ISSN:1991-8178

Australian Journal of Basic and Applied Sciences

Journal home page: www.ajbasweb.com

Preserving Privacy of Cloud Data Using Homomorphic Encryption in Map Reduce Framework

[1]G. Sujitha, [2]T. Rajeshwaran, [2]K. Vidya, [2]R. Thiagarajan and [2]S. Mercy Shalinie

[1]*Department of Information Science and Technology, Anna University, Chennai – 600015, Tamil Nadu, India*
[2]*Department of Computer Science, Thiagarajar College of Engineering, Madurai, Tamil Nadu, India*

ARTICLE INFO

Article history:
Received 12 November 2014
Received in revised form 26 December 2014
Accepted 29 January 2015
Available online 10 February 2015

Keywords:

ABSTRACT

In recent years, outsourcing large amount of data in cloud and managing data raises many challenges with respect to privacy. The concerns of privacy can be addressed if cloud users encrypt the data deployed in it. Among the various cryptographic encryption schemes, the homomorphic scheme allows to perform meaningful computations on encrypted data. This research deals with the homomorphic encryption scheme for maintaining privacy and security in cloud by detecting the error incurred while transferring data using RSA cryptosystem. Three types of homomorphic error detection schemes proved that MapReduce is an efficient model for preserving privacy.

To Cite This Article: G. Sujitha, T. Rajeshwaran, K. Vidya, R. Thiagarajan and S. Mercy Shalinie, Preserving Privacy of Cloud Data Using Homomorphic Encryption in Map Reduce Framework. ***Aust. J. Basic & Appl. Sci.,*** *9(6): 10-15, 2015*

INTRODUCTION

Cloud computing is started off with Grid Computing, where large number of systems are used for solving scientific problem that require high levels of parallel computation. This technology expanded exceptionally, which eventually stimulated concerns over ensuring data security in public networks. There's a lot of fear, uncertainty, and doubt around security of data deployed in public networks. According to a recent Survey conducted by Cisco Global Cloud Networking Academy, it has been revealed that 71 percent of IT professionals stated that security of data is a major hindrance to implement the services in cloud (Kui Ren, *et al*). Recent development in cloud storage and the services rendered by it allows users to outsource storage. As a result, it allows companies or organizations to offload the task of maintaining datacenters. In the past few years, the security requirements for data are very strong and many algorithms have evolved (Kui Ren, *et al*). Only a few algorithms play a comprehensive role in creating and maintaining a secure session over vulnerable public networks. Public key cryptography is one of the commonly used algorithms for this type of operation. The authenticity between the communicating parties is ensured by implementing this technique. These communicating parties share their private keys amongst them before exchanging information. In the case of transmitting a message over a public channel, the work of Diffie Helman (Diffie, 1947) and RSA (Cong Wang, *et al*)] provides way to encrypt a message into cipher text using private key. Consequently, the receiver on the other side has to read the cipher text by decryption with the help of their private key. The encryption scheme shows that the secret decryption key allows retrieving the actual text but if the secret key is lost, the ciphertext is of no use.

Related Work:

In the past years, homomorphic Encryption allows simple computation on encrypted data. Such practice has been existing since long time. The GM (Niu Yukun, 1012) encryption scheme supports addition of encrypted. A number of encryption systems that are either additively or multiplicatively homomorphic followed suit. Encryption sytems of El Gamal encryption scheme (Gentry, 105), Paillier encryption scheme (Liang Chen, 105)(Xingze, 105) and its generalization (Kui Ren, *et al*), a host of lattice-based encryption schemes (Cong Wang, *et al*)(Niu Yukun, *et al*) and others evolved (Niu Yukun, *et al*)(Ping Zhu, 105)(Gentry, 105). A system used in (Niu Yukun ,1012)(Saputro, 105) involved additive and multiplicative encrypted texts which has more number of additions and just one multiplication. Developing an encryption remained a major challenge. The additive and multiplicative homomorphisms form a complete set of operations. This scheme enables to perform any polynomial-time

Corresponding Author: G. Sujitha, Department of Information Science and Technology, Anna University, Chennai – 600015, Tamil Nadu, India.
E-mail: sujitha@auist.net

computation on encrypted data. Later, Gentry (Cong Wang, *et al*) constructed a fully homomorphic encryption which allows evaluation of arbitrary number of additions and multiplications on encrypted data (Liang Chen, 105; Ping Zhu, *et al*.,)(Fangyuan, Jin, *et al*). Rivest *et al*. (Fangyuan, Jin, 105)(Diffie, 1947) proposed the RSA with multiplicative homomorphism.

Homomorphic Property of RSA:

The algorithm to support the homomorphic property of RSA cryptosystem is explained. A public-key encryption scheme E = (Enc, Dec, KeyGen) is homomorphic if for all k and all (pk; sk) output from KeyGen(k), it is possible to define groups M1,C1 so that: The plaintext space M, and all ciphertexts output by Enc_{pk} are elements of C. For any m1, m1€M and c1,c1 € C with m1 = Dec_{sk} (c1) and m1 = Dec_{sk} (c1) it holds that: Dec_{sk} (c1 *c1) = m1 * m1 where the group operations * are carried out in C and M, respectively. and c1,c1 € C1 with m1 = Dec seck (c1) and m1 = Dec seck (c1) it holds that: Dec seck(c1 *c1) = m1 * m1.

On the otherhand, a fully homomorphic scheme is able to output a ciphertext that encrypts f(m1, ...,mt), where f is any desired function, which of course must be efficiently computable. There is no leakage of m1, ...,mt or f(m1, ...,mt), or any intermediate plaintext values.

Map Reduce Model:

Google proposed MapReduce to simplify data processing on large clusters. Hadoop is the most popular open source implementation of the MapReduce framework developed by Yahoo! and the apache software foundation (Guan, 1012). Map Reduce is deployed as a powerful data processing service over open source systems which have become increasingly popular for its parallel programming framework. MapReduce is designed on simple model with two keys such as map and reduce derived from functional programming languages (Guan, 1012). For any kind of application to run using map reduce model the input should be a set of key value pair and the mapper produces a intermediate key pair value that is sent to the reducer to carry out the rest of the application process. The explanation includes map and reduce definition. The functionality of map and reduce is presented. The fundamental unit of data in map reduce computations is represented as <key1, value1> pair, where keys and values are always just binary strings.

Parallelizing Encryption Through Mapreduce Process:

The main advantage of the programming paradigm is its support for parallelization. Since each mapper µr1only operates on one tuple at a time, the system can have many instances of µr1 operating on different tuples in U r−1 in parallel. After the map step, the system partitions the set of tuples output through the instances of µr1 which are created based on their key. That is, part i1 of the partition has all key; value pairs that have key ki1. Since reducer ρr1only operates on one part of this partition, the system based on the application creates many instances of ρr running on different parts in parallel. Data will be stored as contiguous blocks and every block is represented by unique *block id* (I_0, I_1, I_1...I_{n-1}). A MapReduce includes set of mappers (M_1,M_1... M_r) and reducers (R_1,R_1... R_r). The input is given to mapper in the form of <*block id, object*>.This object is the content of the HDFS block or the data stored in the corresponding *block id*. During encryption the mapper and the reducer function based on the key, value pair as discussed above.

Execution of Mapper:

Block <I_{r-1}, object ds1> is given to mapper M_r. The mapper will generate the corresponding encrypted output I'R and send it to the reducer R_r let W= {<I 0, object1>,<I_1,object1>,<I1,object2>.....<In- 1,object n>} then we can say that

I'r= Ir-1€ W<blockid,object> Mr (<blockid,Enc-compobject>)

Execution of Reducer:

The collected outputs from various mappers are written to the disk in the sequential order (I'_1, I'_2...I'_n). The Mapper reads block of equal size which can be optimized based on the available free nodes in the cloud environment as shown below:

Block Size = Input Data Size / Number of nodes configured for mapper

The mapper are assigned with chunks where mapper performs homomorphic encryption on chunk and detects if any error. The reducer writes the content onto HDFS (i.e., output).

Homomorphic Based Error Detection Scheme:

The error detection scheme includes input block that contains all the input chunks. Based on the size of the input, chunks are created. The number of chunks decides the type of error detection scheme. When the count of chunks is even, the basic error detection scheme for even chunks is evaluated as discussed above. When the count of chunks is an odd the error detection scheme for odd chunks is evaluated. Under this scheme a constant chunk is generated and is used during encryption. When the dataset is large involves large number of chunks relatively of the order of n it create follow the enhancement error detection scheme. The enhanced error detection scheme is allowed to run in parallel framework. K denotes number of input chunks taken for all three schemes. Basic ED scheme select two successive chunks and perform encryption of two chunks. It calculates the multiplication of two chunks

and perform the encryption the multiplication of two chunks. Basic ED scheme select two successive chunks and perform encryption of two chunks. Then calculate the multiplication of two chunks then perform the encryption the multiplication. When the chunk count to be processed is an odd number, the error detection scheme for odd chunks is evaluated. Under this scheme a constant chunk is generated and is used during encryption. When the dataset is relatively large of the order of 1 TB the enhanced error detection scheme is evaluated. The enhanced error detection scheme is allowed to run in parallel MapReduce framework. When the homomorphic property not satisfied, it indicates fault on the encrypted data. Figure 1 shows detailed design diagram of homomorphic property based fault detection of RSA algorithm.

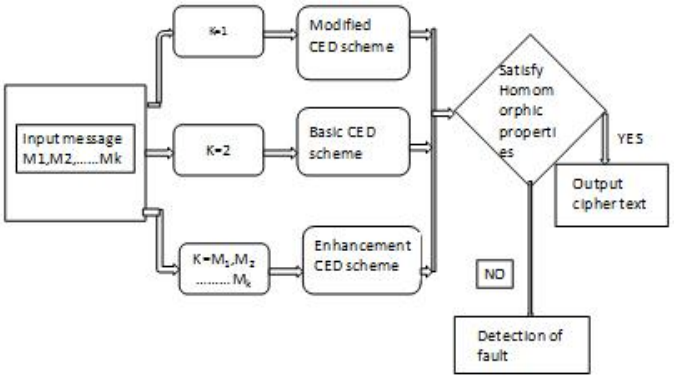

Fig. 1: Detailed Design Diagram of Homomorphic Error Detection Scheme.

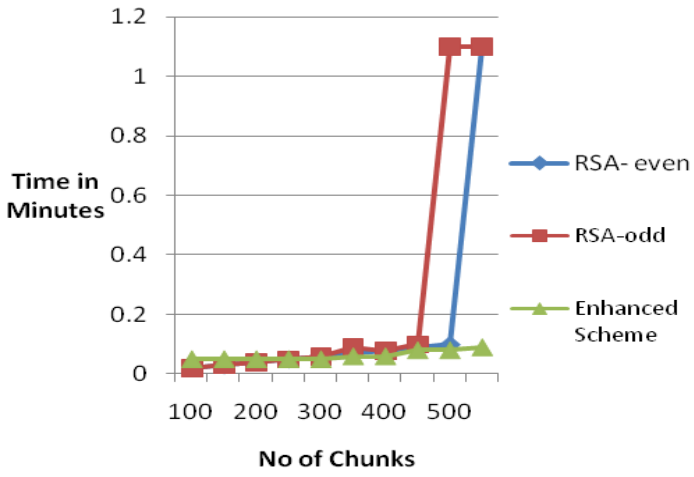

Fig. 2: Showing the comparison graph of the three schemes.

Homomorphic Error Detection Scheme for Even Chunks:

The homomorphic error detection scheme for even chunks operates on encrypted data in three steps namely one verification operation after three normal operations. The process starts with the encryption of two input messages and stores the result. The next step involves the encryption of the product of the two messages which occupies buffer. The comparison of the stored results is achieved through a comparator. The mismatch between the product of the ciphertext (i.e., $C_1 \times C_1$) and the ciphertext of the product of the chunks (C_2) shows that an injection has occurred. Figure 1 represents the basic multiplicative homomorphic property supported by RSA. On the other side , during such operation fault can even occur inside comparator. Such problem can be resolved using a self-checking comparator or a duplicate running in parallel. The results are stored in registers along with the secret information such as p1 and q1. This prevents the attacker to steal the ciphertext before verification is done. The drawback in the scheme is that it requires always even number of chunks. The algorithm for Homomorphic Encryption Scheme is developed as follows:

Step a: $C_1=E(M_1)$
Step b: $C_1=E(M_1)$
Step c: $C_2=E(M_1 \times M_1 \bmod n)$
Step d: If $C_2= (C_1 \times C_1) \bmod n$
No Fault -Output(C_1,C_1)
else
Fault Injected.

Homomorphic Error Detection scheme for odd number of chunks:

The basic ED scheme supports even chunks. Therefore to support odd chunks the next scheme is proposed. Here introduction of padding solves the problem. In such cases the simple chunk M_1 has to follow the previous algorithm proposed with a constant chunk M_1. The chunk M_1 can either be a constant chunk or a random generated value. The scheme encrypts input chunk M_1. It is followed by the encryption of chunk $M_1 \times M_{cons}$ mod n. The comparison of the result with the product of encryption of M_1 and constant chunk M_{cons} helps to identify the fault.

Enhanced Homomorphic Error Detection Scheme:

This scheme is designed for the chunks which are relatively large. In the previous schemes, the time overhead ranges from 50% to 100%. To reduce the time overhead further, the third scheme is proposed by performing the same using the MapReduce concept explained in the previous section. In this parallelized scheme, each mapper holds n chunks where the chunks are split based on the available number of mappers to compute the encrypted form of the input chunks. Mapper i..Mapper k are designed to perform encryption on the chunks resulting in $E(M_i)..E(M_j)$ under Mapper i to $E(M_k)..E(M_p)$ under Mapper k. The leftover odd chunks follow the padding scheme of constant chunks as explained previously. The operations inside the mapper involve homomorphic property check for RSA cryptosystem.

RESULTS AND DISCUSSIONS

Experimental Setup:

The error detection scheme without MapReduce has an environment which has an Intel I5 processor running at 1.2 GHz, 1 GB RAM 150 GB Serial ATA drive with an 4 MB buffer. The error detection of RSA is implemented in Java. The Hadoop's MapReduce is installed in an environment of 6 node cluster. On examining the overall system performance, the proposed scheme efficiently detects fault attack on RSA. Here the time overhead, hardware overhead and memory overhead are considered as performance metrics. Time overhead of the proposed ED scheme varies according to the count of chunks it took for processing. The Basic ED scheme took two chunks for processing. It produces 50 % of the time overhead. The ED for odd chunks produces 100% of the time overhead. The ED scheme using MapReduce produces minimum time overhead compared to the other two schemes.

Analysis of Error Detection Scheme for Even Chunks:

The homomorphic evaluation of the basic scheme involves various metrics which are measured against memory, latency and running time. The memory overhead is due to the values of the buffers that hold the ciphertexts. While running the basic ED scheme the time overhead incurred includes computation of product of the chunks ($M_1 \times M_1$), encrypting the product $E(M_1 \times M_1 \bmod n)$, computation of the product of the ciphertexts of the two chunks ($C_1 \times C_1$) and comparing the results. The time overhead due to multiplications are negligible when compared to the evaluation of encryption of product of chunks. It results in 50% time overhead in the case of basic error detection scheme. Fault detection latency which is defined as the time delay between the time of fault occurrence and the time of its detection is high when the fault occurs on the first chunk. It is therefore equal to the time cost of the three encryptions (i.e., $E(M_1)$, $E(M_1)$ and $E(M_1 \times M_1 \bmod n)$. The output latency of the scheme is high since the ciphertext comes out only after the verification.

Analysis of Error Detection Scheme for Odd Chunks:

In the error detection scheme for odd number of chunks, the introduction of constant chunk has the advantages of holding the constant value in the buffer. It saves time of creating a random value whenever it is required by the comparator. Rather than generating the second chunk value and encrypting the buffer can hold it permanently saves the overall running time. The cipher text of the constant chunk can be precomputed Instead of generating randomly whenever required. The time overhead for encryption of product of single chunk is 100 percent. But such overhead due to the last chunk has small impact as the size of chunk increases.

Analysis of Error Detection Scheme Using MapReduce:

Due to parallelization the time overhead incurred in computing encrypted form of each chunk is considerably reduED. The time taken to compute $c_1, c_1 .. c_n$ using Mappers relatively reduce the 50% overhead incurred before. However after computing the above ciphertext for n chunks the product of the chunks when calculated inside mapper_i ($M_i \times M_j$) and mapper_k computing ($M_k \times M_p$) gives the result which when iterated yields the result of $\text{mapper}_i \times \text{mapper}_j$. The time overhead due to multiplications are negligible when running mapreduce on compute clusters with large number of nodes. This results in less than 10% time overhead which varies with the number of nodes of cluster. The memory overhead involves the usage of buffers to store the results of mappers. The worst case scenario of fault detection latency happens when the fault occurs on the first chunk. When compared to the other two schemes here the fault detection latency cost is reduced to two encryptions. (i.e., one-time encryption calculation of all chunks $c_1 .. c_n$ and $E(M_i \times .. \times M_k)$. The output latency is relatively low even when it comes after

verification as the parallel computation yields one-time calculation of encryption of chunks. The comparison graph shown in the figure 2 suggests that enhanced ED scheme perform better than the other two algorithms because of its parallel framework support. It takes nearly less than 1 minute for 1000 chunks to be processed.

Conclusion:

In this research, an effective low-cost and high-performance error detection scheme that uses multiplicative homomorphic property of RSA is introdued. The time overhead for the error detection scheme for odd and even chunks varied from 50% to 100%. The one-time encryption of individual chunks using mapreduce algorithm has significantly improved the fault detection latency. The memory overhead depends on the mapper output values that are stored in buffers. All the schemes support for output latency is relatively low as the verification to find the fault occurs only after comparing the output of the mapper.

REFERENCES

Kui Ren, Cong Wang, Qian Wang, "Security Challenges for the Public Cloud", IEEE Internet Computing, 16(1): 69-72, 105.

Cong Wang, Ning Cao, Kui Ren, Wenjing Lou, "Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data", IEEE Transactions on Parallel and Distributed Systems, 12(4): 1167-1279, 105.

Aguilar-Melchor, C., S. Fau, C. Fontaine, G. Gogniat, R. Sirdey, 1012. "Recent Advances in Homomorphic Encryption: A Possible Future for Signal Processing in the Encrypted Domain", IEEE Signal Processing Magazine, 11(1): 104-57.

Diffie, W., M.E. Hellman, 1947. "New directions in cryptography" IEEE Transactions on Information Theory, 5(6): 622-652.

Niu Yukun, Tan Xiaobin , Chen Shi , Wang Haifeng , Yu Kai, Bu Zhiyong , 1012. "A security privacy protection scheme for data collection of smart meters based on homomorphic encryption", EUROCON, Digital Object Identifier: 10.509/EUROCON.1012.6615161: 1111-1115.

Guan, D.J., Chen-Yu Tsai, E.S. Zhuang, 1012. "Detect Zero by Using Symmetric Homomorphic Encryption", Eighth Asia Joint Conference on Information Security (Asia JCIS),

Digital Object Identifier: 10.509/ASIAJCIS.1012.4, pp: 1-7.

Han Jing-Li, Yang Ming, Wang Zhao-Li, 105, "Fully Homomorphic Encryption Scheme Extended t o Large Message Space", First International Conference on Instrumentation, Measurement, Computer, Communication and Control, Digital Object Identifier: 10.509/IMCCC.105.114, pp: 522-516.

Gentry, C., S. Halevi, 105. "Fully Homomorphic Encryption without Squashing Using Depth-2 Arithmetic Circuits", 51nd Annual Symposium on Foundations of Computer Science (FOCS),

Digital Object Identifier: 10.509/FOCS.105.92, pp: 107–109.

Liang Chen , Zhang Tong, Wen Liu, Chengmin Gao, 105. "Non-interactive Exponential Homomorphic Encryption Algorithm", International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Digital Object Identifier: 10.509/CyberC.105.22, pp: 52-57.

Saputro, N., K. Akkaya, 105. "Performance evaluation of Smart Grid data Aggregation via homomorphic encryption", Wireless Communications and Networking Conference (WCNC), Digital Object Identifier: 10.509/WCNC.105.65617, pp: 1915–1950.

Xingze, He., Man-On Pun, C.C.J. Kuo, 105. "Secure and efficient cryptosystem for smart grid using homomorphic encryption", PES Innovative Smart Grid Technologies (ISGT), Digital Object Identifier: 10.509/ISGT.105.695676, pp: 1-4.

Jian, Li, Sicong Chen, Danjie Song, 105. "Security structure of cloud storage based on homomorphic encryption scheme", 1nd International Conference on Cloud Computing and Intelligent Systems (CCIS), Digital Object Identifier: 10.509/CCIS.105.666611, pp: 52-57.

Xu Chen, Qiming Huang, 1012. "The data protection of mapreduce using homomorphic encryption", 2th IEEE International Conference on Software Engineering and Service Science (ICSESS), Digital Object Identifier: 10.509/ICSESS.1012.6615214, pp: 69-15.

Ping Zhu, Guangli Xiang, 105. "The Protection Methods for Mobile Code Based on Homomorphic Encryption and Data Confusion",
Fifth International Conference on Management of e-Commerce and e-Government (ICMeCG), Digital Object Identifier: 10.509/ICMeCG.105.14 , Page(s): 156 - 160

Fangyuan, Jin, Yanqin Zhu, Xizhao Luo, 105. "Verifiable Fully Homomorphic Encryption scheme" , 1nd International Conference on Consumer Electronics, Communications and Networks (CECNet), Digital Object Identifier: 10.509/CECNet.105.6101016, pp: 722-716.

Erkin, Z., T. Veugen, T. Toft, R.L. Lagendijk, 105. " Generating Private Recommendations Efficiently Using Homomorphic Encryption and Data Packing",IEEE Transactions on Information Forensics and Security, Digital Object identifier: 10.509/TIFS.105.590716. 7(2): 1052–1066.

Peter, A., E. Tews, S. Katzenbeisser, 1012. " Efficiently Outsourcing Multiparty Computation Under Multiple Keys", IEEE Transactions on Information Forensics and Security", Digital Object Identifier: 10.509/TIFS.1012.54416, 4(5): 1016–1054.

Bianchi, T., A. Piva, M. Barni, 1010," Composite Signal Representation for Fast and Storage-Efficient Processing of Encrypted Signals", IEEE Transactions on Information Forensics and Security, Digital Object Identifier: 10.509/TIFS.1009.1016111, 5(1): 140–147.

Jeffrey Dean and Sanjay Ghemawat. MapReduce:Simplified data processing on large clusters.Commun. ACM, 51(1):107-52,1004.

Hadoop.http://hadoop.apache.org

Andrew Pavlov, "A comparison of approaches to large scale data analysis"In proc of ACM Press,SIGMOD'09.