A Secure Cloud Computing Architecture Using Homomorphic Encryption

Kamal Benzekki
Laboratory of Computer Networks and Systems
Department of Mathematics and Computer Science
Moulay Ismail University, Faculty of Sciences, Meknes, Morocco

Abdeslam El Fergougui
Laboratory of Computer Networks and Systems
Department of Mathematics and Computer Science
Moulay Ismail University, Faculty of Sciences, Meknes, Morocco

Abdelbaki El Belrhiti El Alaoui
Laboratory of Computer Networks and Systems
Department of Mathematics and Computer Science
Moulay Ismail University, Faculty of Sciences, Meknes, Morocco

***Abstract*—The Purpose of homomorphic encryption is to ensure privacy of data in communication, storage or in use by processes with mechanisms similar to conventional cryptography, but with added capabilities of computing over encrypted data, searching an encrypted data, etc. Homomorphism is a property by which a problem in one algebraic system can be converted to a problem in another algebraic system, be solved and the solution later can also be translated back effectively. Thus, homomorphism makes secure delegation of computation to a third party possible. Many conventional encryption schemes possess either multiplicative or additive homomorphic property and are currently in use for respective applications. Yet, a Fully Homomorphic Encryption (FHE) scheme which could perform any arbitrary computation over encrypted data appeared in 2009 as Gentry's work. In this paper, we propose a multi-cloud architecture of N distributed servers to repartition the data and to nearly allow achieving an FHE.**

Keywords—multi-cloud; privacy; fully homomorphic encryption; distributed System; confidentiality

I. Introduction

Cryptosystems supply mechanisms to ensure data confidentiality and integrity. If the data is always encrypted in the cloud, then control is not lost, and the concerns are removed. When an encryption algorithm does not allow arbitrary computation over encrypted data, the encrypted data must be decrypted before the computation, and the decrypted data is no longer under control.

Cloud computing is infeasible for many business organizations if they need to download sensitive data from the cloud to a trusted computer in order to perform operations, and then send the encrypted results backed to the cloud. Encrypted data has historically been impossible to operate on without first decrypting them. There are some encryption algorithms that allow arbitrary computation on encrypted data. For instance, RSA is a multiplicatively homomorphic encryption algorithm where the decryption of the product of two encrypted data will be the product of the two plain data. However, RSA doesn't allow addition operation nor the combination of multiplications and additions. Later, FHE has appeared [1] to perform unlimited chaining of algebraic operations in the cipherspace, which means that an arbitrary number of additions and multiplications can be applied to encrypted operands. Unfortunately, all implementations of FHE schemes showed that this technique is still much too slow for practical applications.

In this work, we will be focusing on the application of N distributed servers and N cloud systems to homomorphic encryption in order to perform a nearly FHE scheme for the security of data and applications, particularly the possibility to execute the calculations of encrypted confidential data without decrypting them.

The remainder of the paper is organized as follows. Homomorphic encryption and related definitions are introduced in section II. In section III, we discuss the Somewhat Homomorphic Scheme. In section IV, we present some examples of partially homomorphic cryptosystems. Finally in the section V, we propose a secure multi-cloud architecture for processing encrypted data. The perspectives are mentioned in section VI with the conclusion.

II. Toward Homomorphic Encryption

The security requirements for data and algorithms have become very strong in the last few years. Due to the vast growth of technology, a great variety of attacks on digital goods and technical devices are enabled. For storing and reading data securely, there exist several possibilities like secure data encryption. The problem becomes more complex when asking for the possibility to compute (publicly) with encrypted data or to modify functions in such a way that they are still executable while the privacy is ensured. That is where homomorphic cryptosystems can be used.

The notion and idea of fully homomorphic schemes was introduced by Rivest, Adle-man and Dertouzos in [2] shortly after the invention of RSA [3]. They asked for an encryption function that permits encrypted data to be operated on without preliminary decryption of the operands, and they called those schemes privacy homomorphisms. Even in 1978 this was a highly important matter, it is even more important nowadays. While the partially homomorphic properties of schemes like RSA, Paillier, ElGamal, etc. have been acknowledged ever since, it was not before 2009 when a young IBM researcher published the first working fully homomorphic cryptosystem based on lattices.

Among the homomorphic encryption we distinguished according to their operation to assess on raw data. The additive homomorphic encryption (addition of the raw data) is the Pailler [4] and Goldwasser-Micalli[5] cryptosystems and the multiplicative homomorphic encryption (only products on raw data) is the RSA [6] and El Gamal [7] cryptosystems.

A. Definition of a Homomorphic Encryption Scheme

A public-key encryption scheme E=(KeyGen, Enc, Dec) is homomorphic if for all k and all (pk,sk) output from KeyGen(k), it is possible to define groups M, C so that:

- The plaintext space M, and all ciphertexts output by Enc_{pk} are elements of C.
- For any m_1 , $m_2 \in M$ and c_1 , $c_2 \in C$ with $m_1 = Dec_{sk}(c_1)$ and $m_2 = Dec_{sk}(c_2)$ it holds that:

$$Dec_{sk}(c_1 * c_2) = m_1 * m_2$$

Where the group operations $*$ are carried out in C and M, respectively.

In other words, a homomorphic cryptosystem is a PKS with the additional property that there exists an efficient algorithm (Eval) to compute an encryption of the sum or/and the product of two messages given the public key and the encryptions of the messages, but not the messages themselves.

Moreover, a fully homomorphic scheme is able to output a ciphertext that encrypts f (m_1,...,m_t), where f is any desired function, which of course must be efficiently computable. No information about m_1,..., m_t or f (m_1,...,m_t), or any intermediate plaintext values should leak. The inputs, outputs and intermediate values are always encrypted, and therefore useless for an adversary. Before we take a closer look on fully homomorphic encryption schemes, we will need another important notion from information theory.

B. Circuits

Informally speaking, circuits are directed, acyclic graphs where nodes are called gates and edges are called wires. Depending on the nature of the circuit the input values are integers, boolean values, etc. and the corresponding gates are set operations and arithmetic operations or logic gates (AND, OR, NOR, NAND, ...). In order to evaluate a function f, we express f as a circuit and topologically arrange its gates into levels which will be executed sequentially.

Example. Assume the function f outputs the expression:

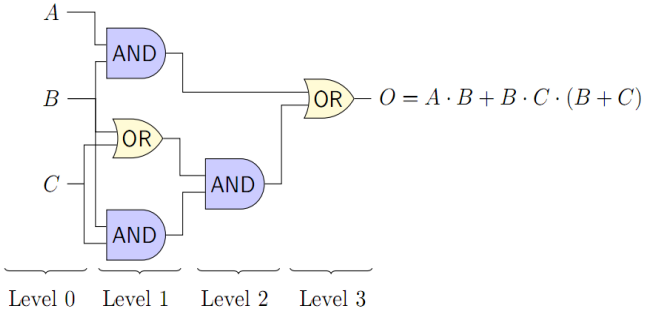

Fig. 1. Example for circuit representation

A·B+B·C·(B+C) on input (A,B,C). Then the following circuit represents the function f, with the logic gates AND and OR.

Two important complexity measures for circuits are size and depth.

The size of a circuit C is the number of its non-input gates. The depth of a circuit C is the length of its longest path, from an input gate to the output gate, of its underlying directed graph.

This yields to another definition of fully homomorphic encryption [8]:

ciphertexts $\Psi = \{c_1, ..., c_t\}$ where $c_i \leftarrow Enc_{pk}(m_i)$, outputs

$$c \leftarrow Eval_{pk}(C, \Psi)$$

under pk.

There is another way to construct fully homomorphic encryption schemes. To understand how this transformation works, we need the following definitions and corollaries.

Definition : A homomorphic encryption scheme E is said to be correct for a family C_E of circuits if for any pair (sk, pk) output by $KeyGen_E(\lambda)$ any circuit $C \in C_E$, any plaintext $m_1,...,m_t$, and any ciphertexts $\Psi = c_1, ...,c_t$

with $c_i \leftarrow Enc_{pk}(m_i)$, it is the case that:

If $c \leftarrow Eval_E(pk, C, \Psi)$, then $Dec_E(sk, c) \rightarrow C(m_1, ...,m_t)$

Except with negligible probability over the random coins in $Eval_E$.

Definition: A homomorphic encryption scheme E is compact, if there is a polynomial f so that, for every value of the security parameter λ, E's decryption algorithm can be expressed as a circuit D_E of size at most f (λ).

A homomorphic encryption scheme E compactly evaluates circuits in C_E if E is compact and also correct for circuits in C_E.

Corollary: A homomorphic encryption scheme E is fully homomorphic if it compactly evaluates all circuits.

This demand is considered to be almost too strong for practical purposes, hence it uses a certain relaxation to include leveled schemes, which only evaluate circuits of depth up to some d, and whose public key length may be poly(d).

Definition: (leveled fully homomorphic). A family of homomorphic encryption schemes $\{E^{(d)} : d \in Z^+\}$ is said leveled fully homomorphic if, for all $d \in Z^+$, it all uses the same decryption circuit, $E^{(d)}$ compactly evaluates all circuits of depth at most d (that use some specified set of gates), and the computational complexity of $E^{(d)}$'s algorithms is polynomial in λ, d, and (in the case of $Eval_E$) the size of the circuit C.

An encryption scheme which supports both addition and multiplication (a fully homomorphic scheme) thereby

preserves the ring structure of the plaintext space and is therefore far more powerful. Using such a scheme makes it possible to let an untrusted party do the computations without ever decrypting the data, and therefore preserving their privacy.

A widely esteemed application of homomorphic encryption schemes is cloud computing. Presently, the need for cloud computing is increasing fast, as the data we are processing and computing on is getting bigger and bigger every day, with the effect that a single person's computation power does not suffice anymore. Hence, it is favorable to use someone else's power without losing the privacy we seek.

Say, Alice wants to store a sensitive file $m \in \{0, 1\}^n$ on Bob's server. So she sends Bob $Enc(m_1)$, ..., $Enc(m_n)$. Assume that the file is a database (a list of people with specific information about them) and Alice wants to find out how many of them are 25 years old. Instead of retrieving the data from Bob, decrypting it and searching for the wanted information, she will ask Bob to do the computations, without him knowing what or who he is computing on.

The answer from Bob comes in form of a ciphertext which only she can decrypt with her secret key.

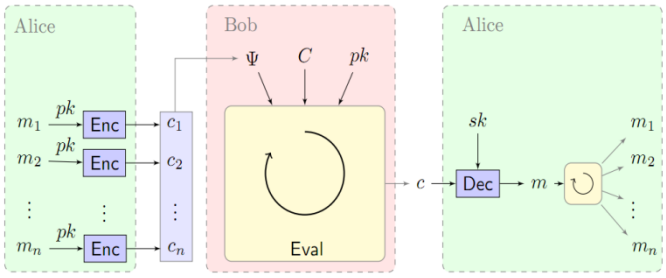

Fig. 2. Diagram of a homomorphic encryption scheme

The benefit of fully homomorphic encryption has long been recognized. The question for constructing such a scheme arose within a year of the development of RSA [2].

For more than 30 years, it was unclear whether fully homomorphic encryption was even achievable. During this period, the best encryption system was the Boneh-Goh-Nissim cryptosystem [9] which supports evaluation of an unlimited number of addition operations but one multiplication at the most.

A common reason why a scheme cannot compute circuits of a certain depth is that after a certain amount of computations too much error accumulates, which causes the decryption to obtain a wrong value. The decryption usually is able to handle small amounts of error within a certain range and bootstrappable encryption enables "refreshing" after some time. The basic idea of "refreshing" is to encrypt under a first key. Compute until right before the error grows too large. Encrypt under a second key. Compute the decryption circuit, which since it stopped before the error grew too large, gives the correct value encrypted under the second key. The first key is no longer required. Continue computation under the second key, and repeat with a new key as often as needed. When the computation has finished, decrypting with the last used key gives the original plaintext.

Gentry's method can be broken down into three major steps:

Step 1: Constructing an encryption scheme using ideal lattices that is somewhat homomorphic, which means it is limited to evaluating low-degree polynomials over encrypted data. This scheme is very similar to the Goldreich-Goldwasser-Halevi scheme published in 1997 [10] which is based on lattice problems as well.

Step 2: "Squashing" the decryption circuit of the original somewhat homomorphic scheme to make it bootstrappable.

Step 3: Bootstrapping the slightly augmented original scheme of step 2 to yield the fully homomorphic encryption scheme. This will be done with a "refreshing" procedure.

The innovative idea of Gentry's method of creating a fully homomorphic scheme out of a somewhat homomorphic scheme is the method of squashing and boot-strapping. Mathematically the most appealing step is the first step.

III. The Somewhat Homomorphic Scheme

The aim of this somewhat homomorphic scheme (SHS) is to construct an encryption scheme that is "almost" bootstrappable with respect to a universal set of gates. The first step is to design a SHE scheme which is a scheme that supports *some* computations over encrypted data. Gentry then showed that if you can manage to design a SHE scheme that supports the evaluation of its own decryption algorithm (and a little more), then there is a general technique to transform the SHE scheme into a FHE scheme. A SHE that can evaluate its own decryption algorithm homomorphically is called *bootstrappable* and the technique that transforms a bootstrappable SHE scheme into a FHE scheme is called *bootstrapping*.

Bootstrapping. So how does bootstrapping work and why is bootstrappability such a useful property? To understand this, you first have to know how the currently-known SHE schemes work. Roughly speaking, the ciphertexts of all these schemes have noise inside of them and unfortunately this noise gets larger as more and more homomorphic operations are performed. At some point, there is so much noise that the encryptions become useless (i.e., they do not decrypt correctly). This is the main limitation of SHE schemes and this is the reason that they can only perform a restricted set of computations. Bootstrapping allows us to control this noise.

The idea is to take a ciphertext with a lot of noise in it and an encryption of the secret key and to homomorphically decrypt the ciphertext. Note that this can only work if the SHE scheme has enough homomorphic capacity to evaluate its own decryption algorithm which is why we need the SHE scheme to be bootstrappable. This homomorphically computed decryption will result in a new encryption of the message but without the noise (or at least with less noise than before). More concretely, say we have two ciphertexts:

$c_1 = E_{pk}(m_1)$ *and* $c_2 = E_{pk}(m_2)$

with noise n_1 and n_2, respectively. We can multiply these encryptions using the homomorphic property of the SHE scheme to get an encryption:

$c_3 = E_{pk}(m_1 \times m_2)$

of $m_1 \times m_2$ under key *pk* ,but c_3 will now have noise $n_1 \times n_2$. The idea behind bootstrapping is to get rid of this noise as follows. First, we encrypt c_3 and *sk* under *pk* .This results in two new ciphertexts

$c_4 = E_{pk}(c_3) = (E_{pk}(m_1 \times m_2))$ *and* $c_5 = E_{pk(sk)}$

Given c_4 and c_5, we now *homomorphically* decrypt c_4 using c_5. In other words, we compute the following operation over c_4 and c_5: "decrypt $c_3 = E_{pk}(m_1 \times m_2)$ using *sk*". This is allowed since the scheme has enough homomorphic capacity to evaluate its own decryption algorithm.

By using this technique throughout a computation whenever the ciphertexts get too noisy, we can remove the main limitation of the SHE scheme and turn it into a FHE scheme.

It turns out that constructing a bootstrappable SHE scheme is difficult. To do this, Gentry had to build his scheme using sophisticated techniques [1] so a lot of the recent work in FHE has tried to figure out how to design simpler bootstrappable SHE schemes.

IV. Partially Homomorphic Cryptosystems

A. RSA-A Multiplicatively Homomorphic Scheme

In 1978, Rivest, Shamir, and Adleman published their public-key cryptosystem which only uses elementary ideas from number theory, in their paper "A Method for Obtaining Digital signatures and Public-Key Cryptosystems" [3]. It was one of the first homomorphic cryptosystem. The RSA cryptosystem is the most widely used public-key cryptosystem. It may be used to provide both secrecy and digital signatures and its security is based on the intractability of the integer factorization problem.

Key Generation: KeyGen(p,q)	
Input: $p,q \in \mathbb{P}$	
Compute	$n = p \cdot q$
	$\varphi(n) = (p-1)(q-1)$
Choose e such that	$\gcd(e, \varphi(n)) = 1$
Determine d such that	$e \cdot d \equiv 1 \bmod \varphi(n)$
Output: (pk, sk)	
public key: $pk = (e, n)$	
secret key: $sk = (d)$	

Encryption: Enc(m, pk)	
Input: $m \in \mathbb{Z}_n$	
Compute	$c = m^e \bmod n$
Output: $c \in \mathbb{Z}_n$	

Decryption: Dec(c, sk)	
Input: $c \in \mathbb{Z}_n$	
Compute	$m = c^d \bmod n$
Output: $m \in \mathbb{Z}_n$	

Fig. 3. RSA Algorithm

The **encryption** algorithm takes as input a message m from the plaintext space Z_n and computes the according ciphertext $c = m^e \bmod n$. This integer $c \in Z_n$ cannot be traced back to the original message without the knowledge of p and q, which will be proved later in this section.

Decryption takes as input the ciphertext c and the secret key (d, n) and computes $m = c^d \bmod n$. Since d is the inverse of e in Z_n this is indeed the original message.

The three steps (key generation, encryption and decryption) can be found in the following table.

B. Paillier - An Additively Homomorphic Scheme

Pascal Paillier introduced his cryptosystem in 1999 published paper "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes" [11]. The proposed technique is based on composite residuosity classes, whose computation is believed to be computationally difficult. It is a probabilistic asymmetric algorithm for public key cryptography and inherits additive homomorphic properties.

The encryption procedure takes as input a message $m \in Z_n$ and randomly chooses an integer r in Z_* , this random number is used to fulfill the probabilistic algorithm's n property, that one plaintext can have many ciphertexts. It is later shown that this random variable does not impede the correct decryption, but has the effect of changing the corresponding ciphertext.

The three steps (key generation, encryption and decryption) can be found in the following table:

Key Generation: KeyGen(p,q)	
Input: $p,q \in \mathbb{P}$	
Compute	$n = pq$
Choose $g \in \mathbb{Z}_{n^2}^*$ such that	$\gcd(L(g^\lambda \bmod n^2), n) = 1$ with $L(u) = \frac{u-1}{n}$
Output: (pk, sk)	
public key: $pk = (n, g)$	
secret key: $sk = (p, q)$	

Encryption: Enc(m, pk)	
Input: $m \in \mathbb{Z}_n$	
Choose	$r \in \mathbb{Z}_n^*$
Compute	$c = g^m \cdot r^n \bmod n^2$
Output: $c \in \mathbb{Z}_{n^2}$	

Decryption: Dec(c, sk)	
Input: $c \in \mathbb{Z}_{n^2}$	
Compute	$m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n$
Output: $m \in \mathbb{Z}_n$	

Fig. 4. Paillier Algorithm

V. Our Architecture

The fully homomorphic encryption schemes [1] are very time consuming. Considering the evaluation of one gate demanding a refresh, the run-time will be significant as well as the processing of security parameters.

A suggestion of a nearlly FHE scheme based architecture for enabling the evaluation of any function and processing

encrypted data is illustrated in Figure 6. In our proposed architecture, the service provider repartitions the processing among the servers to fasten the evaluation process of any function.

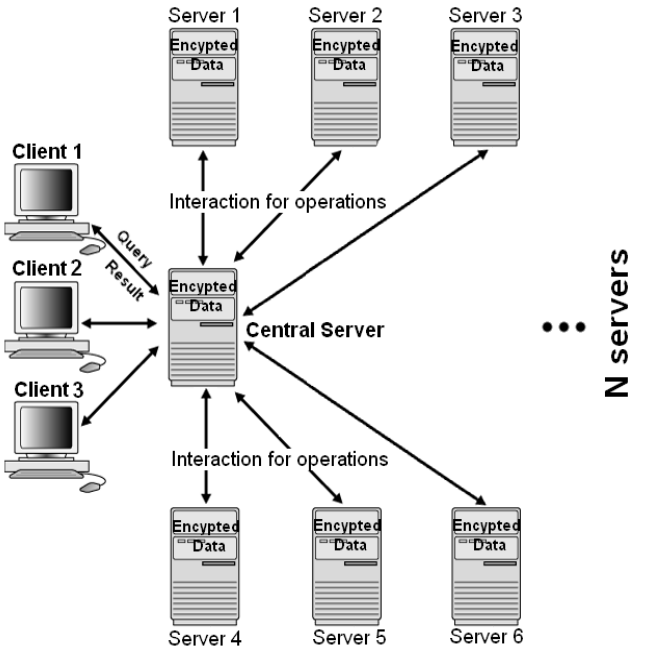

Fig. 5. An architecture of distributed servers for processing encrypted data

In this system, we provide a high leveled architectural scheme through the usage of multiple servers in the computation. This computational system will nearly allow achieving a FHE, and thus a large number of operations including multiplications and additions can be performed. For instance, in Fig. 5, Client 1 requests the results of a given function, let's say $f(x)=ax^2+bx+c$. In this case the function elements are encrypted and divided into several chunks depending on the number of operations (Multiplication and addition), and will be processed separately on N different servers, equivalent to the number of addition operations Finally the result is sent back to a Central Server in order to be forwarded to Client 1 and then decrypted.

The benefit is that no longer chipertext after encryption unlike the classical method. The keys are easily handled and more security is maintained since is it impossible to read relevant information in distributed systems. In the cloud the N servers consists of hypervisors hosting multiple virtual machines which help improving the response time and augment the number of the involved computational entities in the distributed system.

In this suggestion, we analyze the added value of the distributed systems in processing operations requested by clients. The scheme of homomorphic encryption is dispatched within the servers and this can be practical and help improving the security of the cloud in terms of confidentiality of data and performance.

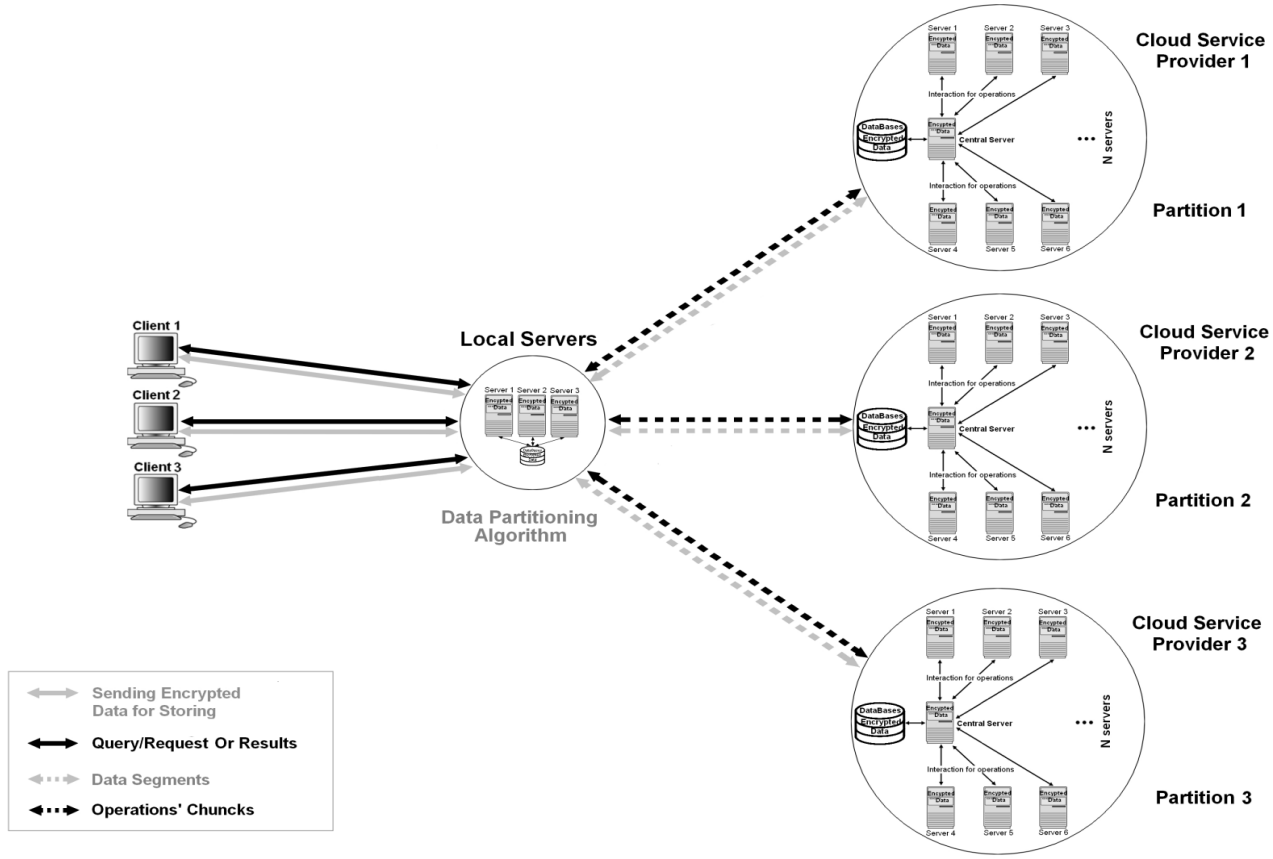

Fig. 6. The proposed architecture to secure data using homomorphic encryption

Another concern that should be considered in our architecture is the confidentiality of the processed data over the distributed systems, which is the main concern of most organization when using third-party hosting. Our approach regarding this matter is the split the stored data among multiple Cloud service providers to decrease the risk of data breaches and increase the parallel processing as well as the number of the servers involved in performing homomorphic encryption. Partitioning and outsourcing the data, applications onto different cloud infrastructures has the advantage of making them ambiguous for third-parties and adversaries, and thus this helps enhancing the confidentiality as well as the privacy.

As the stored encrypted data is repartitioned among a Multi-Cloud Architecture belonging to different Cloud Service Providers (See Fig. 6), Client 1 can perform operations on them and transparently retrieve the intended results. The data is segmented through a Data Partitioning Algorithm (DPA) which allows partitioning, collecting and reconstructing the data. The main operations are chunked into subsets to be handled by the N Clouds/N Servers. The combination of N Clouds and homomorphic encryption using N servers provides an enhanced security strategy which is a safe approach to prevent any potential data breaches even if the data have been already encrypted.

Choosing a trusted CSP requires a Service Level Agreement (SLA), contract negotiation and risk assessments. In most cases it may be logical to believe that a CSP to be trustworthy and handling the clients' sensitive data and applications in a responsible manner.

VI. CONLUSION

As Gentry proposed his construction and blueprint in 2009, there has been a huge effort to make FHE more practical. While a lot of progress has been made, unfortunately, we are still some way from truly practical FHE.

Most FHE schemes are based on Gentry's blueprint which consists of first constructing a SHE and then using Gentry's bootstrapping technique to turn it into a FHE scheme. It turns out that bootstrapping is a major bottleneck and that SHE is actually reasonably efficient. So, if we care about practical applications, then it may be worthwhile to explore what exactly we can do with SHE instead.

Distributed systems and multi-could architectures could bring lots of benefits to the application of homomorphic encryption and making it more practical in the case of the security of data and applications.

In a future work, we will focus on the implementation of our proposal and conduct security and performance tests in order to show its practicability.

REFERENCES

[1] C. Gentry, "A fully homomorphic encryption scheme," Doctoral dissertation, Stanford University, 2009.

[2] R. Rivest, L. Adleman, and M. Dertouzos, "On data banks and privacy homomorphisms," In Foundations of Secure Computation, pages 169-180, 1978.

[3] R. L. 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.

[4] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," In 18th Annual Eurocrypt Conference (EUROCRYPT'99) Prague, Czech Republic , volume 1592, 1999.

[5] J. Bringe and al., "An Application of the Goldwasser-Micali Cryptosystem to Biometric Authentication", Springer-Verlag, 2007.

[6] 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.

[7] T. ElGamal, "A public key cryptosystem and a signature sche based on discrete logarithms," IEEE Transactions on Information Theory, 469-472, 1985.

[8] C. Gentry, "Fully homomorphic encryption using ideal lattices," InSTOC, Vol. 9, pp. 169-178, 2009.

[9] D. Boneh, E. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," In Proceedings of Theory of Cryptography (TCC) '05, LNCS 3378, pages 325-341, 2005.

[10] O. Goldreich, S. Goldwasser, and S. Halevi, "Public-key cryptosystems from lattice reduction problems," In Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology, pages 112-131. Springer-Verlag, 1997.

[11] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," Advances in Cryptology Eurocrypt, 1592:223-238, 1999.

[12] S. Goluch, "The development of homomorphic cryptography: From RSA to Gentry's privacy homomorphism" Doctoral dissertation, Vienna university of Technology, 2010.