Design of Fully Homomorphic Encryption by Prime Modular Operation

Sarah Shihab Hamad and Ali Makki Sagheer

***Abstract* — The meaning of cloud computing is the Information Technology (IT) model for computing, which consists of all the IT components (software, hardware, services and, networking) that are needed to enable the delivery and development of cloud services through a private network or the internet. In cloud computing, the client (user) puts his data in the cloud, and any computations on his stored data will be implemented in the cloud. Security is the main thing in cloud computing because a service provider can access, intentionally change or even delete the stored data. To protect data that is stored in the cloud, it is necessary to use an encryption system that can perform computations on the encrypted data. The scheme that allows executing several computations on the encrypted message without decrypting the message is called homomorphic encryption. The implementation of fully homomorphic encryption over the integer (DGHV scheme) and a Simple Fully Homomorphic Encryption Scheme Available in Cloud Computing (SDC Scheme), are slow in execution time because all of them convert the message to a binary format and then encrypt it. Therefore, we propose another scheme called Fully Homomorphic encryption based on a prime modular operation, this scheme encrypts the message character by character by using a prime secret key without converting that character into a binary format. As a result, we compute the time complexity and compare the execution time among the three schemes and analyse the security of the three schemes.**

***Keywords* — Cloud Computing, Cryptosystem, Fully Homomorphic Encryption, Information Security.**

I. Introduction

THE term “homomorphism” is derived from the Greek word” homos” that means "same" and “morphe” meaning "shape", or “same structure”. Homomorphic encryption in the computer science is used to transform a plaintext to a ciphertext. Plaintext is any information that the senders want to transfer to the receiver, so the plaintext is an input to each encryption algorithm. Several examples of plaintext are email messages, credit card transaction information and images. This plaintext will transform to a ciphertext (the data has been encrypted) and it is difficult to understand unless we use a key that decrypts it [1]. The technique that allows computation on a ciphertext without previous decryption is called homomorphic encryption, HE, by making operations on a ciphertext, the user receives the result (that is in the encrypted form) and decrypts it by a key, which gives the user the first result without knowing the original plaintext[2]. An encryption scheme contains three algorithms: KeyGen, Encrypt, and Decrypt. Homomorphic Encryption (HE) scheme can be either symmetric or asymmetric. In a symmetric (or secret key) encryption scheme, KeyGen utilizes λ (a security parameter that determines the bit length of keys) to produce one key that is used in each of encryption and decryption, first transforms a message to a ciphertext, and later transforms the ciphertext back to the message. In an asymmetric (or public key) encryption scheme, KeyGen utilizes λ to generate two keys: a public encryption key, *pk*, which may be made available to everyone, and a secret decryption key *sk* [3]. HE has a fourth algorithm (Evaluate) that performs a function over the ciphertexts without seeing the messages. Evaluate algorithm takes ciphertexts as input and outputs evaluated ciphertexts. The most crucial point in this homomorphic encryption is that the format of ciphertexts after an evaluation process must be preserved in order to be decrypted correctly. In addition, the size of ciphertext should also be constant to support an unlimited number of operations. Otherwise, the increase in ciphertext size will require more resources and this will limit the number of operations. This limitation of HE continues for more than 30 years, until the first plausible and achievable Fully Homomorphic Encryption scheme which allows any computable function to perform on the encrypted data, was presented by Craig Gentry in 2009[4].

The organization of the remainder of the paper: section 2 provides Related Works of HE, Section 3 displays Fully Homomorphic Encryption FHE and explains some of FHE schemes. Section 4 offers the proposed scheme. Section 5 describes the results and discussion. Finally, our conclusions are mentioned in Section 6.

II. Related Work

The first homomorphism is suggested by Rivest, Adleman, and Dertouzos in [5]. The multiplicative homomorphism is given by RSA [6]. A partial homomorphic encryption scheme is suggested by Yao [7], Goldwasser and Micali [8], ElGamal [9] and Paillier [10]. Fontaine & Galand have presented a survey of homomorphic encryption schemes in [11].Gentry from IBM has proposed fully homomorphic encryption in his thesis and paper [12]. Many researchers propose the variants of Gentry’s model with some improvement. Homomorphic encryption on a smaller size cipher text is proposed by [13] Smart and Vercauteren. The arithmetic operations over integers are proposed by Dijk, Gentry,

Paper received Januar 16, 2018; revised September 14, 2018; accepted December 9, 2018. Date of publication December 25, 2018. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Miroslav Lutovac.

Sarah Shihab Hamad, Msc student at the Department of Computer Science, College of Computer Science and Information Technology, University of Anbar, Iraq.(e-mail:sarah.sh1985@gmail.com).

Ali Makki Sagheer Essaw, professor since 2015 at the College of Computer Science and IT, University of Anbar, Iraq. (phone: 00964-770-0073940; e-mail: ali_makki@uoanbar.edu.iq).

Halevi, and Vaikuntanathan [14]. Faster improvement to Gentry's model is proposed by Stehle and Steinfield [15]. Y Govinda Ramaiah [16] has proposed "Efficient Public Key Homomorphic Encryption over Integer Plaintexts".

III. Fully Homomorphic Encryption (FHE)

An encryption technique is called FHE if it performs both addition and multiplication at the same time, and it can compute any operation. Craig Gentry in 2009 presented the first construction of a fully homomorphic scheme [12].

A. Gentry's Fully Homomorphic Encryption Using Ideal Lattices

Gentry uses lattice-based cryptography. His FHE consists of several steps: start with what was called a somewhat homomorphic encryption scheme SWHE. To understand SWHE scheme's work, the first thing that we must know: all scheme's ciphertexts have noise inside them and this noise unfortunately become larger if we have performed many of homomorphic operations, there is so much noise (at some point) that the encryptions become useless (i.e. they have incorrect decrypt). This is the main limitation of SWHE schemes and this is the reason why they cannot perform just a limited number of computations [17]. Gentry shows that you can manage designing an SWHE scheme that supports the evaluation of its own decryption algorithm (and a little more). Next, "Squashing" the decryption circuit of the original somewhat homomorphic scheme to make it bootstrappable. The last step, there is a general technique to transform the SWHE scheme into an FHE scheme. An SWHE that can evaluate its own decryption algorithm homomorphically is called bootstrappable and the technique that transforms a bootstrappable SWHE scheme into an FHE scheme is called bootstrapping. Bootstrapping allows us to control noise [18].

B. Van Dijk et al: Fully Homomorphic Encryption over the Integers

A much more conceptually simple and true variant of Gentry's scheme was presented by Marten van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan (DGHV scheme) [14].The description of the DGHV scheme is as follows:

KeyGen (λ): generate the secret key p is an odd η-bit integer, $p \in [2^{\eta-1}, 2^{\eta})$.

Encrypt (pk, $m \in \{0, 1\}$): to encrypt a 1-bit message m: generates a large multiple of the secret key, e.g., p q, a small even number $2r$ where r is the noise ($r < p/4$ or $2r < p/2$) , $r \approx 2^{\sqrt{\eta}}$ and generate $q \approx (2^{\eta})^3$ → $c = p\,q + 2r + m$

Evaluate(pk, C, $c_1, \ldots, c_t$): given the (binary) circuit C with t inputs, and t ciphertexts ci, apply the (integer) addition and multiplication gates of C to the ciphertexts, performing all the operations over the integers, and return the resulting integer

Decrypt (sk, c): to decrypt ciphertext c

$m = (c \bmod p) \bmod 2$

Proof:

Let $c_1 = pq_1 + 2r_1 + m_1$, $c_2 = pq_2 + 2r_2 + m_2$

Additive Homomorphism:

- $c_1 + c_2 = (q_1 + q_2)p + 2(r_1 + r_2) + (m_1 + m_2)$
- $[(c_1 + c_2) \bmod p] \bmod 2 \equiv m_1$ XOR m_2 (mod2)

Multiplicative Homomorphism:

- $c_1.c_2 = (q_1q_2p + 2q_1r_2 + q_1m_2 + 2q_2r_1 + q_2m_1)p + 2(2r_1r_2 + r_1m_2 + r_2m_1) + m_1m_2$
- $[(c_1.c_2 \bmod p) \bmod 2] \equiv m_1$ AND m_2 (mod2) [19].

C. Gen10 scheme

In March 2010, a homomorphic encryption scheme (called Gen10 scheme) was presented by Gentry [3], for security parameter λ, Set $N = \lambda$, $P = \lambda^2$ and $Q = \lambda^5$.

The description of the Gen10 scheme is as follows:

KeyGen (λ): generate the secret key p as a random, P-bit odd integer.

Encrypt (p, $m \in \{0, 1\}$): to encrypt a 1-bit message m,

$m' = m \bmod 2$, where m' should be a random N-bit number.

$c = m' + p\,q$, where q is a constant Q-bit big integer.

Decrypt (p, c): $c' = c \bmod p$, where c' is the integer in $(-p/2, p/2)$, such that p divides $c - c'$.

Homormorphic Operations:

Add (c_1, c_2) ➢$c = c_1 + c_2$, Sub (c_1, c_2) ➢$c = c_1 - c_2$,

Mult (c_1, c_2) ➢$c = c_1 * c_2$.

Evaluate (f, $c_1,\ldots,c_t$): given the Boolean function f as a circuit C with (XOR and AND) gates.

To check the ciphertexts output by evaluating decrypt correctly:

Let $c = c_1 * c_2$, where ci's noise is mi', which has the same parity as the message mi. We get:

$c = m_1' * m_2' + p\,q'$, for some integer q'. As long as the noise is small enough so that: $|m_1' * m_2'| < p / 2$, we get:

$(c \bmod p) = m_1' * m_2'$, thus $(c \bmod p) \bmod 2 = m_1 * m_2$ [3].

D. A Simple Fully Homomorphic Encryption Scheme Available In Cloud Computing (SDC Scheme)

In 2012, Jian Li, Danjie Song et al. [20] presented a simple FHE derived from Gentry cryptosystem to ensure the privacy in cloud storage (SDC scheme). Neither DGHV nor Gen10 schemes have indicated to ciphertext retrieval algorithms. The description of SDC scheme is as follows:

KeyGen (λ): generate the secret key p as a random, P-bit odd integer.

Encrypt (p, $m \in \{0, 1\}$): to encrypt a 1-bit message m,

$c = m + p + r * p * q$, where r is a random number of R-bit and q is a Q-bit big integer.

Decrypt (p, c): $m = (c \bmod p)$

Retrieval (c): $R = (c_i - c_{index}) \bmod q$

When the user wants to retrieve contents m_{index}, he encrypts the Keywords $c_{index} = m_{index} + p + r * p * q$ and, delivers c_{index} to the server. On receiving c_{index}, server reads the ciphertexts, computing $R = (c_i - c_{index}) \bmod q$, once $R = 0$, ciphertext retrieval succeeds, and ci is the desired result.

Proof:

Suppose $c_1 = m_1 + p + r_1 * p * q$, $c_2 = m_2 + p + r_2 * p * q$

Additive Homomorphism:

$c_3 = c_1 + c_2 = (m_1 + m_2) + (r_1 + r_2) * p * q + 2p$.

$m_3 = c_3 \bmod p = m_1 + m_2$.

Multiplicative Homomorphism:

$c_4 = c_1 * c_2 = m_1 * m_2 + (m_1 + m_2 + p)\,p + r_1\,(p + m_2 + r_2)\,p\,q + r_2\,(p + m_1)\,p\,q$ → $m_4 = c_4 \bmod p = m_1 * m_2$ [20].

IV. The Proposed Scheme: Fully Homomorphic Encryption by Prime Modular Operation (SAM Scheme)

After reviewing DGHV and SDC schemes, we notice that the message $m \in \{0, 1\}$, in which we should convert each character in plaintext (that will later encrypt) to the binary format which will be the input m to the encryption equation that it is mentioned above. Our proposal scheme (SAM) scheme is: instead of converting each character of the text to (8-bit) binary format and enter each bit into the encryption equation to produce (8- ciphertext) for each character in the plaintext, to take the character directly (ASCII code of the character) and enter it to the proposed encryption equation: $c = m + r\,p + p.\,q$, where c is the ciphertext, $m \in [0, p\text{-}1]$, p is a prime big integer, r is the noise and q is a constant big integer, resulting in one ciphertext for each character in the plaintext.

Algorithm (SAM)

1- Key Generation
   1-1 Generate a prime big integer p
   1-2 Generate q ∈ Z_n
   1-3 Generate r ∈ Z_n

2- Encryption
   2-1 c =m+ r .p+p.q, where m ∈[0, p- 1]

3- Decryption
   3-1 m =c mod p

4- Evaluation
   4-1 Assume there are two ciphertexts:
   $c_1 = m_1 + r_1 p + p.q$
   $c_2 = m_2 + r_2 p + p.q$
   4-2 $c_3 = c_1 + c_2$ ⇨ m_3 = Dec (c_3) (or $m_3 = m_1 + m_2$)
   4-3 $c_4 = c_1 . c_2$ ⇨ m_4 = Dec (c_4) (or $m_4 = m_1 . m_2$)
   Let $c = f(c_1 ... c_i)$, such as:
   $c = [(c_1 . c_3) + c_2] . c_4$
   Let $m = f(m_1 ... m_i)$, such as:
   $m = [(m_1 . m_3) + m_2] . m_4$,
   Where f is any function (Addition and Multiplication) *applied on the ciphertexts or messages.*
   4-4 c mod p ≡ m, where m <p, otherwise that we must take (m mod p)

The implementation interfaces of DGHV, SDC and SAM schemes are illustrated in Appendix. The description of SAM scheme is as follows:

KeyGen (λ): generate the secret key p to be a prime big integer, which is chosen randomly.

Encrypt (pk, $m \in [0, p\text{-}1]$): to encrypt a message m

The ciphertext $c = m + r\,p + p.q$, where r is the noise and r is a random big integer and q is a constant big integer.

Evaluate(pk, $m_1 \ldots m_t$, $c_1, \ldots, c_t$): apply addition and multiplication to t ciphertexts ci, and then decrypt the result of ci, we get an integer number which is the same as the integer number that is a result of applying addition and multiplication to t input mi

Decrypt (sk, c): to decrypt ciphertext c: $m = c \bmod p$

To prove that the SAM scheme supports additive and multiplicative homomorphism:

Suppose, $c_1 = m_1 + r_1 p + p.q$, $c_2 = m_2 + r_2 p + p.q$

Additive Homomorphism:

$c_3 = c_1 + c_2 = (m_1 + m_2) + (r_1 + r_2)\,p + 2\,p.q$

$m_3 = (c_1 + c_2) \bmod p = m_1 + m_2$

Multiplicative Homomorphism:

$c_4 = c_1.c_2 = m_1.m_2 + (m_1 + m_2 + p.q)\,p.q + r_1 (m_2 + r_2 + p.q)\,p + r_2 (m_1 + p.q)\,p.$ → $m_4 = (c_1.\,c_2) \bmod p = m_1.\,m_2.$

SAM Example:

Choose a prime number p =1207645633, and q = 1155942797

Now take two random integers r_1=1828989585 and r_2 =1136953862, and two messages m_1 = 65 and m_2 = 66

Now calculate c_1:

$c_1 = m_1 + r_1 p + p.q$

c_1=65+1828989585*1207645633+1207645633* 1155942797

c_1=3604740555922587871, (65 is ASCII code of character A).

Then calculate c_2:

$c_2 = m_2 + r_2 p + p.q$

c_2 = 66 + 1136953862* 1207645633 + 1207645633 * 1155942797, (66 is ASCII code of the character B)

c_2 =2769006637161640213

Additive Homomorphism:

Let the addition of two encrypted messages be c_3:

$c_3 = c_1 + c_2$

=3604740555922587871+2769006637161640213

= 6373747193084228084

Now decrypt c_3:

$m_3 = c_3 \bmod p \rightarrow m_3$= 6373747193084228084 mod 1207645633

m_3=131, this is equal to $m_1 + m_2$ (i.e. 65 + 66 =131)

Multiplicative Homomorphism:

Let the multiplication of two encrypted messages be c_4:

$c_4 = c_1.c_2$ = 3604740555922587871 * 2769006637161640213

= 9.98155052459538650428376534197797e+36

Now decrypt c_4:

$m_4 = c_4 \bmod p \rightarrow m_4 = c_4 \bmod 1207645633$

m_4= 4290, this is equal to $m_1 * m_2$ (i.e. 65 * 66 = 4290).

V. Results and Discussion

In this section, we will calculate the time complexity of the encryption and decryption function for the DGHV and SDC schemes and our proposed SAM scheme and we will also compute their execution time, and finally analyse the security of the three schemes.

A. Big O Notation (Time Complexity)

The O-notation is very useful in guiding the designers of algorithms in search for the "best" algorithms for an important problem [21]. Before performing the calculations of time complexity, we must first analyze the input numbers of the encryption and decryption algorithm, which are either binary integers or decimal digits, where: the time

complexity of binary integers is $O(n)$, while the time complexity of decimal digits is $O(\log(n))$, except the constant number, with the time complexity of $O(1)$, where n is the size of input numbers.

1) Time Complexity of DGHV scheme

Let n be the size of input message unit.

Encryption function:

$c = m + 2r + p * q$

Then: $T(c) = O(n) + T(2r) + O(n^2)$

$T(2r) = O(n)$, by shift operation

$T(c) = O(2n) + O(n^2) \equiv O(n^2)$ bit operation.

Decryption function:

$m = (c \bmod p) \bmod 2$

Then: $T(m) = O(n^2)$ bit operation.

2) Time Complexity of SDC scheme

Let n be the size of input message unit.

Encryption function:

$c = m + p + r * p * q$

Then: $T(c) = O(n) + O(n) + O(2(n^2))$

$T(c) = O(2(n)) + O(2(n^2)) \equiv O(n^2)$ bit operation.

Decryption function:

$m = c \bmod p$

Then: $T(m) = O(n^2)$ bit operation.

3) Time Complexity of SAM scheme

Let n be the size of input message, n – decimal digit

Encryption function:

$c = m + r * p + p * q$

Then: $T(c) = O(2(\log(n))) + O(2(\log(n))^2)$

$T(c) \equiv O(\log(n)^2)$.

Decryption function:

$m = c \bmod p$

Then: $T(m) = O((\log(n))^2)$,

Where, $(\log_2 n)$ is the number of bits of n.

B. Execution Time

When we use the same length of the message and the same parameter p, q, r the execution time of SAM scheme requires less time than that required for DGHV and SDC schemes which seem approximately similar in execution time because the two schemes were encrypting bit by bit while the proposed scheme was encrypting character by character. As a result, the execution time of SAM scheme is very fast compared to DGHV and SDC schemes, as shown in TABLE 1 and Fig. 1.

TABLE 1: EXECUTION TIME OF DGHV, SDC AND SAM SCHEMES

Length of the message	*DGHV scheme (m.s.)*	*SDC scheme (m.s.)*	*SAM scheme (m.s.)*
12 byte	1118	1180	1007
1.4 k byte	1241820	1283068	20818
2.8 k byte	6715148	4425899	72901

C. Security:

The security of Fully Homomorphic Encryption Scheme over the Integers (DGHV scheme) is based on the strength of the approximate integer greatest common divisors (approximate GCD) problem which is constructed by Howgrave-Graham [22], With appropriate selection of parameters the scheme has proved to resist different types of attacks to recover the secret key including brute-force attack with at least 2λ time. However, it has been proven that the scheme can be attacked to recover the plaintext from ciphertext using lattice reduction algorithm [23]. The parameters settings which have been used in the attack are considered to be appropriate by the DGHV scheme.

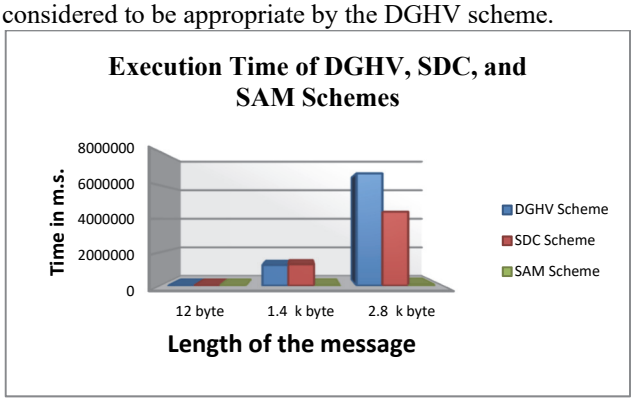


Fig. 1. Execution time of DGHV, SDC, and SAM schemes.

Jian Li, Danjie Song et al. concluded when comparing their scheme (SDC) with the DGHV scheme and the Gen10 scheme, that SDC scheme has a major advantage in the ciphertext retrieval:

1) The DGHV scheme has to submit its private key to the untrusted server to perform the ciphertext retrieval, while the SDC scheme only has to transmit a constant big integer q to the weak server, which is much more secure.

2) To retrieve the ciphertext, the Gen10 scheme does not present the private key p but a random number q instead, which seems quite more secure than the DGHV scheme, yet once q contains the server, utilizing $c \bmod q$, the plaintext m leaks out. While in the SDC scheme, even though the crackers grab the big integer q, using $c \bmod q$, what it can obtain is merely the $m+p$, and the plaintext m could not leak out [20].

Selecting the prime number as a secret key in the proposed scheme gives robustness in the security than when choosing any number. The reason for this lies in that the prime number itself costs to the third party, where first it has to be verified that the number is prime, and then the prime number should be tested on the encryption equation, which requires lots of attempts to break the code. Added to that, the prime number gives one probability of the solution. The non-prime number does not give all probabilities of the solution and it may exist more than one number of the non-prime numbers that provides the same solution. If we analyze SAM scheme as Jian Li, Danjie Song et al. worked, SAM scheme sends a constant big integer q to the server (when the client wants to return some ciphertexts which it stored on the server) using it in the ciphertext retrieval algorithm which is more secure as in SDC scheme. Even though the crackers grab q, by using $c \bmod q$ the plaintext m could not leak out, what it can obtain is simply $m + r\,p$.

VI. CONCLUSION

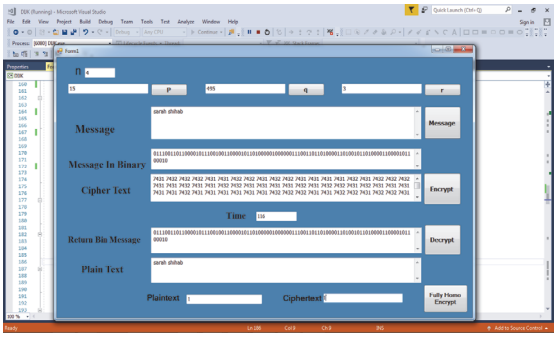
We know what is the meaning of the cloud computing and the cloud must be secured by an encryption system such as homomorphic encryption and explain the categories of Homomorphic Encryption and we present some of FHE schemes and offer examples about it. In this paper, we propose an effective scheme that instead of converting the

character (which is in plaintext) to a binary format and then enter it into the encryption equation, takes the character directly from the text and insert it into the encryption equation. As a result, the time complexity and execution time of SAM scheme is very fast compared to the two schemes DGHV and SDC. Also, the SAM scheme proves its security by using the prime number as a secret key and its security in ciphertext retrieval.

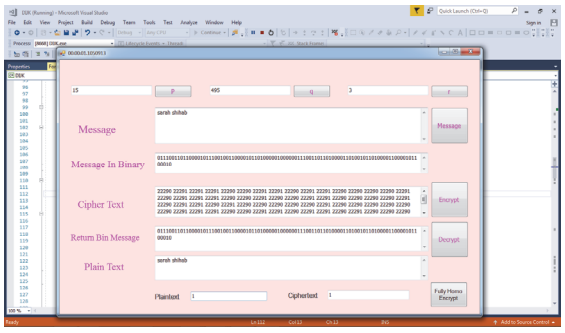
APPENDIX

The Implementation Interfaces

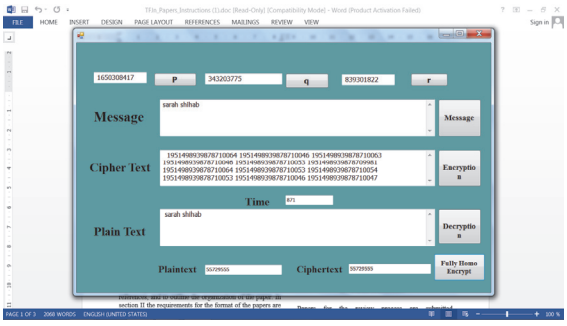
DGHV Scheme



SDC Scheme



SAM Scheme



REFERENCES

[1] Hayes B, Alice, and Bob (2012), Cipherspace, American Scientist, Vol. 100, p. 362.

[2] Yang Jing, Mingyu Wang and Zhiyin Kong (2014), "Simulation Study Based on Somewhat Homomorphic Encryption", *Journal of Computer and Communications*, Vol. 2, No. 2, p. 109.

[3] Craig Gentry (2010)," Computing Arbitrary Functions of Encrypted Data", *Communications of the ACM*, Vol. 53 No. 3, Pages 97-105.

[4] Acar, Abbas, et al. "A Survey on Homomorphic Encryption Schemes: Theory and Implementation." arXiv preprint arXiv:1704.03578 (2017).

[5] Rivest, Ronald L., Len Adleman, and Michael L. Dertouzos. "On data banks and privacy homomorphisms", *Foundations of secure computation* 4.11 (1978): 169-180.

[6] Rivest, Ronald L., Adi Shamir, and Len Adleman. "A method for obtaining digital signatures and public-key cryptosystems." *Communications of the ACM* 21.2 (1978): 120-126.

[7] A. C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science* (FOCS'82), pages 160-164. IEEE, 1982.

[8] Goldwasser, Shafi, and Silvio Micali. "Probabilistic encryption." *Journal of computer and system sciences* 28.2 (1984): 270-299.

[9] ElGamal, Taher. "A public key cryptosystem and a signature scheme based on discrete logarithms." *Advances in cryptology*. Springer Berlin Heidelberg, 1985.

[10] Paillier, Pascal. "Public-key cryptosystems based on composite degree residuosity classes." *Advances in Cryptology—EUROCRYPT'99*. Springer Berlin Heidelberg, 1999.

[11] Fontaine, Caroline, and Fabien Galand. "A survey of homomorphic encryption for nonspecialists." *EURASIP Journal on Information Security 2007* (2007): 15.

[12] Gentry, Craig. "Fully homomorphic encryption using ideal lattices." *STOC*. Vol. 9. 2009.

[13] Smart, Nigel P., and Frederik Vercauteren. "Fully homomorphic encryption with relatively small key and ciphertext sizes." *Public Key Cryptography–PKC 2010*. Springer Berlin Heidelberg, 2010.420-443.

[14] Van Dijk, Marten, et al. "Fully homomorphic encryption over the integers." *Advances in Cryptology–EUROCRYPT 2010*. Springer Berlin Heidelberg, 2010. 24-43.

[15] Stehlé, Damien, and Ron Steinfeld. "Faster fully homomorphic encryption." *Advances in Cryptology-ASIACRYPT 2010*. Springer Berlin Heidelberg, 2010. 377-394.

[16] Ramaiah, Y. Govinda, and G. Vijaya Kumari. "Efficient public key homomorphic encryption over integer plaintexts." *Information Security and Intelligence Control (ISIC), 2012 International Conference on*. IEEE, 2012.

[17] Benzekki, Kamal, Abdeslam El Fergougui, and E. A. Elbelrhiti. "A secure cloud computing architecture using homomorphic encryption." *International Journal of Advanced Computer Science and Applications (IJACSA)* 7.2 (2016): 293-298.

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

[19] Atayero A., Feyisetan O. (2011), "Security Issues in Cloud Computing: The Potentials of Homomorphic Encryption", *Journal of Emerging Trends in Computing and Information Sciences*, VOL. 2, NO.10, October 2011.

[20] J. Li, D. Song, S. Chen, X. Lu, "A Simple Fully Homomorphic Encryption Scheme Available in Cloud Computing", In *Proceeding of IEEE*, (2012).

[21] A. M. Sagheer, "Elliptic Curves Cryptographic Techniques", 2012 *International Conference on. IEEE*, 2012.

[22] N. Howgrave-Graham, —Approximate integer common divisors, in CaLC, pp. 51–66. 2001.

[23] Gu Chunsheng, —Attack on Fully Homomorphic Encryption over the Integers, Cryptology ePrint Archive, Report 2012/157, 2012.