International Journal of Computer Applications (0975 – 8887)
Volume 183 – No. 25, September 2021

Securing Data in the Cloud using the SDC Algorithm

Dennis Redeemer Korda
Kwame Nkrumah University of Science and Technology, Kumasi
Department of Computer Science

Edward Danso Ansong, PhD
Kwame Nkrumah University of Science and Technology, Kumasi
Department of Computer Science

Dickson Kodzo Mawuli Hodowu
Kwame Nkrumah University of Science and Technology, Kumasi
Department of Computer Science

ABSTRACT

The past few years have seen much attention directed towards cloud computing where private and organizational usage has skyrocketed because of the flexibility it provides. Cloud computing may be exploited for a decrease in the limitations of conventional computing architectures in this way profiting an organization as far as a competitive advantage, space, time, power, cost and in any event, giving a disentangled business process. In as much as Cloud computing arises with a lot of benefits, the security challenges that it poses cannot be overestimated. The use of traditional encryption algorithms such as RSA, DGHV andGen10, to scramble the secluded data before transferring it to the cloud provider have been proposed to solve this security loophole. But, the customer ought to give the secret key tothe server to unscramble the data before performing the computations necessitated. This paper proposes a framework for a variant of the Homomorphic encryption known as SDC encryption algorithm which allows performing calculations on scrambled data without decoding. The performance metrics of the proposed framework was carried out using the big-Oh notation and an SDC encryption library in python and then tested for correctness. The results obtained from the proposed framework of the SDC encryption algorithm indicated satisfactory performances. Hence, with the utilization of the SDC encryption algorithm, the customer's data in cloud server is secure and this also allows computations onthis encoded data. These results provide inspiring evidence for securing data in clouds using the SDC encryption algorithm.

General Terms
Homomorphic Encryption Algorithm and Cloud Computing.

Keywords
Fully homomorphic encryption, partially homomorphic encryption, SDC encryption algorithm and big-Oh notation.

1. INTRODUCTION
Cloud computing happens to be the hottest research area in computer science. Theaforementioned empowers clients to get practically boundless computing force and it offerspotential advantages to these clients regarding quick accessibility, scalability and resourceallocation.

Accordingly, a pragmatic, basic completely homomorphic encryption algorithm, utilizingjust basic modular arithmetic, is executed to guarantee the privacy-safeguarding in cloudstorage, such that encoded data might be worked on legitimately upon deprived of the influenceof the secrecy of the encryption frameworks so it can amazingly understand the necessity ofciphertext recovery and extra handling in cloud computing. This simple fully homomorphicencryption algorithm is called the SDC algorithm.

2. PROBLEM STATEMENT
Most cloud computing environments do not give protection from untrusted cloud administrators, which presents a difficulty for organizations and associations that need to store delicate, classified data, for example, clinical records, monetary records, or high-sway business information[1].The use of conventional encryption algorithms to encode remote data before uploading it to the cloud supplier has been the most broadly utilized procedure to connect the security gap in cloud computing environments. Nonetheless, the customer will need to make available the secret key for the server of the CSP in other to decode the data before performing the essential computations on the data. A variant of the Homomorphic encryption known as SDC allows performing calculations on scrambled data without decoding. Hence, with the utilization of the SDC encryption algorithm, the customer's data in cloud server is made secure and this also permits to implement essential calculations on this encoded data.

3. RESEARCH OBJECTIVES
The under-listed are the objectives of the research:

i. To acquire a deep understanding of homomorphic encryption algorithms and their implementation frameworks

ii. To propose an implementation framework for the SDC algorithm in cloud computing

iii. To measure the correctness of SDC encryption algorithm

3.1 RESEARCH QUESTIONS
The questions that when answered will help achieve the objectives of this paper includes:

i. What is the overview of homomorphic encryption algorithms?

ii. What is the implementation framework of the SDC algorithm?

iii. How is the correctness of the SDC encryption algorithm measured?

4. HOMOMORPHIC ENCRYPTION
Although homomorphic encryption algorithms have been in existence for almost half a century, it has received less attention for reasons being, as a result, its computational and storage overhead and also the probability of an operational fully homomorphic encryption algorithm inside a genuine world happened to be a huge query left unanswered. Nevertheless, in this present-day progress in fully homomorphic encryption algorithms has pulled in a great deal of attention into the area of cryptography[2][3][4]. The two

major operations of homomorphic encryptions are addition and multiplication. For addition, $E(d1+d2)=E(d1)+E(d2)\forall d1,d2 \ni n$ and for multiplication, $E(d1*d2)=E(d1)*E(d2)\forall d1,d2 \ni n$ where 'E' corresponds to the encryption and 'n' correspond to the collection of all possible messages.

4.1 SDC Cryptosystem

The SDC encryption algorithm was fully described in [5], thus this section will only focus on the homomorphic characteristics and operations of this SDC encryption algorithm. According to [6], the SDC encryption algorithm encourages both addition and multiplication on ciphertexts and they presented a comprehensive verification for the correctness of the algorithm as shown below.

Suppose m_1, m_2as dual messages, then the ciphertext of these messages as soon as encoding becomes$c_1 = m_1 + p + r_1 * p * q$ and $c_2 = m_2 + p + r_2 * p * q$

Additive Homomorphic Property

From the ciphertexts c_1 , $c_2$above;

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

$$m = cmodp = m_1 + m_2$$

Thus, the algorithm has additively homomorphic property.

Multiplicative Homomorphic Property

Similarly, from the ciphertexts c_1 , $c_2$above;

$$c = c_1 * c_2 = m_1 * m_2 + (m_1 + m_2 + p)p + r_1(p + m_2 + r_2)pq + r_2(p + m_1)pq$$

$$m = cmodp = m_1 * m_2$$

And thus, the algorithm also has multiplicative homomorphic property.

4.2 Tools for Implementation

The computation complexity associated with implementing any encryption algorithm will require a very powerful, flexible, open-source language which is not difficult to learn, easy to use, and possess a powerful library for data manipulation and analysis. Having this in mind, the availability of a library determined the implementation language.

The main library used was the SDC_cryptography which was written with the python programming language. This library includes all SDC encryption algorithms necessary for this implementation, notably the key Generation function, Encryption function, Decryption function and the Evaluation function. This repository is available from PyPi at [sdc-cryptography] and installed with the pip command in python. The Python Package Index (PyPi) is the sanctioned third-party software repository for python which contains modules accepted by the python community. The vast majority of the simulation was carried out on my personal machine running macOS High Sierra with a 2.3GHz Intel Core i7 processor and an 8g RAM, although additional resources were provided on my Desktop computer and even used for backups. It is worth noting that factors such as speed of the processor and size of the RAM play an important role in the performance of the algorithm.

All of the code was executed on the personal machine using PyCharm Community 2019.3 edition. PyCharm is a very powerful IDE equipped with intelligent python editor, graphical debugger and test runner for allowing single-step debugging and even compiling code at runtime, navigation and refactoring, code inspections and more[7].

4.3 Proposed SDC Implementation Model

In this proposed model, the Client first encodes the data utilizing the SDC encryption algorithm which supports homomorphic operations on ciphertexts. The encrypted data is then forwarded to the storage server across the Broker or Cloud Service Provider. The client then Requests the Broker to perform a specific operation F {Enc (d1, d2)} on the ciphertext. The Broker executes this operation on the ciphertext in the fifth stage without unsettling the original content and the results stored in the cloud again. The altered outcomes will be showed when the client decrypts the encrypted data with the secret key in the last step.

The steps involved in the conceptual model shown in fig. 1 are as follows;

Step 1: Key Generation

Step 2: Data Encryption, Enc(d1, d2)

Step 3: Upload Data into the cloud

Step 4: User Computations F(Enc(d1, d2))

The user's computation may either be an Addition or multiplication or both. Thus, the computation becomes F(Enc(d1+d2)) or F(Enc(d1*d2)) depending on the request.

Step 5: SDC Encryption F(Enc(d1, d2))

Step 6: (Enc(d1, d2)) returned

Step 7: User Decrypts (Enc(d1, d2))

Step 8: Evaluation: (Enc(d1, d2))

4.4 Data in Transit Encryption

As mentioned earlier, in securing data in transit, encrypted channels, for instance, Transport Layer Security (TLS) is utilized in conveying data throughout the cloud network to protect its confidentiality but these cryptographical encryption protocols though have been utilized for many years lack support for computations on encrypted data. IPsec is also an alternative transit encryption protocol utilized for securing VPN tunnels using notable algorithms like 3DES and AES but it is also in the vain lack computation for data moving from clients to web-facing service on the cloud and data transiting among devices inside the cloud or amid different Cloud Service Providers (CSPs).

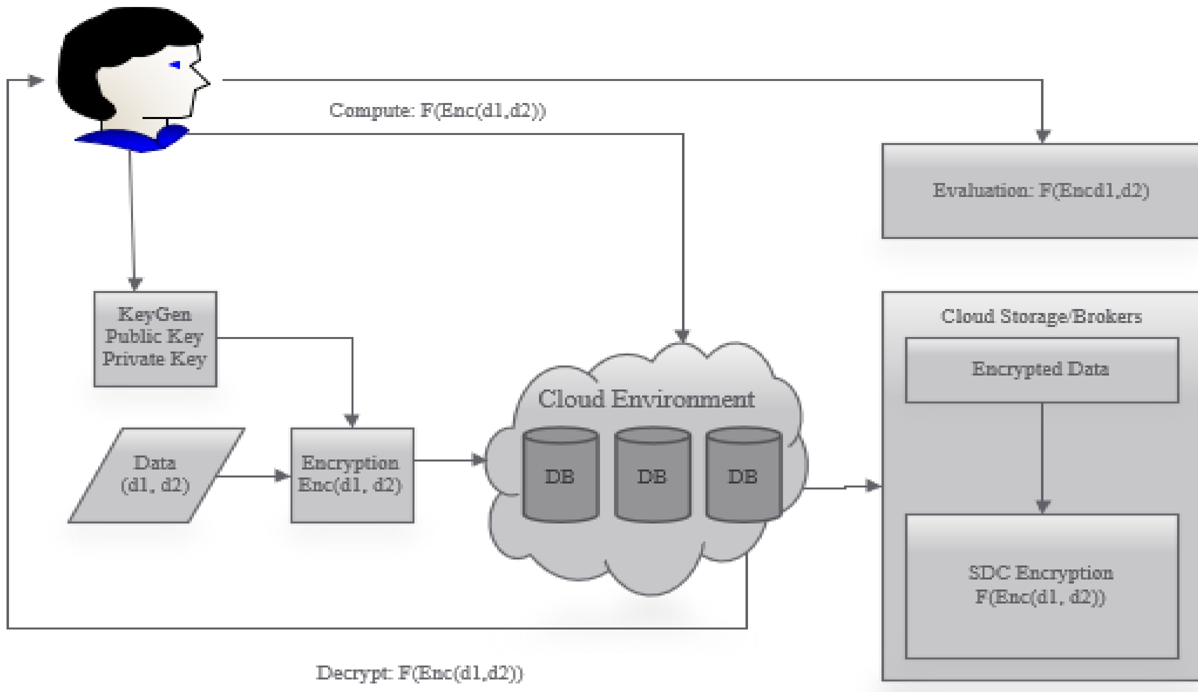

Fig 1: Detailed Conceptual Framework of SDC Encryption Algorithm

5. PERFORMANCE MEASURES

The purpose of this segment is to analyze the efficiency of the algorithm as well as its correctness. The measure of the efficiency of any algorithm is usually determined by the time complexity as well as the space complexity of the implemented algorithm. An analysis of the time needed in carrying out encryptions, decryptions and homomorphic evaluations of a unique size involves time complexity while an analysis of the computer memory needed involves the space complexity of the algorithm. To ascertain the correctness of the algorithm's implementation, the decryption of an encrypted message to its original plaintext and performing homomorphic operations on an encrypted message before the decryption process is varied.

The Θ-notation is key in leading programmers of algorithms in pursuit of efficient algorithms aimed at solving problems. In computing the time complexity of algorithms, the input sizes of both the encryption and decryption algorithms must first be analyzed[8]. The input sizes may either be binary integers with time complexity of Θ (n) or decimal digits with time complexity of Θ(log(n)) with exception of constant numbers having a time complexity of Θ (1) and assume n as the size of entered numbers.

5.1 Big Θ Notation (Time Complexity)

5.1.1 SDC Algorithm

The Encryption Function:

$$C = m + p + r.p.q$$

$$\Rightarrow T(C) = T(m) + T(p) + T(r.p.q)$$

$$T(C) = \Theta(n) + \Theta(n) + \Theta(n^2)$$

$$T(C) = \Theta(n^2)\ \ bit\ operations$$

The Decryption Function:

$$m = (c\ mod\ p)$$

$$\Rightarrow T(m) = T(c\ mod\ p)$$

$$T(m) = T(c)\ .T(\ mod\ p)$$

$$T(m) = \Theta(n)\ .\Theta(n)$$

$$T(m) = \Theta(n^2)\ \ bit\ operations$$

5.1.2 DGHV Algorithm

The Encryption Function:

$$C = p.q + 2r + \ m$$

$$\Rightarrow T(C) = T(p.q) + T(2r) + T(m)$$
$$T(C) = T(p)\ .T(q) + T(2)\ .\ T(r) + T(m)$$
$$T(C) = \Theta(n)\ .\Theta(n) + \Theta(1)\ .\ \Theta(n) + \Theta(n)$$
$$T(C) = \Theta(n^2) + \Theta(n) + \Theta(n)$$
$$T(C) = \Theta(n^2)\ \ bit\ operations$$

The Decryption Function:

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

$$\Rightarrow T(m) = T((c\ mod\ p)\ mod\ 2)$$
$$T(m) = T(c)\ .T(mod\ p)\ .T(\ mod\ 2)$$
$$T(m) = \Theta(n)\ .\Theta(n)\ .\Theta(1)$$
$$T(m) = \Theta(n^2)\ \ bit\ operations$$

5.1.3 Gen10 Algorithm

Suppose the size of the entered message is n, then;

The Encryption Function:

$$C = p.q + m$$

$$\Rightarrow T(C) = T(p.q) + T(m)$$
$$T(C) = T(p)\ .T(q) + T(m)$$
$$T(C) = \Theta(n)\ .\Theta(n) + \Theta(n)$$
$$T(C) = \Theta(n^2) + \Theta(n)$$

$T(C) = \Theta(n^2)$ *bit operations*

The Decryption Function:

$$m = c \bmod p$$

$T(m) = \Theta(c \bmod p)$
$T(m) = \Theta(c) \,.\, \Theta(\bmod p)$
$T(m) = \Theta(n) \,.\, \Theta(n)$
$T(m) = \Theta(n^2)$ *bit operations*

5.2 Space Complexity

It is obviously necessary to know if the algorithm will generate outputs in either microseconds, seconds, minutes, hours, days, weeks or even years so that the essential memory required for that execution process will be allocated. When the length of the message is the same as well as the parameters p, q, and r, the SDC encryption algorithm performs steadily better than the others[9]. Table 1 and Fig.2 shows the performance of the SDC encryption algorithm with other algorithms.

Table 1. Running Time of Gen10, DGHV and SDC algorithms

Message Size	Gen10(m.s.)	DGHV (m.s.)	SDC (m.s.)
11 bytes	906	1017	1079
1.3k bytes	20717	1241719	1282967
2.7k bytes	72800	6715047	4425798
5.6k bytes	145600	13430094	8851596
8.4k bytes	218400	20145141	13277394

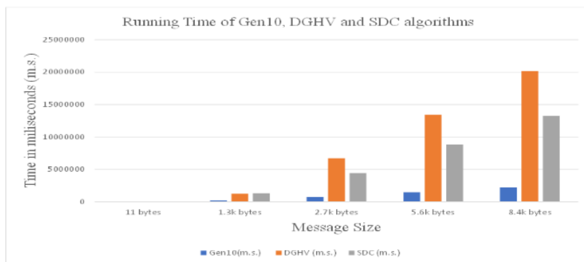

Fig 2: Running Time of Gen10, DGHV and SDC algorithms respectively

5.3 Discussion

The time complexities of the encryption and decryption functions of Gen10, DGHV and SDC algorithms are identical, which is $\Theta(n^2)$.. The growth rate of these functions depends on the size of the message.

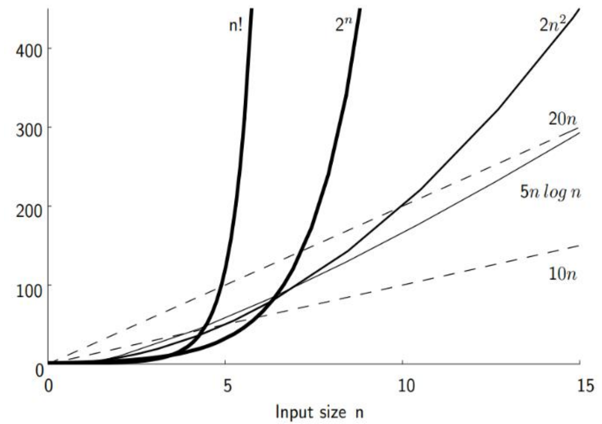

Fig.3: A Graph of Time/Space Against Input Size

Fig.3 illustrates a graph for six equations, each describes the running time for a particular algorithm. The two equations labelled *10n* and *20n* are graphed by straight lines, growth rates of such nature are called linear growth rates or running time. Thus, for a growth rate of *cn* (for c any positive constant), as the value of n grows, the running time of the algorithm grows in the same proportion. Doubling the value of n roughly doubles the running time. An algorithm whose running-time equation has a highest-order term containing a factor of n^2 such as the Gen10, DGHV and SDC are said to have a quadratic growth rate. In Fig. 3, the line labelled $2n^2$ represents a quadratic growth rate. The line labelled 2^n represents an exponential growth rate. The line labelled n! is also growing exponentially.

Also, from Fig. 3, the difference between an algorithm whose running time has a cost of T(n) = 10n and another with cost $T(n) = 2n^2$ becomes tremendous as n grows. For instance, as n > 5, the algorithm with running time $T(n) = 2n^2$ becomes much slower, irrespective of the fact that 10n has a greater constant factor than $2n^2$. Comparing the two curves marked 20n and $2n^2$ shows that varying the constant factor for one of the equations only shifts the point at which the two curves intersect. Considering n > 10, the algorithm with cost $T(n) = 2n^2$ is slower than the algorithm with cost T(n) = 20n. This graph also shows that the equation T(n) = 5n log n grows somewhat more quickly than both T(n) = 10n and T(n) = 20n, but not nearly so quickly as the equation $T(n) = 2n^2$.

It is clear from Table 1 and Fig.2 that Gen10 requires less time than even the SDC encryption but in terms of security and ciphertext retrieval, it stands a zero chance with the SDC encryption algorithm. As mentioned earlier, ciphertext recovery algorithm for the DGHV ought to move the secret key to the server, and this appears to be horrendously shaky. While, the ciphertext recovery algorithm of Gen10 requests to tender q to the server yet uses c mod q, where q is an irregular number and c is the ciphertext, yet the plaintext spills out. Accordingly, [6] proposed algorithm solves the problem of ciphertext recovery well, deprived of plaintext spill out, in light of the fact that the procedure of decoding utilizes the secret key p so that the recovery procedure utilizes the whole number q, that is completely unique.

6. RELATED REVIEW

In early 2009, [2] put forward the initial plausible fully homomorphic encryption (FHE). Gentry's algorithm was based on lattice cryptography and it supported addition and multiplication procedures mutually on ciphertexts. This addition and multiplication operations relate to AND (∧) and XOR (⊕) operations in Boolean algebra respectively. This is

outstanding because it provided the basis for many functions to be derived from them [2]. For instance, ¬A can be derived from A ⊕ 1, and (¬A) ∧ (¬B) can also be derived from A ∨ B, and then transformed to (A ⊕ 1) ^ (B ⊕ 1). The common term for the construction of cryptographic primitives (encryption functions) that involve itself is known as lattice-based cryptography and any basis of R^n the subgroup of every single linear combination with integer coefficients of the basis vectors forms a lattice.

Gentry's lattice-based cryptography comprises of numerous stages which start based on what was suggested to as somewhat homomorphic encryption (SWH) algorithm utilizing ideal lattices which are restricted to assessing low degree polynomials over scrambled data. This restriction to some extent is as a result of the noise in each ciphertext and as more computations (additions and or multiplications) are executed on the ciphertext, this noise grows until eventually, the noise brands the resultant ciphertext undecryptable. Afterwards, it jams the decryption process with the goal that it tends to be communicated as a small degree polynomial that is upheld by the algorithm. Then lastly, it uses a bootstrapping transformation, by means of an iterative self-implanting, to acquire a fully homomorphic algorithm [10].

Subsequently, a fully homomorphic encryption known as the DGHV algorithm, which improves upon the Gentry cryptosystem by showing that the Somewhat Homomorphic constituent of the ideal lattices can be supplanted with an easier homomorphic algorithm which utilizes integers instead [10]. This algorithm is, thus, theoretically uncomplicated as compared with the Gentry cryptosystem, in any case, has comparative qualities as for homomorphic tasks and effectiveness. A DGHV fully homomorphic public-key encryption algorithm consists of a number of sub algorithms. These incorporate the typical KeyGen, Encrypt, Decrypt, and an extra significant algorithm known as Evaluate. KeyGen, as usual, is a large odd integer (for instance p) which is chosen at random and the complexity of the algorithm depends on how easy it is to factorize this odd integer. In order to Encrypt (p, m) a bit of a message, the ciphertext is established as an integer with residue mod p and has a similar equivalence as the plaintext. Viz., set

$$c = pq + 2r + m$$

where the integers q and r are selected indiscriminately in some other recommended intervals, with the end goal that 2r is lesser than p/2 in absolute value. The r represents the noise which is adequately lesser than the private key p and therefore the Decrypt (p, c) outputs (c mod p) mod 2. The Evaluate the public key pk as input. This uncomplicated algorithm is together additive and multiplicative homomorphic with respect to low mathematical computations and one can also utilize bootstrapping and squashing to morph this algorithm into FHE [6] [11]. In the work of [10], he developed a framework for this algorithm that should be easy to use, not too dependent on the security measures taken by end-clients, have the option to handle any cryptographic operations within the trusted infrastructure, be able to send encoded data to the cloud and the public clouds where the encoded data is stored ought not to possess the capability to decode its contents. This proposed implementation structure is exemplified in fig.4.

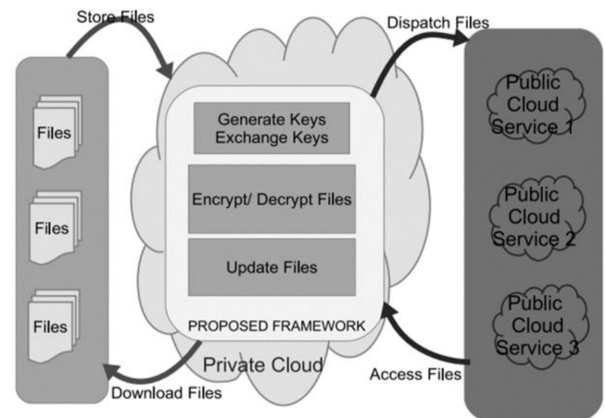

Fig.4: Framework Usage Scenario

An improved homomorphic encryption algorithm known as, Gen10 algorithm was proposed in the publication of the Communications of the ACM [12], making a beeline for far-reaching utilization of cloud computing, which was amazingly basic and of the structure

$$c = pq + m$$

The c represents the encrypted message (ciphertext), m represents the unencrypted message (plaintext), while p represents the key and q an arbitrary numeral [12]. This encryption procedure is, therefore, homomorphic in regard to addition, subtraction and multiplication. There exists a connection among c and m such that m is the remainder of c regarding modulus p, that is,

$$m = c \ mod \ p$$

For this algorithm, the encryption is such that; KeyGen is an arbitrary P-bit odd integer p and to Encrypt (p, m) a bit, let M represent an arbitrary N-bit number such that

$$M = m \ mod \ 2$$

So, the output of the ciphertext becomes

$$c \leftarrow M + pq$$

where q is an arbitrary Q-bit number. The Decrypt (p, c) Outputs c mod p, where (c mod p) is the integer C in (-p/2, p/2) such that p| (c – C).

The Gentry cryptosystem, DGHV and Gen10 encourage the addition and multiplication upon encrypted data, nevertheless, neither of the three have made references on the cyphertext recovery algorithms. Therefore, [6] proposed a simple FHE known as the SDC algorithm, which is also based on the Gentry cryptographic encryption algorithm to safeguard data confidentiality in cloud environments. An illustration for this SDC algorithm is described below:

The KeyGen(p): Where p is an arbitrary P-bit odd integer and to Encrypt (p, m) a message or bit {0,1}

$$c = m + p + rpq$$

where r is an arbitrary R-bit numeral and q is consistent of Q-bit enormous whole integer and c the ciphertext. The Decrypt (p, c) Output (c mod p) and the Retrieval(c):

$Ri = (c_i – c_{index}) \bmod q$

As soon as the customer wishes to retrieve message m_{index}, he encodes the keywords

$c_{index} = m_{index} + p + rpq$

and transports c_{index} to the server. In receipt of c_{index}, the server inspects the ciphertexts, computing

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

once R = 0, ciphertext retrieval works, and C_i is the anticipated outcome [10].

In summary, the ciphertext recovery algorithm in the DGHV requires the movement of the secret key to the server, which appears to be horrendously shaky. While, the ciphertext recovery algorithm of Gentry's work named Gen10 and GSW requests to present q to the server in spite of that uses c mod q, where q is an irregular number and c the ciphertext, yet the plaintext spills out. Consequently, [6] proposed algorithm solves the problem of ciphertext recovery successfully, without plaintext spill out, in light of the fact that the procedure of decoding utilizes the secret key p although the recovery procedure utilizes the whole number q, that is completely unique. Consequently, fulfils both the interest for ciphertext recovery and information security.

7. CONCLUSION, RECOMMENDATION AND FINDINGS

The main points of evidence for this study are that homomorphic encryption enables clients to process confidential data on an untrusted device such as another person's computer which is a server in the clouds and would like to guarantee the confidentiality of data, conventional approaches of encryption can only safeguard their data while it is in transit, however, not when the data is being processed.

Homomorphic encryption, then again, can secure data beginning the time the data packets exit the computer until the time it finally makes its way back. The processes involved in the proposed conceptual framework of SDC Encryption Algorithm are Key Generation, Data Encryption, Enc (d1, d2), Upload Data into the cloud, User Computations F(Enc(d1, d2)), SDC Encryption F(Enc(d1, d2)), (Enc(d1, d2)) returned, User Decrypts (Enc(d1, d2)) and Evaluation: (Enc(d1, d2)) as illustrated in fig 5.

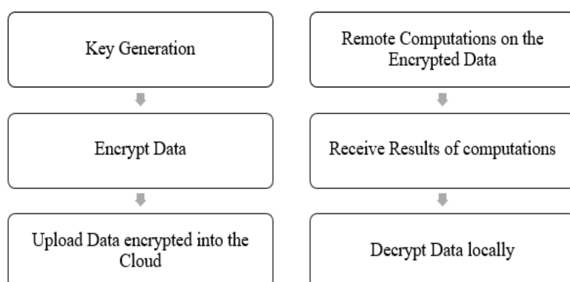

Fig.5: Stream of SDC Encryption Algorithm

The time complexities of the encryption and decryption functions of Gen10, DGHV and SDC algorithms are identical, which is Θ (n^2). The growth rate of these functions depends on the size of the message. Doubling the value of n roughly doubles the running time as seen with functions of Gen10, DGHV and SDC algorithms. It is clear that Gen10 requires less time than even the SDC encryption but in terms of security and ciphertext retrieval, it stands a zero chance with the SDC encryption algorithm. The correctness of this algorithm depends on the time and space which is illustrated.

The key contribution of this research is the proposal of the implementation framework for the SDC algorithm and the measure of the correctness of the SDC algorithm.

The objectives of this research have become contributions to knowledge since they were successfully fulfilled. Knowledge is an ending process, hence in the future, the research will explore the possibility repackaging the SDC encryption algorithm as cloudlet application that can be called in a cloud simulator and also explore the prospect of encrypting data using SDC encryption algorithm with Transport Layer Security (TLS).

8. REFERENCES

[1] N. Lord, "DigitalGuardian," 11 September 2018. [Online]. Available: https://digitalguardian.com/blog/cryptography-cloud-securing-cloud-data-encryption. [Accessed 5 November 2019].

[2] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," ACM, pp. 169-178, 2009.

[3] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in Proceedings of the 31st Annual Conference on Advances in Cryptology, Berlin,Heidelberg: Springer-Verlag, 2011.

[4] C. Gentry, S. Halevi and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques, Berlin, 2012.

[5] D. R. Korda, "Securing Data in the Clouds Using SDC Encryption Algorithm," Department of Computer Science, KNUST, Kumasi, 2020.

[6] J. Li, S. Song, S. Chen and X. Lu, "A Simple Fully Homomorphic Encryption Scheme Available In Cloud Computing," IEEE, pp. 214-217, 2012.

[7] JetNeans, "PyCharm," PyCharm, 2020. [Online]. Available: https://www.jetbeans/pycharm/. [Accessed 13 March 2020].

[8] A. M. Sagheer, "Elliptic Curves Cryptographic Techniques," in IEEE, 2012.

[9] S. S. Hamad and A. M. Sagheer, "Design of Fully Homomorphic Encryption by Prime Modular Operation," Telfor Journal, vol. 10, no. 2, 2018.

[10] I. Jabbar and S. Najim, "Using Fully Homomorphic Encryption to Secure Cloud Computing," Internet of Things and Cloud Computing, pp. 13-18, 2016.

[11] D. K. M. Hodowu, D. R. Korda and E. D. Ansong, "An Enhancement of Data Security in Cloud Computing with an Implementation of a Two-Level Cryptographic Technique, using AES and ECC Algorithm," International Journal of Engineering Research & Technology, vol. 9, no. 9, 2020.

[12] K. M. Mohanty, "Secure Data Storage on the Cloud using Homomorphic Encrypyion," National Institute of Technology, Rourkela, 2013.

[13] C. Gentry, "Computing Arbitrary Functions of Encrypted Data.," Publications of ACM, vol. 53, no. 3, pp. 97-105, 2010.