https://doi.org/10.31449/inf.v46i8.4305 Informatica **46** (2022) 91–104 **91**

Ensuring Privacy and Confidentiality of Data on the Cloud Using an Enhanced Homomorphism Scheme

John Kwao Dawson[1], Frimpong Twum[2], James Hayfron Acquah[3], Yaw Marfo Missah[3]
[1]Sunyani Technical University, [2,3]Kwame Nkrumah University of Science and Technology, Ghana
E-mails: [1]kwaodawson1@yahoo.com, [1]kwaodawson@stu.edu.gh, [2]twumf@yahoo.co.uk, [3]jbha@yahoo.com, [3]ymissah@gmail.com

Keywords: Good Prime Number, Linear Congruential Generator, Homomorphism, Ciphertext, Encryption, Deciphering

Received: July 18, 2022

Abstract: *Cloud computing is one of the widest phenomena embraced in information technology. This result from numerous advantages associated with it making many organizations and individuals offload their data to the cloud. Encryption schemes restrict access to data from unauthorized clients, helping attain confidentiality and privacy. The modification of the ciphertext of clients' data on the cloud demand downloading, deciphering, editing, and finally uploading back to the cloud by sharing their private key with the cloud service provider making it tedious. The application of homomorphism, allows computation to be performed on ciphertext with no decipher activity which helps to avoid the surfacing of sensitive client data stored on the cloud. In this paper, an Enhanced Homomorphism Scheme (EHS) is proposed based on Good Prime Numbers (GPN), Linear Congruential Generator (LCG), Fixed Sliding Window Algorithm (FSWA), and Gentry's homomorphism scheme. A dataset from the Kaggle database was used to test the proposed algorithm. A variety of tests were conducted using the proposed algorithm such as the Uniqueness of ciphertext, addition and multiplication property of full homomorphism, and the execution times using* $2^n (n \in 2,3,4,5)$ *data sizes. A comparison of the execution time of the proposed EHS was conducted with the New Fully Homomorphism Scheme (NFHS), and the Enhanced Homomorphism Encryption Scheme (EHES). From the comparison, the proposed EHS algorithm had the lowest encryption time when a data size of 24kb was executed but with a higher decryption time of 567.6667 ± 96.38911when a data size of 8kb was used. On the other hand, with a data size of 32kb, EHES had the highest decryption time of 1274ms with the proposed EHS having the lowest decryption time of 551.2222 ± 82.68746 indicating a decryption percentage decrease of 56.73%. This confirms that execution times are dependent on the size of the encryption key but not on data size.*

Povzetek: Nov kriptografski algoritem z imenom EHS se je izkazal z izboljšanimi časi izvajanja na nekaj standardnih testnih domenah.

1 Introduction

The use of cloud services has risen due to the convenience associated with their usage. Cloud computing is considered one of the emerging internet-based technologies in the Information Technology industry [1]. Cloud computing is an internet-based system that provides multi-tenancy, scalability, elasticity, pay-as-you-go, and self-provision of resources to the cloud client by the cloud service provider such as Amazon S3, and Google Cloud as shown in figure 1 [2]. This helps cloud clients to be able to distantly distribute the huge amount of information and workloads to the cloud and take advantage of unlimited computing resources and applications in the on-request high-quality services [3].

Cloud computing helps ease the burden of storage management, having access to information independent of regions, and decreases capital use on equipment, programs, and staff. Regardless of these, privacy and confidentiality of data issue is the basic factor that obstructs the far and wide reception of cloud computing [4].

Figure 1: Cloud computing model [39]

There have been numerous attempts by researchers to propose the most robust cryptographic scheme to

secure data at rest and in transit, but there exist many hindrances in achieving this aim, such as security threats and third parties securing data on behalf of organizations [5]. Another consideration that comes into mind is data modification by unauthorized entities and the hacking into systems by hackers when valuable data is outsourced to a third party for storage [6].

To address these security concerns most appropriately, homomorphism encryption schemes are proposed. Homomorphism has the capability that allows the evaluation of ciphertext, which results in encrypted data, and the result can be decrypted when the same function is applied to the plaintext [7]. Privacy and confidentiality become paramount if computations can be performed on ciphertext without knowledge of the plaintext, which is the advantage of homomorphism [8].

1.1 Homomorphism

Homomorphism is a cryptographic scheme that performs computations on ciphertext resulting in an encrypted output with no knowledge of the data upon which computations are performed [13], and [14]. Homomorphism helps in analyzing stored data in the cloud without deciphering it. This is because the resultant computation on the encrypted data after deciphering is similar to the corresponding message from the sender [32].

Gentry in 2009, proposed the maiden Fully Homomorphism scheme which supported additive and multiplicative characteristics which were lattice-based [8], [9], [10], and [11]. This opened the door for variants of fully homomorphism encryption schemes such as Homomorphism Authenticators (HA), and Homomorphism Authenticated Encryption (HAE) to be proposed and has been proven to be one of the security schemes that support the confidentiality and privacy of data on the cloud [12].

Dijk et al. [17] proposed revised fully homomorphism encryption algorithms ((Van Dijk, Gentry, Halevi, and Vaikuntanathan) (DGHV scheme)) using prime numbers in 2012. Their algorithm was based on the computation of integers to achieve full homomorphism. A re-encryption property in their work increased the noise level in their algorithm, resulting in higher complexity time.

There are four (4) stages of homomorphism [15]. These are;

Stage 1: Key Generation

Two keys are selected by the cloud client as indicated in equation 1. The public key P_k and security key S_k. Such that;

$$(P_k, S_k) = Keys \dots\dots\dots\dots\dots\dots\dots\dots \textbf{(1)}$$

Stage 2: Encryption

The plaintext (M) is accepted by the encryption algorithm to produce a ciphertext (C) by applying equation 2 using the private key.

$$C = Enc_{pk}(M) \dots\dots\dots\dots\dots\dots\dots\dots \textbf{(2)}$$

Stage 3: Evaluation

The evaluation function f is performed on the ciphertext C using the generated security key S_k using equation 3.

$$E = Eval_{sk}(f, C) \dots\dots\dots\dots\dots\dots\dots\dots \textbf{(3)}$$

Stage 4: Decryption

The reversal of the ciphertext C using the private key p_k to obtain the plaintext M by applying equation 4.

$$M = DecS_k(\boldsymbol{C}) \dots\dots\dots\dots\dots\dots\dots\dots \textbf{(4)}$$

Based on the four stages of homomorphism operations, homomorphism can be grouped into full homomorphism and partial homomorphism encryption schemes. Partial homomorphism encryption algorithms can perform either addition or multiplicative properties only while fully homomorphism encryption supports both additive and multiplicative properties. A somewhat (SWHE) homomorphism encryption algorithm is also proposed which is a sub-category of a full homomorphism algorithm and can perform addition and or multiplication properties.

1.2 Fully homomorphic encryption algorithm

An encryption algorithm is fully homomorphism if it supports both multiplication and addition properties [16]. Assuming there are five (5) variables (M, F, R, Q, L) such that M is the plaintext, F the ciphertext, R and Q the encryption and decryption algorithms respectively, and L the secret key. It can be deduced that, for every value of $R \in R$, using an encryption algorithm rule $e_k \in Q$ with a corresponding decryption rule $d_k \in L$. Hence, if the plaintext M and ciphertext F, then e_k links from M to F and d_k links from ciphertext F to plaintext M.

This implies

$e_k \Rightarrow M \rightarrow F$ and $d_k \Rightarrow F \rightarrow M$

From this, it can be deduced that considering all characters of plaintext M, the ciphertext will be obtained using equations 5, 6, 7, and 8;

$$\boldsymbol{e_k(a + p^b) = e_k a + p^b e_k} \dots\dots\dots \textbf{(5)}$$

$$\boldsymbol{e_k(a * p^b) = e_k a * p^b e_k} \dots\dots\dots \textbf{(6)}$$

$$\boldsymbol{d_k(c_a * c_b) = d_k c_a * d_k c_b} \dots\dots\dots \textbf{(7)}$$

$$\boldsymbol{d_k(c_a + c_b) = d_k c_a + d_k} \dots\dots\dots \textbf{(8)}$$

The encryption algorithm is considered fully homomorphism from equations 5, 6, 7, and 8.

Shihab and Makki, [12], and Dijk et al. [17] proposed a variant of the homomorphism scheme. Their proposed algorithm, Fully Homomorphism Encryption by Prime Modular Operation (SAM Scheme), pined its security strength on the random selection of big prime integers and

constant random big integers. Again, their algorithm encrypted the plaintext, character by character.

Even though their algorithm was promising the selection of big numbers used as seeds for the encryption and decryption process makes the execution times of their proposed algorithm linear and high. This makes the execution times to be proportional to data size [7], [10], and [12]. Again, the selection of big integers and big prime numbers generates high values which require high Central Processing Unit (CPU) capabilities to execute any data.

This, therefore, raises the need for a full homomorphism scheme that is non-deterministic, with low execution time, and non-linear. A full homomorphism encryption scheme is proposed in this work by integrating Good Prime Numbers (GPN), Linear Congruential Generator (LCG), Sliding Window Algorithm (SWA), and Gentry's algorithm.

1.3 Our contribution

Homomorphism is considered the solution to the security challenges of cloud computing which is hindering organizations and individuals from fully outsourcing to the cloud. There are variants of homomorphism schemes that are proposed, but their execution times are always linear (O (n)), making execution time predictable, and high. A robust high-security full homomorphism algorithm is proposed, which helps to attain privacy and confidentiality of data on the cloud with lower execution (O (log n)), non-linear and non-deterministic execution time. This is achieved through the application of the Fixed Sliding Window Algorithm on the selected numbers from the Linear Congruential Generator which breaks down the secret keys to smaller values to reduce the number of iterations performed by the processor to attain a lower execution time. The uniqueness of ciphertext is computed, the addition and multiplication property of homomorphism are tested, and the generation time for the four (4) arrays for the encryption and decryption of data using $\left(\frac{\boldsymbol{n(a[i])}}{\boldsymbol{3}}\right)$ and the execution time of data sizes of $2^n kb$ $(n \in 2,3,4)$ was conducted using the smaller encryption key aiding in lower execution time.

The paper is organized into five sections supported by sub-sections to give much clarity to the work. Section one discusses the introduction with its sub-sections, section two literature review, section three methodology, section four results and discussion, and section five conclusion.

2 Related work

This discusses the works of other authors that are linked to ensuring data confidentiality and privacy on the cloud by employing homomorphism. The works of Ren et al.
(2014) and Lakhan et al. [21] and [39] respectively are good examples. In the work of Ren et al, they proposed an exclusive – or (XOR) homomorphism encryption scheme. Their scheme encrypts data by randomizing it by performing an XOR operation based on a randomized string of bits. This scheme can protect data against the analysis of ciphertext by randomizing the data in transit thereby ensuring data confidentiality and privacy. Even though the algorithm is fast, its execution time is linear. Lakhan et, al. (2022), also proposed a secured vehicular fog cloud computing (FCN) that uses a Mobility-Aware Multi-Scenario Offloading Phase (MAMSOP) to attain mobility as well as offloading execution in their system. They employed full homomorphism encryption to attain the confidentiality of personal data on the cloud. Their scheme allowed for computation to be performed on locally stored data without decryption but is also linear.
The works of Agwa et.al, [22] and Chang and Li [23] proposed an encryption algorithm based on homomorphism and secret sharing aimed at ensuring cloud data confidentiality and privacy. Their approach was flexible and could add and remove shareholders as well as add them. The integration of the algorithms aimed to reduce the time complexity of Lagrange computations which was achieved by leveraging their data to the cloud. Despite the security strength of their proposed algorithm, the execution time is linear making it predictable. Again, the works of Loyka et, al. (2018), [26] proposed an affine-based homomorphism encryption scheme based on ASCII values of alphabets of plaintext to ensure privacy and confidentiality. Their scheme was the first to use an affine security scheme and its operations considered only strings and integers. Their approach considered the lookup and concatenation of encrypted text which uses the addition and subtraction of ciphertext operations. Their algorithm had linear encryption time with non-linear decryption time.
Torres et al. (2015) [24], proposed a privacy scheme using a protocol, based on iris authentication by employing a lattice-based full homomorphism encryption scheme. Because they adopted homomorphism encryption, their scheme provided an unlimited computation of the ciphertext and also prevented the transfer of private keys to any third party. Their approach is very promising but is still prone to several attacks as indicated in [24] and also linear.

Yang et, al. [29], proposed an optimized re-linearization scheme of CDKS19 to ensure cloud data privacy. Their proposed scheme reduces the linearization complexity thereby decreasing the homomorphism evaluation process through the reorganization of the evaluation key in the key generation process. The security of the proposed security
algorithm is promising but linear which makes it predictable.

The work of Hong et al. [25], ensured the confidentiality of data by employing model inference processes using a secure multi-labeled classifier utilizing an approximate homomorphism encryption algorithm. Their approach allowed for efficient data encryption which reduced the execution time but the data encryption is linear making it predictable.

Aono et al. [27] proposed integrating a key rotatable and security updating homomorphism algorithm (KR-SU-HE) which is a public-key scheme. Their scheme rotated the key for the encryption and rotated the
ciphertext and updated it as well still maintaining its privacy and confidentiality. Their approach seemed good but was also linear.

In 2018 Gazizullina, [28] proposed a probabilistic algorithm using third-degree polynomial time to ensure data privacy. The proposed scheme has a reduced execution time due to reduced calculation times. The security of the algorithm is based on employing product tables which allowed the usage of rational values. There was no comparison with existing algorithms and was not used to encrypt genomic data but is also linear.

A summary of the related works is shown in Table 1, indicating their methodology, weaknesses, and strength associated with the proposed algorithms. Considering the above works, we propose a full homomorphism scheme to ensure the privacy and confidentiality of data on the cloud using Good Prime Numbers (GPN), Linear Congruential Generator (LCG), Fixed Sliding Window Algorithm (FSW), and Gentry's algorithm with lower execution, non-deterministic times and non-linear time.

3 Methodology

A privacy and confidentiality enhancement scheme dubbed Enhanced Homomorphism Scheme (EHS) for cloud data is proposed. EHS is an integration of Good Prime Numbers (GPN), Linear Congruential Generator (LCG), Fixed Sliding Window Algorithm (FSWA), and Gentry's Algorithm. The algorithm is made up of two stages, Key Generation and the homomorphism scheme. The flow and the architectural diagram of EHS are shown in Figures 2 and 3.

The key generation in EHS is computed using three procedural stages. The first stage involves the generation of two Good Prime Numbers. The product of the two

selected numbers are considered the seed for the Linear Congruential Generator to generate twelve integers. The twelve integers are then subjected to the Fixed Sliding Window Algorithm to obtain four numbers using a sub-array of three $\left(\frac{\boldsymbol{n(a[i])}}{\mathbf{3}}\right)$**.** The initial value generated serves as $\boldsymbol{s_i}$, the second computed value $\boldsymbol{s_j}$, the third value $\boldsymbol{s_k}$, the fourth value is $\boldsymbol{s_l}$, and **M** is the plaintext. The encryption of plaintext is achieved through the application of equation 9 and the decryption of the ciphertext is computed using equation 10.

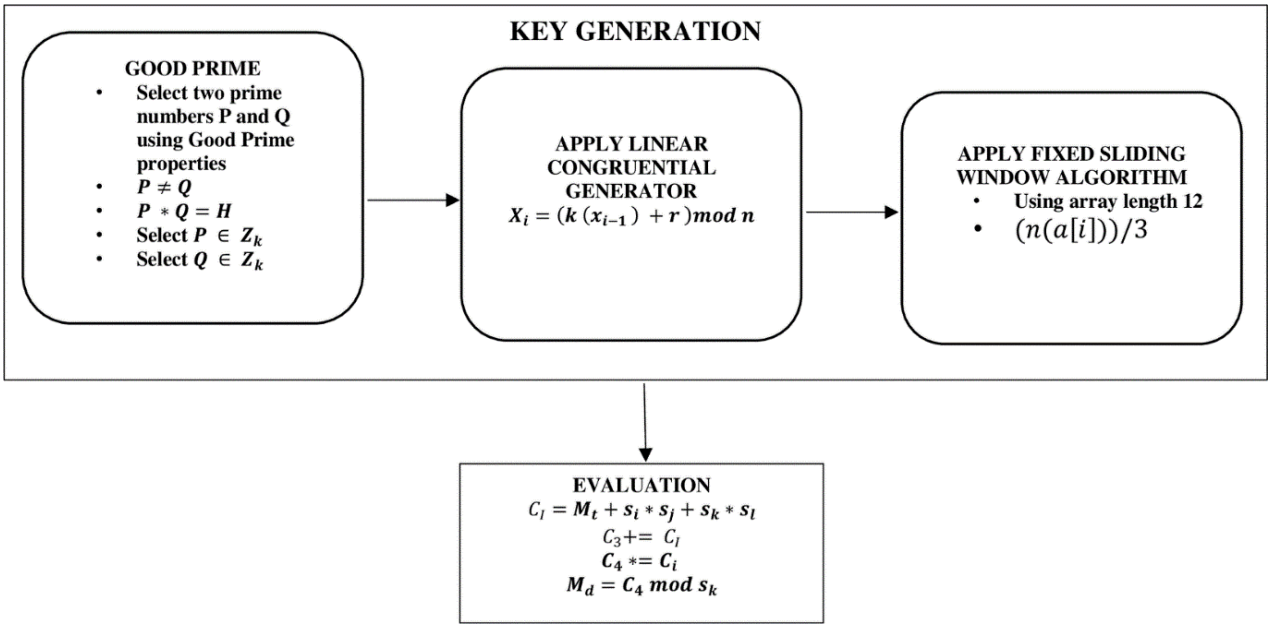
$$C = M + s_i * s_j + s_k * s_l \ldots\ldots\ldots\ldots\ldots (9)$$

$$M = C \; mod \; s_k \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (10)$$

In the architectural diagram of the proposed EHS diagram shown in Figure 3, the cloud client secures data $\boldsymbol{M1}$ and $\boldsymbol{M2}$ using Full Homomorphism encryption and offloads data to the cloud service provider. A common secret key is generated by the cloud client and cloud service provider to allow for synchronization. Upon synchronization, the cloud client requests modification of any data which is delivered as a function of $f(M1,M2)$ without decryption as $\boldsymbol{f(E(M1), E(M2))}$**.**

Table 1: Comparison of homomorphism algorithms to attain confidentiality and privacy on cloud

Serial No	References	Year	Method	Aim	Weakness / Strength
1	Ren et al [21]	2014	XOR- Homomorphism	Privacy and Confidentiality (Security)	Higher execution time resulting from more indexing in the database
2	Agwa et al [22]	2020	Secret sharing and Homomorphism encryption	Privacy and confidentiality	Static execution time
3	Chang and Li [23]	2019	Secret Sharing mechanism	Privacy and confidentiality	Used more variables resulting in higher execution time
4	Torres et al [24]	2015	Iris for biometric authentication using Full homomorphic encryption	Privacy of biometric data in an authentication system	• Linear • Predictable • High
5	Hong et al [25]	2022	Multi-Label Tumor Classification method	Privacy and Confidentiality	• Exponential • Predictable • Execution time increases as data size increases
6	Loyka et al [26]	2018	ASCII Text Encoding with Affine Cipher function	Privacy and Confidentiality	• Non-linear encryption time with text only • Non- deterministic encryption time • Linear encryption time when numbers only are used
7	Aono et al [27]	2017	Key-rotatable and Security-updatable Homomorphic Encryption (KR-SU-HE)	Privacy	• Linear • Predictable • Encryption and decryption are based on data size
8	Gazizullina [28]	2018	Self-Multiplication Complexity Modular arithmetic	Privacy	• Linear • Predictable
9	Yang et al [29]	2021	Optimized re-linearization algorithm of CDKS19	Privacy	• Linear • Higher execution time • Predictable based on larger modular space
10	Lakhan et al [38]	2022	Mobility-Aware Multi-Scenario Offloading Phase (MAMSOP)	Privacy and Confidentiality	For the performance schedule FHE-linear



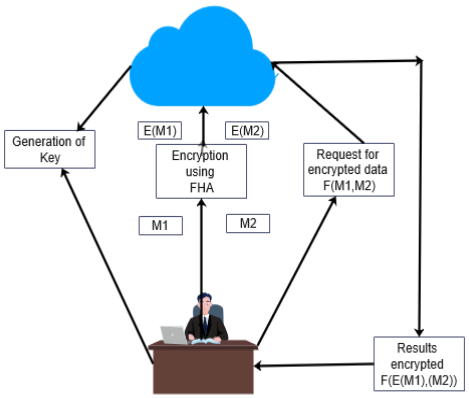


Figure 3: Architectural Framework of proposed EHS algorithm

3.1 Generation of keys

3.1.1 Good prime numbers (GPN)

A number whose square is greater than the product of two prime numbers positioned before and after in the sequence of the prime is considered a good prime [18]. Good Prime Numbers are computed using equation 11.

$$\boldsymbol{P_n^2 > P(n-i) * P(n+i)} \ldots\ldots\ldots\ldots\ldots (11)$$

The n value indicates the prime numbers, $P(n-i)$ represents the initial prime number resulting from the chosen primes while $P(n+i)$ suggests the subsequent prime fulfilling the condition $1 \leq i \leq n-1$. For example, first ten (10) prime numbers are: 2, 3, 5, 7, 11, 13, 17, 19, 23, and 29. Using the formula in equation 11, the first good prime can be computed as;

$5^2 > 3 * 7$, $5^2 > 2 * 11$. From this, the first ten good prime numbers are 5, 11, 17, 29, 37, 41, 53, 59, 67, and 71. Based on equation (11), select any two good prime numbers as P and Q, where $P * Q = H, P \neq Q, P \in Z_k$, and $Q \in Z_k$. The product of $\boldsymbol{P}$ and $\boldsymbol{Q}$ results in $\boldsymbol{H}$ which serves as the seed value for the Linear Congruential Generator in the next stage of the key generation.

3.1.2 Apply linear congruential generator (LCG)

LCG is used to generate random numbers which are computed using a sporadic equation [12]. The formula in equation (12) is used to generate the sequence of numbers between $V1, V2, \ldots . and\ 0, m-1$ based on the condition that;

$$\boldsymbol{m > 0, a < m, c < m, X_i < m}$$
$$\boldsymbol{X_{i+1} = (a\ X_i + C) mod\ m} \ldots\ldots\ldots\ldots (12)$$

In equation (12), $\boldsymbol{a}$ is a multiplier, $\boldsymbol{c}$ is an increment, m is the modulus and X_i is the seed value **(*H*).** The resultant of $\boldsymbol{P}$ and $\boldsymbol{Q}$ is the seed value in equation (12) which is used to generate a hundred thousand random numbers. Twelve numbers are selected at random which serve as the seed numbers for the Fixed Sliding Window Algorithm (FSWA) to generate four numbers as $\boldsymbol{s_i}$, $\boldsymbol{s_j}$, $\boldsymbol{s_k}$, and $\boldsymbol{s_l}$ for the encryption of the plaintext in equation 21.

3.1.3 Apply fixed sliding window algorithm (FSWA)

In identifying the range of numbers in an array the Fixed Sliding Window is applied. This helps to reduce the number of loops in a nested loop aiming at reducing time complexity from $O(n^2)$ to $O(n)$. The main objective of the Fixed Sliding Window algorithm is to generate the

greatest and minor numbers resulting from a continued range based on any given array [20].

9	13	114	18	9	7	17	19	1	3	51	11

Figure 4. Sliding window with 12 array length

	18	9	7	17	19	1	3	51	11

Figure 5. Sliding window with 3 sub-array length

The Fixed Sliding Window algorithm is applied to the array using equations 13, 14, 15, and 16. This is used to generate four numbers $(a_y, a_{y1}, a_{y2,}\, a_{y3,})$ using $(n(a[i]))/3$ based on the twelve arrays depicted in Figures 4 and, 5.

$$a_y = a_i + a_{i+1} + a_{i+2} \quad \ldots\ldots\ldots\ldots(13)$$
$$a_{y1} = a_{i+3} + a_{i+4} + a_{i+5} \quad \ldots\ldots\ldots(14)$$
$$a_{y2} = a_{i+6} + a_{i+7} + a_{i+8} \quad \ldots\ldots\ldots(15)$$
$$a_{y3} = a_{i+9} + a_{i+10} + a_{i+11} \quad \ldots\ldots\ldots\ldots(16)$$

The four arrays are computed using equations 17, 18, 19, and 20. Where $\boldsymbol{s_i}, \boldsymbol{s_j}, \boldsymbol{s_{k,}}$ and $\boldsymbol{s_l}$ are the list of arrays after applying the Fixed Sliding Window Algorithm on the 12 arrays generated.

$$\boldsymbol{s_i} = \sum_{n=0}^{2} \boldsymbol{a_{yn}} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (17)$$
$$\boldsymbol{s_j} = \sum_{n=3}^{5} \boldsymbol{a_{yn}} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (18)$$
$$\boldsymbol{s_k} = \sum_{n=6}^{8} \boldsymbol{a_{yn}} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (19)$$
$$\boldsymbol{s_l} = \sum_{n=9}^{11} \boldsymbol{a_{yn}} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (20)$$

3.1.4 Encryption

This is the conversion of plaintext into ciphertext. To encrypt a message $\boldsymbol{M}$, the ciphertext is computed using equation 21.

$$C_l = M_t + s_i * s_j + s_k * s_l \quad \ldots\ldots\ldots(21)$$

Such that $p_{k,}m \in [0, p-1)$. Where $\boldsymbol{si}$ is the noise (initial value from the fixed sliding window), $\boldsymbol{s_j}$ second value, $\boldsymbol{s_k}$ the third value, $\boldsymbol{s_l}$ fourth value, and, $\boldsymbol{M}$ is the message.

3.1.5 Addition homomorphism

This operation is performed on the ciphertext to authenticate its similarity with the plaintext. This allows for the modification of documents without an idea of the plaintext by applying equation 22 where C_n the storage location and C_I is the subsequent addition of ciphertext values.

$$C_n += C_I \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (22)$$

3.1.6 Decryption

The encrypted data is converted to plaintext by applying equation 23 to the ciphertext. Where $\boldsymbol{M}$ is the plaintext, $\boldsymbol{C}$ is the ciphertext, and $\boldsymbol{s_k}$ is the third value generated from the application of the Sliding Window Algorithm in equation 23.

$$M = C_n \; mod \; s_k \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (23)$$

3.1.7 Multiplication homomorphism

An additional computation can be performed on the ciphertext using multiplication operators to confirm its similarity with the plaintext by the application of equation 24.

$$C_t *= C_i \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (24)$$

The framework for the proposed algorithm is shown in figure 2 and presented in algorithm1.

Algorithm 1 Proposed Algorithm

Procedure EHS
Start of algorithm
  Compute H = P *Q → *H: P, Q ∈ Good Prime*
  $X_z = k\,(X_{x-1}) + r \bmod n$ → *(n > 0, 0 < k < n, 0 ≤ r < n, Compute the CLG*
  $(X_{x-1}) = H$
  $X_z \in 1 \ldots\ldots\ldots\ldots 100{,}000$
  For $i = 0, i < 12, i++$ ***do***
    $\{a[i] = Rand\,(1, 100000)\}$
  end for
  Apply Fixed Sliding Window (FSD) on 12 arrays susing sub − array of 3
  $s_i = \sum_{n=0}^{2} a_{yn}$
  $s_j = \sum_{n=3}^{5} a_{yn}$
  $s_k = \sum_{n=6}^{8} a_{yn}$
  $s_l = \sum_{n=9}^{11} a_{yn}$
  $C_l = M_t + s_i * s_j + s_k * s_l$ → *Encryption*
  $C_n += C_l$ → Addition homomorphism
  $C_t *= C_l$ → Multiplication
  $M_d = C_t \bmod s_k$ → *Decryption*
End

4 Experimental analysis, results, and discussion

4.1 Experimental analysis

The implementation of the proposed algorithm to ensure privacy, and confidentiality of cloud data dubbed Enhanced Homomorphism Scheme (EHS) is presented in this section. In this proposed algorithm, a Fixed Sliding

Window Algorithm (FSWA) is applied to the twelve arrayed numbers generated using the seed value from the Good Prime numbers on Linear Congruential Generator (LCG) to obtain four sequential numbers S_i, S_j, S_k, and S_l as indicated in figure 2 and algorithm 1. The random generation of the resultant values from equations 17, 18, 19, and 20 makes the proposed algorithm unpredictable and again serves as the security basis of this algorithm. The plaintext is encoded by computing the product of the first value S_i by the third S_k value and added to the ASCII value of the alphabet in the message. The snippet of the codes for the encryption, decryption, addition and multiplication properties in the homomorphism algorithm is shown in Figures 6 and 7. This is then added to the product of the second value S_k and the fourth value S_l to obtain the ciphertext using equation 21. An addition homomorphism operation is performed on the ciphertext to confirm their similarity with the plaintext by using equation 22. A multiplication homomorphism is again performed on the ciphertext to confirm its similarity with the plaintext by using equation 24. The snippet for the generation of the keys for the proposed EHS algorithm by the integration of Good Prime numbers (GPN), Linear Congruential Generator, and Fixed Sliding Window Algorithm (FSWA) are shown in Figures 6 and 7.

```
var modulo = rand.Next(3, 100000);
var numbers = new List<long>();
for (int i = 0; i < 12; i++)
{
    var a = rand.Next(1, modulo);
    var b = rand.Next(0, modulo);
    var xnot = rand.Next(1, modulo);
    var x = a * xnot + b % modulo;
    numbers.Add(x);
}
//applying sliding window
int j = 0;
var sums = new List<long>();
while (j < numbers.Count)
{
    long currentSum = 0;
    for (int i = j; i < j + 3; i++)
    {
        currentSum += numbers[i];
    }
    j += 3;
    sums.Add(currentSum);
}

var si = sums[0];
var sj = sums[1];
var sk = sums[2];
var sl = sums[3];
```

Figure 6: Snippet for the generation of the keys for the proposed EHS algorithm.

```
var buffer = Encoding.ASCII.GetBytes(message);
var computedValues = new List<string>();
var sb = new StringBuilder();
BigInteger c3 = 0;
BigInteger c4 = 1;
for (int a = 0; a < buffer.Length; a++)
{
    BigInteger ci = buffer[a] + si * sk + sk * sl;
    c3 += ci;
    c4 *= ci;
}
var m3 = c3 % sk;

BigInteger m4 = c4 % sk;


var encTime = $"{TimeSpan.FromTicks(Stopwatch.GetTimestamp()).Milliseconds}ms";
Console.WriteLine(sb.ToString());
Console.WriteLine("--------Time------");
Console.WriteLine(encTime);
Console.WriteLine("--------Time------");
Stopwatch.StartNew();
```

Figure 7: Snippet of the codes for the encryption, decryption, addition, and multiplication properties in the homomorphism algorithm.

4.2 Environment and data for the experiment

The simulation of the proposed EHS algorithm was conducted on an i7 Lenovo computer with a processor speed of 2.10GHz using a C# language. Predesigned Data sizes of $2^n (n \in 2,3,5)$ using the Kaggle dataset [30] to evaluate the average execution times for EHS. A simulation interface is depicted in Figures 8 and 9. A dataset size of 2kb was taken from the Kaggle database and a simulation was performed using the Enhanced Homomorphism Scheme (EHS). The encryption time was 12ms and 61ms for the decryption time. The data to be transmitted is encrypted from the client's computer to the cloud service provider. The cloud service provider has no idea of the content of the data. The decryption of the content happens only when the cloud client request for retrieval of data as depicted in figure 10.

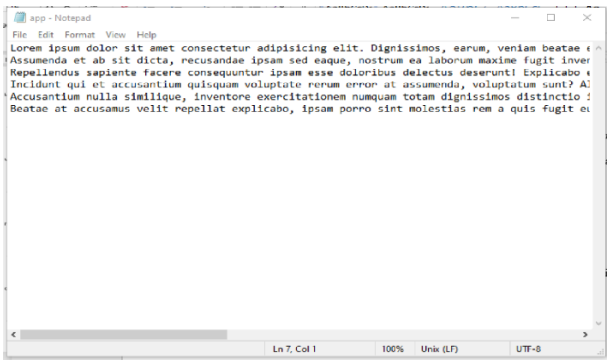


Figure 8: A 2kb dataset from Kaggle for simulation.

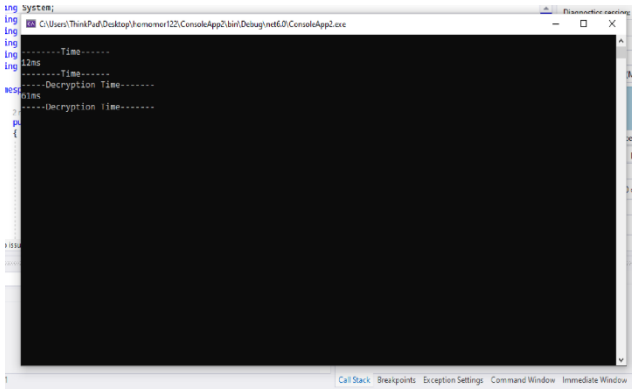


Figure 9: Encryption and Decryption times for a 2kb Kaggle dataset for proposed EHS.

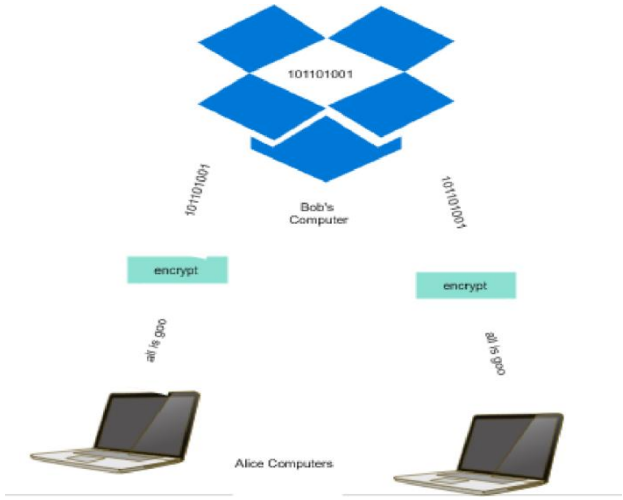


Figure 10: Pictorial view of simulated data representation in cloud using homomorphism.

4.3 Results and discussion

The proposed algorithm was tested for the uniqueness of ciphertext, Homomorphism test property, and the encryption and decryption time for data sizes of $2^n (n \in 2,3,5)$.

4.3.1 Uniqueness of ciphertext test

The uniqueness of ciphertext is the encoded message produced by an algorithm. A different encoded message is produced anytime the same message is encoded using different secret keys. To prove the accuracy of the algorithm the same plaintext should produce different ciphertext anytime the algorithm is executed. As a result, in attempting to control the security challenge in the cloud, anytime data is encrypted, a different encoded text should be produced anytime the algorithm is run. The aim of ensuring data confidentiality and privacy is attained. Compared with all encryption schemes, EHS generates exclusive encoded text for each string of plaintext as shown in Table 2.

Table 2: Uniqueness of ciphertext test

NO	MESSAGE	ENCODED TEXT	DECRYPTED TEXT	EVALUATION
1	CD	10893459098602660107	CD	YES
2	CD	196048421532347120159743969974510	CD	YES
3	GH	2857582766089353527	GH	YES
4	55	12306163028248793884	55	YES

4.3.2 Homomorphic algorithm testing property

Full homomorphism algorithms are distinguished from somewhat and partial homomorphism algorithms because of their ability to perform addition and multiplication operations on ciphertext to obtain another ciphertext without an idea of the plaintext. EHS algorithm supports addition and multiplication operations. Assuming there are M_n, $and\ M_k$ plaintext. When the messages M_n, $and\ M_k$ are encrypted using the formula;

$$C_I = M_t + s_i * s_j + s_k * s_l,$$

results in a ciphertext. An addition operation of M_n, $and\ M_k$ using the equation $C_n += C_I$ should result in another ciphertext. Decrypting the ciphertext using the equation $M_n = C_n \bmod s_k$ results in another ciphertext which is equivalent to adding M_n, $and\ M_k$ as shown in table 3. In multiplicative homomorphism, M_n, $and\ M_k$ are encrypted to get ciphertext by applying the formula; $C_I = M_t + s_i * s_j + s_k * s_l$.

If the ciphertext is decrypted using the equation $M_n = C_n \bmod s_k$, the plaintext is produced

which is equivalent to the product of the plaintext using the equation $\boldsymbol{C_t} *= \boldsymbol{C_i}$ as indicated in Table 3.

4.3.3 Encryption and decryption times for data sizes of $2^n(n \in 2,3,5)$.

Encryption and decryption time explain the conversion of plaintext into ciphertext and the reversal of ciphertext to plaintext. An algorithm with the fastest encryption and decryption time is considered an efficient algorithm [32], and [33]. A comparison of the execution time was conducted with New Fully Homomorphism Scheme (NFHS) [37], Enhanced Homomorphism Encryption Scheme (EHES) [7], Loyka, et al. 2018 [26], and the proposed Enhance Homomorphism Scheme (EHS). The algorithm is executed thirty times using different data sizes and their respective averages and standard errors computed. The comparison analysis of the runtime for encryption and decryption was performed based on data sizes of $2^n(n \in 2,3,4,5)$ using a dataset from Kaggle [30] and the output is indicated in Tables 4 and 5.

From Table 4, with 4kb data, NFHE had the lowest encryption time compared with EHES and the proposed algorithm Enhance Homomorphism Scheme (EHS). On the other hand, when the data size was increased to 24kb, the proposed algorithm had the lowest encryption time of 376.7778 ± 77.37333 followed by NFHE and EHES having the highest encryption time indicating a 52.55% decrease in execution time. From table 5, EHES had the highest decryption time of 653ms, followed by NFHE of 594ms with the proposed EHS algorithm having the lowest decryption time of 503.2222 ± 83.59256 when a data size of 4kb was used. On the other hand, with a data size of 32kb, EHES had the highest decryption time of 1274ms with the proposed EHS having the lowest decryption time of 551.2222 ± 82.68746 indicating a decryption percentage decrease of 56.73%.

From Table 1, it can be observed that authors [21], [22], [23], [24], [25], [27], [28], [29], and [38] have their algorithms producing predictable, high execution and linear execution time, contrarily, Loyka et al. 2018 [26], proposed an algorithm, using homomorphism scheme based on an affine cipher, which produced similar non-linear results as the proposed Enhanced Homomorphism Scheme when text only was executed as shown in table 6, but the encryption and decryption time for numbers only was linear as shown in Table 7, whiles that of EHS is non-linear which makes the work of Loyka et al. (2018) to be defeated by the works of [34], [35], and [36] that execution time depends on the size of security key used for the execution process. From, this it can be concluded that the proposed Enhanced Homomorphism scheme's execution time is not dependent on data size but on the secret key used for the encryption as proposed by authors [34], [35], and [36].

Table 3: Test property for homomorphism algorithm

Plaintext	$M_n + M_k$	$M_n * M_k$	Ciphertext	Decryption	Evaluation
55	106	2809	23024260351412088	55	YES
CD	135	4556	85034133229062951	CD	YES
gh	207	10712	23244261176668167	gh	YES

Table 4: Encryption of plaintext based on different data sizes

Plaintext (kb)	EHES [37] (ms)	NFHE[7] (ms)	Proposed EHS (ms)
4	582	493	558.4444 ± 67.30881
8	634	562	501.7778 ± 93.74089
16	720	693	630.8889 ± 109.938
24	794	754	376.7778 ± 77.37333
32	825	797	540.2222 ± 82.48851

Table 5: Decryption of Ciphertext base of different data sizes

The ciphertext (kb)	EHES [37] (ms)	NFHE (7)ms	Proposed EHS (ms)
4	653	594	503.2222 ± 83.59256
8	864	782	567.6667 ± 96.38911
16	961	842	433.4444 ± 114.3691
24	1049	985	389.3333 ± 77.40047
32	1274	1167	551.2222 ± 82.68746

Table 6. Encryption and decryption time with text only (Loyka et al 2018) [26]

Data Size	Encryption Time	Decryption Time
2^8	3567.829	5117.220
2^{16}	3545.013	5121.104
2^{32}	3635.781	5069.406
2^{64}	3534.376	5084.582
2^{128}	3437.900	5144.918

Table 7. Encryption and decryption time with numbers only (Loyka et al 2018) [26]

No of Digits	Encryption Time	Decryption Time
1	0.017	0.033
2	0.031	0.037
3	0.036	0.046
4	0.038	0.048

5 Conclusion

The benefits of cloud computing are enormous; hence much effort should be applied to ensure its utmost security and sustainability. The most outstanding security challenge in the cloud due to its wide usage is confidentiality and privacy. The homomorphic scheme is considered the best among all the encryption algorithms used to secure data in the cloud. Somewhat, partial or full homomorphism allows varying levels of computation to be performed on encrypted data which is the advantage of homomorphism. This paper proposes a homomorphism algorithm dubbed Enhanced Homomorphic Scheme (EHS) with lower execution, non-deterministic execution time, and non-linear execution time. The proposed algorithm was tested using a dataset from Kaggle [30] and a comparison of its execution time was performed against New Fully Homomorphism Scheme (NFHS), Loyka et al 2018), and Enhance Homomorphism Encryption Scheme (EHES). Other analyses were also conducted such as the ciphertext uniqueness test, and the homomorphism property test (addition and multiplication property) were also conducted. The experiment proved that the proposed algorithm EHS supports the addition and multiplication properties applied in a homomorphism scheme. The proposed algorithm EHS had the lowest encryption time when a data size of 24kb was executed but with a higher decryption time of 567.6667 ± 96.38911with a data size of 8kb. Future work should be conducted on throughput, CPU, and memory usage.

Acknowledgement

We would like to acknowledge the efforts of the staff of the Computer Science Department of Kwame Nkrumah University of Science and Technology and Sunyani Technical University in making this article fruitful.

References

[1] L. Nguyen-Vu, J. Park, M. Park, and S. Jung (2016). Privacy enhancement using selective encryption scheme in data outsourcing, *International Journal of Distributed Sensor Networks*, vol. 12, no. 7, p. 155014771665725, DOI: 10.1177/1550147716657255.

[2] T. Wang, H. Ma, Y. Zhou, R. Zhang, and Z. Song (2021). Fully Accountable Data Sharing for Pay-as-You-Go Cloud Scenes, IEEE Transactions on Dependable and Secure Computing, vol. 18, no. 4, pp. 2005-2016, 1 DOI: 10.1109/TDSC.2019.2947579.

[3] M. Xu, A. N. Toosi, and R. Buyya (2021). A Self-Adaptive Approach for Managing Applications and Harnessing Renewable Energy for Sustainable Cloud Computing. IEEE Transactions on Sustainable Computing, pp. 544-558, DOI: 10.1109/TSUSC.2020.3014943.

[4] H. Zhang, Z. Guo, S. Zhao, and Q. Wen (2021). Privacy-Preserving Linear Region Search Service. IEEE Transactions on Services Computing, pp. 207-221, DOI: 10.1109/TSC.2017.2777970.

[5] O. Akinrolabu, S. New and A. Martin (2019). Assessing the Security Risks of Multicloud SaaS Applications: A Real-World Case Study. 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp. 81-88, DOI: 10.1109/CSCloud/EdgeCom.2019.00-14.

[6] G. S. Gaba, G. Kumar, H. Monga, T. -H. Kim, M. Liyanage, and P. Kumar (2020). Robust and Lightweight Key Exchange (LKE) Protocol for Industry 4.0. IEEE Access, pp. 132808-132824, DOI: 10.1109/ACCESS.2020.3010302.

[7] Z. H. Mahmood and M. K. Ibrahem (2018). New Fully Homomorphic Encryption Scheme Based on Multistage Partial Homomorphic Encryption Applied in Cloud Computing. 2018 1st Annual International

Conference on Information and Sciences (AiCIS), pp.182-186, DOI: 10.1109/AiCIS.2018.00043.

[8] P. Chaudhary, R. Gupta, A. Singh, and P. Majumder (2019). Analysis and Comparison of Various Fully Homomorphic Encryption Techniques. 2019 International Conference on Computing, Power and Communication Technologies (GUCON), pp. 58-62.

[9] T. Shen, F. Wang, K. Chen, K. Wang, and B. Li (2019). Efficient Leveled (Multi) Identity-Based Fully Homomorphic Encryption Schemes, IEEE Access, pp. 79299-79310, DOI: 10.1109/ACCESS.2019.2922685.

[10] P. Zhang, X. Sun, T. Wang, S. Gu, J. Yu, and W. Xie (2016). An accelerated fully homomorphic encryption scheme over the integers, 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), pp. 419-423, DOI: 10.1109/CCIS.2016.7790295.

[11] J. Kim and A. Yun (2021). Secure Fully Homomorphic Authenticated Encryption, IEEE Access, pp. 107279-107297, DOI: 10.1109/ACCESS.2021.3100852.

[12] H. Shihab and S. Makki (2018). Design of fully homomorphic encryption by prime modular operation. *Telfor Journal*, vol. 10, pp. 118–122, DOI: 10.5937/telfor1802118s.

[13] A. M. Abukari, E. K. Bankas, and M. M. Iddrisu (2021). A Hybrid of two Homomorphic Encryption Schemes for Cloud Enterprise Resource Planning (ERP) Data. *International Journal of Computer Applications*, pp. 1–7, DOI: 10.5120/ijca2021921789.

[14] D. Gaidhani (2017). A Survey Report On Techniques for Data Confidentiality In Cloud Computing Using Homomorphic Encryption. *International Journal of Advanced Research in Computer Science*, pp. 389–394, DOI: 10.26483/jars. v8i8.4746.

[15] J. Lin, J. Niu, H. Li, and M. Atiquzzaman (2019). A Secure and Efficient Location-based Service Scheme for Smart Transportation. *Future Generation Computer Systems*, pp. 694–704, DOI: 10.1016/j.future.2017.11.030.

[16] Y. Lu and M. Zhu (2018). Privacy preserving distributed optimization using homomorphic encryption. *Automatica*, pp. 314–325, DOI: 10.1016/j.automatica.2018.07.005.

[17] K. M. M. Aung, H. T. Lee, B. H. M. Tan, and H. Wang (2019). Fully homomorphic encryption over the integers for non-binary plaintexts without the sparse subset sum problem, *Theoretical Computer Science*, pp. 49–70, DOI: 10.1016/j.tcs.2018.11.014.

[18] M. Patel, A. M. Patel, and R. B. Gandhi (2020). Prime numbers and their analysis, *Journal of Emerging Technologies and Innovative Research*, pp. 1–5, DOI: ISSN-2349-5162.

[19] Ankur, Divyanjali and T. Bhardwaj (2015). A dissection of pseudorandom number generators, *2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN)*, pp. 318-323, DOI: 10.1109/SPIN.2015.7095369.

[20] Y. Li, H.-L. Wei, Stephen. A. Billings, and P. G. Sarrigiannis (2015). Identification of nonlinear time-varying systems using an online sliding-window and common model structure selection (CMSS) approach with applications to EEG, *International Journal of Systems Science*, pp.2671–2681, DOI: 10.1080/00207721.2015.1014448.

[21] S. Q. Ren, B. H. M. Tan, S. Sundaram, T. Wang and K. M. M. Aung (2014). Homomorphic exclusive-or operation enhances secure searching on cloud storage, 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, pp. 989-994, DOI: 10.1109/CloudCom.2014.86.

[22] N. A. Agwa, T. Kobayashi, C. Sugimoto and R. Kohno (2020). Security of Patient's Privacy in E-Health using Secret Sharing and Homomorphism Encryption Scheme, 2020 35th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), pp. 155-160.

[23] C.-C. Chang and C.-T. Li (2019). Algebraic secret sharing using privacy home-based health care IoT-based health care systems, *Mathematical Biosciences and Engineering*, pp. 3367–3381, 2019, DOI: 10.3934/mbe.2019168.

[24] W. A. Alberto Torres, N. Bhattacharjee, and B. Srinivasan (2015). Privacy-preserving biometrics authentication systems using fully homomorphic encryption, *International Journal of Pervasive Computing and Communications*, pp. 151–168, DOI: 10.1108/ijpcc-02-2015-0012.

[25] S. Hong, J. H. Park, W. Cho, H. Choe, and J. H. Cheon (2022). Secure tumor classification by shallow neural network using homomorphic encryption, *BMC Genomics*, DOI: 10.1186/s12864-022-08469-w.

[26] K. Loyka, H. Zhou, and S. P. Khatri (2018). A Homomorphic Encryption Scheme Based on Affine Transforms, *Proceedings of 2018 on Great Lakes Symposium on VLSI*, DOI: 10.1145/3194554.3194585.

[27] Y. Aono, T. Hayashi, L. Trieu Phong, and L. Wang (2017). Efficient Key-Rotatable and Security-Updatable Homomorphic Encryption, *Proceedings of the Fifth ACM International Workshop on Security in Cloud Computing*, DOI: 10.1145/3055259.3055260.

[28] A. Gazizullina (2018). Fully homomorphic encryption scheme for secure computation, *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming*, DOI: 10.1145/3191697.3213794.

[29] X. Yang, S. Zheng, T. Zhou, Y. Liu and X. Che (2022). Optimized re-linearization algorithm of the multikey homomorphic encryption scheme," in Tsinghua Science and Technology, pp. 642-652, DOI: 10.26599/TST.2021.9010047.

[30] English to French translations, https://www.kaggle.com/datasets/digvijayyadav/frenchenglish/metadata.

[31] H. Shihab and S. Makki (2018). Design of fully homomorphic encryption by prime modular operation, *Telfor Journal*, pp. 118–122, 2018, DOI: 10.5937/telfor1802118s.

[32] M. U. Sana, Z. Li, F. Javaid, H. B. Liaqat and M. U. Ali (2021). Enhanced Security in Cloud Computing Using Neural Network and Encryption, IEEE Access, pp. 145785-145799, DOI: 10.1109/ACCESS.2021.3122938.

[33] M. Hong, P. Wang, and W. Zhao (2016). Homomorphic Encryption Scheme Based on Elliptic Curve Cryptography for Privacy Protection of Cloud Computing, 2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), pp. 152-157, DOI: 10.1109/BigDataSecurity-HPSC-IDS.2016.51.

[34] M. A. Mohamed, A. Y. Tuama, M. Makhtar, M. K. Awang, and M. Mamat (2016). The Effect of RSA Exponential Key Growth on the Multi-Core Computational Resource, *American Journal of Engineering and Applied Sciences*, pp. 1054–1061, DOI: 10.3844/ajeassp.2016.1054.1061.

[35] S. S. Hamad and A. M. Sagheer (2018). Fully Homomorphic Encryption based on Euler's Theorem, *Journal of Information Security Research*, p. 83, DOI: 10.6025/jisr/2018/9/3/83-95.

[36] O. C. Abikoye, A. D. Haruna, A. Abubakar, N. O. Akande, and E. O. Asani (2019). Modified Advanced Encryption Standard Algorithm for Information Security, *Symmetry*, pp. 1484, DOI: 10.3390/sym11121484.

[37] K. El Makkaoui, A. Ezzati, and A. B. Hssane (2015). Challenges of using homomorphic encryption to secure cloud computing, 2015 International Conference on Cloud Technologies and Applications (CloudTech), pp. 1-7, DOI: 10.1109/CloudTech.2015.7337011.

[38] Mohammed, M. A., Garcia-Zapirain, B., Nedoma, J., Martinek, R., Tiwari, P., and Kumar, N. (2022). Fully Homomorphic Enabled Secure Task Offloading and Scheduling System for Transport Applications. *IEEE Transactions on Vehicular Technology*. 10.1109/tvt.2022.3190490.

[39] M. Chai (2022). Design of Rural Human Resource Management Platform Integrating IoT and Cloud Computing, *Computational Intelligence and Neuroscience*, vol. pp. 1-9, DOI: 10.1155/2022/4133048.