IJARCCE

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

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 7, July 2016

# Homomorphic Encryption Algorithms for Securing Data against Untrusted Cloud

**Gaurav Somani[1], Sourabh Garg[1]**

Department of Mathematical Sciences, Indian Institute of Technology BHU, Varanasi, India [1]

**Abstract:** Cloud Computing has revolutionized businesses and individuals needs by outsourcing computations and storage providing significant cost-effectiveness and flexibility. With its rapid development, the security and privacy are the primary concerns. The homomorphic encryption techniques have provided a good potential in mitigating these issues in recent years. It allows the cloud to perform blind-computations on the encrypted inputs uploaded by the user without prior decryption and return the encrypted results, which can only be decrypted by the user who initiates the proceedings. Thus, clients can rely on the cloud services without compromising the privacy. Some classical, as well as recent practical homomorphic schemes and their algorithms, are discussed in this paper. The main focus is to provide the reader good background knowledge on various schemes for applications in context to practical implementations based on their scope, performance, security and complexity factors.

**Keywords:** Homomorphic encryption, Cloud Security, FHE schemes, LWE.

## I. INTRODUCTION

Cloud computing is termed as “platform for future’s computation”. The global computing infrastructure is rapidly moving towards cloud based architecture so as to expand their business. It seems to give a number of benefits like flexibility, efficiency, scalability, integration, and capital reduction. But security is one of the main challenges that hinder the growth of cloud computing. Researchers and Service provider are trying out different techniques to gain the trust from clients [1]. Some malicious elements always try to peek inside its architecture. Due to this, a client considers cloud platform as a threat and wants to keep his data confidential even from a service provider. A possible solution is to perform encryption such that, computation can be directly applied to encrypted data by cloud services. Homomorphic encryption provides a great potential for the privacy and encrypted computations in the cloud. This paper is aimed at providing the reader background knowledge about various homomorphic schemes for their application in the cloud ecosystem.

In Section II, we briefly introduce cloud computing, its basic services, and the security concerns related to it. Section III presents the basics of homomorphic encryption and its properties. Later in Section IV and V, we present a comparative literature survey on the limited homomorphic encryption algorithms and FHE schemes with their limitations respectively.

## II. CLOUD COMPUTING AND SECURITY

Before the advent of Cloud terminology, the computing was done using “outsourcing” and “server hosting”. However, their performances were low in terms of cost and maintenance of hardware and computing resources. With the recent developments in Virtualization technology paved the way for efficient, on-demand, cost-effective services in the cloud [1].

In a typical cloud architecture, the resources are generally owned by a service provider with remote access to users via the internet or a private network. The resources are powered by distributed and parallel computing incorporating to each other's IT infrastructures (hardware, platform, software). Thus providing quick, convenient data storage and net computing service with reliance on the Internet.

There are three standard service models as per NIST for cloud computing [2]:

- Software as a Service (SaaS): This is software distribution model. The applications are hosted by the service providers and offered as a subscription to the users via the Internet. It is accessible via various interfaces and provider does not manage or control the underlying cloud infrastructure.
- Platform as a Service (PaaS): This refers to the delivery of operating systems and relevant services over the Internet without downloads or installation. The user can deploy their applications using these services and tools by the provider. The user has no manage or control over the underlying cloud infrastructure.
- Infrastructure as a Service (IaaS): This involves outsourcing the physical infrastructures used for storage, hardware, support operations and network components accessible over a network. The user has no manage or control over the underlying cloud infrastructure but has some control over OS and storage.

Although Cloud computing has become a matured service model, but it still has some significant barriers to adoption. One of the main concerns are issues related to security and privacy. The SaaS, PaaS, and IaaS gradually release the responsibility of security control to the cloud users in order. The SaaS model relies on the cloud provider for all security functions while the PaaS model on the provider to for data integrity and availability, but loads the user for confidentiality and privacy control. These leads to two sensitive states that are a concern in the operational context of cloud computing:

- Transmission phase – transfer of the sensitive data to and from the cloud servers and the client systems.
- Static phase – the storage and operations performed in cloud servers on the client's personal data.

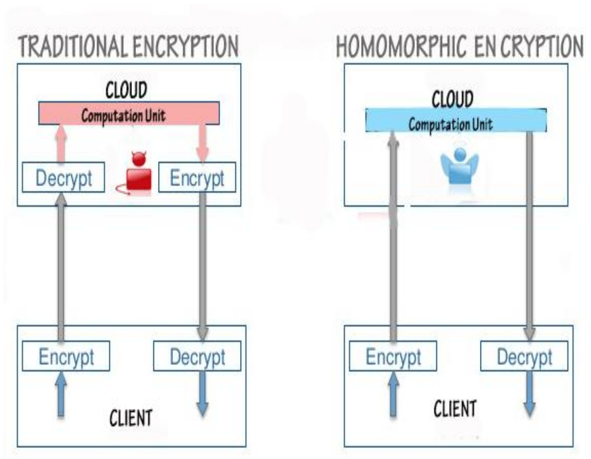

Fig 1: Secure Cloud computing using Homomorphic scheme

The channel of data transmission is well secured with cryptographic key exchanges. The cloud user can encrypt the data and store it in the cloud, however, both the user and the cloud would not be able to operate any computations on data before decrypting it, moreover the cloud provider will not be able to respond to users' queries before decrypting the data first. Considering cloud can be untrustworthy, the client would not consider it reliable over accessing their personal data.

The power that homomorphic encryption brings to preserve privacy in cloud computing is its ability to perform computations on the encrypted data without the need for decrypting it and users to retrieve results that can only be decrypted using his secret key

## III. HOMOMORPHIC ENCRYPTION

This section presents an introduction to the homomorphic encryption, its terminologies, and notations.

### A. Background on Encryption

Let a message $m \in M$ where M denotes the message space and σ be the security parameter. A public-key encryption scheme on M is based on a set of three functions (K, E, D) of probabilistic, polynomial time satisfying the following functionalities:

- Key Generation: Based on security parameter the algorithm K outputs a key pair $k = (pk, sk)$, where pk denotes the encryption key/ public key and sk being the decryption key /secret key.
- Encryption: The encryption algorithm E inputs a message $m \in M$ and a public key pk, outputs a ciphertext $E(m, pk) = c \in C$, where C denotes the ciphertext space.
- Decryption: The decryption algorithm D inputs a ciphertext $c \in C$ and a secret key sk, outputs the message $D(c, sk) = m \in M$.

### B. Homomorphic Property

In mathematics, homomorphic describes the transformation of one data set into another while preserving relationships between elements in both sets. Hence, performing encryption homomorphically will yield equivalent results whether they are performed in encrypted or decrypted data.

More precisely, an encryption scheme is said to be homomorphic for some operations $\circ \in M$ acting in message space M (such as addition) if there are corresponding operations $\diamond \in C$ acting in cipher text space satisfying the property [5]:

$Dec(sk, Enc(pk,m1) \diamond Enc(pk,m2)) = m1 \circ m2$

If the operation $\circ \in M$ is additive {+}, then the scheme is said to be additively homomorphic and multiplicative homomorphic if $\circ \in M$ is multiplicative {*}.

The evaluation algorithm Eval inputs ciphertexts c1, c2 outputs c3 = Eval(c1, c2) such that D(c3, sk) = m3, where $m3 = m1 \circ m2$ , the desired message according to operation applied.

## IV. LIMITED HOMOMORPHIC ENCRYPTION

In this section we look at classical homomorphic encryption schemes limited to partial operations namely, either additive or multiplicative such as RSA, Pallier, El Gamal schemes and discuss their properties and algorithms.

### A. RSA Cryptosystem

Before RSA was developed, the symmetric ciphers were used to encrypt, send and decrypt a message using a single key. It was possible that intruder may attack, it requires the users to send the key hand to hand, which is not possible always. Solution to this problem comes when RSA was invented.

RSA was the first public–key cryptosystem which uses two keys, the first public key which is known to everyone is used to encrypt a message, and second a secret key, which only target user holds and with its help only, he can decrypt the encrypted message. Basic RSA is a multiplicatively homomorphic encryption scheme and is deterministic in the sense that under a fixed public key, a plaintext m is always encrypted to the same ciphertext E(m). It is widely used to transmit data securely. It was introduced by Rivest, Shamir, and Adleman [6].

Algorithm:
(1). Key Generation:
- Generate two large distinct prime numbers, p, and q (should be similar in magnitude).
- Compute n=p*q, where n is the key(both) length
- Compute Euler's totient function on n,

$\varphi(n) = \varphi(p)\varphi(q) = (p - 1)(q - 1) = n - (p + q - 1)$
- Choose a number e ,s.t e must lie between $(1, \varphi(n))$,& i and $\varphi(n)$ are coprime; i.e. $\gcd(e, \varphi(n)) = 1$.
- Compute modular multiplicative inverse of e mod$\varphi(n)$ i.e. $e^{-1} \pmod{\varphi(n)}$ and name it d, d.e=1(mod $\varphi(n)$,equivalent to $d.e = k.\varphi(n) + 1$.
- Now pk = e and sk = d with n as the common modulus.

(2). Encryption:
- Let M be the message and (e,n) be the public key
- First, convert M into an integer m s.t. m ϵ [0,n) & gcd(m,n)=1.
- Compute ciphertext , $E(m)=m^e \pmod n$

(3).Multiplicative Homomorphic operation :
- Let E(m1) and E(m2) be the two ciphertext , $E(m1)= m1^e (\text{mod } n) E(m2)= m2^e (\text{mod } n)$
- $E(m1).E(m2) = m1^e.\ m2^e$ ( mod = $(m1 \cdot m2)^e$ mod n =c3, where c3 is the encryption of (m1.m2) .

(4).Decryption:
- Cipher text c,when raised to the power d(private key) with (mod n) gives the required message
- $E(m)^d=(m^{e.d})\text{mod } n =(m^{k.\varphi(n)+1})\text{mod } n = ((m^{\varphi(n)})^k \cdot m)\text{mod } n \equiv 1^k.m \pmod n = m$ (using Euler and Fermat theorem, for any integer M coprime to n, we have , $M^{\varphi(n)} \equiv 1 \pmod n$))

Limitation**:**
It is not semantically secure. It is a relatively slow algorithm and even when the public key is very large enough, someone with knowledge of prime numbers can easily retrieve the actual message from the encrypted message.

B. Elgamal (Multiplicative Homomorphic encryption)
It is a partially homomorphic public key encryption algorithm as it supports only multiplicative homomorphic property and is based on the Diffe-Hellman key exchange. It was designed by Taher Elgamal [7].

Algorithm:
(1). Key generation:
- Generate a large prime p, s.t (p-1) is divisible by another large prime q.
- Compute a generator g of the multiplicative group of order q in GF(p) via (for some random r) $g \equiv r^{(p-1)/q} \text{mod } p$ until $g \neq 1$.
- Select a natural number 'a' in range of q
- Compute $h \equiv g^a \pmod p$
- Now, pk=(p,g,h) and sk = a

(2). Encryption:
- Generate a random number k $(1 < k < q - 1)$ with $\gcd(k, p - 1) = 1$
- Compute $r \equiv g^k \text{ mod } p$ and $s \equiv h^k \cdot m \pmod p$ $(0 \leq m \leq p - 1)$
- Ciphertext : c= E(m)=(r,s)

(3). Homomorphic operation:
- Let E(m1) and E(m2) be two cipher texts
- $E(m1).E(m2)=(g^{r1}, m1.h^{r1}) (g^{r2}, m2.h^{r2})= (g^{r1+r2}, (m1.m2) h^{r1+r2}) = E(m1.m2)$

(4). Decryption:
- Compute: $m \equiv s \cdot r^{-a} \text{mod } p$ $(r^{-a} \equiv g^{-ka} \equiv h^{-k} \text{ mod } p)$

Limitation:
This homomorphism is multiplicative whereas practical such as E-cash and e-voting application requires addition also. One solution to this to Modify ElGamal : Put the plaintext in the exponent, $E(m)=( r \equiv g^k \pmod p)$ $,s \equiv h^k \cdot g^m \pmod p))$ . This modification introduces the discrete logarithm problem $d_k = g^m$ into the decryption. For large enough texts, this becomes impractical.

C. Paillier(Additive Homomorphic encryption)
It is a probabilistic, additive homomorphic encryption algorithm and invented by Pascal Paillier in 1999. It is based on decisional composite residuosity assumption [8].

Algorithm:
(1). Key generation:
- Choose two large prime number p and q
- Compute n= p.q and y= lcm(p-1,q-1)
- Choose random number g where $g \in Z_n$
- Check the existence of modular multiplicative inverse : $u=( L(g^y \text{ mod } n^2))^{-1} \text{ mod } n$
- pk = (n,g) and sk = (y,u)

(2). Encryption:
- Choose an integer "r"
- Compute $c = E(m)= g^m \cdot r^n (\text{mod } n^2)$

(3).Homomorphic operation:
- $E(m1).E(m2)=( g^{m1} \cdot r1^n )( g^{m2} \cdot r2^n)\text{mod } n^2 = g^{(m1+m2)} (r1.r2)^n \text{ mod } n^2 =E(m1+m2)$

(4). Decryption:
- Compute: $m = L(c^y \text{ mod } n\ 2)/ L(g^y \text{ mod } n2) \text{ mod } n = L(c^y \text{ mod } n\ 2).u \text{ mod } n$

Limitation
It supports only addition cryptographically, and for multiplication, it requires to use two-party computation which makes it dependent and even are limited in number.

## V. FULLY HOMOMORPHIC ENCRYPTION

Classical homomorphic encryption schemes described in the previous section supports some operations on ciphertext (e.g. addition, multiplication, quadratic function. etc.), whereas fully homomorphic encryption support arbitrary computation. As it never requires to decrypt its input, it creates practical opportunity to outsource private computation of client's sensitive data on a untrusted third party.

### A. Gentry's Cryptosystem

Gentry's scheme supports both addition and multiplication operations on ciphertexts. Using these elementary operations, any function can be computed. It is based on lattices.

It is limited to some extent, as noise increase exponentially and when it reaches a threshold, resulting ciphertext is indecipherable. Gentry introduced bootstrappable somewhat homomorphic encryption which refreshes the ciphertext repeatedly, resulting in a new ciphertext with lower noise [9].

Algorithm**:**

(1).Key Generation:

- Takes as input a security parameter $\lambda$ and a positive integer d as the depth of circuit . For length, $l = l(\lambda)$ .
- Sets $(sk_i, pk_i) \leftarrow KeyGen(\lambda)$ for $i \in [0, d]$
- $sk_{ij} \leftarrow EncryptE(pk_{i-1}, sk_{ij})$ for $i \in [1, d]$, $j \in [1, l`]$ where $sk_{i1}, \ldots, sk_{il`}$ is the bit representation of $sk_i$ .
- $sk^{(d)} = sk0$ and $pk^{(d)} \leftarrow (<pk_i>,<sk_{ij}>)$.

(2).Encryption:

- Input a public key $pk_d$ and a plaintext $\pi \in P$.
- Ciphertext $\psi \leftarrow Encrypt(pk_d, \pi)$.

(3).Decryption:

- Input a secret key $sk_d$ and a ciphertext $\psi$ (which should be an encryption under $pk_0$ ).
- Outputs Decrypt $(sk0, \psi)$.

(4).Evaluation:

- Takes as input a public key $pk_\varphi$ , a circuit $C_\varphi$ of depth at most $\varphi$ gates , and a tuple of input ciphertexts $\Psi_\varphi$ (where each input ciphertext should be under $pk_\varphi$ ).
- Check if each wire in $C_\varphi$ connects gate at consecutive levels; if not add identity gates .
- If $\varphi=0$ , it outputs , $\psi^0$ and terminates,

Else

- Sets $(C^\dagger_{\varphi-1}, \psi^\dagger_{\varphi-1}) \leftarrow Augment_\varphi(pk_\varphi, C_\varphi, \psi_\varphi)$
- Sets $(C_{\varphi-1}, \psi_{\varphi-1}) \leftarrow Reduce_{\varphi-1}(pk_\varphi, C_\varphi, \psi_\varphi)$
- $Evaluate_{\varphi-1}(pk_{\varphi-1}, C_{\varphi-1}, \psi_{\varphi-1})$

(5). $Augment_\varphi$ :

- Takes as input a public key $pk_\varphi$ , a circuit $C_\varphi$ of depth at most $\varphi$ gates , and a tuple of input ciphertexts $\Psi_\varphi$ (where each input ciphertext should be under $pk_\varphi$ ).
- Let $\psi^\dagger_{\varphi-1}$ be the tuple of cipher texts formed by replacing each input ciphertext $\psi \in \psi_\varphi$ by the tuple $(sk_\varphi, \psi_\varphi)$, where $\psi_j \leftarrow Encrypt((pk_{\varphi-1}), \psi_j)$ and the $\psi_j$ 's from the properly-formatted representation of $\psi$ as elements of P. It outputs $(C^\dagger_{\varphi-1}, \psi^\dagger_{\varphi-1})$.

(6). $Reduce_\varphi$ :

- Takes as input a public key $pk_\varphi$ , a circuit $C_\varphi$ of depth at most $\varphi$ gates , and a tuple of input ciphertexts $\Psi_\varphi$ .
- sets $C_\varphi$ to be the sub-circuit of $C^\dagger_\varphi$ consisting of the first $\varphi$ levels
- sets $\psi_\varphi$ to be the induced input ciphertext of $C_\varphi$ .

Limitation:

The scheme is impractical, as increasing key size results in a large jump in the graph of ciphertext size and thus, the computation time.

### B. DGHV (Somewhat) Homomorphic Scheme

Van Dijk, Gentry, Halevi and Vaikuntanathan's (DGHV) scheme[10] described a conceptually simpler "somewhat homomorphic" using only modular arithmetic over the integers. The security of the scheme is reduced to finding an approximate integer greatest common divisors (approximate GCD) problem. This scheme can be applied to Gentry's "blueprint" for constructing fully homomorphic schemes out of certain somewhat homomorphic schemes specifically, by "squashing the decryption circuit" and applying Gentry's bootstrapping theorem. Depending upon the security parameter $\lambda$, the scheme uses polynomials in $\lambda$ as four parameters:

$\gamma$ - bit-length of the integers in the public key pk,
$\eta$ - bit-length of the secret key,
$\rho$ - bit-length of the noise in KeyGen,
$\rho'$- bit-length of the noise in Encrypt (secondary noise parameter),
$\tau$ - number of integers in the public key.
The parameters must follow the following constraints [10]:

- $\rho = \omega(\log \lambda)$, to protect against brute-force attacks on the noise;
- $\eta \geq \rho \cdot \Theta(\lambda \log2 \lambda)$, in order to support homomorphism for deep enough circuits to evaluate the "squashed decryption circuit";
- $\gamma = \omega(\eta\ 2 \log \lambda)$, to thwart various lattice-based attacks on the underlying approximate-gcd problem;
- $\tau \geq \gamma + \omega(\log \lambda)$, (see Lemma 4.3 in [DGHV10]);
- $\rho' = \rho + \omega(\log \lambda)$, used as secondary noise parameter.

A suitable set of parameters is $\rho = \lambda$, $\rho' = 2\lambda$, $\eta = O(\lambda^2)$, $\gamma = O(\lambda^5)$ and $\tau = \gamma + \lambda$.

Definition (approximate GCD): Given polynomially many samples from $D\gamma,\rho(p)$ for a randomly chosen $\eta$-bit odd integer p, output p.
$D\gamma,\rho(p) = \{$choose $q \leftarrow Z \cap [0, 2^\gamma/p)$, $r \leftarrow Z \cap (-2^\rho, 2^\rho)$ : output $x = pq + r \}$.

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

Vol. 5, Issue 7, July 2016

Algorithm:
(1). KeyGen($\lambda$): The secret key is a random odd $\eta$-bit integer:

$sk := p \in (2Z + 1) \cap [2^{\eta-1}, 2^{\eta})$.

For the public key,

- sample $x_i \in D\gamma,\rho(p)$ for $i = 0, \ldots, \tau$.
- Relabel $x_i$ so that $x_0$ is the largest.
- Restart unless $x_0$ is odd and $r_p(x_0)$ is even. The public key is $pk = \{x_0, x_1, \ldots, x_\tau\}$.

(2). Encrypt (pk, $m \in \{0, 1\}$) = c.

- Choose a random subset $S \subseteq \{1, 2, \ldots, \tau\}$,
- A random integer r in $Z \cap (-2^{\rho'}, 2^{\rho'})$,
- Compute ciphertext $c = (m + 2r_0 + 2\sum_{i\in S} x_i) \bmod x$.

(3). Evaluate(pk, C, c1, . . . , ct) = c*:

- Given the (binary) circuit CE with t inputs, and t ciphertexts $c_i$, apply the addition and multiplication operations on the ciphertext according to the circuit CE, and return the resulting integer.

(4). Decrypt(sk, c) = m
Output $m' = (c \bmod p) \bmod 2$.

Limitation:
This was fairly an easy scheme as it used only elementary modular arithmetic rather than lattice based encryption to generate FHE. Some open problems posed by the scheme were related to compressing the squashing and improving the efficiency while preserving the hardness of the approximate-gcd problem. The size of the public key pk is around $2^{60}$ bits which was its main drawback.

C. BGV (Leveled) Fully Homomorphic Encryption
The resulting performance out of Gentry's bootstrapping were unsatisfactory and all the existing FHE schemes gave exponential noise growth with depth of the circuit requiring almost $\Omega(\lambda^{3.5})$ computation per gate. Brakerski, Gentry, and Vaikuntanathan introduced the notion of leveled fully homomorphic encryption schemes [11]. The scheme constructs a much effective approach for lattice-based ciphertext using FHE schemes based on LWE (learning with errors) or ring LWE (RLWE) problems that have $2^{\lambda}$ security reducing the per-gate computation to $O(\lambda. L^3)$, $O(\lambda^2)$ and $O(\lambda)$ respectively as described in table 1.

Typically, BGV proved to be the first scheme practical in real-life applications. It describes several improvements to Gentry's original work namely, modulus switching and re-linearisation techniques for reducing the noise growth in multiplication and to growth in ciphertext sizes as shown in table 1.

Definition - Generalised Learning with Errors(GLWE): For security parameter $\lambda$, let $n = n(\lambda)$ be an integer dimension, let $f(x) = x^d + 1$, $d = d(\lambda)$ is a power of 2, let $q = q(\lambda) \geq 2$ be a prime integer, let $R = Z[x]/(f(x))$ and $R_q = R/qR$, and let $\chi = \chi(\lambda)$ be a distribution over R. The $GLWE_{n,f,q,\chi}$ problem is to distinguish the following two distributions: In the first distribution, one samples $(a_i, b_i)$ uniformly from $R_q^{n+1}$. In the second distribution, one first draws $s \leftarrow R^n{}_q$ uniformly and then samples $(a_i, b_i) \in R_q^{n+1}$ by sampling $a_i \leftarrow R^n{}_q$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = \langle a_i, s\rangle + e_i$. The $GLWE_{n,f,q,\chi}$ assumption is that the $GLWE_{n,f,q,\chi}$ problem is infeasible.

TABLE 1: Classifying versions of BGV measured on the basis of security parameter $\lambda$ [11].

| | | | |
|---|---|---|---|
| FHE before BGV | Bootstrapping | $\Omega(\lambda^{3.5})$ | Exponential |
| Leveled BGV 1 | Without bootstrapping | $O(\lambda. L^3)$ | Quasi-linear |
| Leveled BGV 2 | With boot strapping as optimization | $O(\lambda^2)$ independent of depth L | Quasi-polynomial |
| Leveled BGV 3 | Batched bootstrapping | $O(\lambda)$ independent of depth L | Linear |

Algorithm:
(1). E.Setup:

- Input security parameter $\lambda$, a number of levels L, and a bit $b \in \{0, 1\}$ for deciding between LWE or RLWE
- Choose $\mu$-bit modulus q for parameters $d = d(\lambda, \mu, b)$, $n = n(\lambda, \mu, b)$, and $\chi = \chi(\lambda, \mu, b)$.
- $R = Z[x]/(x^d + 1)$ and params = $(q, d, n, N, \chi)$

(2). E.SecretKeyGen:

- Input params
- Sample $s' \leftarrow \chi^n$
- Set $sk = s \leftarrow (1, s'[1], \ldots, s'[n]) \in R^{n+1}{}_q$.

(3). E.PublicKeyGen:

- Inputs params and sk
- Generate matrix $A' \leftarrow R^{NxM}{}_q$ and vector $e \leftarrow \chi^N$.
- Set $b \leftarrow A's' + 2e$, A to be (n+1)-columned consisting of b followed by n columns from $-A'$.
- Set the public key pk = A.

(4). E.Encrypt :

- Input params, pk, and message $m \in R_2$.
- Set $m \leftarrow (m,0,\ldots\ldots,0) \in R^{n+1}{}_q$.
- Sample $r \leftarrow R^N{}_2$.
- Output ciphertext $c \leftarrow m + A^T r \in R^{n+1}{}_q$.

(5). E.Decrypt :

- Output $m \leftarrow [\,[(c,s)]_q\,]_2$.

Implementation:
Halevi and Shoup's HElib implemented the BGV scheme and proved to be the foremost among all the implementations [13]. Several optimizations such as ciphertext packing have been introduced to make homomorphic encryption practically faster. In 2014, bootstrapping was introduced in the library [14]. Multi-threading support was provided in March 2015.

Limitations:
The BGV scheme was performed on homomorphic evaluation of the AES circuit [15]. But to evaluate the circuit using the modulus switching technique, the public key required multiple versions evaluation-keys that need to be shared between the client (data owner) and the server(computing entity). Hence, very large memory requirements for storing the public key.

D. Towards Scale Invariant Fully Homomorphic Encryption
Brakerski provided a new notion of scale-invariance for leveled homomorphic encryption schemes[12]. Unlike modulus switching, the cipher-texts in this scheme keep the same modulus during the whole homomorphic evaluation and only one scale invariant evaluation key has to be stored.

This technique has been adapted to the BGV scheme by Fan and Vercauteren [17] called as FV scheme and to L´opez-Alt, Tromer and Vaikuntanathan's scheme [16] by Bos, Lauter, Loftus, and Naehrig [18] called as YASHE scheme respectively. A comparative work has been theoretically done and applied on a lightweight block cipher SIMON between both the scheme [19]. It is observed that FV has a smaller noise growth as compared to YASHE while the latter is faster in terms of performance.

**Fully Homomorphic Scheme FV**
Fan and Vercauteren [17] applied the scale- invariant technique introduced by Brakerski in [12] to the RLWE. Using the message encoding as in RLWE encryption scheme, we can avoid the modulus switching to obtain a leveled homomorphic encryption scheme. A brief description is given below.

Algorithm:
(1).FV.ParamsGen:
- Inputs security parameter $\lambda$.
- Set a positive integer d.
- Determine R, moduli q and t, distributions $\chi_{key}$, $\chi_{err}$ on R, and an integer base $w > 1$.
- Output params = $(d,q,t, \chi_{key}, \chi_{err}, w)$.

(2). FV.KeyGen:
- Input params.
- Sample $s \leftarrow \chi_{key}$, $a \leftarrow R_q$(uniformly,random), $e \leftarrow \chi_{err}$
- Compute $b = [-(as + e)]_q$ .
- Sample , $a \leftarrow R_q^{\,lw,q}$ (uniformly,random), $e \leftarrow \chi_{err}^{\,lw,q}$.
- Compute $\gamma = ( [\mathrm{PowersOf}_{w,q}(s^2) - (e + a \cdot s) ]_q, a ) \in R_q^{\,lw,q}$.
- Output (pk, sk, ek) $= ((b, a), s, \gamma)$.

(3). FV.Encrypt:
- Input ((b, a), m), $m \in R/tR$.
- Sample $u \leftarrow \chi_{key}$, $e_1$ and $e_2 \leftarrow \chi_{err}$ .
- Output $c = ( [\Delta[m]_t + bu + e_1]_q, [au + e_2]_q ) \in R^2$ .

(4). FV.Decrypt:
- Input s, $c = (c_0, c_1)$.
- Output m = = hj t q · [c0 + c1s]q mi t ∈ R

(5). FV.Add:
- Input $c_1 = (c_{10}, c_{11})$ and $c_2 = (c_{20}, c_{21})$ .
- Output $c_{add} = ([c_{10} + c_{20}]_q, [c_{11} + c_{21}]_q)$ .

(6). FV.ReLin:
- Input (b, a) = ek and $\hat{c}_{mult} = (c_0, c_1, c_2)$.
- Output ciphertext ( $[c_0 + \langle \mathrm{WordDecomp}_{w,q}(c_2), b\rangle]_q$ , $[c_1 + \langle \mathrm{WordDecomp}_{w,q}(c_2), a\rangle]_q$ ), where $\mathrm{WordDecomp}_{w,q}(a) = ([a_i]_w)^{lw,q-1}_{i=0} \in R^{lw,q}$ .

(7). FV.Mult :
- Input $c_1$, $c_2$, ek.
- Output $c_{mult}$ = FV.ReLin ($\hat{c}_{mult}$, ek).

**Fully Homomorphic Scheme YASHE**
A fully homomorphic encryption scheme is introduced in [18] based on modified NTRU by Stehl´e and Steinfeld [20] and the multi-key fully homomorphic encryption scheme presented in [16]. A brief description of the scheme is illustrated below.

Algorithm:
(1). YASHE.ParamsGen($\lambda$) :
- Inputs security parameter $\lambda$.
- Set a positive integer d.
- Determine R, moduli q and t, $1< t < q$, distributions $\chi_{key}$, $\chi_{err}$ on R, and an integer base $w > 1$.
- Output params = $(d,q,t, \chi_{key}, \chi_{err}, w)$.

(2). YASHE.KeyGen:
- Inputs params
- Sample f', $g \leftarrow \chi_{key}$. Let $f = [tf' + 1]_q$ .If f not invertible mod q, choose a new f'.
- Compute the inverse $f^{-1} \in R$ of f mod q.
- Set $h = [tgf^{-1}]_q$ .
- Sample e, $s \leftarrow \chi^{lw,q}_{err}$ and compute $\gamma = [\mathrm{PowersOf}_{w,q}(f) + e + h \cdot s]_q \in R^{lw,q}$ where $\mathrm{PowersOf}_{w,q}(a) = ([aw^i]_q)^{lw,q-1}_{i=0} \in R^{lw,q}$.
- Output (pk,sk, evk) = $(h, f, \gamma)$.

(3). YASHE.Encrypt:
- Input (h, m), $m \in R/tR$.
- Sample s, $e \leftarrow \chi_{err}$ .
- Output $c = ( [\Delta[m]_t + e + hs]_q ) \in R$.

(4). YASHE.Decrypt:
- Input (f, c).
- Output $m = [\frac{t}{q}.[f\,c]_q]_t \in R$.

(5). YASHE.Add:
- Inputs $(c_1, c_2)$
- Output $c_{add} = [c_1 + c_2]_q$ .

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 7, July 2016

(6). YASHE.KeySwitch:

- Inputs ($\hat{c}_{mult}$ , ek)
- Output ciphertext $[\langle \text{WordDecomp}_{w,q}\ (\hat{c}_{mult}),\ ek\rangle]_q$ , where $\text{WordDecomp}_{w,q}(a) = ([a_i\ ]_w)\ ^{lw,q-1}_{i=0} \in R^{lw,q}$.

(7). YASHE.Mult:

- Inputs ($c_1$, $c_2$, ek)
- Output ciphertext $c_{mult}$ = YASHE.KeySwitch($\hat{c}_{mult}$ , ek) where $\hat{c}_{mult} = [\frac{t}{q}.c_1c_2]_q$ .

Implementation**:**
Researchers from Microsoft presented a more complete and accurate encryption scheme using YASHE algorithm with a small variant in the form of a library SEAL (Simple Encrypted Arithmetic Library) [22]. In another work, Gilad-Bachrach, Ran, et al. [23] discussed an implementation for applying neural network on encrypted data with high throughput and accuracy.

Limitation:
One of the main drawback found is the inability to stay in Double-CRT form during multiplication, at the cost of computation overhead. Despite being more elegant and less memory consuming, the benefits of scale-invariant schemes are still questioned [21].

## VI. CONCLUSION

The security of Cloud Computing based on homomorphic has gained attention in recent years. It provides potential for carrying out large-scale computations and storage, statistical analysis and query processing directly on encrypted forms of private data, thus respecting the confidentiality of the data.

This paper analyzes some prominent homomorphic schemes (RSA, El Gamal, Pallier, Gentry's, DGHV, BGV, FV, and YASHE) and their significance and approach in context to application in cloud computing. Finally, a comparative table is constructed based on their class of operation (partial, somewhat, leveled fully), underlying approach and problems.

Fully homomorphic encryption is a promising aspect in cryptography. Despite its interesting properties, it is quite limited regarding its computation abilities and practical implementations. Future work of this research is planned towards experimenting these techniques in the cloud environment and analyzes them based on the time of response of the cloud for different key sizes.

TABLE 2: Comparision of Homomorphic Encryption (HE) Schemes

| **HE Scheme** | **Type** | **HE Operation** | **Underlying Principle** | **Concern** |
|---|---|---|---|---|
| **RSA** | Partial HE | Addition | Factoring problem of product of large primes | Not semantically secure |
| **ELGAMAL** | Partial HE | Multiplication | Discrete Logarithms | Impractical for addition |
| **PAILLIER** | Partial HE | Addition | Decisional composite residuosity assumption | Does not support multiplication |
| **Gentry's FHE** | First fully HE | Any circuit | Lattice-Based using Bootstrapping | Implementation is impractical |
| **DGHV** | Some-What HE | Circuit up to a certain depth | Modular Arithemetic & Approximate GCD | Key size is too large( ~$2^{60}$ bits) |
| **BGV** | Leveled FHE | Any circuit | Modulus switching & LWE, RLWE | Large memory requirement for storing multiple keys |
| **FV** | Leveled FHE | Any circuit | Scale invariant RLWE on BGV | Inability to stay in double CRT form |
| **YASHE** | Leveled FHE | Any circuit | Scale invariant RLWE on NTRU | Inability to stay in double CRT form |

## REFERENCES

[1] Winkler, Vic JR. Securing the Cloud: Cloud computer Security techniques and tactics. Elsevier, 2011.
[2] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011).
[3] Hashizume, Keiko, et al. "An analysis of security issues for cloud computing."Journal of Internet Services and Applications 4.1 (2013): 1.
[4] Patrascu, Alecsandru, Diana Maimu, and Emil Simion. "New Directions in Cloud Computing: A Security Perspective." Communications (COMM), 2012 9th International Conference on. 2012.
[5] Aslett, Louis JM, Pedro M. Esperança, and Chris C. Holmes. "A review of homomorphic encryption and software tools for encrypted statistical machine learning." arXiv preprint arXiv:1508.06574 (2015).
[6] Rivest, Ronald L., Len Adleman, and Michael L. Dertouzos. "On data banks and privacy homomorphisms." Foundations of secure computation 4.11 (1978): 169-180.
[7] ElGamal, Taher. "A public key cryptosystem and a signature scheme based on discrete logarithms." Workshop on the Theory and Application of Cryptographic Techniques. Springer Berlin Heidelberg, 1984.
[8] Paillier, Pascal. "Public-key cryptosystems based on composite degree residuosity classes." International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, 1999.
[9] Gentry, Craig. A fully homomorphic encryption scheme. Diss. Stanford University, 2009.
[10] Van Dijk, Marten, et al. "Fully homomorphic encryption over the integers."Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, 2010.
[11] Brakerski, Zvika, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping." Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. ACM, 2012.
[12] Brakerski, Zvika. "Fully homomorphic encryption without modulus switching from classical GapSVP." Advances in Cryptology–CRYPTO 2012. Springer Berlin Heidelberg, 2012. 868-886.
[13] Halevi, Shai, and Victor Shoup. "Algorithms in helib." International Cryptology Conference. Springer Berlin Heidelberg, 2014.
[14] Halevi, Shai, and Victor Shoup. "Bootstrapping for helib." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, 2015.
[15] Gentry, Craig, Shai Halevi, and Nigel P. Smart. "Homomorphic evaluation of the AES circuit." Advances in Cryptology–CRYPTO 2012. Springer Berlin Heidelberg, 2012. 850-867.
[16] López-Alt, Adriana, Eran Tromer, and Vinod Vaikuntanathan. "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption." Proceedings of the forty-fourth annual ACM symposium on Theory of computing. ACM, 2012.
[17] Fan, Junfeng, and Frederik Vercauteren. "Somewhat Practical Fully Homomorphic Encryption." IACR Cryptology ePrint Archive 2012 (2012): 144.
[18] Bos, Joppe W., et al. "Improved security for a ring-based fully homomorphic encryption scheme." IMA International Conference on Cryptography and Coding. Springer Berlin Heidelberg, 2013.
[19] Lepoint, Tancrede, and Michael Naehrig. "A comparison of the homomorphic encryption schemes FV and YASHE." International Conference on Cryptology in Africa. Springer International Publishing, 2014.
[20] Stehlé, Damien, and Ron Steinfeld. "Making NTRU as secure as worst-case problems over ideal lattices." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, 2011.
[21] Barthelemy, Lucas. “A brief survey of Fully Homomorphic Encryption, computing on encrypted data” Quarkslab’s blog, 29 June 2016. Web.
[22] Dowlin, Nathan, et al. Manual for using homomorphic encryption for bioinformatics. Technical report MSR-TR-2015-87, Microsoft Research, 2015.
[23] Gilad-Bachrach, Ran, et al. "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy." Proceedings of The 33rd International Conference on Machine Learning. 2016.