IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 5, May 2016

Cloud Computing Security Applied by Homomorphic Encryption

Santosh Kumar Singh[1], Dr. P.K. Manjhi[2], Dr. R.K. Tiwari[3]

Research Scholar, Department of Computer Applications, Vinoba Bhave University, Hazaribag, India [1]

Assistant Professor, University Department of mathematics, Vinoba Bhave University, Hazaribag, India [2]

Professor, H.O.D CSE, R.V.S College of Engg & Tech., Jamshedpur, India [3]

Abstract: Cloud Computing has been envisioned as the next generation architecture of IT Enterprise. In contrast to traditional solutions, where the IT services are under proper physical, logical and personnel controls, Cloud Computing moves the application software and databases to the large data centers, which provides the capability to use computing and storage resources on a rented basis and reduce the investments in an organization's computing infrastructure. With all its benefits, cloud computing also brings with it concerns about the security and privacy of information extant on the cloud as a result of its size, structure, and geographical dispersion. Cloud computing security challenges and it's also an issue to many researchers; first priority was to focus on security which is the biggest concern of organizations that are considering a move to the cloud. There are two major challenges to the Cloud Computing providers: first is How to guaranty a better data security and second is how we can keep the client private information confidential. When the data transferred to the Cloud we use standard encryption methods to secure the operations and the storage of the data. But to process data located on a remote server, the Cloud providers need to access the raw data. In this paper we are proposing an application of a method to execute operations on encrypted data without decrypting them which will provide us with the same results after calculations as if we have worked directly on the raw data.

Keywords: Homomorphic Encryption; Cloud Computing; Security; Confidentiality; Integrity.

I. INTRODUCTION

Homomorphic encryption is the conversion of data into cipher text that can be analyzed and worked with as if it were still in its original form. Homomorphic encryptions allow complex mathematical operations to be performed on encrypted data without compromising the encryption. In mathematics, homomorphic describes the transformation of one data set into another while preserving relationships between elements in both sets. The term is derived from the Greek words for "same structure." Because the data in a homomorphic encryption scheme retains the same structure, identical mathematical operations whether they are performed on encrypted or decrypted data will yield equivalent results. Homomorphic encryption is expected to play an important part in cloud computing, allowing companies to store encrypted data in a public cloud and take advantage of the cloud provider's analytic services [1].

Here is a very simple example of how a Homomorphic Encryption scheme might work in cloud computing:

- Business XYZ has a very important data set (VIDS) that consists of the numbers 5 and 10. To encrypt the data set, Business XYZ multiplies each element in the set by 2, creating a new set whose members are 10 and 20.
- Business XYZ sends the encrypted VIDS set to the cloud for safe storage. A few months later, the government contacts Business XYZ and requests the sum of VIDS elements.
- Business XYZ is very busy, so it asks the cloud provider to perform the operation. The cloud provider, who only has access to the encrypted data set, finds the sum of 10 + 20 and returns the answer 30.
- Business XYZ decrypts the cloud provider's reply and provides the government with the decrypted answer, 15.

Our basic concept was to encrypt the data before sending them to the Cloud provider. But, this one will have to decrypt them each time he has to work on them. The client will need to provide the private key to the server to decrypt the data before execute the calculations required, which might affect the confidentiality of data stored in the Cloud. The Homomorphic Encryption method is able to perform operations of encrypted data without decrypting them.
In this work we focus on the application of Homomorphic Encryption method on the Cloud Computing security, particularly the possibility to execute the calculations of confidential data encrypted without decrypting them. In Section II, we are introducing the concept of Cloud Computing and the necessity to adopt Homomorphic Encryption to secure the calculation of data hosted by the Cloud provider. In section III, we will define Homomorphic Encryption and we will illustrate some examples of existing Homomorphic cryptosystems. In section IV, we will present our scheme and our implementation. The conclusion and perspectives will be mentioned in section V.

II. CLOUD COMPUTING

Definition [2]: By cloud computing we mean: The Information Technology (IT) model for computing, which is composed of all the IT components (hardware, software, networking, and services) that are necessary to enable development and delivery of cloud services via the Internet or a private network.
This definition doesn't mention any security notion of the data stored in the Cloud Computing even being a recent definition. Therefore we understand that the Cloud Computing is lacking security, confidentiality and visibility. To Provide Infrastructure (IaaS), Platform Service (PaaS) or Software (SaaS) as a Service is not sufficient if the Cloud provider does not guaranty a better security and confidentiality of customer's data.
By convention, we consider as Cloud Computing any treatment or storage of personal or professional information which are realized outside the concerned structure (i.e outside the company), to secure the Cloud means secure the treatments (calculations) and storage (databases hosted by the Cloud provider). Cloud providers such as IBM, Google and Amazon use the virtualization on their Cloud platform and on the same server can coexist a virtualized storage and treatment space that belong to concurrent enterprises.
The aspect of security and confidentiality must intervene to protect the data from each of the enterprises. Secure storage and treatment of data requires using a modern aspect of cryptography that has the criteria for treatment such as, the necessary time to respond to any request sent from the client and the size of an encrypted data which will be stored on the Cloud server.
Our proposal is to encrypt data before sending it to the cloud provider, but to execute the calculations the data should be decrypted every time we need to work on it. Until now it was impossible to encrypt data and to trust a third party to keep them safe and able to perform distant calculations on them. So to allow the Cloud provider to perform the operations on encrypted data without decrypting them requires using the cryptosystems based on Homomorphic Encryption.

III.HOMOMORPHIC ENCRYPTION

Homomorphic Encryption systems are used to perform operations on encrypted data without knowing the private key (without decryption), the client is the only holder of the secret key. When we decrypt the result of any operation, it is the same as if we had carried out the calculation on the raw data.

Definition: An encryption is homomorphic, if: from Enc (a) and Enc (b) it is possible to compute Enc (f (a, b)), where f can be: +, ×, $\oplus$ and without using the private key. Among the Homomorphic encryption we distinguish, according to the operations that allows to assess on raw data, the additive Homomorphic encryption (only additions of the raw data) is the Pailler [3] and Goldwasser-Micalli [4] cryptosystems, and the multiplicative Homomorphic encryption (only products on raw data) is the RSA [5] and El Gamal [6] cryptosystems.

A. History of Homomorphic Encryption

The homomorphic property of various cryptosystems can be used to create secure voting systems,[7] collision-resistant hash functions, private information retrieval schemes and enable widespread use of cloud computing by ensuring the confidentiality of processed data. There are several efficient, partially homomorphic cryptosystems, and two fully homomorphic, but less efficient cryptosystems. Although a cryptosystem which is unintentionally homomorphic can be subject to attacks on this basis, if treated carefully homomorphism can also be used to perform computations securely.

Partially homomorphic cryptosystems: In the following examples, the notation $\mathcal{E}(x)$ is used to denote the encryption of the message x.
Unpadded RSA: If the RSA public key is modulus m and exponent e, then the encryption of a message x is given by $\mathcal{E}(x) = x^e \bmod m$. The homomorphic property is then

$$\mathcal{E}(x_1)\cdot\mathcal{E}(x_2) = x_1^e x_2^e \bmod m = (x_1 x_2)^e \bmod m = \mathcal{E}(x_1 \cdot x_2 \bmod m)$$

ElGamal: In the ElGamal cryptosystem, in a cyclic group G of order q with generator g, if the public key is (G, q, g, h), where $h = g^x$, and x is the secret key, then the encryption of a message m is $\mathcal{E}(m) = (g^r, m \cdot h^r)$, for some random $r \in \{0, \ldots, q-1\}$. The homomorphic property is then

$$\mathcal{E}(m_1)\cdot\mathcal{E}(m_2) = (g^{r_1}, m_1 \cdot h^{r_1})(g^{r_2}, m_2 \cdot h^{r_2})$$

$$= (g^{r_1+r_2}, (m_1 \cdot m_2)h^{r_1+r_2}) = \mathcal{E}(m_1 \cdot m_2).$$

Goldwasser–Micali: In the Goldwasser–Micali cryptosystem, if the public key is the modulus m and quadratic non-residue x, then the encryption of a bit b is $\mathcal{E}(b) = x^b r^2 \bmod m$, for some random $r \in \{0, \ldots, m-1\}$. The homomorphic property is then

$$\mathcal{E}(b_1)\cdot\mathcal{E}(b_2) = x^{b_1} r_1^2 x^{b_2} r_2^2 \bmod m = x^{b_1+b_2}(r_1 r_2)^2 \bmod m = \mathcal{E}(b_1 \oplus b_2)$$

Where $\oplus$ denotes addition modulo 2, (i.e. exclusive-or).
Benaloh: In the Benaloh cryptosystem, if the public key is the modulus m and the base g with a blocksize of c, then the encryption of a message x is $\mathcal{E}(x) = g^x r^c \bmod m$, for some random $r \in \{0, \ldots, m-1\}$.
The homomorphic property is then

$$\mathcal{E}(x_1)\cdot\mathcal{E}(x_2) \bmod m = (g^{x_1} r_1^c)(g^{x_2} r_2^c) \bmod m = g^{x_1+x_2}(r_1 r_2)^c = \mathcal{E}(x_1+x_2 \bmod m)$$

Paillier: In the Paillier cryptosystem, if the public key is the modulus m and the base g, then the encryption of a message x is $\mathcal{E}(x) = g^x r^m \bmod m^2$, for some random $r \in \{0, \ldots, m-1\}$.
The homomorphic property is then

$$\mathcal{E}(x_1)\cdot\mathcal{E}(x_2) = (g^{x_1} r_1^m)(g^{x_2} r_2^m) \bmod m^2 = g^{x_1+x_2}(r_1 r_2)^m \bmod m^2 = \mathcal{E}(x_1+x_2 \bmod m^2)$$

Other partially homomorphic cryptosystems:
(a) Okamoto–Uchiyama cryptosystem, (b) Naccache–Stern cryptosystem,(c) Damgård–Jurik cryptosystem, (d) Boneh–Goh–Nissim cryptosystem, (e) Ishai-Paskin cryptosystem.

Fully homomorphic encryption**:** Each of the examples listed above allows homomorphic computation of only one operation (either addition or multiplication) on plaintexts. A cryptosystem which supports both addition and multiplication (thereby preserving the ring structure of the plaintexts) is known as fully homomorphic encryption (FHE) and is far more powerful. Using such a scheme, any circuit can be homomorphically evaluated, effectively allowing the construction of programs which may be run on encryptions of their inputs to produce an encryption of their output. Since such a program never decrypts its input, it can be run by an untrusted party without revealing its inputs and internal state. The existence of an efficient and fully homomorphic cryptosystem would have great practical implications in the outsourcing of private computations, for instance, in the context of cloud computing. [8]

The "homomorphic" part of a fully homomorphic encryption scheme can also be described in terms of category theory. If C is the category whose objects are integers (i.e., finite streams of data) and whose morphisms are computable functions, then (ideally) a fully homomorphic encryption scheme elevates an encryption function to a functor from C to itself.

The utility of fully homomorphic encryption has been long recognized. The problem of constructing such a scheme was first proposed within a year of the development of RSA. [9] A solution proved more elusive; for more than 30 years, it was unclear whether fully homomorphic encryption was even possible. During this period, the best result was the Boneh-Goh-Nissim cryptosystem which supports evaluation of an unlimited number of addition operations but at most one multiplication.

Craig Gentry [10] using lattice-based cryptography showed the first fully homomorphic encryption scheme as announced by IBM on June 25, 2009. [11][12] His scheme supports evaluations of arbitrary depth circuits. His construction starts from a somewhat homomorphic encryption scheme using ideal lattices that is limited to evaluating low-degree polynomials over encrypted data. (It is limited because each ciphertext is noisy in some sense, and this noise grows as one adds and multiplies ciphertexts, until ultimately the noise makes the resulting ciphertext indecipherable.) He then shows how to modify this scheme to make it bootstrappable in particular, he shows that by modifying the somewhat homomorphic scheme slightly, it can actually evaluate its own decryption circuit, a self referential property. Finally, he shows that any bootstrappable somewhat homomorphic encryption scheme can be converted into a fully homomorphic encryption through a recursive self-embedding. In the particular case of Gentry's ideal-lattice-based somewhat homomorphic scheme, this bootstrapping procedure effectively "refreshes" the ciphertext by reducing its associated noise so that it can be used thereafter in more additions and multiplications without resulting in an indecipherable ciphertext. Gentry based the security of his scheme on the assumed hardness of two problems: certain worst-case problems over ideal lattices and the sparse (or low-weight) subset sum problem. Regarding performance, ciphertexts in Gentry's scheme remain compact insofar as their lengths do not depend at all on the complexity of the function that is evaluated over the encrypted data. The computational time only depends linearly on the number of operations performed. However, the scheme is impractical for many applications, because ciphertext size and computation time increase sharply as one increases the security level. To obtain 2k security against known attacks, the computation time and ciphertext size are high-degree polynomials in k. Recently, Stehle and Steinfeld reduced the dependence on k substantially [13]. They presented optimizations that permit the computation to be only quasi-k3.5 per boolean gate of the function being evaluated.

Gentry's Ph.D. thesis [14] provides additional details. Gentry also published a high-level overview of the van Dijk et al. construction (described below) in the March 2010 issue of Communications of the ACM [15].

In 2009, Marten van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan presented a second fully homomorphic encryption scheme,[16] which uses many of the tools of Gentry's construction, but which does not require ideal lattices. Instead, they show that the somewhat homomorphic component of Gentry's ideal lattice-based scheme can be replaced with a very simple somewhat homomorphic scheme that uses integers. The scheme is therefore conceptually simpler than Gentry's ideal lattice scheme, but has similar properties with regards to homomorphic operations and efficiency. The somewhat homomorphic component in the work of van Dijk et al. is similar to an encryption scheme proposed by Levieil and Naccache in 2008,[17] and also to one that was proposed by Bram Cohen in 1998.[18] Cohen's method is not even additively homomorphic, however. The Levieil-Naccache scheme is additively homomorphic, and can be modified to support also a small number of multiplications. In 2010, Nigel P. Smart and Frederik Vercauteren presented a refinement of Gentry's scheme giving smaller key and ciphertext sizes, but which is still not fully practical.[19]

At the rump session of Eurocrypt 2010, Craig Gentry and Shai Halevi presented a working implementation of fully homomorphic encryption (i.e. the entire bootstrapping procedure) together with performance numbers[20] .

In 2010 Riggio and Sicari presented a practical application of homomorphic encryption to a hybrid wireless sensor/mesh network. The system enables transparent multi-hop wireless backhauls that are able to perform statistical analysis of different kinds of data (temperature, humidity, etc.) coming from a WSN while ensuring both end-to-end encryption and hop-by-hop authentication [21].

Recently, Coron, Naccache and Tibouchi proposed a technique allowing reducing the public-key size of the van Dijk et al. scheme to 600 KB [22].

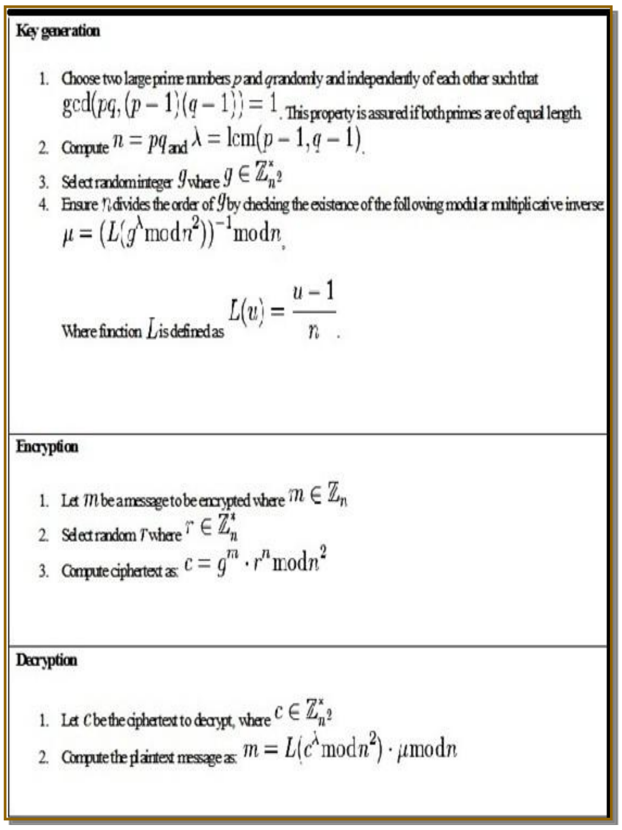
B. Additive Homomorphic Encryption

A Homomorphic encryption is additive, if:

$$\mathrm{Enc}(x \oplus y) = \mathrm{Enc}(x) \otimes \mathrm{Enc}(y)$$

$$\mathrm{Enc}\left(\sum_{i=1}^{l} m_i\right) = \prod_{i=1}^{l} \mathrm{Enc}(m_i)$$

Example: Fig. 1 shows how Paillier Cryptosystem (1999) generates Key, Encrypt and Decrypt.



Key generation

1. Choose two large prime numbers p and q randomly and independently of each other such that $\gcd(pq,(p-1)(q-1))=1$. This property is assured if both primes are of equal length
2. Compute $n = pq$ and $\lambda = \operatorname{lcm}(p-1, q-1)$.
3. Select random integer g where $g \in \mathbb{Z}^{*}_{n^2}$
4. Ensure n divides the order of g by checking the existence of the following modular multiplicative inverse: $\mu = (L(g^{\lambda} \bmod n^2))^{-1} \bmod n$,

Where function L is defined as $L(u) = \frac{u-1}{n}$.

Encryption

1. Let m be a message to be encrypted where $m \in \mathbb{Z}_n$
2. Select random r where $r \in \mathbb{Z}^{*}_{n}$
3. Compute ciphertext as: $c = g^m \cdot r^n \bmod n^2$

Decryption

1. Let c be the ciphertext to decrypt, where $c \in \mathbb{Z}^{*}_{n^2}$
2. Compute the plaintext message as: $m = L(c^{\lambda} \bmod n^2) \cdot \mu \bmod n$

Fig.1. Paillier Algorithm

Suppose we have two ciphers C_1 and C_2 such that:

$C_1 \;=\; g^{m1} . r_1^{\,n} \bmod n^{\,2}$

$C_2 \;=\; g^{m2} . r_2^{\,n} \bmod n^{\,2}$

$C_{1.}\, C_2 = g^{m1} . r_1^{\,n} \;.\; g^{m2} \;.\; r_2^{\,n} \bmod n^{\,2} = g^{m1+m2}\, (r_1\, r_{2)}^{\,n} \bmod n^2$

So, Pailler cryptosystem realizes the property of additive Homomorphic encryption. An application of an additive Homomorphic encryption is electronic voting: Each vote is encrypted but only the "sum" is decrypted.

C. Multiplicative Homomorphic Encryption

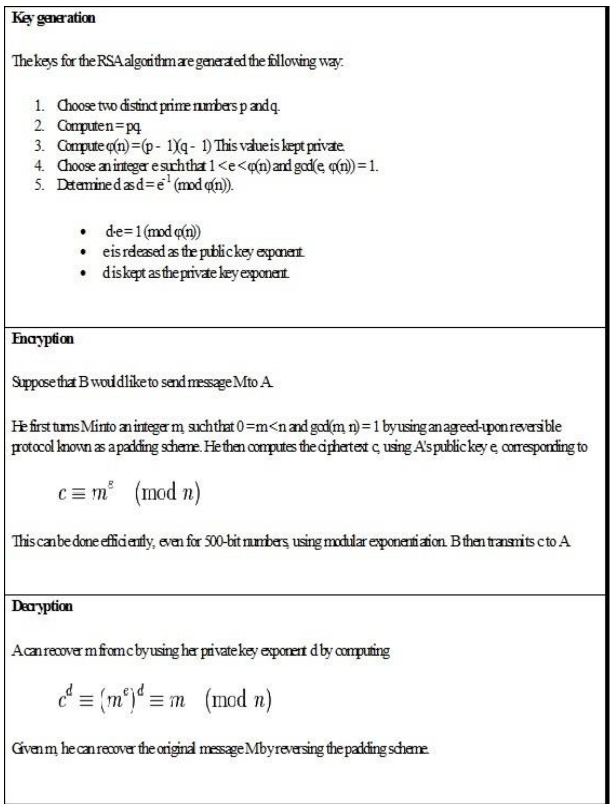
A Homomorphic encryption is multiplicative, if:

Enc $(x \otimes y)$ = Enc$(x) \otimes$ Enc(y)

$$\text{Enc}\left(\prod_{i=1}^{l} mi\right) = \prod_{i=1}^{l} \text{Enc}\,(mi)$$

Example: Fig. 2 shows how RSA Cryptosystem (1978) generates Key, Encrypt and Decrypt.

The RSA algorithm involves four steps: key generation, key distribution, encryption and decryption. RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key.



Key generation

The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.
2. Compute n = pq.
3. Compute φ(n) = (p - 1)(q - 1) This value is kept private.
4. Choose an integer e such that 1 < e < φ(n) and gcd(e, φ(n)) = 1.
5. Determine d as d = e⁻¹ (mod φ(n)).

- d·e = 1 (mod φ(n))
- e is released as the public key exponent.
- d is kept as the private key exponent.

Encryption

Suppose that B would like to send message M to A.

He first turns M into an integer m, such that 0 = m < n and gcd(m, n) = 1 by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c, using A's public key e, corresponding to

$$c \equiv m^e \pmod{n}$$

This can be done efficiently, even for 500-bit numbers, using modular exponentiation. B then transmits c to A.

Decryption

A can recover m from c by using her private key exponent d by computing

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

Given m, he can recover the original message M by reversing the padding scheme.

Fig.2. RSA Algorithm

The basic principle behind RSA is the observation that it is practical to find three very large positive integers' e, d and n such that with modular exponentiation for all m:

$$(m^e)^d \equiv m \pmod{n}$$

and that even knowing e and n or even m it can be extremely difficult to find d. Additionally, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies: $(m^d)^e \equiv m \pmod{n}$

Suppose we have two ciphers C_1 and C_2 such that:

$C_1 = m_1^{\,e} \bmod n$

$C_2 = m_2^{\,e} \bmod n$

$C_{1.}\, C_2 = m_1^{\,e}\, m_2^{\,e} \bmod n = (m_1\, m_2)^{\,e} \bmod n$

RSA cryptosystem realize the properties of the multiplicative Homomorphic encryption, but it still has a lake of security, because if we assume that two ciphers $C_{1,}$ C_2 corresponding respectively to the messages $m_{1,}\, m_2$, so:

$C_1 = m_1^{\,e} \bmod n$

$C_2 = m_2^{\,e} \bmod n$

The client sends the pair ($C_{1,}$ C_2) to the Cloud server the server will perform the calculations requested by the client and sends the encrypted result ($C_1 \times C_2$) to the client. If the attacker intercepts two ciphers C_1 and C_2, which are encrypted with the same private key, he/she will be able to decrypt all messages exchanged between the server and the client. Because the Homomorphic encryption is multiplicative, i.e. the product of the ciphers equals the cipher of the product.

Example: The application of RSA multiplicative Homomorphic encryption on two messages m_1 and m_2.

Let, for p=3, q=5, e=9 and d=1 with block size = 1. Two messages m_1 and m_2 and their ciphers C_1 and C_2 respectively, obtained using the RSA encryption.

m_1 = 589625 → C_1= 00 05 00 08 00 09 00 06 00 02 00 05

m_2 = 236491 → C_2= 00 02 00 03 00 06 00 04 00 09 00 01

The Block of C_1 In binary system	The Block of C_2 In binary system
00 05 => 00 0101	00 02 => 00 0010
00 08 => 00 1000	00 03 => 00 0101
00 09 => 00 1001	00 06 => 00 0110
00 06 => 00 0110	00 04 => 00 0100
00 02 => 00 0010	00 09 => 00 1001
00 05 => 00 0101	00 01 => 00 0001

The binary multiplication of the ciphers block by block is as follow:

00 0101 x 00 0010 = 00 1010	00 10
00 1000 x 00 0101 = 00 11000	00 24
00 1001 x 00 0110 = 00 110110	00 54
00 0110 x 00 0100 = 00 11000	00 24
00 0010 x 00 1001 = 00 10010	00 18
00 0101 x 00 0001 = 00 0101	00 05

If we decrypt the cipher C_1 X C_2 with the private key, we get:
C_1 C_2 = 00 10 00 02 00 04 00 05 00 04 00 02 00 04 00 01 00 08 00 05
So: m_1m_2 = 10 2 4 5 4 2 4 1 8 5

This is the exactly the same row message obtained by multiplying: m_1 x m_2

m_1 = 5 8 9 6 2 5
m_2 = 2 3 6 4 9 1

m_1 m_2 = 10 24 54 24 18 5 (we are multiplying m_1 x m_2 block by block).

IV. SCHEME AND IMPLEMENTATION

For all types of calculation on the data stored in the cloud, we must opt for the fully Homomorphic encryption which is able to execute all types of operations on encrypted data without decryption as shown in Fig. 3.

In 2009 Craig Gentry of IBM has proposed the first encryption system "fully homomorphic" that evaluates an arbitrary number of additions and multiplications and thus calculate any type of function on encrypted data [23].

The application of fully Homomorphic encryption is an important stone in Cloud Computing security more generally, we could outsource the calculations on confidential data to the Cloud server, keeping the secret key that can decrypt the result of calculation.

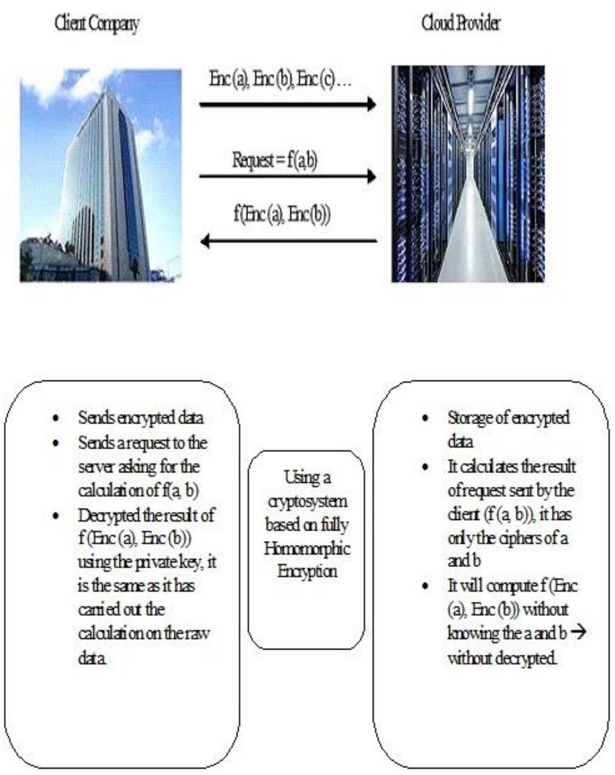

Client Company
Cloud Provider
Enc(a), Enc(b), Enc(c)…
Request = f(a,b)
f(Enc(a), Enc(b))
Sends encrypted data
Sends a request to the server asking for the calculation of f(a, b)
Decrypted the result of f (Enc (a), Enc (b)) using the private key, it is the same as it has carried out the calculation on the raw data
Using a cryptosystem based on fully Homomorphic Encryption
Storage of encrypted data
It calculates the result of request sent by the client (f (a, b)), it has only the ciphers of a and b
It will compute f (Enc (a), Enc (b)) without knowing the a and b → without decrypted

Fig.3. Homomorphic Encryption applied to the Cloud Computing

In our implementation, we analyze the performance of the existing Homomorphic encryption cryptosystems, we are working on a virtual platform with ESX as a Cloud server, a VPN network that links the Cloud to the client (enterprise), then later we started by simulating different scenarios using the Computer Algebra System Magma tools [24], focusing on:

- The size of the public key and its impact on the size of the encrypted message.
- The server delay of the request treatment according to the size of the encrypted message.
- The result decrypting time of the request according to the cipher text size sent by the server.

V. CONCLUSION AND PERSPECTIVE

The cloud computing security based on fully Homomorphic encryption is a new concept of security which enables providing results of calculations on encrypted data without knowing the raw data on which the calculation was carried out, with respect of the data confidentiality.

Our work is based on the application of fully Homomorphic encryption to the Cloud Computing security considering: the analyse and the improvement of the existing cryptosystems to allow servers to perform various operations requested by the client, and The improvement of the complexity of Homomorphic encryption algorithms and compare the response time of the requests to the length of the public key.

REFERENCES

[1] http://searchsecurity.techtarget.com/definition/homomorphic-encryption
[2] Vic (J.R.) Winkler, "Securing the Cloud, Cloud Computer Security Techniques and Tactics", Elsevier, 2011. ISBN-9781597495929.
[3] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In 18th Annual Eurocrypt Conference (EUROCRYPT'99), Prague, Czech Republic , volume 1592, 1999
[4] Julien Bringe and al. An Application of the Goldwasser-Micali Cryptosystem to Biometric Authentication, Springer-Verlag, 2007.
[5] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM, 21(2) :120-126, 1978. Computer Science, pages 223-238. Springer, 1999.
[6] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, 469-472, 1985.
[7] Ron Rivest (2002-10-29). "Lecture Notes 15: Voting, Homomorphic Encryption" (http:/ / web. mit. edu/ 6. 857/ OldStuff/ Fall02/ handouts/ L15-voting. pdf).
[8] Daniele Micciancio (2010-03-01). "A First Glimpse of Cryptography's Holy Grail" (http:/ / cacm. acm. org/ magazines/ 2010/3/76275-a-first-glimpse-of-cryptographys-holy-grail/ fulltext). Association for Computing Machinery. p. 96, 2010.
[9] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. In Foundations of Secure Computation, 1978.
[10] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices (http:/ / domino. research. ibm. com/ comm/ research_projects. nsf/ pages/ security. homoenc. html/ $FILE/ stocdhe. pdf). In the 41st ACM Symposium on Theory of Computing (STOC), 2009.
[11] http:/ / www-03. ibm. com/ press/ us/ en/ pressrelease/ 27840. Wss
[12] Michael Cooney (2009-06-25). "IBM touts encryption innovation" (http:/ / www. computerworld. com/ s/ article/ 9134823/ IBM_touts_encryption_innovation?taxonomyId=152& intsrc=kc_top& taxonomyName=compliance). Computer World, 2009.
[13] Damien Stehle; Ron Steinfeld (2010-05-19). "Faster Fully Homomorphic Encryption" (http:/ / eprint. iacr. org/ 2010/ 299) (PDF). International Association for Cryptologic Research, 2010.
[14] Craig Gentry. "A Fully Homomorphic Encryption Scheme (Ph.D. thesis)" (http:/ / crypto. stanford. edu/ craig/) .
[15] Craig Gentry. "Computing Arbitrary Functions of Encrypted Data" (http:/ / crypto. stanford. edu/ craig/ easy-fhe. pdf). Association for Computing Machinery.
[16] Marten van Dijk; Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan 2009, "Fully Homomorphic Encryption over the Integers" (http:/ / eprint. iacr. org/ 2009/ 616) (PDF). International Association for Cryptologic Research, 2010.
[17] "Cryptographic Test Correction" (http:/ / en. wikipedia. org/ wiki/ Cryptographic_Test_Correction). .
[18] Bram Cohen. "Simple Public Key Encryption" (http:/ / en. wikipedia. org/ wiki/ Cohen's_cryptosystem). .
[19] News report (http:/ / www. zdnet. co. uk/ news/ emerging-tech/ 2010, british-researcher-cracks-crypto-problem-40089057) http:/ / www. info. unicaen. fr/ M2-AMI/ articles-2009-2010/ smart. pdf paper] pdf slides (http:/ / www. math. leidenuniv. nl/ ~dfreeman/ smart. pdf) .
[20] Craig Gentry; Shai Halevi. "A Working Implementation of Fully Homomorphic Encryption" (http:/ / eurocrypt2010rump. cr. yp. to/ 9854ad3cab48983f7c2c5a2258e27717. pdf) .
[21] Roberto Riggio; Sabrina Sicari. "Secure Aggregation in Hybrid Mesh/Sensor Networks" (http:/ / disi. unitn. it/ ~riggio/ lib/ exe/ fetch. php?media=publications:sasn2009. pdf) .
[22] Jean-Sébastien Coron; David Naccache, Mehdi Tibouchi. "Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers" http:/ / eprint. iacr. org/ 2011/ 440.
[23] Craig Gentry Thesis, A Fully Homomorphic Encryption Scheme, 2009.
[24] WiebBosma, John Cannon, and Catherine Playoust. The Magma algebra system I: The user language. J. Symbolic Comput., 24(3-4): 235-265, 1997. Computational algebra and number theory, London, 1993.

BIOGRAPHY

Santosh Kumar Singh is a Research Scholar in the Department of Computer Applications, Vinoba Bhave University, Hazaribag, Jharkhand. He received M. Phil (Computer Science) degree in 2011 and Qualified Doctoral (Ph. D) Eligibility Test 2014 Vinoba Bhave University, Hazaribag. His research interests are Cloud Computing, Parallel and Distributed Computing etc.