Y.P. Hu, Z. Lian, Double batch for RLWE-based leveled fully homomorphic encryption. Chin. J. Electron. **24**(3), 661–666 (2015)
40. J.H. Cheon, J. Kim, M.S. Lee, A. Yun, CRT-based fully homomorphic encryption over the integer. Inform. Sci. An Intern. J **310**(C), 149–162 (2015)
41. S. Garg, C. Gentry, S. Halevi, A. Sahai, B. Waters, in *Proc. of 33rd Annual Cryptology Conf , Berlin*. Attribute-based encryption for circuits from multilinear map (2013), pp. 479–499
42. D. Tan, H. Wang, Fully homomorphic encryption based on the parallel computing. KSII Trans. Int. Inform . Syst **12**(1), 497–522 (2018)
43. D. Liu, Practical Fully Homomorphic Encryption without Noise Reduction. Cryptology ePrint Archive, [Online]. Available: http://eprint.iacr.org/2015/468.pdf

44. D. Liu, *Efficient processing of encrypted data in honest-but-curious clouds, IEEE cloud (2017), IEEE Computer Society* (2017), pp. 970–974
45. X. Liu, R.H. Deng, K.K.R. Choo, J. Weng, An efficient privacy-preserving outsourced calculation toolkits with multiple keys. IEEE Trans. Inform. Forensics Sec. **11**(11), 2401–2414 (2016)
46. X. Liu, R. Choo, R. Deng, R. Lu, J. Wengl, *Efficient and Privacy-Preserving Outsourced Calculation of Rational Numbers, IEEE Trans on Dependable and Secure Computting, PP(99), 27–39* (2018)
47. M. Brenner, J. Wiebelitz, G.V. Voigt, M. Smith, in *Proc. of IEEE International Conference on Digital Ecosystems and Technologies Conference(DEST)*. Secret program execution in the cloud applying homomorphic encryption (2011), pp. 114–119
48. C.S. Lu, Homomorphic encryption-based secure SIFT for privacy-preserving feature extraction. Proc. of SPIE **7880**(2), 788005 (2011)
49. M. Naehrig, K. Lauter, V. Vaikuntanathan, in *Proc. of ACM CCSW, ACM, Chicago*. Can homomorphic encryption be practical? (2011), pp. 113–124
50. K. Gjøsteen, M. Strand. Fullyhomomorphic encryption must be fat or ugly? Cryptology ePrint Archive, [Online]. Available: http:// eprint.iacr.org/2016/105.pdf
51. M. Chase, K. Lauter, J. Benaloh, A.Z. Horvitz, *Patient controlled encryption: Patient privacy in electronic medical records , in Proc of Cloud Computing Security Workshop* (2009), pp. 103–114
52. Y. Tu, Y. Lin, J. Wang, J.U. Kim, Semi-supervised learning with generative adversarial networks on digital signal modulation classification. Comp. Mater. Continua **55**(2), 243–254 (2018)
53. D.J. Zeng, Y. Dai, F. Li, R.S. Sherratt, J. Wang, Adversarial learning for distant supervised relation extraction. Comp. Mater. Continua **55**, 121–136 (2018)
54. J. Wang, J.Y. Cao, S. Ji, J.H. Park, Energy efficient cluster-based dynamic routes adjustment approach for wireless sensor networks with mobile sinks. J. Supercomput. **73**(7), 3277–3290 (2017)
55. J. Yao, K. Zhang, Y.T. Yang, J. Wang, Emergency vehicle route oriented signal coordinated control model with two-level programming. Soft. Comput. **22**(13), J4283–J4294 (2018)
56. Y.J. Ren, Y.P. Liu, S. Ji, A.K. Sangaiah, J. Wang, Incentive Mechanism of Data Storage Based on Blockchain for Wireless Sensor Networks, Mobile Information Systems, Volume 2018, Article ID 6874158 2018, 10.1155/2018/6874158/
57. Z. Min, G. Yang, J.Q. Shi, A privacy-preserving parallel and homomorphic encryption scheme. Open Physics **15**(1), 135–142 (2017)
58. J.H Cheon , A. Kim , M. Kim and Y Song. Floating-Point Homomorphic Encryption, [Online]. Available: http:// eprint.iacr.org /2016/421.pdf
59. Arita S, Nakasato S. Fully Homomorphic Encryption for Point Numbers , [Online]. Available: http://eprint.iacr.org /2016/402.pdf
60. Costache A, Smart N P, Vivek S, et al. Fixed point arithmetic in she schemes, [Online]. Available: http://eprint.iacr.org /2016/250.pdf
61. Armknecht F, Boyd C, Carr C, et al. A Guide to Fully Homomorphic Encryption, [Online]. Available: http://eprint.iacr.org /2015/1192.pdf