Research Article

Zhaoe Min*, Geng Yang, and Jingqi Shi

A privacy-preserving parallel and homomorphic encryption scheme

DOI 10.1515/phys-2017-0014
Received December 2, 2016; accepted December 16, 2016

Abstract: In order to protect data privacy whilst allowing efficient access to data in multi-nodes cloud environments, a parallel homomorphic encryption (PHE) scheme is proposed based on the additive homomorphism of the Paillier encryption algorithm. In this paper we propose a PHE algorithm, in which plaintext is divided into several blocks and blocks are encrypted with a parallel mode. Experiment results demonstrate that the encryption algorithm can reach a speed-up ratio at about 7.1 in the MapReduce environment with 16 cores and 4 nodes.

Keywords: privacy protection, homomorphic encryption, parallel encryption, parallel homomorphic encryption, cloud computing, MapReduce

PACS: 9.70.-a, 89.70.EG

1 Introduction

Since 2009, cloud servers such as Microsoft, Amazon and Google have experienced security incidents and the security of cloud computing has become the paramount problem constraining its development. In 2011, Cloud Security Alliance (CSA) presented the cloud security model, emphasising the issue of privacy protection in cloud computing. The privacy problem in cloud computing results from the features of data outsourcing and service renting. Users lose their direct control of data while they store the data in the cloud, which can lead to the leakage and abuse of personal privacy data. Some privacy preserving methods are proposed in Papers [1, 2]. Furthermore, it is considered that data privacy can be protected from the two aspects of data encryption and user identity management. The homomorphic encryption (HE) technique supports the management of ciphertext data under the privacy protection, and can directly apply to ciphertexts for retrieval, computation and statistics in the clouds. Afterwards, the results are returned to the user in the form of ciphertexts, in contrast to the conventional encryption algorithm, which can eliminate the need for frequent operations of encryption and decryption between the clouds and the users, thus reducing the communication overhead and computation resources.

Cloud computing clusters possess large numbers of computational nodes, and they can be used for storing and processing mass data. It is easy to construct a cloud computing environment, therefore, many enterprises and research institutes have built private cloud computing platforms. We can conduct encryption on the data which need to be stored in the public clouds using the parallel features of the private cloud clusters and the computational capacities of the various nodes. Therefore, with pertinence to the problem of mass data and the frequent visits and clouds utilization, it is of great importance to speed up the processing capacities of mass data encryption, and then conduct parallel process of data encryption computations in combination with the features of cloud computing and the networks.

2 Related works

HE was first proposed by Rivest et al. in 1978 as a concept of “privacy homomorphism” [3], and is an encryption scheme which can directly conduct computations on the ciphertexts. Subsequently, RSA [4] and the ElGamal algorithm [5] with multiplicative homomorphism as well as Paillier [6] and the BNG algorithm [7] with additive homomorphism were successively proposed. Particularly, the above algorithms belong to the category of homomorphic encryption algorithms.
By 2009, Gentry proposed the first fully homomorphic scheme based on the ideal lattice [8], which can conduct any number of addition and multiplication operations on the ciphertext. Since then, more fully homomorphic scheme acts have been proposed, such as the fully homomorphic scheme DGHV [9] based on integers, BGV scheme [10] based on the RLWE, GSW13 scheme [11] based on the approximate eigenvectors, the multiple keys fully homomorphic ciphertext scheme [12] based on NTRU, and so on.

The low encryption efficiency is a problem which is mutually faced by the somewhat homomorphic encryption (SHE) algorithm and the fully homomorphic encryption (FHE) algorithm. In order to elevate the encryption efficiency, various kinds of schemes have been proposed. For example, papers [13–16] introduce a scheme to elevate the efficiency of the RSA algorithm through different parallel methods, such as GPU, CPU or tree architecture. Wang et al. elaborated on the scheme of parallel technique on MapReduce so as to elevate the efficiency of Hill Cipher [17].

As for the fully homomorphic scheme, Gu proposed a kind of HE scheme which is based on the deal ellipsoid lattice together with SIMD technique [18]. Cheon improved the scheme of DGHV, and proposed an optimal method of batch processing plaintext [19]. Considering that the addition and multiplication calculations in the ciphertext homomorphism in the GSW scheme are only based on the addition and multiplication of the matrices, Hiromasa proposed a method to compress ciphertexts and optimize bootstrap [20]. Moreover, the scheme was the first fully homomorphic method which can conduct encryptions of the matrices and support the homomorphic operations on the matrices.

Among all different kinds of the above mentioned HE algorithms, Hill Cipher has been proven to be lacking sufficient security. For the Paillier and RSA algorithms with the same length key, the Paillier algorithm has higher security. On the other hand, the Paillier algorithm is more suitable for parallel due to its property of additive homomorphism. More research has been done on the fully homomorphic encryption algorithm in recent years. However, due to the problem of encryption efficiency, most of them are still in the experimental stage and are not widely applied in practice.

With the goal of privacy protection, this paper realizes a kind of parallel encryption algorithm based on the MapReduce. Firstly, we use the additive homomorphism of the Paillier algorithm to divide the data that requires encryption into several approximately equal numbers according to the parity of each data packet. Then, using the parallelism in MapReduce, we realize the parallel encryption by grouping the plaintext. Experimental results show that the proposed scheme can achieve the speedup of 7.1 while maintaining security, thus solving the problem of low computational efficiency for a somewhat homomorphic encryption algorithm.

### 3 Background Knowledge

#### 3.1 Paillier algorithm

In 1999, Paillier proposed a kind of probabilistic public key crypto system based on a high order residual class, i.e. the Paillier algorithm [6]. The concrete encryption scheme is defined as follows.

Generating key: Set \( n = p \times q \) and \( \lambda = \text{lcm}(p - 1)(q - 1) \), where \( p \) and \( q \) are large primes, and then we randomly select a base \( g \in \mathbb{Z}_n^* \), this can be done efficiently by checking whether

\[
gcd(L(g^3 \mod n^2), n) = 1
\]  

(1)

where the function \( L \) is defined as: \( L(u) = \frac{u - 1}{n} \). Now, \((n, g)\) is regarded as public keys while the pair \((p, q)\) (or equivalently \( \lambda \)) remains private.

Encryption: Plaintext \( m \in \mathbb{Z}_n \), a random number of \( r \in \mathbb{Z}_n^* \) is selected, then the ciphertext is defined as follows:

\[
c = E(m) = g^m r^n \mod n^2
\]

(2)

Decryption: Ciphertext \( c \in \mathbb{Z}_n^* \), Plaintext \( m \) is defined as follows:

\[
m = D(c) = \frac{L(c^3 \mod n^2)}{L(g^3 \mod n^2)} \mod n
\]

(3)

It can be observed that Paillier’s public key encryption algorithm has additive homomorphism, and the proving process is listed as follows:

\[
E(m_1) \times E(m_2) = (g^{m_1} r_1^n \mod n^2)(g^{m_2} r_2^n \mod n^2) = g^{m_1 + m_2} (r_1 r_2)^n \mod n^2
\]

\[
E(m_1 + m_2) = g^{m_1 + m_2} r^n \mod n^2
\]

(4)

where \( r \) is a random number. Set \( r = r_1 \times r_2 \), then \( E(m_1 + m_2) = E(m_1) \times E(m_2) \) are satisfied.

#### 3.2 MapReduce Model

From the perspective of the software system, the hadoop system includes two parts: the distributed storage and the
parallel computations. The Hadoop Distributed File System (denoted as HDFS) is mainly used for the distributed storage of massive-scale data, and among them, the master node controls and manages the whole system of distributed files which are named as NameNodes. Furthermore, each slave node which is responsible for data storage is defined as a DataNode.

Inorder to realizetheparallelcomputation processing of the large data stored in the HDFS, Hadoop is used to propose a parallel computational framework (named MapReduce). The computational framework contains a master node (named JobTracker) and a multitude of slave nodes are defined as TaskTrackers. The JobTracker is mainly responsible for scheduling and managing the tasks in the operations, while the TaskTracker is responsible for executing the tasks distributed by the JobTracker.

The MapReduce parallel computing framework is a parallelized program execution system that provides a parallel processing model and a process comprising two stages of Map and Reduce. In the Map phase, the input tuples are partitioned by each Mapper, and tuples with the same partition value are passed to the same Reducer for correlation calculation.

4 An Encryption Scheme Based on the Additive Homomorphism

The encryption algorithm in this work mainly uses the additive homomorphism of the Paillier algorithm to encrypt, and is defined as follows.

\[ E(m) = E(m_1 + m_2) = E(m_1) \times E(m_2) \]

To ensure the security of the encryption algorithm, the length of the public key is set to 256 bit, and the specific encryption scheme is listed as follows.

1. According to the definition of the Paillier encryption algorithm, public keys \((n, g)\) and private keys \((p, q)\) are generated.

2. Assume that the length of the plaintext \((L\ MB)\) can be divided into \(x\) groups of 256 bits in length, and the number of packets \(x\) is computed as follows

\[ x = \left\lfloor \frac{L \times 2^{20}}{256} \right\rfloor = \left\lfloor \frac{L \times 2^{20}}{32} \right\rfloor \]

3. For each packet, the plaintext is firstly converted to BigInteger type, the value of which is represented as \(m\), and \(m = m_1 + m_2\) is satisfied. Afterwards, the values of \(m_1\) and \(m_2\) are computed. If \(m\) is an even number \((m \ mod\ 2 = 0)\), then \(m_1 = m_2 = m \div 2\), else the values of \(m_1\) and \(m_2\) are given as follows.

\[ \begin{cases} m_1 = \frac{m - 1}{2} \\ m_2 = \frac{m + 1}{2} \end{cases} \quad (m \ mod\ 2) = 1 \]

4. \(E(m_1)\) and \(E(m_2)\) are calculated respectively, and the specific formula is given as follows.

\[ \begin{align*} E(m_1) &= g^{m_1}r_1^n \ mod\ n^2 \\ E(m_2) &= g^{m_2}r_2^n \ mod\ n^2 \end{align*} \]

The ciphertexts are \(E(m_1)\) and \(E(m_2)\). We let \(r_1 = r_2\) while \(m_1 = m_2\), hence, we just need to calculate \(E(m_1)\) for \(E(m_1) = E(m_2)\).

During the decryption process, according to the additive homomorphism of Paillier, the algorithm can be figured out as follows.

\[ E(m) = E(m_1) \times E(m_2) \mod\ n^2 \]

Then, using the decryption formula (Eq. 3), the plaintext can be established.

5 Parallel Encryption Scheme based on the MapReduce

As is illustrated in this paper, the parallel encryption scheme based on the MapReduce combines the cloud environment and additive homomorphism of the Paillier algorithm, and realizes parallel encryptions by conducting blocking on the plaintexts. Thus, the encryption efficiency can be greatly enhanced.

5.1 Algorithm Flow

In the programming model of MapReduce, Split function is used to divide the input data into splints with fixed sizes based on the user’s needs. Afterwards, the master node distributes them to different processors based on the relevant scheduling mechanism. The Map function conducts relevant operations on each splint after splitting blocks based on an algorithm self-defined by the user, with each Map accomplishing one part of the results. The Reduce function is responsible for integrating the results of all the Maps. The encryption calculation of each splint of the parallel scheme is independent. Therefore, it can be distributed to multiple Maps for encryption at the same time, and it can be defined as a tuple of polynomial time algorithms

\[ \Pi = (\text{Split, Map, Reduce}) \]
The specific process is as shown in Fig. 1.

![Diagram showing the parallel encryption process of Paillier](image)

**Figure 1:** Parallel encryption process of Paillier

5.2 Split Algorithm

We assume that $p$ is the number of the cores for set processing, and the primary file is divided into $n$ blocks ($n > 1$), and $\text{blocksize}_i$ indicates the size of the $i$-th block ($i \leq n$). As the length of the encryption key is 256 bits, the size of the plaintext split is an integral multiple of 32 bits, and specifically the size of each data block can be calculated with Eq. 10.

$$B_{\text{size}}_i(B) = \begin{cases} \lfloor \frac{L \times 2^{20}}{32 \times n} \rfloor \times 32 & i < n \\ \lfloor \frac{L \times 2^{20} - (n - 1) \times \frac{L \times 2^{20}}{32 \times n}}{32} \rfloor \times 32 & i = n \end{cases}$$ (10)

Fig. 2 shows the specific block algorithm. The content of the $i$-th split is stored using the file buffer, and Key indicates the offset of the initial part of $i$-th split in the file. Moreover, the value is stored in each file buffer.

5.3 Map Functions and Reduce Functions

Map ( ) is a single task, and the Mapper class refers to the one for realizing the Map tasks. Hadoop provides an abstract Mapper base class. We inherit this base class, and realize the functions of the relevant interfaces among them. The detailed interface of the Map ( ) method is defined as follows: public void map (Object key, Text value, Context context) throws IOException Interrupted Exception. The parameter key is the key value transmitted into the map, and the values is the corresponding key value, the context is parameter of the environment subject that an environmental subject for the access of Hadoop by the programs. Map functions repeatedly on each group of data in each block data. Fig. 3 shows the encryption algorithm implemented in each Map ( ).

Input: key,value
Output: key,ci
if ($m \mod 2 = 0$) then
{ $m_1 = m/2$ $c_i = g^{m_1} \mod n^2$ }
else
{ $m_1 = (m - 1)/2$ $m_2 = m_1 + 1$ $c_{i1} = g^{m_1} \mod n^2$ $c_{i2} = g^{m_2} \mod n^2$ }

Figure 3: Map function
environment subjects for the accesses of Hadoop by the programs.

The <key, value> are merged using the key values of the Map functions outputs, and the different values under the same keys are merged into the same list, they are sequenced based on the key values. When the files are written, only the value portion is output without recording the key values. As the key values are the offsets of the texts, the local ciphertext can be spliced into the complete ciphertext output in order.

5.4 Performance Analysis

We assume that the entire encryption time of the Paillier algorithm, key generation and the encryption times are $T_{seq}$, $T_{key}$ and $T_{Enc}$ respectively. Then $T_{seq}$ can be expressed as $T_{seq} = T_{key} + T_{Enc}$. We can know that the main computation overhead in the encryption scheme is the modular exponentiation computation.

Assuming that the time cost for a single modular exponentiation operation is $T_{pd}$, then the time cost for conducting one encryption computation is represented as $2T_{pd}$. Based on the above assumption, the plaintext with a length of $L(MB)$ is divided into $x$ groups. Hence, the time needed to encrypt the plaintext of length $L(MB)$ is defined as $T_{Enc} = 2 \times x \times T_{pd}$, the entire encryption time is $T_{seq} = T_{key} + 2 \times x \times T_{pd}$. Based on Eq. 5, the following equation can be obtained

$$T_{seq} = T_{key} + 2 \times \left\lfloor \frac{L \times 2^{20}}{32} \right\rfloor \times T_{pd} \quad (11)$$

When the value of $L$ is bigger, $T_{key}$ can be ignored, and it can be seen that with the increase of $L$ value, the encryption time $T_{seq}$ basically presents a trend of linear rise.

During the process of the parallel encryptions, we assume that the operational process has no overlapping, the execution time $T_n$ of the parallel encryption algorithm is composed of four parts, that is

$$T_n = T_{comm} + T_{key} + T_{Map} + T_{reduce} \quad (12)$$

where $T_{comm}$ is the communication time, $T_{Map}$ is the parallel encryption time of Map, and $T_{reduce}$ is the time for conducting merging of the ciphertext based on the key values after encrypting.

Assume that the CPU has $p$ cores, the plaintext with a length of $L(MB)$ is divided into $n$ blocks, and the processing time of the i-th block on the CPU core by a map is $T_i$, then

$$T_n = T_{comm} + T_{key} + \text{Max}(T_i) \times \left\lfloor \frac{n}{p} \right\rfloor + T_{reduce} \quad (13)$$

Based on the size of each splint given by Eq. 10, it can be inferred that each block of plaintext contains the number of packets $B_x = \left\lfloor \frac{L \times 2^{20}}{32 \times n} \right\rfloor$, then

$$T_i = 2 \times \left\lfloor \frac{L \times 2^{20}}{32 \times n} \right\rfloor \times T_{pd} \quad (14)$$

Assuming the same size of plaintext data, the size of the generated ciphertext is the same, and the Reduce time needed by each group of data for the generation of the ciphertext is $T_{ric}$. Afterwards, the Reduce time $T_{reduce}$ needed by the entire ciphertexts is listed as follows

$$T_{reduce} = x \times T_{ric} = \left\lfloor \frac{L \times 2^{20}}{32} \right\rfloor \times T_{ric} \quad (15)$$

It can be learned that $T_{reduce}$ presents a proportional relation with the size of the generated ciphertext, and it can be derived as follows.

$$T_n = T_{comm} + T_{key} + 2 \times \left\lfloor \frac{L \times 2^{20}}{32 \times n} \right\rfloor \times T_{pd} \times \left\lfloor \frac{n}{p} \right\rfloor$$

$$+ \left\lfloor \frac{L \times 2^{20}}{32} \right\rfloor \times T_{ric} \quad (16)$$

If the value of $L$ is bigger, $T_{key}$ and $T_{comm}$ can be ignored. According to Eq. 16, we can conclude that with the value increasing, the main overhead of the encryption time $T_n$ is shifted from being dominated by Map to Reduce.

The integral speed-up ratio $S_p$ can be expressed as follows

$$S_p = \frac{T_{seq}}{T_n} = \frac{T_{key} + T_{Enc}}{T_{comm} + T_{key} + T_{Map} + T_{reduce}} \quad (17)$$

As $T_{comm}$ and $T_{key}$ can be ignored, then the following equation can be obtained.

$$S_p = \frac{2 \times \left\lfloor \frac{L \times 2^{20}}{32} \right\rfloor \times T_{pd}}{2 \times \left\lfloor \frac{L \times 2^{20}}{32 \times n} \right\rfloor \times T_{pd} \times \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{L \times 2^{20}}{32} \right\rfloor \times T_{ric} \times \frac{1}{n \times \left\lfloor \frac{n}{p} \right\rfloor + \frac{T_{ric}}{2 \times T_{pd}}} \quad (18)$$

We can know that when the total core number $p$ is fixed and $n \in (kp, kp + p)$, the speed-up ratio $\eta_i$ presents an increasing trend, and the maximum will not exceed $p$ with $k$ a natural number.

5.5 Security Analysis

This scheme is homomorphic, and under the selection of the plaintext attacks, it is semantically secure. In paper [6],
the encryption scheme proposed by Paillier is designed based on the calculations among the $Z_{N^2}$ group, of which $N$ is a module of RSA. As $< N, g > \in P_n$ is satisfied, the following equation can be satisfied $P_{N, g} : Z \times Z_N \rightarrow Z_{N^2}$.

Paillier has proved that $P_{N, g}$ is a one-way trapdoor permutation. As $P_n$ is a bijection, it is known as $< N, g > \in P_n$. For an element $c \in Z_{N^2}$, there exists a sole $(m, r) \in Z_N \times Z_N$, which satisfies $c = g^m \times r^N \mod N^2$, $m$ is known as a class that $c$ relates to $N$ and $g$, and is recorded as $ClassN, g(c)$. In case the decomposition of $N$ is known, it is easy to calculate $ClassN, g(c)$. On the other hand, if the decomposition of $N$ is not known, up till now, no polynomial time algorithm for calculating $ClassN, g(c)$ has been found. We represent the unidirectional polarity of $P_{N, g}$ as $Pailier_{1}[N, g]$. This is where the hardness in the calculations of composite residuosity class, i.e. $Pailier_{1}[N, g] = Class[ClassN, g]$.

The safety of the MapReduce operation mainly derives from Job Submission and Task shuffle. In the Submission Job phase all of the work submitted or the operation of the track are using RPC (Remote Procedure Call Protocol) with the Kerberos implementation of the certification. In the stage of Task, each task submitted by a user is initiated by the user’s identity, so that a user’s task will not be sent the operating system of signal to Task Tracker or other user’s task. In the shuffle stage, when the end of a map task is running, it may provide the results to management Task Tracker. Afterwards, each reduce task will request the data processed by itself from the Task Tracker by HTTP, and then Hadoop should ensure that other users can’t obtain the intermediate results of map task. Based on the above definitions, the safety of data in the MapReduce process can be guaranteed.

6  Experiment Testing and Analysis

6.1  Experiment settings

The hardware platform used in this experiment includes a Master node and three slave nodes. Particularly, the Master node is responsible for the monitoring and scheduling of jobs, while the slave node is in charge of storage file distribution and calculation. The specific hardware configurations and software environment of each node are listed in the node configurations of experiment cluster in Table 1.

<table>
<thead>
<tr>
<th>Product name</th>
<th>The parameter and model</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon E3-1225v3</td>
</tr>
<tr>
<td>Cash</td>
<td>3.2 GHz/8M</td>
</tr>
<tr>
<td>Memory bank</td>
<td>16 GB(2×8 GB)1333 MHzDual Ranked RDIM</td>
</tr>
<tr>
<td>Hard disk</td>
<td>1 TB 3.5-inch 7200 RPM SATA II</td>
</tr>
<tr>
<td>Operating System</td>
<td>CentOS Linux Server 6.6</td>
</tr>
<tr>
<td>JAVA VM</td>
<td>JAVA 1.7.0</td>
</tr>
<tr>
<td>Hadoop</td>
<td>Hadoop-2.5.2</td>
</tr>
</tbody>
</table>

6.2  Experiment Results Analysis

In this experiment, encryption tests are conducted on the plaintexts with respective sizes of 256 MB, 512 MB, 768 MB, 1024 MB, 1280 MB, 1536 MB, 1792 MB and 2048 MB. The experiment involves two kinds of serial and the parallel test. In the serial environment, direct encryption is conducted on the plaintexts without the need of splinting. In contrast, the data needs to be splinted into lots of blocks in the parallel environment. The block size used in this experimental environment is set to 64MB, therefore, these files are divided into different data blocks, with the number of data blocks being 4, 8, 12, 16, 20, 24, 28 and 32 respectively.

In the experiments, four computational nodes are used, with the core number of the CPU being 4, and the total core number for the CPU is 16. Table 2 respectively records the sizes of the different files, the encryption time of the files and the total speed-up ratio under the serial and the parallel environment: FS, SET, PET, MMT and RT respectively mean File size, Serial encryption time, Parallel encryption time, Max Map time and Reduce time. Fig. 4 indicates the encryption time of files of different sizes under the serial and the parallel environments, Fig. 5 indicates the integral speed-up ratio of files of different sizes under the serial and the parallel environments.

![Table 1: Software and hardware configuration

<table>
<thead>
<tr>
<th>Product name</th>
<th>The parameter and model</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon E3-1225v3</td>
</tr>
<tr>
<td>Cash</td>
<td>3.2 GHz/8M</td>
</tr>
<tr>
<td>Memory bank</td>
<td>16 GB(2×8 GB)1333 MHzDual Ranked RDIM</td>
</tr>
<tr>
<td>Hard disk</td>
<td>1 TB 3.5-inch 7200 RPM SATA II</td>
</tr>
<tr>
<td>Operating System</td>
<td>CentOS Linux Server 6.6</td>
</tr>
<tr>
<td>JAVA VM</td>
<td>JAVA 1.7.0</td>
</tr>
<tr>
<td>Hadoop</td>
<td>Hadoop-2.5.2</td>
</tr>
</tbody>
</table>

![Table 2: The test results of different files

<table>
<thead>
<tr>
<th>FS(MB)</th>
<th>SET (s)</th>
<th>PET(s)</th>
<th>MMT(s)</th>
<th>RT(s)</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>1473</td>
<td>478</td>
<td>382</td>
<td>87</td>
<td>3.1</td>
</tr>
<tr>
<td>512</td>
<td>3013</td>
<td>587</td>
<td>401</td>
<td>178</td>
<td>5.1</td>
</tr>
<tr>
<td>768</td>
<td>4528</td>
<td>728</td>
<td>431</td>
<td>286</td>
<td>6.2</td>
</tr>
<tr>
<td>1024</td>
<td>6045</td>
<td>864</td>
<td>441</td>
<td>389</td>
<td>7.0</td>
</tr>
<tr>
<td>1280</td>
<td>7734</td>
<td>1231</td>
<td>464</td>
<td>543</td>
<td>6.2</td>
</tr>
<tr>
<td>1536</td>
<td>8916</td>
<td>1382</td>
<td>445</td>
<td>664</td>
<td>6.5</td>
</tr>
<tr>
<td>1792</td>
<td>10607</td>
<td>1524</td>
<td>461</td>
<td>787</td>
<td>6.9</td>
</tr>
<tr>
<td>2048</td>
<td>12666</td>
<td>1788</td>
<td>453</td>
<td>902</td>
<td>7.1</td>
</tr>
</tbody>
</table>

Unauthenticated
From the above experimental results (shown in Table 2, Fig. 4 and Fig. 5), the following conclusions can be drawn.

(1) Under the circumstance of fixed node numbers, the time needed by the serial encryption presents a proportional relation with the plaintext size.

(2) The time needed by the parallel encryption increases along with the increase of the plaintext.

(3) When $n < p$, the time consumption of the map functions is the main time consumption of the encryption process, and when $n$ is increasingly approaching $p$, the difference between the time consumed by the reduce functions and the map functions gradually decreases.

(4) When $n > p$, and along with the increase of $n$, the time consumption of reduce functions becomes the main time consumption during the encryption process. The main reason being that only one node is responsible for Reduce, and this also constrains the speed-up ratio.

(5) When $n < p$, the speed-up ratio $S_p$ increases faster, and when $n = p$, the maximum value of $S_p$ is reached. When $n > p$, at each $n \in (kp, kp + p)$, the speed-up ratio presents a trend of slow increase, and when $n = (k + 1) \times p$, the maximum value is reached.

### 7 Conclusion

In this paper, we propose a novel privacy-preserving PHE scheme based on MapReduce, which uses the additive homomorphism of the Paillier algorithm. The proposed scheme is suitable for cloud computing environments due to its parallelism. Experimental results indicate that the scheme enabled the speed-up ratio to reach 7.1 through the parallelism of clusters.

In the future, we will extend this work in the following aspects: (1) To conduct the optimization of the Reduce functions and then adopt two or more Reducers. (2) To study the parallelism of other HE algorithms, and then compare them with the efficiency of this algorithm so as to find out the optimal HE algorithm.

### Acknowledgement:

This research is supported by the National Natural Science Foundation of China under the Grants nos. 61572263, 61502251 and 61502243, Natural Science Foundation of Jiangsu Province under the Grants nos. BK20151511, BK20141429 and BK20161516, Colleges and Universities in Jiangsu Province plans to graduate research and innovation under Grant no. KYLX_0816, Natural Science Research Project of Jiangsu University under the Grants nos.(14KJB520031 and 15KJB520027),the Nature Science Project of Nanjing University of Posts and Telecommunications under the Grants nos.(NY215097 and NY214127).

### References


