A Privacy-preserving BGN-type Parallel Homomorphic Encryption Algorithm Based on LWE

Zhaoe Min[1], Geng Yang[1, 2], Jin Wang[3,4], Gwang-jun Kim[5]

[1] School of Computer Science, Nanjing University of Posts and Telecommunications, China
[2] Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, China
[3] School of Computer and Communication Engineering, Changsha University of Science & Technology, China
[4] School of Information Science and Engineering, Fujian University of Technology, China
[5] Dept. of Computer Engineering, Chonnam National University, Korea
minzhaoe@njupt.edu.cn, yangg@njupt.edu.cn, jinwang@csust.edu.cn, kgj@chonnam.ac.kr

Abstract

Although the rapid development of cloud computing brings many conveniences to people's lives, it also leads to the problems of user data privacy protection and the massive bandwidth resource consumption caused by frequent access to cloud servers. A feasible solution is to combine the Homomorphic Encryption (HE) technique to realize the efficient operation of ciphertext without decryption. The low encryption efficiency is a common issue faced by both Partially Homomorphic Encryption (PHE) algorithm and Fully Homomorphic Encryption (FHE) algorithm. To this end, based on the cryptosystem of Boneh, Goh and Nissim (BGN), we propose an efficient BGN-type parallel homomorphic encryption algorithm to address this issue, which security is based on the hardness of the Learning with Errors problem (LWE). Specifically, the proposed algorithm utilizes the characteristics of multi-nodes in cloud environment to conduct parallel encryption through block matrix multiplication, and simultaneously conduct the 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 speedup up to 5.3, which meets the practical requirements for the implementing efficient homomorphic encryption in cloud computing environment.

Keywords: Privacy protection, Homomorphic encryption, Learning with errors, Matrix multiplication, Parallel encryption

1 Introduction

Open network environment can provide strong computing and storage ability to the cloud computing users, which has broad applications in the industry. However, with the fast development of cloud computing technology, the faced security problems are also becoming more prominent [1-3]. On the one hand, the attacker may circumvent the authentication mechanism of cloud platform, and obtain the user's data by directly accessing file on lower layer or the original data, which may cause disclosure of privacy. On the other hand, the cloud service provider is an unreliable third party, and this characteristic also causes more sever privacy problem during data computation faced by cloud computing [4-5].

In order to effectively solve the data privacy protection problem, encrypted storage of cloud data is a very outstanding solution [6-7]. After encryption, the data is stored in the server provided by cloud service provider in the form of ciphertext, and in the meantime, the server is also required to return the data to user when the user requires. When the user needs to use the data frequently, it requires a lot of network bandwidth and user's time to conduct communication with the server and realize data encryption and decryption, which will significantly reduce the usability of cloud computing. In the meantime, after the encrypted data stored in cloud server has developed to a certain scale, effective retrieval of encrypted data has become a new problem that needs to be solved, while the traditional information retrieval technology can no long satisfy the requirement of mass data retrieval in the cloud storage environment.

The homomorphic encryption technology is an encryption method which can directly process the encrypted data [8], and it can effectively protect the security of user's data content, which has very broad development potential under the background of cloud storage application. By utilizing the homomorphic encryption algorithm, it can not only ensure that the encrypted data won't be statistically analyzed to decrypt corresponding plaintext, but also conduct homomorphic operation (such as addition and multiplication) to the ciphertext, while maintaining corresponding plaintext order of this ciphertext during

*Corresponding Author: Gwang-jun Kim; E-mail: kgj@chonnam.ac.kr
DOI: 10.3966/160792642019122007016

operation. During the retrieval process, the used index file and keyword are both in the form of ciphertext, and the cloud server cannot obtain any information of user data from the retrieval results. The index file is small, which will not increase the storage pressure of cloud server. In the meantime, the retrieval speed is fast, and it supports retrieval of multiple keywords, which will be convenient for the users and provides high security and strong practicability.

The homomorphic encryption algorithm can be divided into partially homomorphic encryption algorithm and fully homomorphic encryption algorithm. However, the low encryption efficiency is a common issue faced by both partially homomorphic encryption algorithm and fully homomorphic encryption algorithm. As for the problem that the mass data in cloud environment all require encryption, the traditional serial encryption cannot satisfy the requirement in terms of efficiency. The cloud computing cluster has many computational nodes which can be used to process mass data, and the cloud computing environment is easy to build. Many companies and scientific research institutions have deployed private cloud computing platforms, so we can use the parallel characteristic of private cloud cluster and the computing power of various nodes to encrypt the data that need to be stored in public cloud. In this paper, by utilizing the parallel characteristic of cloud computing cluster, we design and realize a parallel homomorphic encryption algorithm based on MapReduce, which can solve the low efficiency problem of homomorphic encryption algorithm.

2 Related Works

The homomorphic encryption method was initially proposed by Rivest et al. in the concept of “privacy homomorphism” in 1978 [9], which is an encryption scheme which can be directly used in the operation of ciphertext. In the same year, they also put forward that the RSA public key encryption algorithm has multiplication homomorphism [10], and the security of this scheme is based on integer factorization. Later, various homomorphic encryption schemes have been proposed, such as the ElGamal encryption scheme [11] with multiplication homomorphic characteristic and the Paillier encryption scheme [12] with addition homomorphic characteristic. Because they can only satisfy addition homomorphism or multiplication homomorphism, they are called partially homomorphic encryption algorithm.

In 2005, Boneh, Goh, and Nissim built a homomorphic encryption scheme [13] for BGN cryptosystem based on the bilinear pair mapping. By introducing the bilinear pair, it enables the scheme to support arbitrary times of homomorphic addition operations and one homomorphic multiplication operation of the ciphertext. The BGN scheme does not have ciphertext length extension during the encryption process, which also provides semantic security. It is the scheme closest to the concept of full homomorphism, but it is still not fully homomorphic encryption algorithm.

In 2009, Gentry proposed the fully homomorphic encryption scheme [14] based on ideal lattice for the first time. This scheme can conduct arbitrary times of homomorphic addition and multiplication operations of the ciphertext, which is an important milestone in the field of homomorphic encryption, and it has provided a new research direction for fully homomorphic encryption scheme. Later, the fully homomorphic encryption technology entered the fast development period.

According to the development period of fully homomorphic encryption scheme and the hardness assumption it is to construct fully homomorphic encryption scheme, the fully homomorphic encryption schemes can mainly be divided into four types: the first type is the fully homomorphic encryption scheme based on ideal lattice proposed by Gentry, and this scheme is the ideal scheme for various rings; the second type is the homomorphic encryption scheme based on Learning With Errors (LWE) and Learning With Errors over Ring (R-LWE) [15-16], which is constructed using nonlinearization technique; the third type is the fully homomorphic encryption scheme based on the Number Theory Research Unit (NTRU) [17]; the last type is the fully homomorphic encryption scheme based on Approximate Greatest Common Divisor (AGCD) [18].

The low encryption efficiency is a critical problem faced by homomorphic encryption algorithm. In order to improve the efficiency of homomorphic encryption algorithm, various schemes have been proposed: for example, Literatures [19-23] introduce the schemes which utilize different parallel methods (GPU, CPU or OPEN MP) to improve the efficiency of RSA algorithm; Literature [24] discusses the scheme which utilizes the MapReduce parallel technique to improve the efficiency of Hill encryption algorithm. According to the problem of low encryption efficiency of Paillier encryption algorithm, [25] proposes a homomorphic encryption algorithm capable of parallel encryption in the cloud environment.

For the fully homomorphic encryption scheme, [26] proposes an ideal lattice-based fully homomorphic encryption scheme which can be combined with the Single Instruction Multiple Data(SIMD) technique; by improving the fully homomorphic encryption scheme of DGHV(Dijk- Gentry-Halevi-Vaikuntanathan), [27] proposes an optimized scheme which can conduct batch processing of multiple plaintext bits. According to the problem that the ciphertext homomorphic addition and multiplication computations in GSW(Gentry-Sahai-Waters) scheme are only addition and multiplication based on matrix, Literature [28] proposes a method of compressed ciphertext and optimized bootstrap, and this scheme is the first fully homomorphic encryption scheme which can simultaneously encrypt the matrix and support homomorphic operation of matrix.

At the Eurocrypt meeting held in 2015, the FHEW scheme proposed in [29] has the problem of involving many matrix and vector operations. In order to address this problem, in 2017, Yang et al. designed and realized the CPU multi-core parallel algorithm of FHEW scheme in [30] by utilizing the characteristic that the CPU multi-core can adapt to many independent data operations. In the same experimental environment, the key generation efficiency is increased by 4.3 times, and the efficiency of one homomorphic NAND gate circuit operation is increased by 2.68 times.

In 2018, Shi et al. conducted analysis and research of large-number multiplication operation which takes the longest time in fully homomorphic encryption algorithm [31], and realized the FPGA design of FFT algorithm in the finite field of 16×24. By constructing tree-based large-number summation unit and parallel processing scheme, they realized the design and optimization of key module of finite field in the large-number algorithm and improved the efficiency of algorithm. In the same year, in order to solve the dilemma that FHE schemes can't be put into the practical applications, Tan et al. optimize FHE schemes by the parallel computing [32]. The main principle is to improve the performance of homomorphic operations by sacrificing hardware resources.

According to the characteristic of computationally expensive operations of the Fan-Vercauteren(FV) homomorphic encryption scheme on the FPGA,in 2019, Roy et.al design a custom co-processor and make the Arm processor a server for executing different homomorphic applications in the cloud [33]. In the hardware architecture, they used parallel computation cores to minimize cycle count, and applied circuit-level and block-level pipeline strategy to benefit parallel processing and reach a clock frequency of 200 MHz.

Among various homomorphic encryption algorithms discussed above, it is proved that the Hill encryption algorithm is not sufficiently secure; RSA only has multiplication homomorphism, while the Paillier algorithm only has addition homomorphism. Although there have been many researches on fully homomorphic encryption algorithm in recent years, however, due to the problems of encryption efficiency and ciphertext expansion, most are still at the experimental stage, and they have not be completely applied in real life. The GHV homomorphic encryption algorithm supports many additions and one multiplication. In the meantime, the main operation of this algorithm is matrix multiplication, and the parallel multiplication characteristic of matrix can be fully utilized to design parallelizable encryption algorithm to be used in privacy protection of cloud data.

In this paper, with the objective of privacy preservation, the cloud computing environment is combined to realize a BGN-type parallel homomorphic encryption algorithm based on LWE. According to the characteristic that the main operation of algorithm is matrix multiplication, we propose a parallelizable block matrix multiplication, and by utilizing the parallelism of MapReduce, parallel encryption can be realized through block encryption of plaintext data. The experimental results show that in addition to guaranteeing security, this scheme can also achieve a speed-up ratio of 5.3, which has solved the problem of low computational efficiency of partially homomorphic encryption algorithm.

3 Background

In 2005, Boneh, Goh and Nissim proposed the BGN cryptosystem with semantic security, and this scheme has the characteristics of multiple-addition homomorphism and one-multiplication homomorphism. At the Eurocrypt 2010, Gentry et al. proposed the GHV scheme for binary matrix encryption [34]. This scheme also supports polynomial times of addition operations and one multiplication homomorphic operation. Because its security is based on the hardness of the Learning With Errors problem (LWE), it is called the BGN-type encryption scheme based on LWE. Each sensor node has a unique identifier (ID) to differ from others.

3.1 Trapdoor Sampling

The GHV encryption algorithm is mainly based on the trapdoor sampling algorithm first constructd by Ajtai in 1999 [35] and later improved by Peikert in 2009 [36]. The specific algorithm is as follows:

Lemma1. (TrapSamp) Let the security parameter of algorithm be n, q=$poly(n)$, q>2, and m=8*n*logq. Then, there is a probability polynomial time algorithm with 1^n as the input, generate a matrix $A \in \mathbb{Z}_q^{m\times n}$ with almost random uniform distribution statistically speaking and a full-rank matrix et $S \subset \Lambda^{\perp}(A^T)$, and for $T \in S$, $\| T \|_\infty \leq O(n \cdot \log q)$. Matrixes A and T satisfy the following conditions:

- The rows of T form a basic of the lattice $\Lambda^{\perp}(A) \overset{def}{=} \{w \in \mathbb{Z}^m : w \cdot A = 0(\bmod q)\}$;
- Various elements of T are all "small", i.e., $\| T \|_\infty \leq O(n \cdot \log q)$;
- $\boldsymbol{T}$ is invertible $\bmod 2$.

3.2 BGN-type Cryptosystem from LWE

Assume the security parameter of encryption system is n, q=$poly(n)$, and q is an odd prime;

$p > n^{3\varepsilon+1} \cdot \log^5 n$ is a prime number distinctive from q, and $p < q < p^3$. The algorithm mainly consists of three steps: generation of key, encryption and decryption. The specific steps are as follows:

Generation of key. Run the trapdoor sampling algorithm TrapSamp of in Lemma 1 to obtain random matrix $A \in \mathbb{Z}_q^{m\times n}$ and trapdoor matrix $T \in \mathbb{Z}_q^{m\times m}$, which satisfy $T \cdot A = 0(\mathrm{mod}\, q)$, and $m > n$, i.e., $(A,T) \leftarrow TrapSamp(1^n, q, m)$. Then, matrix A is the public key, and matrix T is the private key.

Encryption. For plaintext M, encode it to a binary $m \times m$ matrix first, i.e., $B \in \mathbb{Z}_2^{m\times m}$. Choose matrix S with random uniform distribution from $\mathbb{Z}_q^{n\times m}$, i.e., $S \leftarrow \mathbb{Z}_q^{n\times m}$. Select an "error matrix" X from Gaussian distribution $\overline{\Psi}_\beta(q)^{m\times m}$, i.e., $X \leftarrow \overline{\Psi}_\beta(q)^{m\times m}$. Use Formula (1) to obtain ciphertext C through calculation, and the ciphertext matrix C satisfies $C \in \mathbb{Z}_q^{m\times m}$.

$$C = AS + 2X + B(\mathrm{mod}\, q) \tag{1}$$

Where, $2X$ represents each component in matrix X multiplied with 2.

Decryption. set $E = TCT^t \bmod q$, and utilize Formula (2) to calculate plaintext B.

$$B = T^{-1}E(T^t)^{-1} \bmod 2 \tag{2}$$

Proof. Because $T \bullet A = 0(\mathrm{mod}\, q)$ and therefore $TCT^t = T(2X+B)T^t(\mathrm{mod}\, q)$. If q is a sufficiently big prime number so that $T(2X+B)T^t$ is significantly smaller than q, then, we also have the equality over the integers $E = TCT^t(\mathrm{mod}\, q) = T(2X+B)T^t$, and hence $T^{-1}E(T^t)^{-1} = B(\mathrm{mod}\, 2)$. This means that we have correct decryption as long as we set the parameter β small enough so that with high probability all the entires of $T(2X+B)T^t$ are smaller than $q/2$.

3.3 Proof of Homomorphism

Addition homomorphism: assume C_1 and C_2 are the ciphertexts generated after encryption of plaintexts B_1 and B_2 respectively. Specifically, if we have $C_1 = AS_1 + 2X_1 + B_1$ and $C_2 = AS_2 + 2X_2 + B_2$ then

$$\begin{aligned} C^* &= (C_1 + C_2) \bmod q \\ &= (AS_1 + 2X_1 + B_1) \bmod q + (AS_2 + 2X_2 + B_2) \bmod q \\ &= A(S_1 + S_2) + 2(X_1 + X_2) + (B_1 + B_2) \bmod q \end{aligned} \tag{3}$$

After decryption, we can obtain:

$$\begin{aligned} B^* &= T^{-1}(TC^*T^t(\mathrm{mod}\, q))(T^t)^{-1}(\mathrm{mod}\, 2) \\ &= (2(X_1 + X_2) + (B_1 + B_2))(\mathrm{mod}\, 2) \\ &= (B_1 + B_2) \end{aligned} \tag{4}$$

As long as we choose suitable parameter, we can ensure various elements of $T(2(X_1+X_2)+(B_1+B_2))T^t$ are all smaller than $q/2$. Then, we can obtain plaintext through correct decryption. Similarly, we can prove the characteristic of polynomial times of addition homomorphism.

Multiplication homomorphism: assume C_1 and C_2 are the ciphertexts generated after encryption of plaintexts B_1 and B_2 using Formula (1) respectively. Specifically, if we have $C_1 = AS_1 + 2X_1 + B_1$ and $C_2 = AS_2 + 2X_2 + B_2$ then

$$\begin{aligned} C^* &= (C_1 \cdot C_2^t) \bmod q \\ &= (AS_1 + 2X_1 + B_1) \cdot C_2^t (\mathrm{mod}\, q) \\ &= AS_1 \cdot C_2^t + (2X_1 + B_1) \cdot (AS_2 + 2X_2 + B_2)^t (\mathrm{mod}\, q) \\ &= AS_1 \cdot C_2^t + 2X_1 \cdot A^t S_2^t + 2X_1 \cdot (2X_2 + B_2)^t \\ &\quad + B_1 \cdot A^t S_2^t + B_1 \cdot 2X_2^t + B_1 \cdot B_2{}^t (\mathrm{mod}\, q) \\ &= A \cdot \underbrace{(S_1 C_2^t)}_{S} \cdot + 2\underbrace{(X_1(2X_2 + B_2)^t + B_1 X_2^t)}_{X} \\ &\quad + \underbrace{(2X_1 + B_1) \cdot S_2^t}_{S'} \cdot A^t + \underbrace{(B_1 \cdot B_2^t)}_{B}(\mathrm{mod}\, q) \end{aligned} \tag{5}$$

Hence the product ciphertext has the form $AS + 2X + B + S'A^t$. The obtained ciphertext result has the structural form, and we can obtain the following formula through decryption:

$$\begin{aligned} B^* &= T^{-1}(TC^*T^t(\mathrm{mod}\, q))(T^t)^{-1}(\mathrm{mod}\, 2) \\ &= T^{-1}(T(AS + 2X + B + S' \cdot A^t)T^t(\mathrm{mod}\, q))(T^t)^{-1}(\mathrm{mod}\, 2) \\ &= (2X + B)(\mathrm{mod}\, 2) \\ &= B_1 \cdot B_2^t \end{aligned} \tag{6}$$

Similarly, as long as we choose suitable parameter, we can ensure various elements of $T(2X+B)T^t$ are all smaller than $q/2$. Then, we can correctly obtain the plaintext, and further prove that the algorithm has multiplication homomorphism.

3.4 Block Matrix Multiplication

Set matrix $A \in R^{m\times l}$, matrix $B \in R^{l\times n}$ and matrix $C = A \times B$, then matrixes A,B can be divided into the blocks of:

$$A = \begin{pmatrix} A_{11} & \cdots & A_{1t} \\ \vdots & & \vdots \\ A_{s1} & \cdots & A_{st} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & \cdots & B_{1k} \\ \vdots & & \vdots \\ B_{t1} & \cdots & B_{tk} \end{pmatrix} \tag{7}$$

where, the column number of A_{i1}, A_{i2},..., A_{it} equals to the row number of B_{1j}, B_{2j},…, B_{tj} respectively. Then, the block matrix can be represented as:

$$C = A \times B = \begin{pmatrix} C_{11} & \cdots & C_{1k} \\ \vdots & & \vdots \\ C_{s1} & \cdots & C_{sk} \end{pmatrix} \quad \textbf{(8)}$$

where, $C_{ij}=A_{i1}\ B_{1j}+A_{i2}\ B_{2j}+\ldots+\ A_{it}\ B_{tj}$ (i=1,…s ; j=1,…,k).

4 Parallel Homomorphic Encryption Scheme

The GHV encryption scheme proposed in this paper is based on the MapReduce architecture, and the GHV encryption algorithm mainly involves the matrix addition operation and matrix multiplication operation, of which, the matrix multiplication operation is the main operation of this algorithm. The execution process of GHV encryption algorithm mainly consists of three steps: generate the key sequence, use the public key matrix to encrypt the plaintext data file, and use the private key to decrypt the ciphertext file. The matrix multiplication parallel algorithm proposed in this paper is realized based on block matrix multiplication, which mainly describes the realization of block matrix multiplication in MapReduce.

4.1 Algorithm Procedure

A complete MapReduce programming model includes 3 operations: Split, Map and Reduce. The Split function is used to divide the input data into data blocks of fixed size according to the user’s requirement, and then, the master node allocates them to different child nodes according to corresponding scheduling mechanism. For each data block after fragmentation, the Map function conducts corresponding operation in accordance with the encryption algorithm defined by the user, and each Mapper completes part of the final result. Reduce is responsible of integrating all results of Mapper part. Each encryption computation of parallel GHV encryption scheme is independent, so it can be allocated to multiple Mappers for simultaneous encryption. It can be defined as a trinomial time algorithm $\prod = (Split, Map, \mathrm{Re}duce)$, and the specific process is as shown in Figure 1.

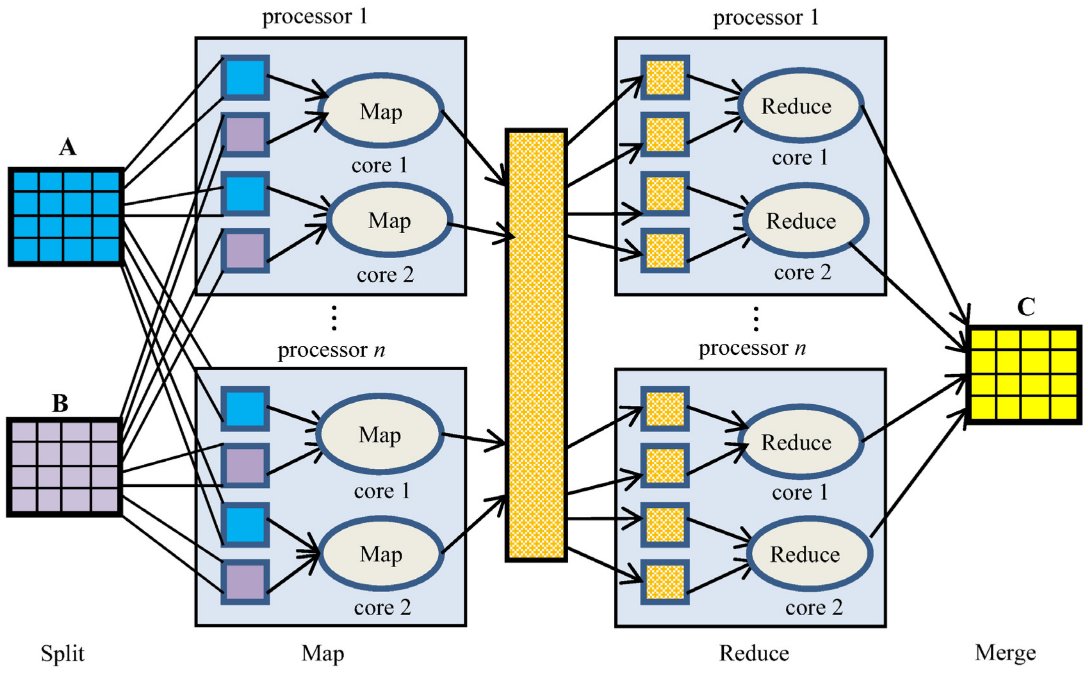


Figure 1. Algorithm procedure

4.2 Data Partitioning

Utilize the key generation algorithm in Section 3.2 to generate the public key matrix $A \in \mathbb{Z}_q^{m\times n}$ and private key matrix $T \in \mathbb{Z}_q^{m\times m}$, and select the matrix $S \leftarrow \in \mathbb{Z}_q^{n\times m}$ with random uniform distribution. Assume the public key matrix A consists of s rows and t columns, and the random distribution matrix S consists of t rows and k columns. The specific partitioning strategy is: the public key matrix A can be divided into $s\times t$ blocks, and each block is denoted as $A_{\mathrm{ip}}^{+}\left(i \le s, \mathrm{p} \le t\right)$. The boundary of each block can be represented by the four variables of $\left(ir, er, ic, ec\right)$, which refer to the start row number, end row number, start column number and end column number, respectively. The public key matrix A is:

$$A_{\text{ip}}^{+}\left(1\le i\le s,1\le p\le t\right)$$
$$ir=\left\lceil\frac{m}{s}\right\rceil\times(i-1)+1 \quad er=\min\left(\left\lceil\frac{m}{s}\right\rceil\times i,m\right)$$
$$ic=\left\lceil\frac{n}{t}\right\rceil\times(p-1)+1 \quad ec=\min\left(\left\lceil\frac{n}{t}\right\rceil\times p,n\right) \tag{9}$$

The distribution matrix S is divided into $t\times k$ blocks, each block is denoted as $S_{\text{pj}}^{+}\left(1\le \text{p}\le t,1\le j\le k\right)$, and the block is:

$$S_{\text{pj}}^{+}\left(1\le p\le t,1\le j\le k\right)$$
$$ir=\left\lceil\frac{n}{t}\right\rceil\times(p-1)+1 \quad er=\min\left(\left\lceil\frac{n}{t}\right\rceil\times p,n\right)$$
$$ic=\left\lceil\frac{m}{k}\right\rceil\times(j-1)+1 \quad ec=\min\left(\left\lceil\frac{m}{k}\right\rceil\times j,m\right) \tag{10}$$

Then, the product matrix M consists of $s\times k$ blocks in total, each block is denoted as $M_{\text{ij}}^{+}\left(1\le \text{i}\le s,1\le j\le k\right)$, and the block is:

$$M_{\text{ij}}^{+}\left(1\le i\le s,1\le j\le k\right)$$
$$ir=\left\lceil\frac{m}{s}\right\rceil\times(i-1)+1 \quad er=\min\left(\left\lceil\frac{m}{s}\right\rceil\times i,m\right)$$
$$ic=\left\lceil\frac{m}{k}\right\rceil\times(j-1)+1 \quad ec=\min\left(\left\lceil\frac{m}{k}\right\rceil\times j,m\right) \tag{11}$$

After partitioning, the block matrix multiplication can be used to conduct parallel computation of GHV homomorphic encryption algorith.

4.3 Parallel Encryption

After partitioning of public key matrix A and random uniform distribution matrix S, encrypt the plaintext according to the GHV encryption formula $C=AS+2X+B(\bmod q)$.

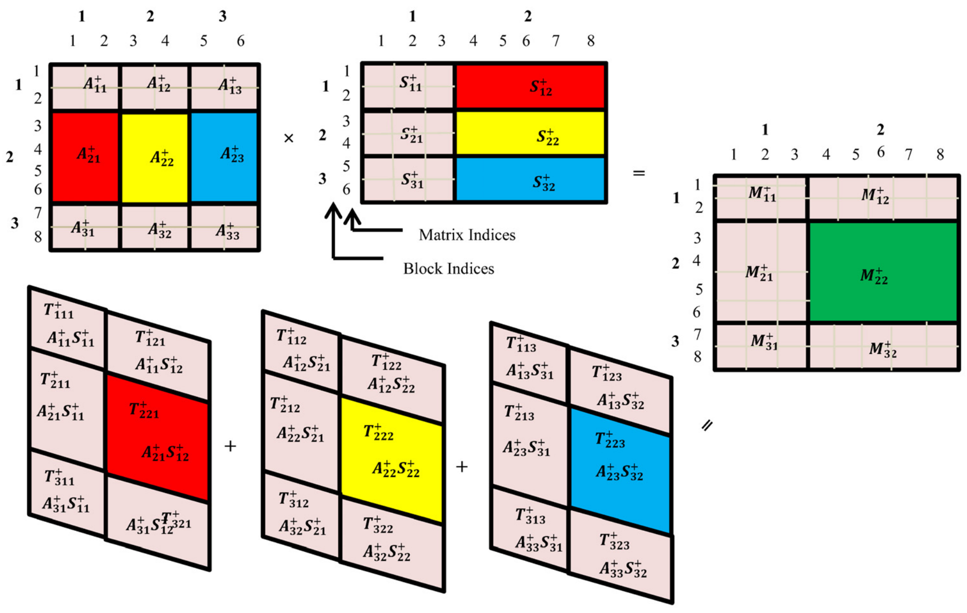
The main operation of GHV encryption formula is the matrix multiplication operation. The block matrix algorithm can be employed for parallel encryption, and the block matrix multiplication strategy is: conduct matrix multiplication operation to corresponding blocks; finally, conduct summation operation of blocks to obtain the product block M_{ij}^{+}; combine the product blocks to obtain an $m\times m$ product matrix $M=A\times S$. Assume the public key matrix A consists of s rows and t columns, and the random distribution matrix S consists of t rows and k columns, then, $s\times t\times k$ blocks are generated in total, which is denoted as $T_{\text{i,j,p}}^{+}\left(1\le i\le s,1\le \text{p}\le t,1\le j\le k\right)$, then:

$$T_{\text{i,j,p}}^{+}=\text{A}_{i,p}^{+}\times S_{p,j}^{+} \tag{12}$$

After calculation of intermediate result, conduct matrix addition operation of blocks with the same p value in $T_{\text{i,j,p}}^{+}$ to obtain $s\times k$ product blocks M_{ij}^{+}, i.e. (as shown in Figure 2),

$$M_{ij}^{+}=\sum_{p=1}^{k}T_{i,j,p}^{+} \tag{13}$$

Assume the plaintext matrix $B\in\mathbb{Z}_2^{m\times m}$ consists of s rows and k columns, then the plaintext matrix B can be divided into $s\times k$ blocks, and each block denoted as $B_{\text{ij}}^{+}\left(i\le s,\text{j}\le k\right)$. The boundary of each block can be represented by the four variables of $\left(ir,er,ic,ec\right)$,



Matrix Indices
Block Indices
A_{11}^{+} A_{12}^{+} A_{13}^{+} A_{21}^{+} A_{22}^{+} A_{23}^{+} A_{31}^{+} A_{32}^{+} A_{33}^{+}
S_{11}^{+} S_{12}^{+} S_{21}^{+} S_{22}^{+} S_{31}^{+} S_{32}^{+}
M_{11}^{+} M_{12}^{+} M_{21}^{+} M_{22}^{+} M_{31}^{+} M_{32}^{+}
T_{111}^{+} $A_{11}^{+}S_{11}^{+}$ T_{121}^{+} $A_{11}^{+}S_{12}^{+}$ T_{211}^{+} $A_{21}^{+}S_{11}^{+}$ T_{221}^{+} $A_{21}^{+}S_{12}^{+}$ T_{311}^{+} $A_{31}^{+}S_{11}^{+}$ $A_{31}^{+}S_{12}^{+}T_{321}^{+}$
T_{112}^{+} $A_{12}^{+}S_{21}^{+}$ T_{122}^{+} $A_{12}^{+}S_{22}^{+}$ T_{212}^{+} $A_{22}^{+}S_{21}^{+}$ T_{222}^{+} $A_{22}^{+}S_{22}^{+}$ T_{312}^{+} $A_{32}^{+}S_{21}^{+}$ T_{322}^{+} $A_{32}^{+}S_{22}^{+}$
T_{113}^{+} $A_{13}^{+}S_{31}^{+}$ T_{123}^{+} $A_{13}^{+}S_{32}^{+}$ T_{213}^{+} $A_{23}^{+}S_{31}^{+}$ T_{223}^{+} $A_{23}^{+}S_{32}^{+}$ T_{313}^{+} $A_{33}^{+}S_{31}^{+}$ T_{323}^{+} $A_{33}^{+}S_{32}^{+}$

Figure 2. Matrix block multiplication

which refer to the start row number, end row number, start column number and end column number, respectively. The plaintext matrix *B* is:

$$B_{\text{ij}}^{+}\left(1\le i\le s,1\le j\le k\right)$$
$$ir=\left\lceil \frac{m}{s}\right\rceil\times\left(i-1\right)+1 \quad er=\min\left(\left\lceil \frac{m}{s}\right\rceil\times i,m\right)$$
$$ic=\left\lceil \frac{m}{k}\right\rceil\times\left(j-1\right)+1 \quad ec=\min\left(\left\lceil \frac{m}{k}\right\rceil\times j,m\right) \qquad (14)$$

Use the partitioning method for plaintext matrix *M* to partition the error matrix *X*, and each block is denoted as $X_{\text{ij}}^{+}\left(i\le s,\text{j}\le k\right)$. Conduct addition operation of product matrix block M_{ij}^{+}, twice the error matrix block X_{ij}^{+} and plaintext matrix block B_{ij}^{+}. In the matrix sum, through modulo operation of various elements to *q*, we can obtain the ciphertext matrix block C_{ij}^{+}, i.e.,

$$C_{ij}^{+}=M_{ij}^{+}+2X_{ij}^{+}+B_{ij}^{+}\bmod q \qquad (15)$$

4.4 Performance Analysis

The encryption formula of GHV encryption algorithm is $C=AS+2X+B(\bmod q)$, in which, *A* and *S* are the *m*×*n* and *n*×*m* matrixes respectively. According to the encryption formula, the time complexity of GHV algorithm is O(m^2×*n*). Assume the total encryption time for sequential execution of GHV algorithm is T_{seq}, then T_{seq} consists of two parts: one part is the key matrix generation time T_{key}; the other part is the time T_{Enc} required by sequential execution encryption algorithm. Then, T_{seq} can be expressed by the following formula: $T_{seq}=T_{key}+T_{Enc}=T_{key}+O(m^2\times n)$.

Set the total encryption time T_{par} for parallel execution of GHV algorithm consisting of two parts: one part is the key matrix generation time T_{key}; the other part is the parallel execution time of encryption algorithm, i.e., the matrix multiplication and matrix addition time T_{matrix_par}, then, T_{seq} can be expressed by the following formula: $T_{par}=T_{key}+T_{matrix\_par}$.

We can see that the increase of encryption speed mainly depends on the acceleration process of parallel matrix multiplication. The computation time of MapReduce-based matrix multiplication equals to the computation time of matrix multiplication for each block plus the time consumed by summing up various blocks. In the Reduce phase, the results of all Map modules need to be read, i.e.:

$$T_{matrix\_par}=T_{comp}+T_{wait} \qquad (16)$$

Assuming the total number of CPU cores is p, then the serial time complexity can be reduced as through partitioning:

$$T_{comp}=\frac{m^2\times n}{p} \qquad (17)$$

According to the algorithm, it can be seen that because $m\times\left(k\times m+s\times n\right)$ key/value pairs are generated during the Map process, and *m***n* Reduce modules are responsible of integrating the output result of Map. Assume the number of Map nodes is $p=s\times t\times k$, and it takes time t_w to write the results generated by Map process into the disk. Then, it can be known that the time of waiting for all results generated by Map is:

$$\begin{aligned}T_{waits}&=m\times\left(k\times m+s\times n\right)t_w\\&=\left(km^2+mnk\right)t_w\end{aligned} \qquad (18)$$

Set $s=t=k=\sqrt[3]{p}$, $m=n$, and we can obtain the parallel encryption time of GHV as:

$$\begin{aligned}T_{par}&=T_{key}+T_{matrix\_par}\\&=T_{key}+\frac{m^2\times n}{p}+\left(km^2+mnk\right)t_w\\&=T_{key}+\frac{m3}{p}+2\sqrt[3]{p}m^2t_w\end{aligned} \qquad (19)$$

When the m is big, T_{key} can be ignored, so the speed-up ratio η is:

$$\eta=\frac{T_{seq}}{T_{par}}=\frac{m^2\times n}{\left(\dfrac{m^3}{p}+2\sqrt[3]{p}m^2t_w\right)}=\frac{p}{1+\dfrac{2p^{3/4}t_w}{m}} \qquad (20)$$

As *m* approaches infinity, we can obtain $T_{seq}\approx pT_{par}$.

4.5 Security Analysis

The security of this scheme can defend Indistinguishable Chosen Plaintext Attack (IND-CPA), which is built based on the hardness of the learning with errors problem. The specific proof is as follows.

Theorem 1: If there is a distinguishable algorithm which can solve this scheme of parameters *n*, *m*, *q*, *p* and *β* with non-negligible advantage ε, then such distinguishable algorithm can be utilized to build a distinguisher. Within the same time, if the DLWE difficulty problem of parameters *n*, *m*, *q*, *p* and *β* can be solved with a probability no less than *ε* /2*m*, then this scheme will be regarded as CPA-secure.

Proof: Let *A* be a CPA-adversary that distinguishes between encryptions of messages of its choice with advantage *ε*. First, build a distinguisher *D* with advantage of no less than *ε*/2 probability, which can distinguish the following 2 distributions:

$\{(A, AS+X): A \leftarrow Z_q^{m\times n}, S \leftarrow Z_q^{m\times n}, X \leftarrow \overline{\Psi}_\beta(q)^{m\times m}\}$ and $\{Unif(Z_q^{m\times n} \times Z_q^{m\times m})\}$ $(A \in Z_q^{m\times n}, C \in Z_q^{m\times n})$.

The distinguisher *D* takes as input a pair of matrices $(A \in Z_q^{m\times n}, C \in Z_q^{m\times n})$, and operate adversary *A* with A as the public key. After obtaining information B_0 and B_1 provided by the adversary, the distinguisher *D* randomly chooses $i \in_R \{0,1\}$, and returns the challenge ciphertext $2C + B_i (\bmod q)$. If the adversary *A* is able to guess right *i*, it will output 1; otherwise, it will output 0.

On the one hand, if *C* is a matrix with uniform random distribution, then the challenge ciphertext must also have uniform distribution, which is irrelevant to the selection of *i*, Hence in this case *D* outputs 1 with probability at most 1/2. On the other hand, if the ciphertext matrix is $C = AS + X(\bmod q)$, the challenge ciphertext is $2C + B = AS' + 2X + B \bmod q$, in which, $S' = 2S \bmod q$ has uniform random distribution (since *q* and 2 are relatively prime). Therefore, the probability of correctly guessing *i* through opponent *A* is $(1+\varepsilon)/2$, which indicates that the probability of distinguisher *D* outputting 1 is also $(1+\varepsilon)/2$. To sum it up, the probability of distinguisher *D* correctly distinguishing 2 distributions is at least $1/2 \times \varepsilon/2 + 1/2 \times (1+\varepsilon)/2 = 1/2 + \varepsilon/2$, and in other words, the advantages is no less than $\varepsilon/2$. Therefore, the advantage of using such distinguisher *D* to solve the LWE hardness problem of parameters *n*, *m*, *q*, *p* and β is $\varepsilon/2m$.

5 Experimental Results and Analysis

5.1 Experimental Environment

The hardware platform used in experiment mainly consists of the Master and Slave nodes, of which, the Master node is mainly responsible of monitoring and scheduling, while the Slave node is mainly responsible of the distributed storage data file and computation tasks.

The Master node nodes include the NameNode and JobTracker nodes. The Slave nodes include the TaskNode and DataNode nodes. The cluster consists of 4 Slave nodes in total. See "Table 1 Software and hardware configuration" for the specific hardware configuration and software environment for each node.

Table 1. Software and hardware configuration

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

5.2 Analysis of Experimental Results

In this paper, data test is conducted from two different perspectives: first, the plaintext data of different sizes are selected to compare their encryption speed and speed-up ratio in serial and parallel environment; second, the plaintext data of fixed size are selected to compare their encryption speed and speed-up ratio under different numbers of processor cores.

In the first experiment, the selected plaintext data have the sizes of 256MB, 512MB, 768MB, 1024 MB, 1280 MB, 1536 MB, 1792 MB and 2048 MB respectively, the default data block size is 64MB in the parallel environment, and encryption test is conducted in the serial and parallel environment respectively.

In the second environment, the plaintext data with the size of 2G are selected, and their encryption speeds are tested when there are 1, 4, 8, 12, 16, 24 and 32 processors respectively.

4 computational nodes are used in the experiment, each node has 4 CPU cores, and there are 16 CPU cores in total. In the parallel experiment, it is found that with the increase of plaintext data volume, the time occupied by Reduce function also increases. In order to improve the efficiency, the number of parallel Reduce is set at 15.

Table 2 summarizes the encryption speed and overall speed-up ratio of files with different sizes in serial and parallel environment, respectively. Figure 3 presents the curve of how the file encryption time changes with the increase of plaintext in the parallel environment. Figure 4 shows the curve of how the Map speed-up ratio and overall speed-up ratio change with the increase of plaintext in the parallel environment.

Table 3 records the overall encryption time of file, the execution time of Map process, the overall speed-up ratio and the speed-up ratio of Map process during encryption of file with plaintext size of 2GB under different numbers of processor cores.

Figure 5 shows the curve of how the file encryption speed changes with the increase of core number during encryption of file with plaintext size of 2GB. With the increase of processor number, the encryption time presents exponential decline. With the increase of core number, during the early period, the encryption speed quickly declines; during the later period, the encryption time is becoming relatively stable. This mainly depends on the time of Reduce process. Figure 6 reflects the comparison of overall speed-up ratio and Map speed-up ratio for files of 2GB.

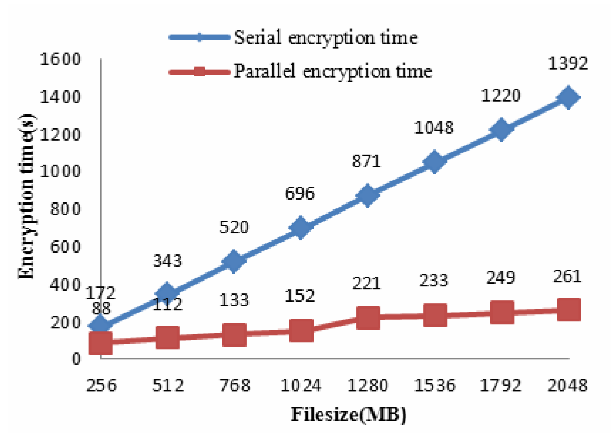


Figure 3. The encryption time of different size file

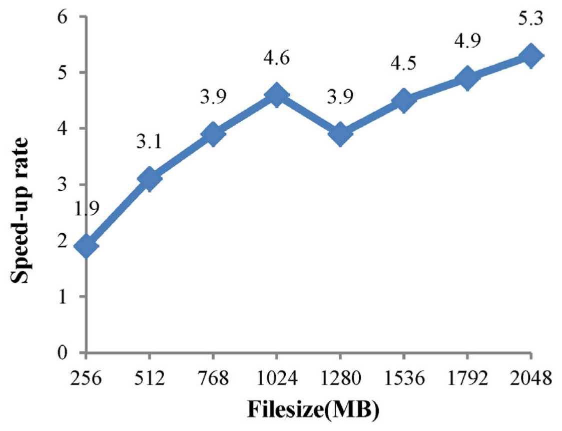


Figure 4. The speed-up rate of different size file

Table 2. The test results of diffent size file

Filesize (MB)	Serial time (s)	Parallel time (s)	Max Map time (s)	Reduce time (s)	Speed-up
256	172	88	44	38	1.9
512	343	112	45	61	3.1
768	520	133	44	83	3.9
1024	696	152	44	103	4.6
1280	871	221	45	122	3.9
1536	1048	233	46	138	4.5
1792	1220	249	45	155	4.9
2048	1392	261	44	169	5.3

Table 3. The test results of 2GB files on different cores

No P	General time (s)	Max Map time (s)	Reduce time (s)	Map SP	General SP
1	1418	1243	171	1.0	1.0
4	501	326	169	3.8	2.8
8	343	167	170	7.4	4.1
12	302	125	170	9.9	4.7
16	267	92	171	13.5	5.3
24	312	71	170	17.6	4.5
32	266	48	169	25.8	5.3

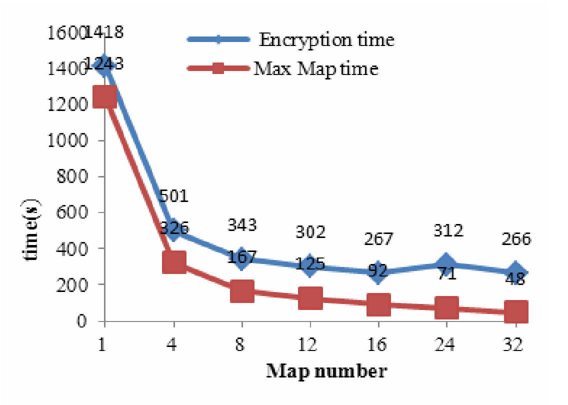


Figure 5. The encryption time of 2GB on different cores

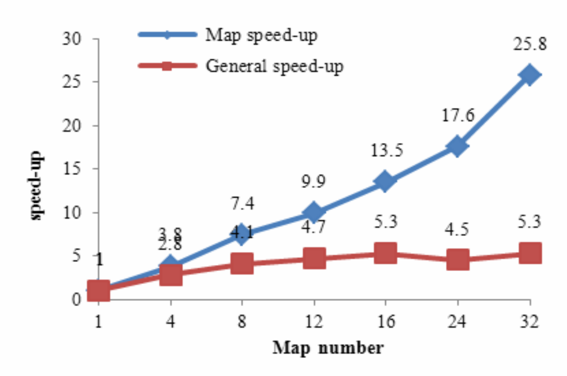


Figure 6. The speed-up rate of 2GB on different cores

According to Table 2, Figure 3 and Figure 4, it can be seen that under fixed node number and fixed size of file piece: (1) The time required by serial encryption is basically in direct proportion to the size of plaintext, while the time required by parallel encryption increases slowly with the increase of plaintext; (2) With the increase of plaintext data volume, the time consumption of Reduce function gradually increases.

While the time consumption of Map function almost doesn't change, the proportion of the time required by the Reduce function gradually increases in the entire parallel encryption process; (3) When the number of plaintext pieces is smaller than the node number, the increasing speed of speed-up ratio S_P is fast, and when the number of plaintext pieces equals to the node number, the speed-up ratio reaches the highest value. When the number of plaintext pieces is bigger than the node number, the speed-up ratio presents the trend of slow increase.

By observing Table 3, Figure 5 and Figure 6, we can see that with the increase of the numbers of available cores and file partitions in cluster: (1) The time consumption of Map gradually decreases. This is because with the increase of Map number, each Map data block becomes smaller, and the time consumption of Map mainly concentrates on the encryption operation, so the time consumption of will gradually decline. The time consumption of Reduce basically doesn't change, because the number of Reduce is a fixed number of 15. (2) The Map speed-up ratio increases with the increase of Map number, but it is smaller than the Map number, which is consistent with the theoretical analysis in previous section. (3) With the continuous increase of the speed of Map process, the ratios of the time required by the Map process and the time required by the Reduce process keep declining, and the weight of the influence of Map process on the final acceleration speed is relatively small, which results in certain gap between the final overall speed-up ratio and the speed-up ratio of Map process. (4) For plaintext data file of fixed size, with the increase of Map number, the file encryption time presents decline trend in general. When the Map number equals to the node number, it takes the least time. With the increase of the number of available cores, the file encryption time presents significant decline, and the clustering performance can be effectively carried out.

6 Conclusion

To address the low encryption efficiency issue of homomorphic encryption algorithm, this paper proposes a BGN-type parallel homomorphic encryption algorithm which utilizes parallel matrix to improve the encryption speed. This encryption algorithm realizes block matrix

multiplication and addition operations by partitioning matrix into multiple blocks during encryption process, which can improve the efficiency of encryption algorithm. Furthermore, we also design and implement an effective parallel algorithm executed on the MapReduce platform. During the encryption process, the file is divided into different number of data blocks, and the parallelism of algorithm can be controlled by specifying the numbers of available cores and partitions. Meanwhile, parallel execution of multiple Reduce functions can reduce the high real-time cost of Reduce operation. The experimental results show that, compared to the traditional linear encryption algorithm, the proposed algorithm can achieve higher speed-up ratio when processing big data files in the MapReduce cluster.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under the Grant 61972209, in part by the Colleges and Universities in Jiangsu Province plans to graduate research and innovation under Grant KYLX_0816, in part by Institute of Population and Health Nanjing collage for Population Program Management general fund (2012C06).

References

[1] S. Mahalle, R. Jaiswal, Cloud Computing Security: A Survey, *International Journal of Computer Applications*, Vol. 115, No. 6, pp. 21-25, April, 2015.

[2] M. Ali, S. U. Khan, A. V. Vasilakos, Security in Cloud Computing: Opportunities and Challenges, *Information Sciences*, Vol. 305, pp. 357-383, June, 2015.

[3] A. P. Bodkhe, C. A. Dhote, Cloud computing Security: An Issue of Concern, *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 5, No. 4, pp. 1337- 1342, April, 2015.

[4] Z. Y. Li, X. L. Gui, Y. J. Gu, X. S. Li, H. J. Dai, X. J. Zhang, Survey on Homomorphic Encryption Algorithm and Its Application in the Privacy-preserving for Cloud Computing, *Journal of Software*, Vol. 29, No. 7, pp. 1830-1851, July, 2018.

[5] C. X. Zhou, Z. M. Cui, G. Y. Gao, On the Security of an Improved Identity-based Proxy Signature Scheme without Random Oracles, *Journal of Interne Technology*, Vol. 19, No. 7, pp. 2057-2068, December, 2018.

[6] Y. Xie, L. B. Wu, J. Shen, L. Li, Efficient Two-party Certificateless Authenticated Key Agreement Protocol under GDH Assumption, *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 30, No. 1, pp. 11-25, September, 2019.

[7] Y. Huang, W. Li, J. Lei, Concatenated Physical Layer Encryption Scheme Based on Rateless Codes, *IET Communications*, Vol. 12, No. 12, pp. 1491-1497, July, 2018.

[8] D. Micciancio, A First Glimpse of Cryptography's Holy Grail, *Communications of the ACM*, Vol. 53, No. 3, pp. 96-96, March, 2010.

[9] R. L. Rivest, L. Adleman, M. L. Dertouzos, On Data Banks and Privacy Homomorphisms, in: R. A. DeMillo, D. P. Dobkin, A. K. Jones, R. L. Lipton (Eds.), *Foundations of Secure Computation*, Academic Press, 1978, pp. 169-179.

[10] R. L. Rivest, A. Shamir, L. Adleman, A Method for obtaining Digital Signatures and Public-key Cryptosystems, *Communications of the ACM*, Vol. 21, No. 6, pp. 120-126, February, l978.

[11] T. Elgamal, A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *IEEE Transactions on Information Theory*, Vol. 31, No. 4, pp. 469-472, July, 1985.

[12] P. Paillier, Public-key Cryptosystems Based on Composite Degree Residuosity Classes, *International Conference on the Theory and Applications of Cryptographic Techniques*, Prague, Czech Republic, 1999, pp. 223-238.

[13] D. Boneh, E. J. Goh, K. Nissim, Evaluating 2-DNF Formulas on Ciphertexts, *Proceedings of the Second international Conference on Theory of Cryptography*, Cambridge, MA, USA, 2005, pp. 325-341.

[14] C. Gentry, Fully homomorphic encryption using ideal lattices, *Proc. of the Annual ACM Symposium on Theory of Computing*, Bethesda, MD, USA, 2009, pp. 169-178.

[15] Z. Brakerski, V. Vaikuntanathan, Efficient Fully Homomorphic Encryption from (standard) LWE, *2011 IEEE 52nd Annual Symposium* on *Foundations of Computer Science*, Palm Springs, CA, USA, 2011, pp. 97-106.

[16] Z. Brakerski, V. Vaikuntanathan, Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages, *Proceedings of the 31st Annual Conference on Advances in Cryptology*, Santa Barbara, CA, USA, 2011, pp.505-524.

[17] A. López-Alt, E. Tromer, V. Vaikuntanathan, On-the-fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption, *Proceedings of the Annual ACM Symposium on Theory of Computing*, New York, NY, USA, 2012, pp. 1219 -1234.

[18] M. Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan, Fully Homomorphic Encryption over the Integers, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, French Riviera, France, 2010, pp. 24-43.

[19] C. H. Lin, J. C. Liu, C. C. Li, Speeding Up RSA Encryption Using GPU Parallelization, *5th International Conference on Intelligent Systems, Modelling and Simulation*, Langkawi, Malaysia, 2014, pp. 529-533.

[20] C. H. Lin, J. C. Liu, C. C. Li, P. W. Chu, Parallel Modulus Operations in RSA Encryption by CPU/GPU hybrid computation, *9th Asia Joint Conference on Information Security*, Wuhan, China, 2014, pp. 71 -75.

[21] S. Saxena, B. Kapoor, An Efficient Parallel Algorithm for Secured Data Communications Using RSA Public Key Cryptography Method, *4th IEEE International Advance Computing Conference*, Gurgaon, India, 2014, pp. 850-854.

[22] K. Guo, Z. Liang, R. Shi, C. Hu, Z. Li, Transparent Learning: An Incremental Machine Learning Framework Based on Transparent Computing, *IEEE Network*, Vol. 32, No. 1, pp. 146-151, January-February, 2018.

[23] K. H. Guo, T. Li, R. H. Huang, J. Kang, T. Chi, DDA: A Deep Neural Network-based Cognitive System for IoT-aided Dermatosis Discrimination, *Ad Hoc Networks*, Vol. 80, pp. 95-103, November, 2018.

[24] X. Y. Wang, Z. Min, Parallel Algorithm for Hill Cipher on MapReduce, *2014 International Conference on Progress in Informatics and Computing (PIC)*, Shanghai, China, 2014, pp. 493- 497.

[25] Z. Min, G. Yang, J. Q. Shi, A Privacy-preserving Parallel and Homomorphic Encryption Scheme, *Open Physics*, Vol. 15, No. 1, pp. 135-142, June, 2017.

[26] C. S. Gu, Fully Homomorphic Encryption from Approximate Ideal Lattices, *Journal of Software*, Vol. 26, No. 10, pp. 2696-2719, October, 2015.

[27] J. H. Cheon, J. S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi and A. Yun, Batch Fully Homomorphic Encryption over the Integers, *32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Athens, Greece, 2013, pp. 315-335.

[28] R. Hiromasa, M. Abe, T. Okamoto, Packing Messages and Optimizing Bootstrapping in GSW-FHE, *International Conference on Practice and Theory in Public-Key Cryptography*, Gaithersburg, MD, USA, 2015, pp. 699-715.

[29] L. Ducas, D. Micciancio, FHEW: Bootstrapping Homomorphic Encryption in less than a Second, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Sofia, Bulgaria, 2015, pp. 617-640.

[30] X. Y. Yang, Y. T. Ding, T. P. Zhao, Parallel FHEW Based on Multi-core CPU, *Journal of Cryptologic Research*, Vol. 4, No. 6, pp. 620-626, December, 2017.

[31] Q. Shi, S. Han, X. M. Huang, L. Sun, X. Xie, T. Z. Tang, Design of Finite Field FFT for Fully Homomorphic Encryption Based on FPGA, *Journal of Electronics & Information Technology*, Vol. 40, No. 1, pp. 57-62, January, 2018.

[32] D. Tan, H. Wang, Fully Homomorphic Encryption Based On the Parallel Computing, *Ksii Transactions on Internet & Information Systems*, Vol. 12, No. 1, pp. 497-522, January, 2018.

[33] S. S. Roy, F. Turan, K. Jarvinen, F. Vercauteren, I. Verbauwhede, FPGA-based High-performance Parallel Architecture for Homomorphic Computing on Encrypted Data, *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Washington, DC, USA, 2019, pp. 387-398.

[34] C. Gentry, S. Halevi, V. Vaikuntanathan, A Simple BGN-Type Cryptosystem from LWE, *9th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, French Riviera, France, 2010, pp. 506 -522.

[35] M. Ajtai, Generating Hard Instances of the Short Basis Problem, *Proceedings of the 26th International Colloquium on Automata, Languages, and Programming*, Prague, Czech Republic, 1999, pp.1-9.

[36] J. Alwen, C. Peikert, Generating Shorter Bases for Hard Random Lattices, *26th International Symposium on Theoretical Aspects of Computer Science*, STACS 2009, Freiburg, Germany, 2009, pp. 75-86.

Biographies

Zhaoe Min received the master's degree in software engineering from Suzhou University, Suzhou, China, in 2007. She is currently a Ph.D. student of Nanjing University of Posts and Telecommunications, Nanjing, China. Her main research interest is information security and parallel computing.

Geng Yang born in 1961. Professor and Ph.D. supervisor with the School of Computer Science, Nanjing University of Posts and Telecommunications. His current research interests include computer communication and networks, parallel and distributed computing, and information security.

Jin Wang received the M.S. degree from Nanjing University of Posts and Telecommunications, China in 2005. He received Ph.D. degree from Kyung Hee University Korea in 2010. Now, he is a professor at Changsha University of Science and technology. He has published more than 300 international journal and conference papers. His research interests mainly include wireless ad hoc and sensor network, network performance analysis and optimization etc. He is a senior member of the IEEE and a member of ACM.

Gwang-jun Kim received the B.E, M.E and Ph.D. degrees in computer engineering from Chosun University in 1993, 1995 and 2000, respectively. He joined the department of computer engineering, Chonnam National University, in 2003 and became an Associate Professor in 2009. Since 2015, he has been an Professor in computer engineering at Chonnam National University. During 2000-2001, he was a researcher in the department of electrical and computer engineering at university of California, Irvine. His current research interest lie in the area sensor network, IoT, real-time communication and various kinds of communication systems.