International Journal of Computer Networks and Applications (IJCNA)
DOI: 10.22247/ijcna/2022/212332 Volume 9, Issue 2, March – April (2022)

RESEARCH ARTICLE

A Symmetric, Probabilistic, Non-Circuit Based Fully Homomorphic Encryption Scheme

George Asante
Department of Information Technology Education, Akenten Appiah-Menka University of Skills Training and Entrepreneurial Development, Ghana
geosante@yahoo.com

James Ben Hayfron-Acquah
Department of Computer Science, Kwame Nkrumah University of Science and Technology, Ghana
jbha@yahoo.com

Michael Asante
Department of Computer Science, Kwame Nkrumah University of Science and Technology, Ghana
mickasst@yahoo.com

Joshua Caleb Dagadu
Department of Information Technology Education, Akenten Appiah-Menka University of Skills Training and Entrepreneurial Development, Ghana
joscaldag@yahoo.com

Received: 04 January 2022 / Revised: 22 February 2022 / Accepted: 06 March 2022 / Published: 30 April 2022

Abstract – Traditional encryption allows encrypted data to be decrypted before any computation could be performed on such data. This approach could compromise the security of the data when an untrusted party is involved in the computation. To be able to work on data in its encrypted form, a homomorphic encryption approach is recommended. Homomorphic encryption allows computation to be done on data that has been encrypted and yields the same results that would have been obtained if the computation had been performed on the unencrypted form of the data. Most of the Homomorphic encryption (HE) algorithms are deterministic. These deterministic algorithms produce the same ciphertext for a given data on different occasions. This could allow an adversary to easily predict a plaintext from a ciphertext. Probabilistic algorithms, however, resolve the aforementioned challenge of deterministic algorithms. A probabilistic encryption algorithm ensures different ciphertexts for the same plaintext on different occasions. Another challenge of most homomorphic encryption schemes is the way data is encrypted. Most algorithms encrypt data bit-by-bit (i.e. circuit-based). Circuit-based encryption makes the encryption and decryption complex, thereby increasing the running time. To reduce the running time, Non-Circuit based encryption and decryption are preferred. Here, numeric data need not be converted to binary before any encryption is done. To ensure a very secure, efficient but simpler HE scheme, the authors have offered a fully homomorphic encryption (FHE) scheme that is Probabilistic, Non-Circuit based, and uses symmetric keys. Results from the experiment conducted show that the proposed scheme is faster than Fully Homomorphic Encryption over the Integer (DGHV), A simple Fully Homomorphic Encryption Scheme Available in Cloud Computing (SDC), and Fully Homomorphic Encryption by Prime Modular Operation (SAM) schemes. The proposed scheme has a time complexity of $O(\log(n^2))$ and consumes less memory space. Even though HE schemes are naturally slow, the less memory space consumed by the proposed scheme and the time complexity of $O(\log(n^2))$, makes the proposed scheme suitable for real-life implementation such as auction, electronic voting, and in other applications that make use of private data.

Index Terms – Probabilistic Encryption, Homomorphic Encryption, Non-Circuit Based Encryption, Information Security, Data, Symmetric Keys, Cryptography.

1. INTRODUCTION

The use of Information Technology in our everyday life poses risks and associated threats that affect the confidentiality and integrity of data [1]. Cryptography is highly recommended when one wants to ensure data confidentiality, data integrity, or authentication [2]. There are always security challenges when an untrusted user is involved in the computation of data [1]. With an untrusted user, the data must be worked on in its encrypted form. Thus, the need for HE. HE helps one to compute encrypted data without decrypting ciphertext [3]. With HE, all computations that could be performed on

RESEARCH ARTICLE

plaintexts could also be performed on ciphertexts and yield the same results.

Most of the HE schemes operate on binary data. These include: [4, 5,6,7,8,9,10,11,12,13]. These schemes are considered circuit-based and are complex in structure [9]. HE schemes that operate on real numbers are considered to be faster since data need not be converted to binary format before any operation could be performed on it [14]. These non-circuit-based schemes use simple mathematical functions and do not require additional circuit computation overheads thereby reducing ciphertext size and speeding up computation time [15]. Most of the encryption schemes that operate on binary data [4, 5,6,7,8,9,10,11,12,13] are deterministic [16] in nature. These deterministic algorithms produce the same ciphertext for a given data on different occasions. This could allow an adversary to easily predict a plaintext from a ciphertext. Probabilistic algorithms, however, resolve the aforementioned challenge of deterministic algorithms. A probabilistic encryption algorithm ensures different ciphertexts for the same plaintext on different occasions.

Furthermore, the public-key cryptography technique is used by most proposed homomorphic encryption schemes. Many suggest that public-key cryptography is more secure than private-key cryptography. However, public-key cryptography is more complex and consumes more resources comparatively [17]. Also, private-key cryptography could be secured if the private key involved could be exchanged securely. Moreover, some applications would inherently require a secret key [9].

Though there have been many proposed HE schemes, much has not been exploited on an FHE scheme that is probabilistic, symmetric [17, 18], and non-circuit based with function privacy [19]. The main focus of this study was to secure data using an FHE scheme. Specifically, the concern was to design an FHE scheme that uses a probabilistic approach in generating ciphertexts; that makes use of symmetric keys, and that is non-circuit based. HE schemes are naturally slow. This inhibits its use in real-life applications. The less memory space consumed by the proposed scheme and the time complexity of O ($\log(n^2)$) makes the proposed scheme appropriate for real-life implementation.

This paper has been structured as follows: Related studies are reviewed in section 2. The tools and methods used in designing the proposed scheme are explained in section 3. Discussion of Results was done in section 4, and finally, the paper was concluded in section 5.

2. RELATED WORK

Since the inception of the HE scheme by Rivest, Adleman, and Dertouzous in 1978 (Rivest et al, 1978 as cited in [1]), there has been improvement in either speed, security, efficiency, or simplicity of the scheme [1].

Before Gentry's proposal of an FHE scheme in 2009[4], HE schemes were either partial or somewhat homomorphic encryption schemes [1]. Since then, numerous variants of Gentry's encryption scheme have been introduced.

For this study, this section reviews three FHE schemes: DGHV, SDC, and SAM Schemes. The review is mainly based on the description of algorithms, and the Time and Space complexities of these Schemes.

2.1. DGHV Scheme

[5] Proposed DGHV Scheme in 2010 [5]. Their scheme was a true version of Gentry's (Gentry, 2009) scheme [14]. Their scheme has the following components: Key generation (KeyGen), Encryption (Encrypt), Decryption (Decrypt), and Evaluation (Evaluate).

KeyGen: An odd integer p is generated as key, $p \in [2^{n-1}, 2^n)$

Encrypt (p, m): set $c = pq + 2r + m$, $m \in \{0,1\}$; p, q is a very big multiple of the key and r is a smaller even number. "2r is always smaller than p/2." [5]

Decrypt (p, c): set m = (c mod p) mod 2

Evaluate: Given the binary circuit with ciphertexts, the ciphertexts are added or multiplied.

The time complexity of the DGHV scheme is $O(n^2)$. The space complexity is also O(1).

The number of homomorphic operations is limited in DGHV. DGHV operations are restricted to binary operations. It means DGHV works only on plaintext in bits form. Processing in bits requires a higher number of operations in computation. This makes the scheme slower. DGHV also uses the public-key technique. This makes the algorithm complex with a lot of overheads. DGHV uses a very long public key which makes it difficult to be used in practice. DGHV scheme is, therefore, not fit for real-life applications.

2.2. SDC Scheme

SDC scheme was proposed by [20] in 2012. Their scheme was a version of the Gentry's cryptosystem. Their scheme was to ensure privacy in storing data in the cloud. SDC's scheme involves the effective retrieval of encrypted data. The scheme uses only elementary modular arithmetic [20]. The scheme has the following components:

Key Generation (KeyGen), Encryption (Encrypt), Decryption (Decrypt) and Retrieval (Retrieval).

In Key Generation, a random odd integer P is generated as a key. During the encryption stage, the ciphertext is generated using the expression c = m + p + r * p * q. $m \in \{0,1\}$. Where r and q are integers. The decryption stage uses the algorithm m = c mod p.

RESEARCH ARTICLE

The ciphertext is retrieved at the final stage using the algorithm $R = (c_i - c_{index}) \bmod q$.

The time complexity of the SDC scheme is $O(n^2)$. Similar to the DGHV scheme, the SDC scheme also has five variables in the encryption algorithm. So total memory requirement is 4 *5 = 20 bytes. The 20 bytes space is fixed, it does not change in the process of executing the algorithm. The space complexity is, therefore, O(1).

Similar to DGHV, SDC operations are also restricted to binary data. This makes the algorithm slower since there are a lot of operations that are performed in computation.

2.3. SAM Scheme

SAM Scheme was proposed by Sarah Shihab Hamad and Ali Makki Sagheer in 2018. Unlike DGHV and SDC schemes that convert plaintext to 8-bit binary format before encryption, the SAM scheme takes the plaintext directly and encrypt using the algorithm: c = m + r*p + p*q, where c is the ciphertext and m is the plaintext. $m \in [0, p-1]$, p is a big prime integer, q is a constant integer, and r is the noise [17]. The scheme has the following components:

KeyGen: generate secret key p as a big prime integer

Encrypt(pk, $m \in [0, p-1]$): encrypt plaintext using the equation c = m + r*p + p*q.

Evaluate (pk, $m_1 \ldots m_t$, $c_{1, \ldots} c_t$): add or multiply t ciphertexts to get c_i, and then decrypt the c_i.

Decrypt (sk, c): Set m=c mod p.

The time complexity of the SAM scheme is $O(\log(n^2))$. Similar to DGHV and SDC schemes, the SAM scheme also has five variables in the encryption algorithm. So total memory requirement is 4 *5 = 20 bytes. The 20 bytes space is fixed, it does not change in the process of executing the algorithm. The space complexity is, therefore, O(1).

Though SAM's scheme is faster than DGHV and SDC, the number of primitive operations involved in the encryption algorithm will increase the running time of the algorithm. Therefore, an algorithm with less primitive operations is preferred.

3. METHODS AND MATERIALS

This section describes the various approaches used in achieving the stated objectives.

3.1. The Proposed Scheme

The proposed scheme takes plaintext (m) and generates two random numbers (p and r). One of the random numbers (p) is taken as a private key. This key (p) is used to decrypt the ciphertext when generated. The algorithms for encryption and decryption are discussed in Sections 3.1.2 and 3.1.3.

The scheme uses only numeric values. Therefore, a non-numeric plaintext is converted to numeric using its ASCII code equivalent. Thus, Plaintext in non-numeric form is converted to its corresponding ASCII code before any encryption could be done. For instance, a plaintext "a" is converted to its ASCII code "65", the 65 is then encrypted using the appropriate algorithm. To apply the homomorphic encryption scheme to numeric data, four basic algorithms were used. They are Key generation, encryption, decryption, and evaluation algorithms.

These algorithms have the following description:

KeyGen: select the secret key p to be a randomly generated large prime positive integer. The key must have a fixed length and must be within a given range.

Encrypt(p, $m \in Z^+$): Encrypt a plaintext (m) using the encryption equation $c = m + (p * r)$. Here c is the ciphertext, m is the plaintext (numeric or ASCII code of a character), p is the randomly generated large positive prime integer and r is the noise, which is also a randomly generated large positive prime integer.

Evaluate ($c_1 \ldots c_t$): add or multiply t ciphertexts and get c_i. When c_i is decrypted it yields the same results as adding or multiplying t plaintexts.

Decrypt (p, c): decrypt ciphertext c using the decryption equation m = c mod p.

These components of the scheme are explained in the next sections. Section 3.1.1, 3.12, 3.13, 3.14 gives a detailed explanation of the four algorithms.

3.1.1. The Proposed Key Generation Algorithm

The proposed key generation algorithm selects randomly a symmetric key for the encryption and decryption. The random selection of the symmetric key makes the encryption scheme probabilistic. The random number selected must also be a prime number. The key's size depends on the number of digits used as the parameter in the key generation algorithm. A minimum of 10 digits is preferred. The longer the size of the key, the more secure the key is. However, the size of the key affects the ciphertext size.

3.1.2. The Proposed Encryption Algorithm

The proposed encryption algorithm takes the plaintext m, the symmetric key p, and a random number r and transforms the plaintext into ciphertext c. The inclusion of a random number makes it difficult for an adversary to predict the plaintext even when the symmetric key is known. The encryption algorithm is C = m + (p * r)

The plaintext must be less than the encryption key. If the plaintext is equal to or greater than the encryption key, the decryption results will be undesirable. For this reason, the

RESEARCH ARTICLE

encryption key must be large enough. The plaintext must also be a positive integer. Using negative values yield wrong results. The programmer can take care of negative values by appending the negative sign to the decrypted results.

3.1.3. The Proposed Decryption algorithm

The proposed decryption algorithm uses modular arithmetic. The decryption algorithm uses the symmetric key to convert the ciphertext into plaintext. ciphertext modulo symmetric key gives the plaintext value. That is, the remainder of dividing the ciphertext by the symmetric key is the plaintext. m = c % p

3.1.4. The Evaluation Operations

The evaluation operations, for example, Addition and multiplication, are performed on ciphertext in the same manner as it is done on the plaintext. For example, for ciphertexts c1, c2, and c3, addition is performed as c1+c2+c3, and multiplication is performed as c1*c2*c3. Other computations such as comparing ciphertexts or ordering ciphertexts could be performed on the ciphertexts. The evaluation is simply applying the various mathematical operations on the ciphertexts.

3.2. The Experimentation

This section deals with the implementation of the scheme. The various components of the scheme were tested with actual data.

3.2.1. Experimental Set-Up

All the algorithms stated in this study were implemented using the Java Programming Language. The Integrated Development Environment (IDE) used was NetBeans. The codes were run on an intel® Core ™ i7-7500U CPU @ 2.70 GHz 2.90 GHz on 64-bit Windows 10 Operating System, an x64-based processor with 8.00 GB Installed memory (RAM).

3.2.2. Key Generation

Table 1: Results of p and r Values Generated Using the Key Generation Algorithm

Count	*P value*	*r* value
1	1089333481	1329746303
2	1680005731	1283849519
3	1440361501	1333286827
4	1532538439	1067249569
5	1573836631	1911291269
6	1277045633	1153954427

Math. random () method in Java was used to generate the random numbers. Minimum and maximum values of 10-digits long were used as arguments in the Math. random method to serve as the range to generate the random numbers from. A number generated was then verified as a prime. The result is shown in Table 1.

3.2.3. Encryption

The encryption algorithm is c = m + (p*r). Assuming m = 4, p = 1453609901 and r = 1851786169.

Then c = 4 + (1453609901 * 1453609901)

c = 2691774709793259273

3.2.4. Decryption

The decryption algorithm is m = c % p. Assuming c=2691774709793259273 and p =1453609901

Then m = 2691774709793259273 % 1453609901

m = 4

3.2.5. Evaluation

The ciphertexts were evaluated the same way addition and multiplication are performed on numeric values.

Assuming we have three numeric plaintexts m1, m2, and m3. Let m1 =4, m2=6, and m3=8. Let the symmetric key (p) and random number (r) be p = 1358972137; r = 1487563225. Let c1, c2, and c3 be the ciphertext of m1, m2, and m3 respectively.

//Encryption Formulas

c1 = m1 + (p*r)

c2 = m2 + (p*r)

c3 = m3 + (p*r)

// Encrypting Plaintexts

c1 =4 + (1358972137 * 1487563225)

c2 =6 + (1358972137 * 1487563225)

c3 =8 + (1358972137 * 1487563225)

// ciphertexts

c1 = 2,021,556,974,800,861,829

c2 = 2,021,556,974,800,861,831

c3 = 2,021,556,974,800,861,833

//addition of ciphertext

adding c1, c2 and c3 will produce the following:

CombinedCiphertext = c1 +c2 +c3

CombinedCiphertext = 2,021,556,974,800,861,829 + 2,021,556,974,800,861,831 + 2,021,556,974,800,861,833

RESEARCH ARTICLE

CombinedCiphertext = 6,064,670,924,402,585,493

//multiplication of ciphertext

Multiplying c1, c2, and c3 will produce the following:

ProductCiphertext = c1 * c2 * c3

ProductCiphertext = 2,021,556,974,800,861,829 * 2,021,556,974,800,861,831 * 2,021,556,974,800,861,833

ProductCiphertext = 8261481934180097218200820440722338368364284767742471867.

//Decrypting ciphertexts

Decrypting CombinedCiphertext will yield the following:

DecCombinedCiphertext = CombinedCiphertext mod p

DecCombinedCiphertext = 6,064,670,924,402,585,493 mod 1358972137

DecCombinedCiphertext = 18 // proof: 4+6+8 = 18

Decrypting ProductCiphertext will yield the following:

DecProductCiphertext = ProductCiphertext mod p

DecProductCiphertext = 8261481934180097218200820440722338368364284767742471867 mod 1358972137

DecProductCiphertext = 192 // proof: 4*6*8 = 192

4. RESULTS AND DISCUSSION

This section analyzes the results obtained when the various proposed algorithms were implemented. The main discussion is on performance and security. Specifically, key generation time, encryption time, decryption time, evaluation time, and ciphertext size were measured and analyzed. A comparison was made with other fully homomorphic encryption schemes, in respect of time and space complexity. The algorithms were implemented using the Java Programming Language. BigDecimal class in Java was used to handle the large ciphertext sizes generated. This allows for large data to be handled well. The random () method of the Java MathClass was used to generate the various random numbers. NetBeans Integrated Development Environment was used to write the codes. The codes were run on an intel® Core ™ i7-7500U CPU @ 2.70 GHz 2.90 GHz on 64-bit Windows 10 Operating System, an x64-based processor with 8.00 GB Installed memory (RAM).

4.1. Performance of the Proposed SCHEME

4.1.1. Size of Ciphertext

Ciphertext size was measured by casting the ciphertext to String and finding the length of the String. When a 10-digit p and r are chosen, the ciphertext size is always 19-digit long. Table 2 shows different ciphertexts of 19-digits long.

Table 2 Results of Encrypting Plaintext, 4, Several Times Using Different Values of p and r

Count	p value	r value	Ciphertext
1	1453609901	1851786169	2691774709793259273
2	1769358203	1712993479	3030899063754158241
3	1737304277	1235295917	2146084879964737013
4	1672281913	1179802847	1972962961944006315
5	1527127999	1194191879	1823683854599320125
6	1863357499	1832429831	3414471866985152673

4.1.2. Execution Time

Table 3 Summary of the Execution Time of the Proposed Encryption Algorithm

Size of Plaintext	Execution time (in milliseconds)	Execution time (in Nanoseconds)
1 byte	0.0002	200
6 bytes	0.0012	1200
12 bytes	0.0024	2400
1 kilobyte	0.2048	2048
1.4 kilobyte	0.28672	28672

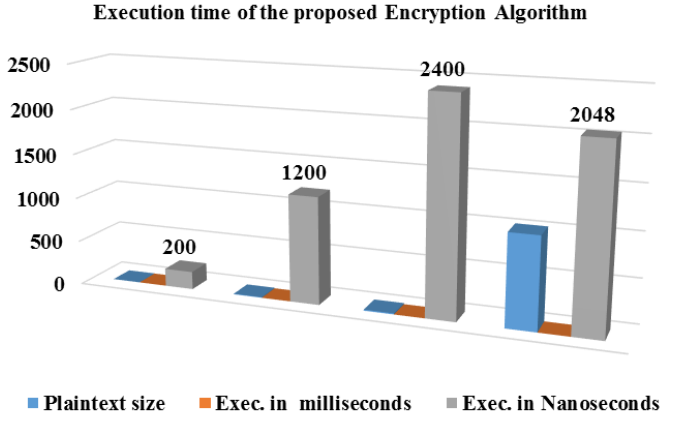

Figure 1 Execution time of the Proposed Encryption Algorithm

RESEARCH ARTICLE

The execution times were measured using Java System.nanoTime(). To make the time useful, the time needs to be generated before and after the statements whose execution time needs to be measured. The difference between the two times generated is the execution time of that Java statement. Table 3 and Figure 1 summarize the execution time of the proposed algorithm using different sizes of plaintext.

4.1.3. Time and Space Complexities

The time complexity of the encryption algorithm could be deduced as follows:

Assuming n is the plaintext's size. Where $n \in Z^+$. Using the encryption algorithm c = m + (p*r), Let T be the time complexity. The time complexity of c will be T(c) = T(m) + T(p*r).

All inputs are in decimal digits. The time complexity of decimal digit is O(log(n))[17].

Therefore, $T(c) = T(m) + T(p*r)$

$= O(\log(n)) + O(\log(n^2))$

$\equiv O(\log(n^2))$

The time complexity of the encryption algorithm is, therefore, $O(\log(n^2))$.

There are four variables in the encryption algorithm. c, p, r, and $m \in Z^+$. All four variables are integer data types; hence these variables will consume a memory space of 4 bytes each. The total memory space required will, therefore, be 4 *4 = 16 bytes. The 16 bytes space is fixed, it does not change in the process of executing the algorithm. The space complexity is, therefore, O(1)

4.2. Security of the Proposed Scheme

4.2.1. Random Prime Number Generation

To make the proposed scheme withstand a Chosen Plaintext Attack (CPA), large random prime numbers p and r were generated. By Prime Number Theorem, there are nearly $\frac{x}{\ln x}$ prime numbers. Where $p \leq x$ [20]. Making p and r random prime numbers, thwart the effort of the attacker to guess the p when having the ciphertext.

One benefit of using a prime as a symmetric key is that there will be only one probable solution when a prime number is used [14]. That is, two plaintexts can never generate the same ciphertext.

4.2.2. Symmetric Key Length

The length of the symmetric key is large enough to withstand a brute force attack. The symmetric key length is Ten decimal digits which are 80 bits long. Even though the key length could be increased to say 12-decimal digits or 15-decimal digits, a very long key length could also affect the multiplication operations. Multiplication operation generates longer ciphertext compared to the Addition operation. It is, therefore, appropriate to keep the length moderate.

4.2.3. One Way Security

One of the securities of the proposed scheme is one-way Security. The knowledge of a given ciphertext does not allow adversaries to retrieve the decryption key. The reason is that the ciphertext does not disclose anything about the decryption key. Therefore, there is one-way security. That is, when an adversary has a ciphertext, an adversary would find it difficult to recover the associated plaintext.

4.2.4. The Indistinguishability Under Chosen-Plaintext Attack (IND-CPA)

The proposed scheme's security could also be described using the indistinguishability property of symmetric encryption. A Symmetric encryption scheme is usually said to possess an indistinguishability property if when given a ciphertext of one of two plaintexts, an adversary will find it difficult to guess which of the two plaintexts generated the given ciphertext [21, 14, 13]. The proposed scheme is Indistinguishable under a chosen-plaintext attack (IND-CPA). Because when given a ciphertext, an attacker will find it difficult to guess its corresponding plaintext due to the randomization of the symmetric key.

4.2.5. Hardness of Large Integer Factorization

Factoring large integers is difficult. The principle of the hardness of large integer factorization explains that Factoring N in polynomial time is unattainable if Large Integer Factorization is unattainable [20]. To ensure that the encryption and decryption key is secure, large integer values p and r were generated.

4.3. Comparison of the Proposed Scheme to Other Schemes

This section compares the performance of the proposed scheme with the other schemes such as DGHV, SDC, and SAM Schemes. The comparison is mainly based on the Time and Space complexity of these Schemes.

4.3.1. Comparison of Time Complexities of Proposed scheme, SAM, SDC, and DGHV Schemes

The time complexity for the encryption algorithm of the proposed scheme is $O(\log(n^2))$. This is a logarithmic time which is quite more efficient than exponential and quadratic time. That is, as quadratic time multiplies the time it takes to run n inputs, the logarithmic time divides. Therefore, reducing the time complexity. From the review, the time complexity of the DGHV and the SDC encryption schemes is $O(n^2)$. This is a quadratic time. It could be easily deduced that the proposed scheme is faster than the DGHV and the SDC schemes. The

RESEARCH ARTICLE

time complexity of the proposed scheme is the same as the SAM scheme. It could be deduced from Table 4 that the proposed scheme has less execution time comparatively. Even though the proposed scheme has the same time complexity as SAM, it is faster than SAM. This is due to a few primitive operations that are performed in the proposed scheme. The proposed scheme is, therefore, suitable for most real-life implementation.

4.3.1.1. Comparison of Space Complexities of the Proposed Scheme, SAM, SDC, and DGHV Schemes

From the above analysis, even though the proposed scheme and SAM scheme have the same Time complexity, the proposed scheme is more efficient than the SAM scheme due to the number of primitive operations involved in the SAM Scheme.

Unlike DGHV, SDC, and SAM schemes which have five variables each in its encryption algorithm, the proposed scheme has four variables in the encryption algorithm. c, m, p, and r. The four variables are all integer types; hence they will consume a memory space of 4 bytes each. The total memory space required is, therefore, 4 *4 = 16 bytes. The 16 bytes space is fixed, it does not change in the process of executing the algorithm. The space complexity is, therefore, O(1). The Summary of the comparison is shown in table 4 and Figure 2.

It could be concluded that the memory requirement of the proposed scheme is lesser than DGHV, SDC, and SAM schemes which take a total memory requirement of 4*5 =20 bytes. The less memory requirement contributed to the speed of the algorithm.

Table 4: Summary of Results

Scheme	DGHV[5]	SDC[14]	SAM[14]	PROPOSED SCHEME
Encryption Algorithm	$c = m + 2r + p * q$	$c = m + p + r * p$	$c = m + r * p + p * q$	$c = m + p * r$
Time Complexity	$O(n^2)$	$O(n^2)$	$O(\log(n^2))$	$O(\log(n^2))$
Space Complexity	O(1)	O(1)	O(1)	O(1)
Memory Requirement	20 $bytes$	20 $bytes$	20 $bytes$	16 bytes
Operation	Bit operation	Bit operation	Decimal operation	Decimal operation
Execution time (12 bytes)	1118 [14]	1180 [14]	1007 [14]	0.0024

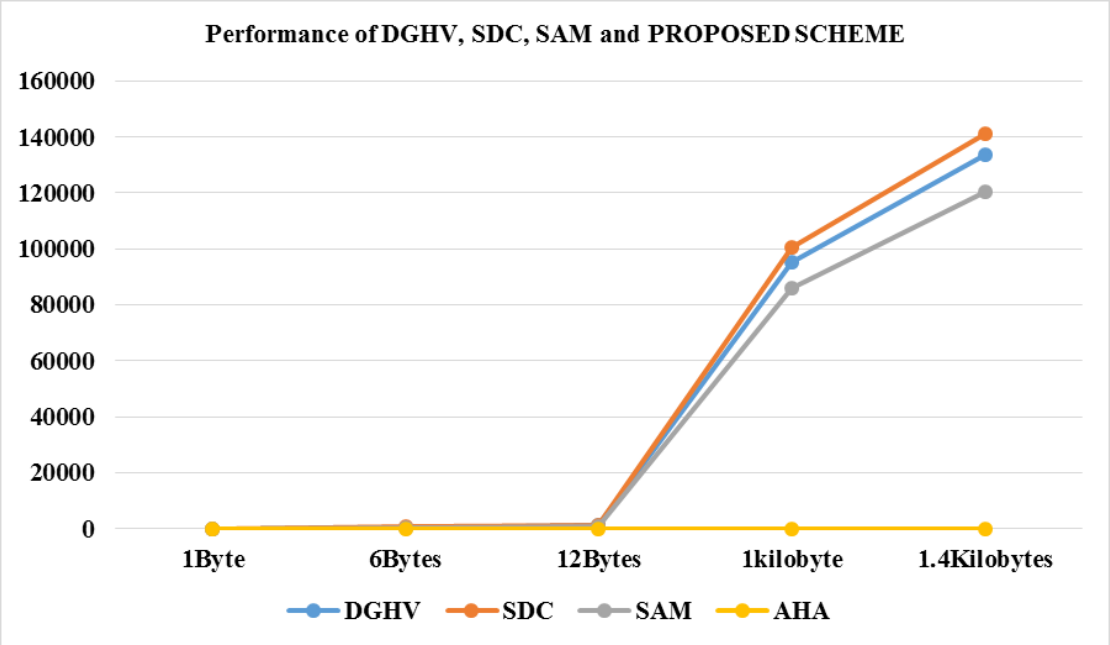

Figure 2 Comparison of the Execution Time of DGHV [5], SDC [14], SAM [14], and PROPOSED SCHEME

RESEARCH ARTICLE

5. CONCLUSION

The main focus of this study was to secure data using a probabilistic, symmetric, and non-circuit-based fully homomorphic encryption scheme. The scheme works with numeric data. Therefore, plaintext in non-numeric format has to be converted to numeric before encryption. Symmetric keys were used for both encryption and decryption. The keys were large prime integers generated randomly from a given range of integers. Generating a random large prime integer as a symmetric key makes the encryption probabilistic. The randomness also improves upon the security of the scheme and makes it withstand a Chosen Plaintext Attack. The encryption scheme possesses Indistinguishability property. The scheme uses fewer memory spaces as compared to DGHV, SDC, and SAM schemes. The time complexity of the proposed scheme is less than DGHV, and SDC but the same as SAM. However, the proposed Scheme runs faster than SAM because the SAM scheme has more primitive operations. Even though homomorphic encryption is seen to be slow in performance, the less execution time of the proposed scheme makes it appropriate for real-life implementation in most data-centric applications. The running time of the encryption algorithm of the proposed scheme is O (log (n^2)). The running time of the proposed scheme is quadratic. Further studies could be conducted to improve the scheme's running time to linear, that is O (log(n)).

REFERENCES

[1] Asante, G., Hayfron-Acquah, J.B., & Asante, M. (2021). Evolution of Homomorphic Encryption. International Journal of Computer Applications 183(29):37-40.

[2] Kahate, A. (2013). Cryptography and network security. Tata McGraw-Hill Education.

[3] Al Badawi, A. Q. A., Polyakov, Y., Aung, K. M. M., Veeravalli, B., & Rohloff, K. (2018). Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme. IEEE Transactions on Emerging Topics in Computing. 2018: pp. 70-95

[4] Gentry, C. (2009). A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University.

[5] Van Dijk, M., Gentry, C., Halevi, S., and Vaikuntanathan, V. (2010) Fully homomorphic encryption over the integers," in Advances in Cryptology {EUROCRYPT 2010, pp. 24{43, Springer, 2010.

[6] Gentry, C., & Halevi, S. (2011, May). Implementing gentry's fully-homomorphic encryption scheme. In Annual international conference on the theory and applications of cryptographic techniques (pp. 129-148). Springer, Berlin, Heidelberg.

[7] Brakerski, Z., & Vaikuntanathan, V. (2011, August). Fully homomorphic encryption from ring-LWE and security for key-dependent messages. In Annual cryptology conference (pp. 505-524). Springer, Berlin, Heidelberg.

[8] Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2014). (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT), 6(3), 1-36.

[9] Gupta, C. P., and Sharma, I. (2013, October). A fully homomorphic encryption scheme with symmetric keys with application to private data processing in clouds. In 2013 Fourth International Conference on the Network of the Future (NoF) (pp. 1-4). IEEE.

[10] Cheon, J. H., Coron, J. S., Kim, J., Lee, M. S., Lepoint, T., Tibouchi, M., & Yun, A. (2013, May). Batch fully homomorphic encryption over the integers. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 315-335). Springer, Berlin, Heidelberg.

[11] Coron, J. S., Lepoint, T., & Tibouchi, M. (2014, March). Scale-invariant fully homomorphic encryption over the integers. In International Workshop on Public Key Cryptography (pp. 311-328). Springer, Berlin, Heidelberg.

[12] Dasgupta, S., & Pal, S. K. (2016). Design of a polynomial ring-based symmetric homomorphic encryption scheme. Perspectives in Science, 8, 692-695.

[13] Ahmed, E. Y., and Elkettani, M. D. (2019), An Efficient Fully Homomorphic Encryption Scheme, International Journal of Network Security, Vol.21, No.1, PP.91-99, Jan. 2019 (DOI: 10.6633/IJNS.201901 21(1).11)

[14] Hamad, S.S & Sagheer, Ali. (2018). Design of Fully Homomorphic Encryption by Prime Modular Operation. Telfor Journal. 10. 118-122. 10.5937/telfor1802118S.

[15] Xiao, Liangliang & Bastani, Osbert & Yen, I-ling. (2012). An Efficient Homomorphic Encryption Protocol for Multi-User Systems. IACR Cryptology ePrint Archive. 2012. 193.

[16] Bellare, M., Fischlin, M., O'Neill, A., & Ristenpart, T. (2008, August). Deterministic encryption: Definitional equivalences and constructions without random oracles. In Annual International Cryptology Conference (pp. 360-378). Springer, Berlin, Heidelberg.

[17] Bali, P. (2014). Comparative study of private and public-key cryptography algorithms: A Survey. IJRET: International Journal of Research in Engineering and Technology, 2319-1163.

[18] Nurhaida, I., Ramayanti, D., & Riesaputra, R. (2017). Digital signature & encryption implementation for increasing authentication, integrity, security, and data non-repudiation. vol, 4, 4-14.

[19] Zhang, L., Zheng, Y., & Kantoa, R. (2016, June). A review of homomorphic encryption and its applications. In Proceedings of the 9th EAI International Conference on Mobile Multimedia Communications (pp. 97-106).

[20] J. Li, D. Song, S. Chen, X. Lu, "A Simple Fully Homomorphic Encryption Scheme Available in Cloud Computing", In Proceeding of IEEE, (2012), pp. 214-217.

[21] Das A., Dutta S., Adhikari A. (2013) Indistinguishability against Chosen Ciphertext Verification Attack Revisited: The Complete Picture. In: Susilo W., Reyhanitabar R. (eds) Provable Security. ProvSec 2013. Lecture Notes in Computer Science, vol 8209. pp. 347-356Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-41227-1_6.

Authors

Mr. George Asante is a Senior Lecturer and the Head in the Department of Information Technology Education at the Akenten Appiah-Menka University of Skills Training and Entrepreneurial Development, Kumasi - Ghana. His research areas include Information Security and Education.

Prof. J.B Hayfron-Acquah is an Associate Professor in Computer Science at the Department of Computer Science in the Kwame Nkrumah University of Science and Technology. His research interests include Image Processing, Pattern Recognition, Artificial Intelligence, and Information security.

RESEARCH ARTICLE

Prof. Michael Asante is an Associate Professor in Computer Science at the Department of Computer Science in the Kwame Nkrumah University of Science and Technology. His research areas include Computer Security, Cyber Security, and networking.

Joshua Caleb Dagadu holds a Ph.D. in Computer Science and Technology from the University of Electronic Science and Technology of China (UESTC). He is currently a Lecturer of Computing and Information Technology at the Akenten Appiah-Menka University of Skills Training and Entrepreneurial Development (AAMUSTED). He has a research interest in Information security, Artificial intelligence, and Telemedicine.

How to cite this article:

George Asante, James Ben Hayfron-Acquah, Michael Asante, Joshua Caleb Dagadu, "A Symmetric, Probabilistic, Non-Circuit Based Fully Homomorphic Encryption Scheme", International Journal of Computer Networks and Applications (IJCNA), 9(2), PP: 160-168, 2022, DOI: 10.22247/ijcna/2022/212332.