Society for Science and Education
United Kingdom
TMLAI Transactions on Machine Learning and Artificial Intelligence
Volume 5 No 4, 2017
ISSN 2054-7390

Computing on Encrypted Data into the Cloud though Fully Homomorphic Encryption

Samiha Jlilab, Hassan Satori, Khalid Satori
LIIAN, Department of Mathematics and Computer Science, FSDM, USMBA,
Fez, Morocco
Department of Mathematics and Computer Science, FPN, UMP
Nador, Morocco
samihajlilab00@gmail.com, hsatori@yahoo.com

ABSTRACT

Securing Data in the cloud based on Fully Homomorphic Encryption (FHE) is a new and potential form of security that allows computing on encrypted data without decrypted it first. However, a practical FHE solution is not available for implementation today. In this work, we propose a platform based on open source solutions to perform data computations (addition and multiplication) on encrypted form. In addition, taking account of efficiently and the security component, the most popular partially homomorphic encryption algorithms (RSA, Paillier and ElGamal) are studied to analyze the process times of encryption, decryption and computation of each algorithm. Furthermore, to compromise between performance and security, we need to study different key sizes and different data sizes as parameters.

Keywords: Cloud Computing, Third-Party;Data Privacy, Fully Homomorphic Encryption, Partially Homomorphic Encryption, RSA, ElGamal, Paillier, Gentry's scheme.

1 Introduction

Cloud computing holds enormous potentials (cost reduction, shared computing-resources, elasticity, good performance, etc.). As a matter of fact, the cloud has few challenges like any other innovative technology. Data security is a key challenge when moving to the cloud[1]. In fact, when you move your data to the cloud, you are losing control. Cloud gives access to data, but you have no way to make sure that someone else does not have access to it as well. In addition, exposing your data in a public cloud environment shared with other companies may put your data at risk and may you lose the confidentiality of your sensitive data.The key solution is tostore and perform computations on cipher-text over a public area. This goal can be achieved using the homomorphic encryption proprieties[2]. Homomorphic encryption is the mathematical process of performing calculations on encrypted data which cannot be read or analyzed by anyone other than the client who retains the encryption and decryption keys. This kind of cryptography is an efficient solution to protect from internal and externals threats (curious cloud administrator, malicious cloud users, attackers).

Recently, many researchers are interested in fully homomorphic encryption solutions to improve data privacy and eliminate trusted third parties i.e. performing computational operations on encrypted data without learning anything about it by a third party. Naveed Islam et al[3] have exploited the multiplicative

DOI: 10.14738/tmlai.54.3225
Publication Date: 15th August, 2017
URL: http://dx.doi.org/10.14738/tmlai.54.3225

and the additive homomorphic proprieties of RSA and Paillier algorithms to secure image sharing between players. M.Tebaa et al. [4-5] have implemented the two previous asymmetric algorithms, on a VMware vSphere environment, to secure computations on sensitive data (bank data). A.Chantterjee and I.Sengupta [6]applied FHE technique to search and store encyrpted data on the cloud environnement and they have re-used the homomorphic modules proposed in scarab library [7]to perform search and store over encrypted data.

Our contribution is to develop a practical, efficient and secure platform based on different homomorphic algorithms (Paillier, RSA and ElGamal) to allow performing calculations, by a third party, with encrypted data. The performance of our platform is analyzed by studying the processing time of encryption, computation and decryption phases.

The rest of this paper is divided into seven sections. In the 2nd section, we start by introducing cloud computing concept and the security component as issue for cloud adaptation by companies and users. In the 3rd section, we define the homomorphic encryption propriety, its types and some examples of homomorphic cryptosystems. In the 4th section, we present the system conception. The experimental platform is described in the 5th section. In the 6th section we present our simulation procedure and results. Finally, the conclusion and perspectives will be mentioned in 7th section.

2 Cloud Computing

Cloud Computing is the new form of Information Technology (IT) outsourcing that provides its solutions as an on-demand consumable services. It consists of using applications, platforms or distributed virtual infrastructures which are not necessarily located in the company's premises. The value of Cloud computing is to introduce a new way of managing IT, providing better control of Direction of information systems (DIS) expenses and allowing enterprises to renovate the core of their business without worrying about time and money constraints related to the integration of new technologies.

2.1 Official Definition of Cloud Computing

The National Institute of Standards and Technology (NIST)defined cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction[8].

The NIST definition lists five essential characteristics of Cloud Computing: on-demand self-service, broad network access, resource pooling, rapid elasticity or expansion, and measured service. It also lists three service models (IaaS, PaaS and SaaS), and four deployment models (public, private, community and hybrid).

2.2 Security issue in cloud computing

The NIST's cloud definition did not include security such as an essential characteristic of the cloud computing however security is a potential concept which must intervene for preserving and protecting the confidential and sensitive data of each enterprises. In fact, every technology has a flaw so does cloud computing. The client has also participate to protect his data from harm (attackers, malicious administrator…). So, cloud security should be a shared responsibility between cloud provider and cloud users.

Cloud security requires a profound questioning of corporate security policies. They must go beyond the narrow managing passwords and login privileges (access control). It is necessary to take the next step and think safety in terms of use and data types [9]. For this, Fully Homomorphic Encryption will be a good solution to protect the data and an efficient way to take advantage of cloud services.

3 Homomorphic Encryption

In traditional encryption schemes, Bob encrypts a plaintext message to obtain a cipher text. Alice decrypts the cipher-text to find the plaintext. So, with this standard encryption, we can only secure the following steps: establishment of communication and data transfer. In Homomorphic Encryption, parties that do not know anything about the plaintext data can perform computations on it by performing computations on the corresponding cipher-text.

Definition of Homomorphic Encryption:

The encryption algorithm E () is homomorphic if given E(x) and E(y), one can obtain E (x□y) without decrypting x; y for some operation □ (+, ×...) [10].

3.1 Partially Homomorphic Encryption (PHE):

A cryptosystem is partially homomorphic if it supports adding or multiplying of cipher-texts but not both operations at the same time. The Goldwasser-Micali [11] and Paillier [12] schemes supported addition operations, while the RSA [13] and ElGamal [14]schemes supported multiplication operations.

3.1.1 Additive Homomorphic Encryption (AHE):

A scheme is an additive Homomorphic Encryption (AHE) if you give only the public-key and the encryption of $\mathrm{m1}$ and$\mathrm{m2}$, one can compute the encryption of$\mathrm{m1} + \mathrm{m2}$.

Mathematically, E is an AHF if:

$$E(x + y) = E(x) \times E(y)$$

$$E\left(\sum_{i=1}^{n} m_i\right) = \prod_{i=1}^{n} E(m_i)$$

3.1.2 Multiplicative Homomorphic Encryption (MHE):

A scheme is a Multiplicative Homomorphic Encryption (MHE) if you give only the public-key and the encryption of $\mathrm{m1}$ and$\mathrm{m2}$, one can compute the encryption of$\mathrm{m}_1\mathrm{m}_2$.

Mathematically, E is a MHE if:

$$E(x \times y) = E(x) \times E(y)$$

$$E\left(\prod_{i=1}^{n} m_i\right) = \prod_{i=1}^{n} E(m_i)$$

3.2 Asymmetric Homomorphic Encryption Algorithms

3.2.1 Paillier Scheme; an AHE:

Paillier cryptosystem is an asymmetric algorithm for public key cryptography and it is an AHE.The algorithm consists of three components: the key generator, the encryption algorithm and the decryption algorithm. The three components work as follow:

Table 1. Paillier Cryptosystem

Key Generator : KeyGen(p, q)
Input: $\boldsymbol{p, q \in \mathbb{Z}, \gcd(p, q) = 1}$
Compute $\boldsymbol{n = p \times q}$, $\boldsymbol{\lambda = lcm(p-1, q-1) = \frac{(p-1)(q-1)}{pgc(p-1,q-1)}}$ Choose $\boldsymbol{g} \in \mathbb{Z}^*_{n^2}$ such that $\boldsymbol{\gcd(L(g^{\lambda} mod n^2), n) = 1 with L(u) = \frac{u-1}{n}}$
Output: $\boldsymbol{(pk, sk)}$ Public key: $\boldsymbol{pk = (n, g)}$ Secret key: $\boldsymbol{sk = (p, q)}$
Encryption: E(m, pk)
Input: $\boldsymbol{m \in \mathbb{Z}_n with m < n}$
Choose $\boldsymbol{r} \in \mathbb{Z}^*_{n^2}$ Compute $\boldsymbol{C = g^m r^r mod(n^2)}$
Output: $\boldsymbol{C} \in \mathbb{Z}_{n^2}$
Decryption: D(c, sk)
Input: $\boldsymbol{C} \in \mathbb{Z}_{n^2}$
Compute $\boldsymbol{m = \frac{L(c^{\lambda} mod n^2)}{L(g^{\lambda} mod n^2)} mod n}$
Output: $\boldsymbol{m} \in \mathbb{Z}_n$

Paillier's algorithm realizes the property of additive homomorphic encryption.

Proof: suppose we have two ciphers $E(m_1) = C_1 et E(m_2) = C_2$

$$E(m_1) = C_1 = g^{m1}.r_1^n mod n^2$$

$$E(m_2) = C_2 = g^{m2}.r_2^n \bmod n^2$$

So:

$$C_1.C_2 = g^{m1}.r_1^n \times g^{m2}.r_2^n \bmod n^2 = g^{m1+m2}.(r_1 r_2)^n \bmod$$

i.e.

$$\boldsymbol{E(m_1).E(m_2) = E(m_1 + m_2)}$$

3.2.1 Paillier cryptosystem can be a MHE?

Suppose we have two plaintexts m_1 et m_2:

$E(m_1) = g^{m1}.r_1^n mod n^2$.

$$E(m_1)^{m_2} = g^{m1\, m_2}.r_1^n \bmod n^2 = E(m_1 m_2).$$

So with this trick, we can realize multiplicative operation with Paillier cryptosystem.

3.2.2 RSA Scheme, a MHE:

RSA is an asymmetric cryptosystem. It was described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman. The scheme summarizes in three steps: the key generator, the encryption algorithm and the decryption algorithm.The three components work as follow:

Table 2 RSA cryptosystem

Key Generations : KeyGen(p, q)
Input: $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{Z}, \mathbf{gcd}(\boldsymbol{p}, \boldsymbol{q}) = \mathbf{1}$
Compute $\boldsymbol{n} = \boldsymbol{p} \times \boldsymbol{q}$ $\boldsymbol{\varphi}(\boldsymbol{n}) = (\boldsymbol{p} - \mathbf{1})(\boldsymbol{q} - \mathbf{1})$ Choose $\boldsymbol{e} \in \mathbb{Z}_{\boldsymbol{n}}$ such that $\mathbf{gcd}(\boldsymbol{e}, \boldsymbol{\varphi}(\boldsymbol{n})) = \mathbf{1}$ Determine $\boldsymbol{d} \in \mathbb{Z}_{\boldsymbol{n}}$such that $\boldsymbol{e} \times \boldsymbol{d} = \mathbf{1}\, \boldsymbol{mod}(\boldsymbol{\varphi}(\boldsymbol{n}))$
Output: $(\boldsymbol{pk}, \boldsymbol{sk})$ Public key: $\boldsymbol{pk} = (\boldsymbol{n}, \boldsymbol{e})$ Secret key: $\boldsymbol{sk} = \boldsymbol{d}$
Encryption: E(m,pk)
Input: $\boldsymbol{m} \in \mathbb{Z}_{\boldsymbol{n}} \boldsymbol{with m} < \boldsymbol{n}$
Compute $\boldsymbol{C} = \boldsymbol{m}^{\boldsymbol{e}} \boldsymbol{mod}(\boldsymbol{n})$
Output: $\boldsymbol{C} \in \mathbb{Z}_{\boldsymbol{n}}$
Decryption: D(c. sk)
Input: $\boldsymbol{C} \in \mathbb{Z}_{\boldsymbol{n}}$
Compute $\boldsymbol{m} = \boldsymbol{C}^{\boldsymbol{d}} \boldsymbol{modn}$
Output: $\boldsymbol{m} \in \mathbb{Z}_{\boldsymbol{n}}$

RSA realizes the property of Multiplicative Homomorphic encryption.

Proof: Suppose we have two hers $E(m_1) = C_1 et E(m_2) = C_2$

$$E(m_1) = C_1 = {m_1}^e mod(n)$$

$$E(m_2) = C_2 = {m_2}^e mod(n)$$

$$E(m_1) \times E(m_2) = {m_1}^e {m_2}^e mod(n) = (m_1 m_2)^e mod(n)$$

And $E(m_1 m_2) = (m_1 m_2)^e mod(n)$

So: $\boldsymbol{E}(\boldsymbol{m}_1) \times \boldsymbol{E}(\boldsymbol{m}_2) = \boldsymbol{E}(\boldsymbol{m}_1 \boldsymbol{m}_2)$

3.2.3 ElGamal cryptosystem; a MHE:

The ElGamal encryption system is an asymmetric key encryption algorithm for public-key cryptography which is based on the Diffie-Hellman key exchange. It was described by Taher El-Gamal in 1985.

The scheme summarizes in three steps which are described in table3.

Table 3 ElGamal cryptosystem

Key Generator : KeyGen(p, g, s)
Input: $\boldsymbol{p} \in \mathbb{Z}, \boldsymbol{g} \in \mathbb{Z}_{\boldsymbol{p}}$
Choose $\boldsymbol{s} \in \mathbb{Z}, \boldsymbol{with}\ \mathbf{1} < \boldsymbol{s} < \boldsymbol{p} - \mathbf{1}$ Choose $\boldsymbol{h} \in \mathbb{Z}_{\boldsymbol{n}}$ Compute $\boldsymbol{h} = \boldsymbol{g}^{\boldsymbol{s}}$
Output: $(\boldsymbol{pk}, \boldsymbol{sk})$ Public key: $\boldsymbol{pk} = (\boldsymbol{p}, \boldsymbol{g}, \boldsymbol{h})$ Secret key: $\boldsymbol{sk} = \boldsymbol{s}$
Encryption: E(m, pk)
Input: $\boldsymbol{m} \in \mathbb{Z}_{\boldsymbol{p}}$
Compute $\boldsymbol{E}(\boldsymbol{m}) = (\boldsymbol{C1}, \boldsymbol{C2}) = (\boldsymbol{g}^{\boldsymbol{k}}, \boldsymbol{m}\boldsymbol{h}^{\boldsymbol{k}})\ \boldsymbol{mod}(\boldsymbol{p})$
Output: $\boldsymbol{C1},\ \boldsymbol{C2} \in \mathbb{Z}_{\boldsymbol{p}}$
Decryption: D(c, sk)
Input: $\boldsymbol{C1},\ \boldsymbol{C2} \in \mathbb{Z}_{\boldsymbol{p}}$
Compute $\boldsymbol{m} = \frac{\boldsymbol{C2}}{\boldsymbol{C1}^{\boldsymbol{s}}}\boldsymbol{modp i.e.}\ \boldsymbol{m} = \boldsymbol{C_1}^{\boldsymbol{p}-\mathbf{1}-\boldsymbol{s}}.\boldsymbol{C_2 modp}$
Output: $\boldsymbol{m} \in \mathbb{Z}_{\boldsymbol{p}}$

The scheme realizes the property of MHE.

<u>Proof:</u> Suppose we have two ciphers $E(m_1)etE(m_2)$

$$E(m_1) = (g^{k1}, m_1 h^{k1})$$

$$E(m_2) = (g^{k2}, m_2 h^{k2})$$

So: $E(m_1) \times E(m_2) = \left(g^{k1}, m_1 h^{k1}\right) \times \left(g^{k2}, m_2 h^{k2}\right)$

$= (g^{k1+k2}, m_1 m_2 \times h^{k1+k2})$.

And: $E(m_1 m_2) = (g^k, m_1 m_2 \times h^k)$.

⇨ $E(m_1) \times E(m_2) = E(m_1 m_2)$.

3.2.4 ElGamal cryptosystem can be an AHE?

If we practice some modifications on El-Gamal scheme such as replacing the plaintextmbyg^{m}, this scheme will be realized the property of an additive homomorphic cryptosystem.

<u>Proof:</u> Suppose we have two ciphers$E(m_1)etE(m_2)$:

$$E(m_1) = (g^{k1}, g^{m_1}.h^{k1})$$

$$E(m_2) = (g^{k2}, g^{m_2}.h^{k2})$$

So:

$$E(m_1) \times E(m_2) = \left(g^{k1}, g^{m_1} h^{k1}\right) \times \left(g^{k2}, g^{m_2} h^{k2}\right)$$

$$= (g^{k1+k2}, g^{m_2+m_1} \times h^{k1+k2})$$

⇨ $E(m_1) \times E(m_2) = E(m_1 + m_2)$

3.3 Fully Homomorphic Encryption(FHE)

Fully Homomorphic Encryption (FHE) allows performed arbitrary operations on encrypted data without actually observing the raw data.

Being fully homomorphic means that whenever f is a function composed of additions and multiplications in the ring, then $\mathrm{Dec}(\mathrm{f}(\mathrm{C}_1, \dots, \mathrm{C}_n) = \mathrm{f}(\mathrm{m}_1, \dots, \mathrm{m}_n)$. So, if the cloud provider can efficiently compute $\mathrm{f}(\mathrm{m}_1, \dots, \mathrm{m}_n)$ from cipher-texts $\mathrm{C}_1, \dots, \mathrm{C}_n$, without learning any information about the corresponding plaintexts$\mathrm{m}_1, \dots, \mathrm{m}_n$, then the system is efficient and secure [15].

In 2009, Craig Gentry has invented the first fully homomorphic encryption scheme (based on ideal lattices and LWE) that evaluates an arbitrary number of additions and multiplications and thus calculate any type of function on encrypted data [16].

Gentry's Algorithm contains the following four steps:

Key-generator: three keys $(\boldsymbol{S_k}, \boldsymbol{P_k}, \boldsymbol{E_k})$
Encryption: Enc (pk, m) → $\boldsymbol{C}$
Evaluation:$\boldsymbol{Eval_{E_k}(f, C)}$
Decryption: Dec($\boldsymbol{S_k}$,$\boldsymbol{C}$) → f(m)

Gentry's model includes several steps. The first one is constructing the Somewhat Homomorphic Encryption (SWHE) that supported a limited number of additions and multiplications on cipher-text in order to limit the noise component. The next step is squashing the decryption function to be expressed as a low degree polynomial. Then, using the Bootstrapping technique to refresh the cipher-texts before every homomorphic operation. Using this technique, the number of homomorphic operations becomes unlimited and we get a FHE scheme.

Gentry's solution improves the efficiency of secure multiparty computation. But, until today, his theoretical solution has a practical problem: it is complicated and imposes a large public keys size. Also, perform computations on encrypted data is too slow than on raw data[17].

3.3.1 DGHV Scheme;a FHE over integer

Based on Gentry's construction, V. Dijk et al[18] have proposed a simple algorithm applying simple addition and multiplication in order to reduce size of keys by using subset of the original public key. The table 5 as below, describe the various steps of DGVH scheme.

Table 4 DGVH scheme

Parameters
p: *secret key*, p *is a prime odd integer* $ev_k = \{x_i = pq_i + r_i\}$
Encryption
Input : $m \in \{0,1\}$ $compute: \ c = pq + 2r + m$
Evaluation
$compute: \ c_{add} = (c_1 + c_2) \ mod \ x_0$ $compute: \ c_{mul} = (c_1 c_2) \ mod \ x_0$
Decryption
$Input: c$ $compute: c \ modp \ mod2 = (2r + m) mod2 = m$ $Output: m$

3.3.2 BGV Scheme; FHE without bootstrapping

Z. Brakerski et al.[19] have proposed a new approach without gentry's bootstrapping technique. For managing the noise component of cipher-text as homomorphic operations are performed, the BGV is based on new techniques developed by Brakerski and vaikuntanathan[20]. The table 6 as below, describes the BGV scheme.

Table 5 BGV scheme

Parameters
λ: *secuity parameter* n, k: *positives integers* q : *odd integer* e: *noise* A: $k \times n$ *marix* $\in Z_q^{k\times n}$ s: *secret key* $v = As + 2e$: *public key*
Encryption
Input : $m \in \{0,1\}$, r $\in \{0,1\}^k$ Compute: $c_1 = A^T r and c_2 = v^T + m$ Output: $(c_1, c_2) \in Z_q^{n+1}$
Decryption
Input : (c_1, c_2) Compute: $C = c_2 - < a, s >$ $m = C mod 2$ Output: $m \in Z_q$

All these algorithms can be applied on several domains and applications (e-voting System, Cloud Computing services, banking and Financial Applications, Medical Applications, Securing Treatment of Personal data, etc.)[21]–[24]

4 System conception

As we previously mentioned, this work focus on finding a scheme to treat encrypted data and perform complex calculations, in a cloud environment, without first decrypted it. So, the main goal is to create an algorithm on which we can use both cryptosystems when we have arbitrary operations. We can summarize our approach on three principal steps:

4.1 Encryption

we create, locally, a program based on PHE schemes to create an encrypted files or encrypted databases. In this program, we take into account the concept of parallelism because a parallel calculation is performed on more computing units. I.e. the computation is divided into parts that can run concurrently in order to reduce the processing time, solve problems of large size and to be able to handle multiple things at one time. Then, we send encrypted data into the cloud provider to store them.

4.2 Treatment

includes three sub-steps*:*

- Sends a request to cloud to perform calculations on encrypted data.
- Cloud's compute server has a function f for doing computations of cipher-texts. This computations are performed in order of priority.

- Send the result, of the request, which will be decrypted by Client Company.

4.3 Decryption:

Decrypts the result using the private key and comparing the results obtained with our results performed on raw data.

The Figure.1 presents the architecture of our conception system.

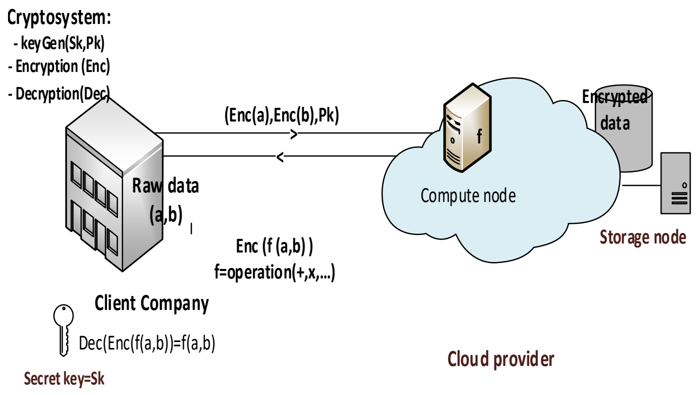

Figure 3. The proposed architecture of computing on encrypted data.

5 Experimental setup

The execution results are taken on Dell laptop having Intel® core ™ i5-4310M (2.70 GHz) Processor, 8 GB DDR3L (RAM), 320GB HDD, and Ubuntu 16.04 LTS Desktop (64-bits) operating system.

Concerning the platform of this work, our implementation is divided into two main parts:

5.1 Cloud platform:

using a cloud environment is a main part of our thesis. So, we implement a virtual cloud platform based on open-source solutions: OpenStack and KVM. The installation and the configuration of OpenStack's services implemented on four nodes (controller node, compute node, networking node and storage-Swift node). Each node is a VM under KVM having minimums material characteristics and the Ubuntu server 14.04 LTS 64 bits as operating system. We had tested our cloud functionality by creating network devices, creating images and instances, uploading and downloading raw data (files and images).

5.2 Java platform*:*

we use java language to test various homomorphic encryption algorithms in order to analyze and compare between them. This way, we're going to take account of the process times of encryption, computation and decryption time, size of keys and size of data. In addition, java platform provides an excellent base for writing secure applications by integrating JCA and JCE frameworks, and Bouncy Castle provider to implement the algorithms easily and with few lines of code. In this work, we implemented manually the cryptosystems in order to do some modifications on the encryption equations of Paillier and ElGamal algorithms.

The experiment is diagrammed as follow:

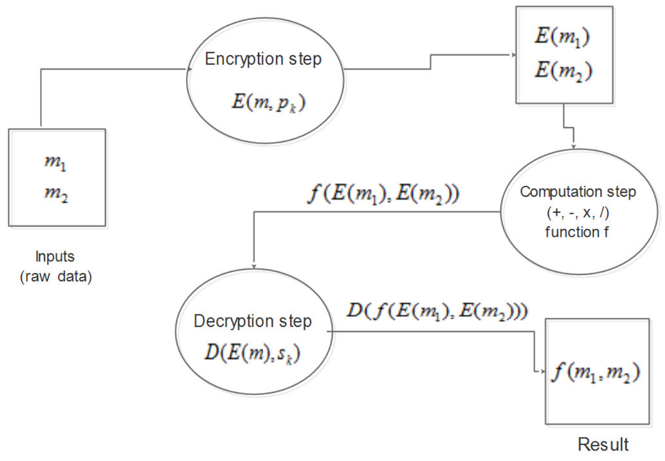

Figure 4. Diagram of homomorphic encryption operations processing.

6 Simulation procudure and results

6.1 Procedure

Our experimental starts by generating a set of data with different size (from 32bytes up to 1MB). In addition we are using digital data type to perform our mathematical calculations. In fact, the type of the data doesn't affect our results because encryption depends only on data size and not data type.

Depending on the input parameters and the security component, the simulation execution of our cryptosystem algorithms was on two experiments:

- In the 1st experiment: we fixed data size and varied key length in order to analyze the efficiently of the most popular asymmetric homomorphic algorithms (RSA, Paillier and ElGamal) by analyzing the process times of each algorithm's phases (encryption, computation and decryption).
- In the 2nd experiment, we fixed the key size i.e. we used the default key length recommended by NIST to preserve the security component for each algorithm, and we varied the data length.

After a successful execution, encrypted and decrypted files are created. To make sure that all the data are processed in the right way, we made a comparison between the original data file and the decrypted data file. Thus, we execute the mathematic operations performed on encrypted data for the previous cryptosystems in order to achieve our goal (efficient computations with encrypted data).

6.2 Results

6.2.1 The 1st experiment:

the simulation results of the encryption step are shown in fig.2, fig.3, table 8 and table 9, and those of the decryption step are shown in fig.4, fig.5, table 10 and table 11. The results show that the process time of RSA is less than the one of Paillier and ElGamal algorithms. For more clarification, bellow is a statistic study:

- *For small data (1KB):* the process time of the previous schemes cannot exceed a few milliseconds with a key-length less than 1024 bits. With a key >= 2048 bits, the process time of Paillier system can reach more than 4 seconds.

- *For large data (1MB):* The process time (key= 2048) of the encryption step can reach more than 42 minutes for Paillier, more than 12 minutes for ElGamal and 5 second for RSA. Concerning the process time of the decryption step, it can reach more than 72 minutes for Paillier, more than 18 minutes for ElGamal and 9 minutes for RSA.

6.2.2 The 2nd experiment

The simulation results of the encryption step are shown in table 12 and fig. 6, and those of the decryption are shown in table 13 and fig. 7. As the first execution, the results show that the time process of RSA is very less than the other Algorithms. However, with a large key, the process time take a long time to execute the steps (encryption and decryption).

6.2.3 Computations

For the mathematic operations performed on encrypted data, we multiplied/added two data cipher blocks of 32 with different key sizes:

- The simulation results of Multiplication encryption and decryption steps are shown on table12, table13, fig. 6 and fig.7.
- The additive homomorphic results for Paillier are shown in fig.8 for the encryption step and in fig.9 in the decryption step.

Table 6 The process time (ms) of the encryption / decryption of small data (1KB).

Scheme \ Key size	Encryption step				Decryption step			
	256	512	1024	2048	256	512	1024	2048
Paillier	32	74	391	2520	39	90	589	4381
RSA	4	5	43	67	20	22	95	567
ElGamal	18	40	156	1213	30	70	309	786

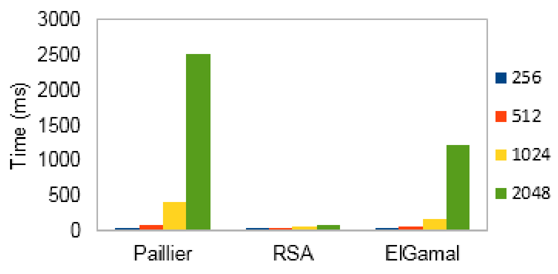

Figure 3. Encryption of small data using various schemes and different size of key

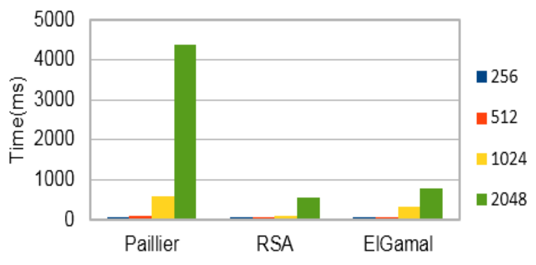

Figure 4. Decryption of small data using various scheme with different size of key

Table 7 The process time (ms) of the encryption / decryption of large data (1MB)

Key \ scheme	Encryption step				Decryption step			
	256	512	1024	2048	256	512	1024	2048
Paillier	17936	63054	236350	2633354	26030	81857	552725	4340671
RSA	711	663	1195	4855	4855	12192	76930	555541
ElGamal	12789	30082	174039	1099392	7156	15992	87118	745873

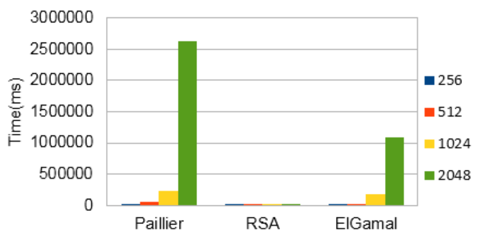

Figure 5. Encryption of large data using partially HE schemes with different sizes of key

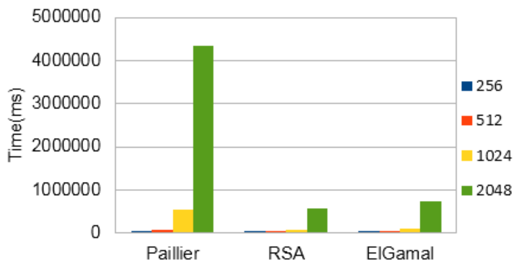

Figure 5. Decryption step of large data using various scheme with different size of key

Table 8 Time of encryption of different data size with a fixed key size (key= 2048).

Scheme \ data	32B	1KB	10KB	100KB	1MB
Paillier	79	2520	22131	243235	2633354
RSA	1	9	67	558	4855
ElGamal	43	1213	11847	109040	1099392

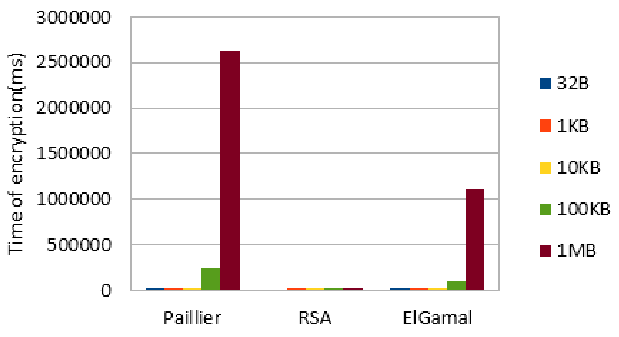

Figure 6. Time of encryption of different data size with a fixed key=2048 bits

Table 9 Decryption time of different data size with a fixed key length (key= 2048 bits).

Scheme \ data	32B	1KB	10KB	100KB	1MB
Paillier	0	4381	42129	416205	4340671
RSA	25	567	5598	56912	555541
ElGamal	19	786	6423	67532	745873

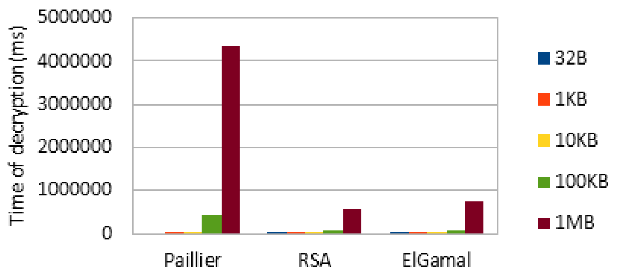

Figure 7. Time of decryption of different data sizes with a fixed key length (key= 2048)

Table 10 Time of multiplication encryption of a data block of 32B using PHE schemes.

Scheme \ Key	256	512	1024	2048
Paillier	0	0	1	4
RSA	1	1	2	2
ElGamal	1	1	1	2

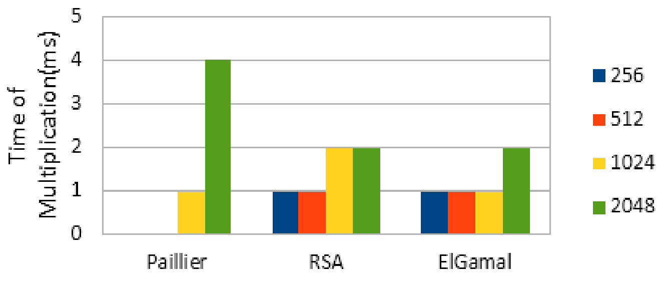

Figure 8. Time of Multiplication Decryption of a data block of 32 bytes using PHE schemes.

Table 11 Time of Multiplication Decryption of a data block of 32 bytes using PHE schemes.

Cryptosystems	256	512	1024	2048
Paillier	2	3	23	145
RSA	1	2	5	25
ElGamal	2	3	6	26

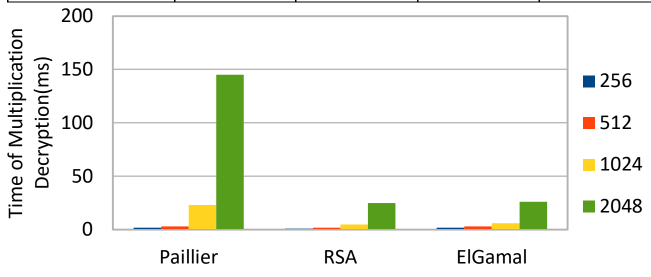

Figure 9. Time of Multiplication Decryption of a data block of 32B using PHE schemes.

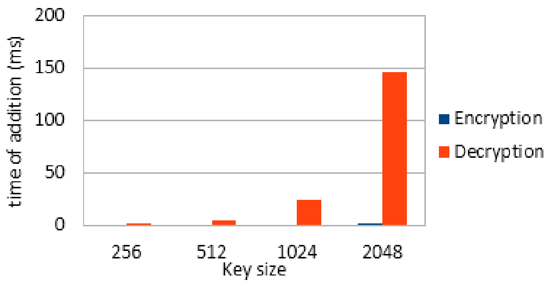

Figure 10. Addition time of encryption/decryption steps of a data block (32B) for Paillier cryptosystem.

As an analytical study of our results, RSA is faster than ElGamal and Pallier algorithms but it's less secure than the previous cryptosystems. In fact, RSA is a deterministic algorithm (we get the same cipher-text when we execute the same plaintext for many time). Contrary to ElGamal and Paillier, they are probabilistic algorithms i.e. using a random variable, you can get different cipher-texts for the same plaintext.

7 Conclusion

In this work, we described the basic concept of cloud computing and how security becomes a major issue of the delay in the widespread adoption of this technology. In addition, we presented the concept of homomorphic encryption and various encryption algorithms: additive, multiplicative and fully homomorphic encryption as well a good solution to secure data stored and computed by a third party.

This paper provides an analytic study of the most popular homomorphic encryptions schemes (RSA, Paillier and ElGamal). RSA is the faster algorithm but Paillier and ElGamal are the most efficient on term of security.

Finally, and as a reminder, our objective is how to store and enable computation on encrypted data using homomorphic encryptions proprieties in a cloud environment.

The next work is to optimize our tests of homomorphic programs by storing encrypted data on swift node and perform both of additions and multiplications operations on the cloud platform.

REFERENCES

[1] R. Chow, Ph. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Msuoka and J. Molina, “Controlling Data in the Cloud : Outsourcing Computation without Outsourcing Control,” pp. 85–90, 2009.

[2] P. V. Parmar, S. B. Padhar, S. N. Patel, N. I. Bhatt and R. H. Jhaveri, “Survey of Various Homomorphic Encryption algorithms and Schemes,” vol. 91, no. 8, p. 8887, 2014.

[3] N. Islam, W. Puech, K. Hayat and R. Brouzet, “Application of Homomorphism to Secure Image Sharing To cite this version :,” OPTICS, vol. 284, no. 19, pp. 4412–4429, 2013.

[4] M. Tebaa, S. El Hajji, and A. El Ghazi, “Homomorphic Encryption Applied to the Cloud Computing Security,” vol. I, pp. 8–11, 2012.

[5] M. Tebaa, K. Zkik, and S. El Hajji, “Hybrid Homomorphic Encryption Method for Protecting the Privacy of Banking Data in the Cloud,” vol. 9, no. 6, pp. 61–70, 2015.

[6] A. Chatterjee and I. Sengupta, “Searching and Sorting of Fully Homomorphic Encrypted Data on Cloud,” pp. 1–14.

[7] “https://github.com/hcrypt-project/libScarab” .

[8] P.Mell and T.Grance, The NIST Definition of Cloud Computing,” vol. 145, no. September, p. 6028, 2011.

[9] A. A. Atayero and O. Feyisetan, “Security Issues in Cloud Computing : The Potentials of Homomorphic Encryption,” vol. 2, no. 10, pp. 546–552, 2011.

[10] V. Vaikuntanathan, “Computing Blindfolded : New Developments in Fully Homomorphic Encryption,” 1978.

[11] S. Goldwasser, “Probabilistic Encryption *,” pp. 270–299, 1984.

[12] P. Paillier, “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,” pp. 223–238, 1999.

[13] M. Seetha and A. K. Koundinya, “Comparative Study and Performance Analysis of Encryption in RSA , ECC and Goldwasser- Micali Cryptosystems,” vol. 3, no. 1, pp. 111–118, 2014.

[14] T. ElGamal, “A public key cryptosystem and a signature scheme based on the discrete logarithm,” springer, 1985.

[15] C. Gentry, “Computing Arbitrary Functions of Encrypted Data.”

[16] C. Gentry, “A FULLY HOMOMORPHIC ENCRYPTION SCHEME,” no. September 2009.

[17] R. Meissen, “A Mathematical Approach to Fully Homomorphic Encryption ”.

[18] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, “Fully Homomorphic Encryption over the Integers,” pp. 1–28, 2010.

[19] Z. Brakerskiand C. Gentry, “Fully Homomorphic Encryption without Bootstrapping.”

[20] Z. Brakerski and V. Vaikuntanathan, “Efficient Fully Homomorphic Encryption from (Standard) LWE,” pp. 97–106, 2011.

[21] K. Lauter, M. Naehrig, and V. Vaikuntanathan, “Can Homomorphic Encryption be Practical ?,” pp. 1–18, 2011.

[22] J. W. Bos, K. Lauter, and M. Naehrig, “Private Predictive Analysis on Encrypted Medical Data.”

[23] E. Magkos, M. Burmester, and V. Chrissikopoulos, “Receipt-freeness in Large-scale Elections without Untappable Channels.”

[24] S. Iftene, “General Secret Sharing Based on the Chinese Remainder Theorem with Applications in,” Electron. Notes Theor. Comput. Sci., vol. 186, no. 3, pp. 67–84, 2007.