Enhanced Integer-Based Homomorphic Encryption Scheme with Windowing Mechanism for Securing Electronic Health Records

Abdulrahman Tawfeeq Jalal, Mohammed Anwar Mohammed

Department of Computer Science, College of Science, University of Sulaimani, Sulaymaniyah, Iraq

ABSTRACT

The frequent breaches of healthcare data annually make robust encryption mechanisms crucial, especially those that preserve the usefulness of the data while ensuring privacy. This study addresses specific integer-based homomorphic encryption systems and their critical vulnerabilities. The vulnerability identified in these systems is the possibility of decryption using other values, such as factors or primes, instead of the claimed unique secret key. We propose an enhanced cryptographic formula to address this vulnerability using a double random value technique that ensures decryption depends solely on the designated secret key. We also apply a windowing technique for prime selection to enhance the key properties against pattern detection attacks. Security analysis shows that the enhanced system prevents decryption using values other than the dedicated key while maintaining additive and multiplicative homomorphism. Performance evaluations show that the improved system maintains decryption times and ciphertext expansion ratios similar to the original system, with a reasonable decryption time reduction. Statistical testing results using the National Institute of Standards and Technology tests demonstrate the robustness of the proposed approach compared to the original, with the windowing technique exhibiting superior randomness properties.

Index Terms: Homomorphic Encryption, Windowing Mechanism, Electronic Health Records, Privacy Preserving, Synthetic Healthcare Data

1. INTRODUCTION

According to estimates by clinical pathology laboratories, the number of patients whose personal information was revealed reached 2.2 million, and the number of patients whose banking information was affected reached 34,500 [1]. With the increasing demand for outsourced applications, data privacy on the Internet has become crucial. While customers benefit from these services by uploading their data to be evaluated and obtaining results, their data are exposed to third-party services [2]. Hospitals and healthcare organizations increasingly collect data due to the increasing use of this data in health research and personalized medicine based on big data technologies [3]. The amount and risk of cyberattacks on patients' medical data stored in the cloud are increasing daily.

The healthcare sectors focus on digitizing healthcare information because it improves care efficiency, making it more affordable and accessible. In the healthcare sector, hospitals or healthcare facilities keep a digital record of each

Access this article online

DOI:10.21928/uhdjst.v9n2y2025.pp77-91	E-ISSN: 2521-4217 P-ISSN: 2521-4209

Corresponding author's e-mail: Abdulrahman Tawfeeq Jalal, Department of Computer Science, College of Science, University of Sulaimani, Sulaymaniyah, Iraq. E-mail: abdulrahman.jalal@univsul.edu.iq

Received: 15-05-2025 Accepted: 28-07-2025 Published: 25-08-2025

patient called an electronic health record (EHR) that contains all demographic, clinical, historical, reporting data, progress notes, problems, medications, vital signs, immunizations, laboratory data, and radiology reports [2], [4]. A large percentage of patients allow their data to be used to support research and improve the quality of healthcare, but at the same time, they are concerned about privacy, misuse, and mishandling of their data [5]. For information to be secure, it must have the security characteristics of confidentiality, integrity, and availability [6]. To prevent data breaches, encryption is a vital component of information security systems, thus protecting private data from unauthorized access and ensuring integrity [7].

2. LITERATURE REVIEW

2.1. Data Security and Privacy Challenges

In multi-party environments such as the cloud, all parties are at risk of threats, whether the provider or the user. Therefore, the minimum requirement for data security is to be stored in its encrypted state [8]. Customers are very concerned about the content of data sent to the cloud or stored in the service provider's servers because it is easily accessible to service providers. If digital records are handled in a way that ensures their privacy during acquisition, storage, and computing, then complete privacy has been achieved. However, while privacy can be achieved during acquisition and storage, achieving privacy is challenging during processing [2].

2.2. Homomorphic Encryption as a Solution

When encrypted data are required to be used, it must be decrypted, which exposes it to risk at that moment, so homomorphic encryption is a solution to ensure its privacy and security [2]. Fully homomorphic encryption (FHE) is one of the most widely adopted encryption techniques, focusing on privacy at the highest levels [9]. This technique focuses on two aspects: data encryption and the ability to handle data that has been encrypted through mathematical operations without the need to decrypt it, thus ensuring its complete confidentiality.

The main types of homomorphic encryption schemes are divided according to the number and type of operations into Partial, Somewhat, and Full. In Partial, only one operation, either addition or multiplication (not both), is applied to the encrypted data an unlimited number of times. Examples of these are Rivest-Shamir-Adleman (RSA), ElGamal, and Paillier. In Somewhat, a limited number of mathematical operations are applied to the encrypted data a limited number of times. Finally, the full technique involves both addition and multiplication utilized simultaneously and an unlimited number of times [10].

As for the functions and use cases, the benefit of FHE is that it supports functions, such as searching, sorting, max, and min. However, despite the proposed attempts that have improved the efficiency and performance of FHE schemes, it is still difficult to implement them in a real service without disturbing the user. Although somewhat homomorphic encryption or leveled-FHE has achieved acceptable performance in some applications, the bootstrapping techniques still need improvement, as do the multiplication operations, which negatively affect performance [10]. Due to this feature, data encrypted with this technique can be handed over to third parties to perform any meaningful operation [3].

2.3. Related Work

In certain cases, user data becomes vulnerable, such as during pandemics when patients cannot reach hospitals, so they contact doctors using technology over the network to obtain information about the diagnosis [11]. This is done remotely through the presence of devices that provide accurate medical data and support the Internet of Things, which can then send readings of blood pressure, oxygen, temperature, etc. [12]. Carpov *et al.* developed a mobile application that encrypts users' data with a FHE algorithm and transmits it to the cloud [9]. Their implementation focused on analyzing and evaluating cardiac risk factors in the cloud by processing encrypted health data uploaded in a secure and privacy-preserving manner. They processed the data exchange between the doctor and the patient through an architecture that enables the patient to grant authorization to the doctor to access the results through a private FHE key.

Based on the increasing need for secure remote healthcare during pandemics, Kumar *et al.* proposed a hybrid approach of multi-party computing and homomorphic encryption to secure health data [11]. Their model addressed the aspect of secure architecture that ensures the results between the patient and the doctor. This prevents eavesdropping across different systems' entities and ensures trust among the patient community. They achieved the partial homomorphic encryption approach by adopting the Paillier encryption scheme. Despite the security, reliability, and patient-centric approach, it is relatively slow due to processing long ciphertexts instead of plain data.

Scheibner *et al.* presented a system that combines homomorphic encryption and secure multiparty computation to address the limitations of traditional approaches to enhancing privacy during medical data sharing [13]. The authors developed a new homomorphic multiparty encryption approach that enables secure, flexible processing while meeting data anonymization and legal requirements under the General Data Protection Regulation (GDPR). This is done by efficiently transitioning between homomorphic encryption, represented by local computation, and interactive protocols, defined by multiparty computation. Their analysis showed that homomorphic multiparty encryption provides an effective solution compared to using the two techniques alone, and they acknowledged the limitations of computational complexity as one of the drawbacks of full homomorphic encryption.

Guo *et al.* stated that privacy issues exist in the model training and the pre-diagnosis stage in machine learning [14]. They designed a pre-diagnosis scheme that provides healthcare services using logistic regression called Privacy-Preserving Online Medical Pre-diagnosis in a privacy-preserving manner by employing homomorphic encryption techniques. A homomorphic encryption scheme called Boneh-Goh-Nissim was applied to protect the confidentiality of the feature vector x and the pre-diagnosis model ω. Their analysis showed that the proposal resisted security threats and privacy concerns. Furthermore, it is efficient and cost-effective regarding computational and communication burdens [14].

Anwar and Salman proposed a FHE algorithm to protect data privacy on the cloud based on a Super-increasing Sequence. Each character in the plaintext is converted to its American Standard Code for Information Interchange (ASCII) code and encrypted using the $C = S\ (2 \times q \times r) + \mathrm{m}$ algorithm to produce a ciphertext. They added r as a random value for the noise added to the ciphering process and evaluated the randomness through well-known National Institute of Standards and Technology (NIST) tests [15]. Their research results showed that their algorithm performs better than other proposed algorithms, as it can encrypt larger files. As for security, a super-increasing sequence with a subset sum problem that was used provides high security. However, they faced challenges regarding the encrypted file's size, which is larger than the plaintext file. Furthermore, the decryption process takes longer than the encryption process [15].

Sinha *et al.* proposed a privacy-aware surveillance system that ensures secure patient data sharing, leading to timely and effective responses to crisis scenarios (e.g., the COVID-19 pandemic). The proposed new encryption mechanism is based on a fully homomorphic and secure encryption scheme using the ElGamal algorithm. The effectiveness of the contact tracing method was analyzed, and the results showed that the proposed solution effectively provides security and adequate support for the computational needs of contact tracing [16].

Mohammed and Abed proposed a FHE algorithm that uses n prime numbers instead of two. They convert each character in the plaintext to its ASCII code and encrypt it using the $C = M + r \times L \times n$ model [17]. The key consists of L (a large prime integer), r (a random number added as noise), and n (a set of prime numbers multiplied together). Results showed better performance in encrypting different file sizes. The proposed method was compared with the DGHV and SDC schemes, showing superior performance. The security level is high due to using n as a multiplier for the i-Prime Modular Operation. However, the authors mention limitations related to encrypted text size, affecting storage space and the decryption process [17].

2.4. Compliance with Regulations and Standards

18 elements can directly and uniquely identify an individual as defined by the Health Insurance Portability and Accountability Act (HIPAA), such as name, social security number, and phone number. Until the mid-2000s, and based on HIPAA, removing all these fields was considered the simplest and most adequate way to protect privacy and de-identify data [18]. The same case concerns the GDPR, which focuses on personal data relating to a specific, identified, or identifiable person [19]. Homomorphic encryption thus complies with data protection regulations and privacy laws such as HIPAA in healthcare and GDPR in finance [20]. The provisions of the regulation guarantee the right to access information related to processing and allow data subjects to monitor how their data are used [21].

3. PROBLEM STATEMENT

Despite encryption being a vital component of information security systems, specific existing integer-based homomorphic encryption schemes contain critical mathematical vulnerabilities. Specifically, schemes using formulations:

$$C = S\cdot\ (q\cdot 2\cdot r)+M \tag{1}$$

$$C = M+r\cdot L\cdot n \tag{2}$$

$$C = M+L(rK+i) \tag{3}$$

Where:
- C = Ciphertext

- M = Message/Plaintext
- L = Large prime integer
- K = Secret key
- r = Noise/random number
- i = Counter/iteration variable
- n = Product of multiple prime numbers
- S = Parameter greater than the sum of the super-increasing sequence
- q = Sum of chosen numbers from the transformed sequence

Equation (1) in [15], Equation (2) in [17], and Equation (3) in [22], [23] are susceptible to unauthorized decryption when attackers utilize factors that exceed m, effectively bypassing the intended secret prime key. This vulnerability contradicts the fundamental security model of FHE, which aims to protect sensitive healthcare data from privacy disclosure risks during server-side processing. In addition, present implementations fail to specify secure methods for random number generation and lack resistance to pattern detection attacks. With the increasing digitization of healthcare information and application outsourcing, this security gap poses significant risks to patient data privacy. Therefore, an enhanced encryption scheme that eliminates these vulnerabilities while maintaining homomorphic properties is urgently needed.

4. THEORETICAL FRAMEWORK

4.1. Math Model

The encryption scheme is homomorphic when: E(m1) ★ E(m2) = E (m1 ★ m2), ∀m1, m2 ∈ M. Where E is the cipher algorithm, M is the plaintext to be encrypted, and ★ is the mathematical operation. This means encrypting m1 and m2 and then performing the operation ★ on their results is the same as performing the operation ★ on m1 and m2 first and then encrypting the result.

4.2. Noise and Circuit Depth

The noise component is an essential element in the FHE encryption process to ensure that many possible ciphertexts are obtained for each message that needs to be encrypted and thus ensure semantic security [9]. In FHE encryption, there is a direct proportion between noise and the number of homomorphic operations, where a substantial increase in noise has a negative effect and makes it difficult to decrypt correctly. In particular, multiplication increases noise significantly compared to addition, and the encrypted data has to be refreshed regularly by bootstrapping [2]. This depends on the circuit depth; the number of operations (addition and multiplication) can be performed in sequence while still being able to decrypt correctly.

4.3. Architecture and Quality

The security model on which FHE is based essentially consists of two parties: The user who owns the data with the privacy and the software service provider. The purpose and function of FHE is to protect user data from the risks of privacy disclosure that may be exposed to it on the server [9]. The quality and performance of this model depend on the evaluation results of three main criteria: security, speed, and simplicity. The scheme should be consistent, understandable, and within the standards and best practices followed by other practitioners in the field, and thus, be simple, applicable, and producible [10].

5. MATERIALS AND METHODS

5.1. Data Collection

There are significant challenges in accessing high-quality datasets due to data use agreements, privacy concerns, ethics reviews, and costs. Therefore, synthetic data offers an attractive alternative to address these concerns [19]. Synthetic data is artificially generated using a trained model that replicates real data according to its distributions (shape and variance) and structure (attributes' correlations) [24]. When synthetic data is derived from sensitive medical information, the challenge is to ensure that it cannot access details that would lead to the re-identification of individuals [19]. Synthea is an open-source synthetic health simulator that generates synthetic EHRs, and it was proposed by Walonoski *et al.* as an approach to simulate patient information in JSON and CSV format [25].

5.2. Original Scheme Analysis

In the proposed encryption scheme [17], the text is encrypted in a homomorphic way using equation (2), where m is the plaintext message; specifically, the character's ASCII value.

The plaintext is recovered using the modular operation. Let us derive this step-by-step:

$$C \bmod L = (M + r \cdot L \cdot n) \bmod L \tag{4}$$

Then, using the basic property of modular arithmetic:

$$(a+b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n \tag{5}$$

We get:

$$C \bmod L = ((M \bmod L)+((r\cdot L\cdot n) \bmod L)) \bmod L \tag{6}$$

Based on the fact that any number multiplied by a coefficient is divisible by it, then the noise term becomes zero:

$$((r\cdot L\cdot n) \bmod L) = 0 \tag{7}$$

Then the equation simplifies to:

$$C \bmod L = ((M \bmod L)+ 0) \bmod L \tag{8}$$

Since m ∈ [0, L-1] according to [17], it concludes:

$$C \bmod L = (M \bmod L) = M \tag{9}$$

The advantage of this scheme is that it supports homomorphic encryption. It does not require noise management and bootstrapping to reduce noise growth, especially with the multiplication process, as the modulus operation eliminates it.

For addition, taking *mod L*, let the noise term disappear:

$$C\_add = C1+C2 = (M1+r1\cdot L\cdot n)+(M2+r2\cdot L\cdot n) = M1+M2+nL(r1+r2) \tag{10}$$

$$C\_add \bmod L = (M1+M2+nL(r1+r2)) \bmod L = M1+M2 \tag{11}$$

For multiplication, taking *mod L* eliminates any term contains noise:

$$C\_mul = C1\times C2 = (M1+r1\cdot L\cdot n)(M2+r2\cdot L\cdot n) \tag{12}$$

$$C\_mul = (M1\cdot M2)+(M1\cdot r2\cdot L\cdot n)+(M2\cdot r1\cdot L\cdot n)+(r1\cdot r2\cdot L^2\cdot n^2) \tag{13}$$

$$C\_mul \bmod L = M1\cdot M2 \tag{14}$$

However, the authors did not specify how the random numbers are generated and how secure they are. As for security, no proven and tested method has been adopted to produce secure random numbers for cryptographic use. In addition, we identified a critical mathematical vulnerability in this scheme and similar schemes. Any factor of the terms used in the key then m can be used in decryption instead of L, which is claimed to be the only secret big prime integer key used in decryption.

Let f be any factor of the term $r \times n$ such that $f > M$. Then:

$$C \bmod f = (M+r\cdot L\cdot n) \bmod f \tag{15}$$

Since $r \cdot L \cdot n \equiv 0 \pmod f$ (as f divides r or n), this simplifies to:

$$C \bmod f = M \tag{16}$$

For example, using n = 114197120264498507, which is the product of (1009, 1013, 2153, 6553, 7919) primes, then all possible keys that can be used in decryption other than L are 31 keys (primes and products of primes). When the attacker does not need to know all the primes used, then using n of them will be a disadvantage. Unlike RSA, someone needs to get the value of p and q to calculate d and decrypt the message [26].

5.3. Enhanced Scheme Design

Encryption process:

$$C = M+(r1\cdot L)+(r2\cdot L\cdot W(n)) \tag{17}$$

Where:

- L = Secret large prime key
- n = Composite large value of prime numbers
- $r1$, $r2$ = Large random numbers
- $W(n)$ = Product of prime numbers in the current window

Decryption process:

$$M = C \bmod L \tag{18}$$

$$M = (M+(r1\cdot L)+(r2\cdot L\cdot W(n))) \bmod L \tag{19}$$

$$M = (M+(0)+(0)) = M, \text{ when } M< L \tag{20}$$

5.4. Random Number Generator (RNG)

We evaluate two approaches for generating the random values $r1$ and $r2$: First, Counter_Deterministic Random Bit Generator (CTR_DRBG) is a counter-mode deterministic random bit generator implemented based on NIST specifications. Second, Java SecureRandom is a built-in cryptographically secure pseudo-RNG in Java.

5.5. Windowing Technique for Prime Selection

One of the reasons for increasing the size of the ciphertext is to adopt a set n that contains an uncontrolled number of prime numbers. Therefore, we propose an enhanced FHE scheme that provides a technique for adopting specific prime numbers from the n set. This is done by employing a dynamic prime window mechanism while maintaining homomorphic properties. Instead of using a full fixed set of prime numbers when encrypting each character, a sliding window W of size k is defined, where the size of $k < n$. A subrange of primes is extracted for each ASCII character, and the window shifts according to a deterministic function.

We implemented a random windowing approach for the prime selection to enhance security. For each encryption operation, we select a subset of *n* primes based on:

Window size:

$$K(\text{seed}) = k_{\min}+((\text{seed}) \bmod (k_{\max}-k_{\min}+1)) \quad (21)$$

Window offset:

$$O(seed)=\left[\frac{seed}{(k_{max}-k_{min}+1)}\, mod\left(n-K(seed)+1\right)\right] \quad (22)$$

Where:

- k_{min} = Minimum window size
- k_{max} = Maximum window size
- seed = Deterministic value

5.6. Experimental Setup

To evaluate the performance and security of our enhanced scheme, we designed a comprehensive test suite with the following components:

1. Security analysis: We do a vulnerability check regarding the identified gap with a demo example
2. Homomorphic property verification: We test the enhanced scheme regarding homomorphic encryption properties (addition and multiplication)
3. We evaluate the selected RNGs randomness properties using the NIST Statistical Test Suite and choose the more secure one
4. Performance testing: We measure encryption and decryption times and ciphertext expansion ratios across file sizes ranging from 100 bytes to 10 MB
5. We check the role and effectiveness of the windowing technique
6. We apply the technique to practical use cases.

5.7. Testbed

1. Hardware Configuration
   a. Operating System: Windows 10
   b. Architecture: 64-bit operating system, x64-based processor
   c. Processor: Intel(R) Core (TM) i7-6600U CPU @ 2.60GHz 2.81 GHz
   d. Memory: 8.00 GB installed RAM (7.88 GB usable)

2. Software Environment
   a. Programming language: Java (version 20.0.2)
   b. Cryptographic libraries:
      i. Java Security API for SecureRandom implementation
      ii. Custom implementation of CTR_DRBG based on NIST SP 800-90A
   c. Analysis Tools:
      i. NIST Statistical Test Suite (STS)
      ii. Cygwin64 terminal environment for running NIST tests

6. EXPERIMENTS

6.1. Random Number Generation

Hardware or software can produce binary sequences based on cryptographic randomness for random number generation. The generated random number can be used to create keys in cryptographic applications, and common cryptosystems may generate keys randomly and employ them. However, the produced number may have patterns that can be predicted. Hence, tests must be done to check for non-randomness in the generated sequence. The NIST statistical package can be used with 15 tests to test the randomness and identify weaknesses or patterns in the produced binary sequences.

A RNG uses a non-deterministic source based on entropy and processing functions. Any resulting weakness in the entropy source is overcome using the distillation process, thus avoiding producing non-random numbers such as those containing long strings of zeros or ones. On the other hand, the pseudo-RNG relies on deterministic functions to generate pseudorandom numbers through one or more random and unpredictable seed inputs [27]. Pseudorandom numbers appear more random compared to random numbers generated from physical sources. These transformations introduce additional randomness and eliminate the associations between inputs and outputs, so the statistical properties of the output of a pseudo-RNG may be better. They can be generated faster than an RNG [27].

Key requirements such as randomness, unpredictability, uniformity, scalability, and consistency should be satisfied to develop a new random bit sequence generation technique [27]. Regarding unpredictability, there are two types: forward and backward unpredictability. Forward unpredictability means that if the seed is unknown and the previous random numbers in the sequence are known, the next number must be unpredictable. Backward unpredictability means that knowing which values were generated does not lead to the possibility of determining the seed.

Based on NIST SP 800-90A, we can use approved DRBG techniques, such as Hash, Hash-based Message Authentication Code, or CTR. We chose Counter-Deterministic Random Bit Generator for its performance; it uses an approved block cipher with Advanced Encryption Standard and runs at its native speed [28].

A special key K, a counter V, and a block cipher lock box are the basic components of CTR-DRBG. The counter V is encrypted with key K using our block cipher; the output becomes the first random number. The key is kept secret, and the counter will not be reset until it reaches the specified large number limit. In addition, knowing one random number does not lead to the next number without knowing the key. CTR-DRBG has four core functions: Update, Instantiate, Reseed, and Generate (Fig. 1) [28].

On the other hand, we have SecureRandom library that complies with the statistical RNG tests specified in FIPS 140-2 and security requirements for cryptographic modules [29]. For seed management, it is handled internally by Java SecureRandom based on platform security standards. As mentioned, even if set seed is not called, the first call to next bytes will force the SecureRandom object to create a self-seed [29]. SecureRandom automatically seeds itself from operating system entropy sources such as/dev/urandom on Linux/Unix systems and Microsoft CryptoAPI on Windows [30]. Thus, when we don't have a better seeding data source, and SecureRandom uses a highly random data source to seed automatically and securely, we can rely on it, avoiding manual seed management.

Regarding testing RNG using NIST, the recommended minimum input size for the Frequency (Monobit) Test, Runs Test, and Cumulative Sums Test is 100 bits. On the other hand, the input size for the Universal test ranges from 387,840 bits to 1,059,061,760 bits based on the block size [27]. Hence, based on that, we choose to generate sequences of 10,342,400 bits in total at least to test the generated numbers by all tests in the suite properly. For each statistical test, the minimum rate to pass the test is approximately equal to 8 for a sample size of 10 binary sequences [27].

6.2. Vulnerability Check and Security Enhancement

$$C \bmod f = (M+(r1\cdot L)+(r2\cdot L\cdot n)) \bmod f \tag{23}$$

Case 1: Decryption failure when f is a factor of n

$$C \bmod f = (M+(noise)+(0)) \neq M \tag{24}$$

Case 2: Decryption failure when f is a factor of $r1$

$$C \bmod f = (M+(0)+(noise)) \neq M \tag{25}$$

Demo Example: Message $M1 = 2$, Secret key $L = 151$ (prime number greater than $M1$), $n = 10403$ (101×103), $r1 = 847$, and $r2 = 548$:

$$Equation\ (17){:}\ C1 = 2+(847\times151)+(548\times151\times10403) = 860955343$$

$$Equation\ (18){:}\ M1 = C1 \bmod L = 860955343 \bmod 151 = 2$$

Testing decryption with potential factor values of $r1$ and $r2 \times n$ factors (e.g., 1, 7, 11, 77, 121, 847, 101, 103, 274) does not successfully recover $M1$. Hence, decryption now relies solely on the value of the secret key L.

6.3. Homomorphic Properties Verification

Additive Property, based on Equation (17):

$$C\_add = C1+C2 = (M1+(r1\cdot L)+(r2\cdot L\cdot n))+(M1+(r3\cdot L)+(r4\cdot L\cdot n))$$

$$C\_add = (M1+M2)+L(r1+r3)+nL(r2+r4)$$

Decryption of Sum, based on Equations (5), (18):

$$C\_add \bmod L = ((M1+M2)+ L\cdot(r1+r3)+L\cdot n\cdot(r2+r4)) \bmod L$$

$$C\_add \bmod L = ((M1+M2)+0+0) = M1+M2 \quad (assuming\ M1+M2<L)$$

Multiplicative Property, based on Equation (17):

$$C\_mul = C1\times C2 = (M1+(r1\cdot L)+(r2\cdot L\cdot n))\times(M1+(r3\cdot L)+(r4\cdot L\cdot n))$$

$$C\_mul = M1M2+L(r3M1+r1M2)+nL(r4M1+r2M2)+L^2(r1r3)+nL^2(r1r2+r2r3)+L^2n^2(r2r4)$$

Decryption of Product, based on Equations (5), (18):

$$C\_mul \bmod L = M1M2+0+0+0+0+0 = M1M2\ (assuming\ M1\cdot M2<L)$$

7. RESULTS AND DISCUSSION

7.1. NIST Results for RNGs

We evaluated both CTR_DRBG and Java SecureRandom using the NIST statistical test suite. The tests were conducted using 10 separate bitstreams, each containing 1,034,240 bits in a binary file. Table 1 summarizes the comparative results.

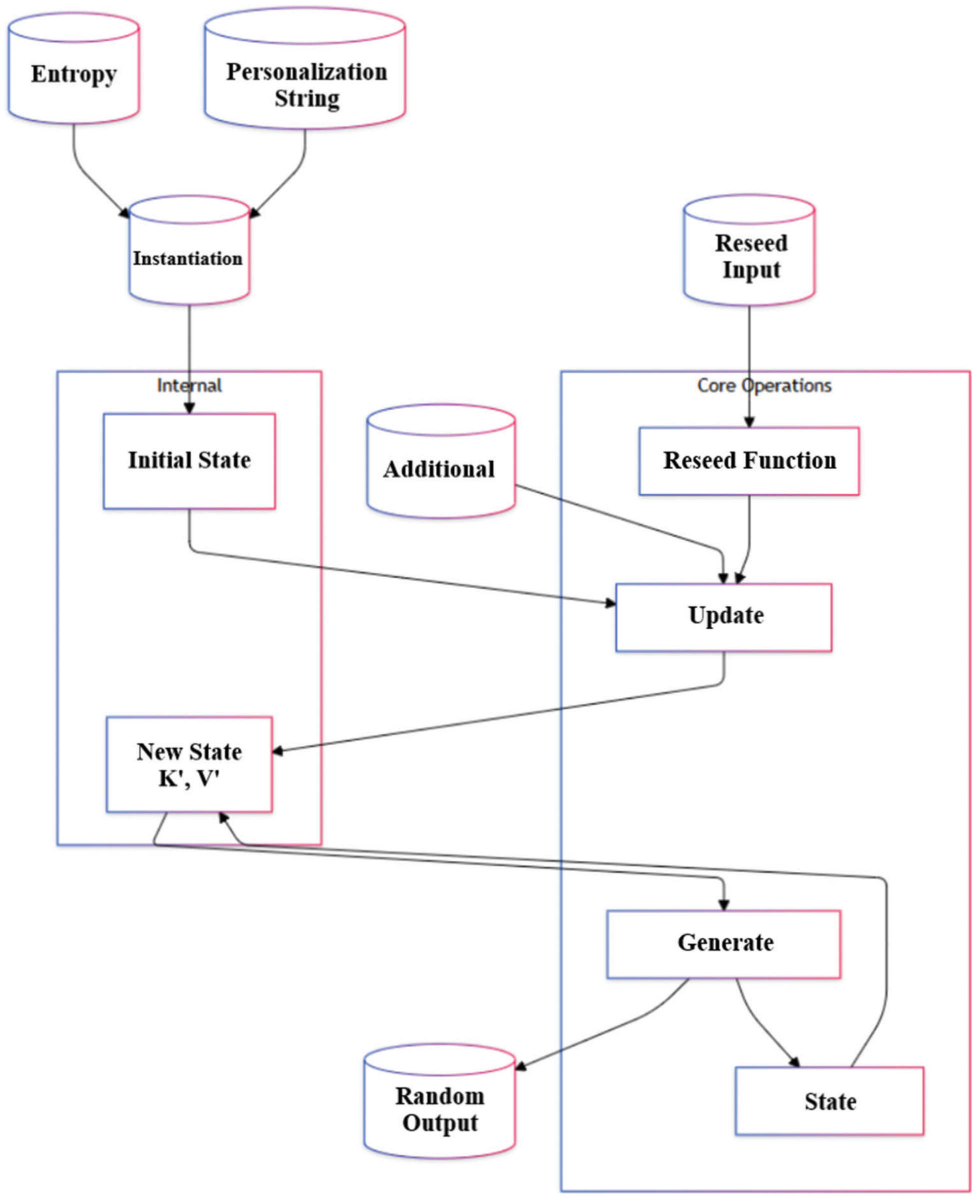


Fig. 1. Counter-deterministic random bit generator (CTR-DRBG) architecture.

A *P*-value of "0.01" is considered the minimum acceptable threshold to pass the NIST test, with stronger random properties when it is close to 1 (Fig. 2). We note that both generators passed all statistical tests successfully, demonstrating their suitability for cryptographic applications. In the Runs test, while CTR_DRBG has a higher P-value, SecureRandom passes with all 10 sequences (10/10), whereas CTR_DRBG failed in one with 9/10. Based on these results, we chose SecureRandom for implementation.

7.2. Performance Comparison

We tested the original *n*-Primes Homomorphic Encryption Algorithm for encryption and decryption times, ciphertext size, and the expansion ratio (Table 2). We use the following parameters: L = 115792089237316195423[truncated] (256 bits), n (product of primes) = 26959942480324040277[truncated] (224 bits), random number bit length = 128 bits. Hence, the combined security level is ~608 bits.

Then we tested the Enhanced Scheme using the following parameters (Table 3): L = 115792089237316195423[truncated] (256 bits), available primes pool = 32 primes (each ~75 bits), window parameters k_min = 2, k_max = 4 (uses 2-4 primes per encryption), security level of n = 151-301 bits (depending on window size), and random number bit length = 128 bits

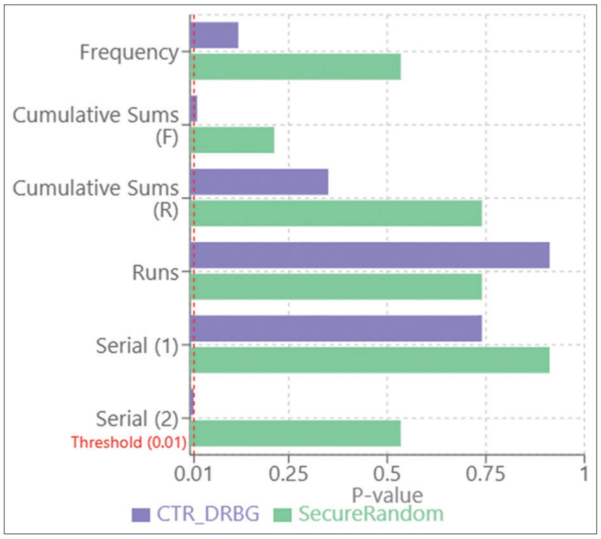


Fig. 2. National Institute of Standards and Technology statistical test *P*-values for CTR_DRBG and SecureRandom generators across selected tests (minimum threshold $P \geq 0.01$ for passing). CTR_DRBG: Counter_Deterministic random bit generator.

for each $r1$ and $r2$. Hence, the effective combined security level is ~535-685 bits and ~610 on average.

For larger files such as the tested file (10 MB), the improved system requires more time in encryption, representing a performance cost 2.89 times higher than the original system (Fig. 3). This additional processing time is due to the computational complexity of the enhanced scheme, which focuses on improving security. Two random values were used, in addition to a windowing approach, which means that the increased security impacted performance.

The decryption time is similar between the two schemes, but the enhanced scheme shows a slightly better advantage for the largest scanned file size. Consequently, decryption times remain comparable between the two schemes (Fig. 4). It should be noted that the calculated time is only for the decryption process using mod; the time required to retrieve the encrypted data from storage is not included.

As for ciphertext size, both systems achieved an expansion size of around 183× for all file sizes, indicating that the added security measures do not significantly impact this factor. The y-axis range is intentionally narrowed to (182–185) to highlight the slight differences (Fig. 5).

7.3. Windowing Technique Role

To test the effectiveness of this technique for security, we chose to test the generated keys using the NIST test suite.

TABLE 1: Comparison of NIST test results for RNGs

Statistical tests	CTR_DRBG		SecureRandom	
	P-value	Proportion	*P*-value	Proportion
Frequency (monobit)	0.122325	10/10	0.534146	10/10
Block frequency	0.739918	10/10	0.350485	10/10
Cumulative sums (forward)	0.017912	10/10	0.213309	10/10
Cumulative sums (reverse)	0.350485	9/10	0.739918	10/10
Runs	0.911413	9/10	0.739918	10/10
Longest run of ones	0.739918	10/10	0.739918	10/10
Rank	0.213309	10/10	0.350485	10/10
Fast Fourier transform	0.350485	10/10	0.534146	10/10
Overlapping template	0.534146	10/10	0.122325	10/10
Universal statistical	0.911413	10/10	0.739918	10/10
Approximate entropy	0.122325	10/10	0.213309	10/10
Serial (1)	0.739918	10/10	0.911413	10/10
Serial (2)	0.008879	10/10	0.534146	9/10
Linear complexity	0.350485	10/10	0.534146	10/10

NIST: National Institute of Standards and Technology, RNGs: Random number generators, CTR_DRBG: Counter_Deterministic random bit generator

TABLE 2: N-prime scheme performance test

File size	Encryption (ms)	Decryption (ms)	Cipher size	Expansion
100.00 Bytes	44	0	17.87 KB	183.01×
500.00 Bytes	69	2	89.35 KB	182.99×
1.00 KB	8	39	182.97 KB	182.97×
2.00 KB	24	5	365.90 KB	182.95×
5.00 KB	56	8	914.78 KB	182.96×
10.00 KB	39	7	1.79 MB	182.96 ×
20.00 KB	54	25	3.57 MB	182.95×
50.00 KB	106	31	8.93 MB	182.95×
100.00 KB	214	92	17.87 MB	182.95×
200.00 KB	635	119	35.73 MB	182.95 ×
500.00 KB	770	300	89.33 MB	182.95×
1.00 MB	1364	608	182.95 MB	182.95×
2.00 MB	3583	1356	365.91 MB	182.95×
5.00 MB	8358	3238	914.77 MB	182.95×
10.00 MB	14668	8578	1.79 GB	182.95×

For the first original n Prime technique, we generated keys of a 1,000,000-bit sequence binary file using the $r \times L \times n$

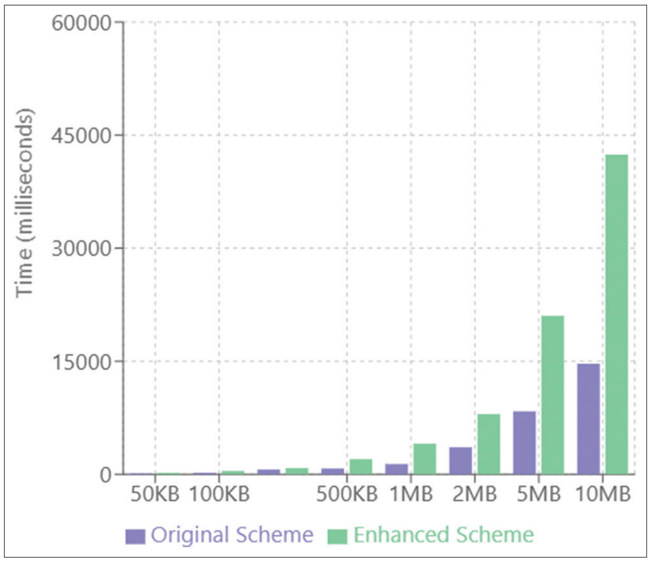


Fig. 3. Encryption time comparison between original and enhanced *n*-Primes homomorphic encryption schemes across different file sizes.

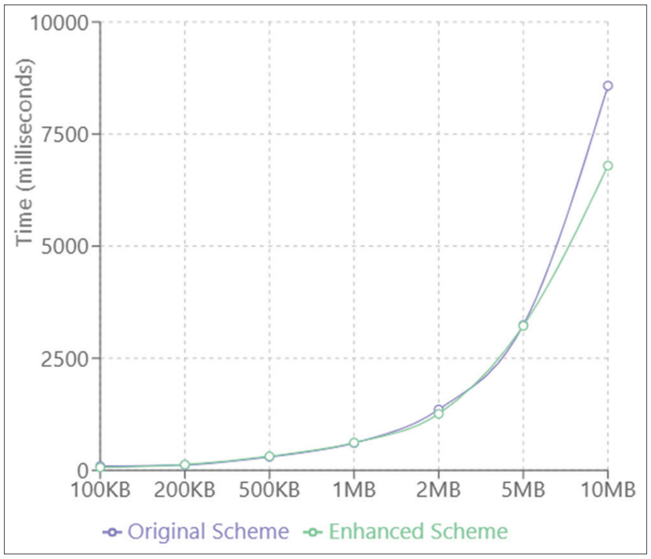


Fig. 4. Decryption time comparison between original and enhanced *n*-Primes homomorphic encryption schemes across different file sizes.

term, where n is pre-computed by multiplying three 76-bit primes. Then, we generated another similar file using the $r1 \times L + r2 \times L \times W(n)$ term of the enhanced technique, but without windowing; n is fixed, pre-computed, and equal to the one used first. Finally, we generated the third bin file based on the enhanced technique with windowing (window size = 3 primes) applied over n from a set that contains 32 different 76-bit length primes. We tested all files with 10 separate bitstreams, each containing 100,000 bits (Table 4).

TABLE 3: Enhanced scheme performance test

File size	Encryption (ms)	Decryption (ms)	Cipher size	Expansion
100.00 Bytes	71	0	17.98 KB	184.13×
500.00 Bytes	37	1	89.57 KB	183.44×
1.00 KB	22	2	183.76 KB	183.76×
2.00 KB	21	4	365.23 KB	182.61×
5.00 KB	48	6	919.09 KB	183.82×
10.00 KB	66	7	1.79 MB	183.66×
20.00 KB	108	14	3.58 MB	183.34×
50.00 KB	205	36	8.96 MB	183.52×
100.00 KB	447	65	17.90 MB	183.34×
200.00 KB	829	126	35.83 MB	183.43×
500.00 KB	2013	312	89.56 MB	183.43×
1.00 MB	4060	617	183.42 MB	183.42×
2.00 MB	7994	1260	366.86 MB	183.43 ×
5.00 MB	21024	3222	917.15 MB	183.43×
10.00 MB	42416	6793	1.79 GB	183.41×

TABLE 4: Comparison of NIST test results regarding key generation

NIST test	Fixed *n* prime	Enhanced with W (n)	Enhanced without W (n)
Frequency	0.002043 (9/10)	0.350485 (10/10)	0.122325 (10/10)
Block frequency	0.017912 (9/10)	0.017912 (10/10)	0.350485 (10/10)
CUMULATIVE sums (1)	0.008879 (9/10)	0.213309 (10/10)	0.739918 (10/10)
Cumulative sums (2)	0.002043 (10/10)	0.534146 (10/10)	0.122325 (10/10)
Runs	0.534146 (10/10)	0.739918 (10/10)	0.739918 (10/10)
Longest run	0.350485 (10/10)	0.350485 (10/10)	0.066882 (10/10)
Rank	0.534146 (10/10)	0.911413 (10/10)	0.534146 (10/10)
FFT	0.350485 (10/10)	0.534146 (10/10)	0.911413 (10/10)
Overlapping template	0.739918 (10/10)	0.911413 (10/10)	0.739918 (10/10)
Approximate entropy	0.213309 (10/10)	0.350485 (10/10)	0.066882 (10/10)
Serial (1)	0.739918 (10/10)	0.213309 (10/10)	0.035174 (10/10)
Serial (2)	0.534146 (10/10)	0.213309 (10/10)	0.213309 (10/10)
Linear complexity	0.017912 (10/10)	0.739918 (10/10)	0.035174 (10/10)

NIST: National Institute of Standards and Technology, FFT: Fast Fourier Transform

The enhanced windowing technique shows the best randomness properties of keys. It won the most tests (Fig. 6), has the highest average P-value (Fig. 7), shows the most consistent randomness across different tests (Fig. 8), and has the largest area on the polar graph (Fig. 9).

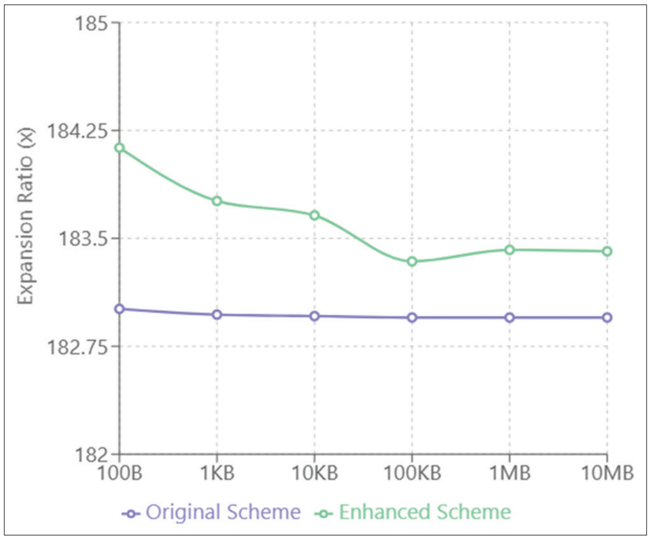


Fig. 5. Ciphertext expansion ratio comparison between original and enhanced *n*-Primes schemes across different file sizes.

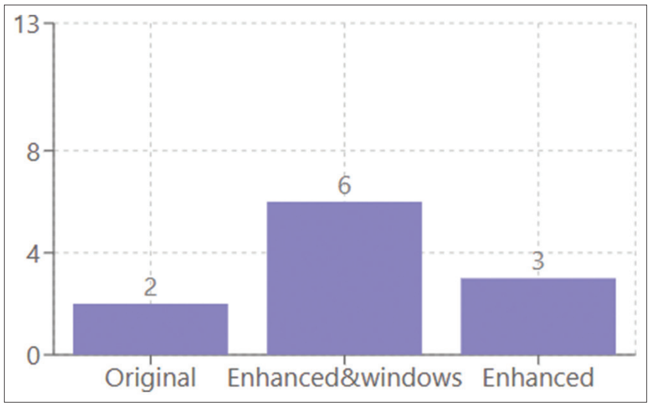


Fig. 6. Number of National Institute of Standards and Technology tests won by original, enhanced without windowing, and enhanced with windowing techniques.

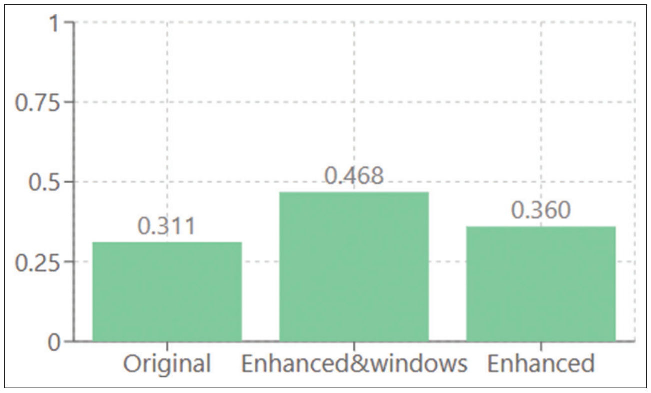


Fig. 7. Average National Institute of Standards and Technology *P*-values for original, enhanced without windowing, and enhanced with windowing techniques.

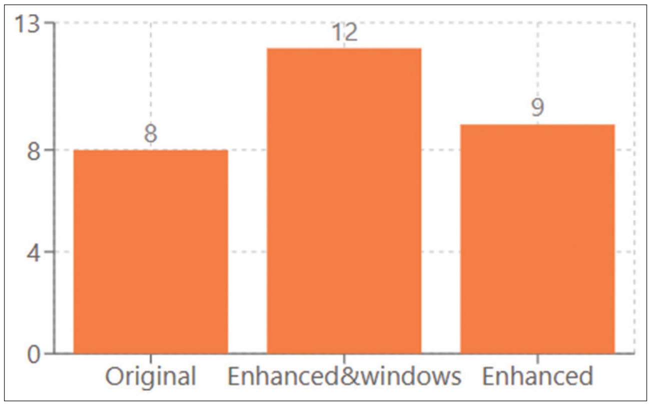


Fig. 8. Number of National Institute of Standards and Technology tests achieving solid randomness ($P > 0.1$) for each key generation technique.

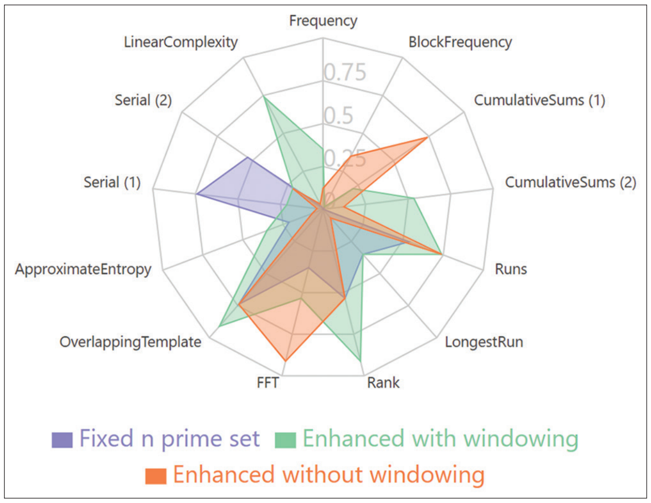


Fig. 9. Security level of the generated keys based on National Institute of Standards and Technology statistical tests.

Logically, using dynamic subsets adds variability, not just mathematical robustness. This technique adds a layer of randomness because each encryption operation uses a different value of n, making it more difficult to detect patterns in ciphertexts. We also chose to have a set of primes and then extract subsets from it instead of trying to generate the primes on the fly when needed, which is computationally expensive.

7.4. Security versus Performance Tradeoff

There is a clear trade-off between security and performance in the enhanced encryption scheme, as computational overhead is added to address vulnerability in decryption attacks using factors instead of a dedicated key. Further computational overhead is added when using windowing techniques to make the ciphertexts' own random properties resistant to pattern detection attacks (Fig. 10). Our improved scheme offers greater security benefits and may be preferred in applications where confidentiality is the priority and performance is secondary. The windowing approach also provides a limited solution by reducing the ciphertext size to a minimum acceptable level or increasing it to a certain level by modifying the window size parameters. This allows for a tailored balance between security and performance, but its impact remains limited.

8. USE CASES

8.1. Medication Cost Analysis

Patient records contain numerical data, such as body mass index, blood pressure, glucose levels, medication doses, treatment rates, and healthcare expenditures, that a third party may need to perform operations on. Synthea™ provides a dataset in CSV and JSON files containing patient demographics, clinical notes, medications, encounters, and regulatory data [31].

Analyzing medication costs typically requires access to sensitive financial data. Research needs (such as access to spending patterns, cost variations, and improvement opportunities) conflict with privacy requirements. We implemented our integer-based homomorphic encryption to enable statistical analysis of costs, such as measuring total spending and average costs for each type of medication, without revealing individual patient data.

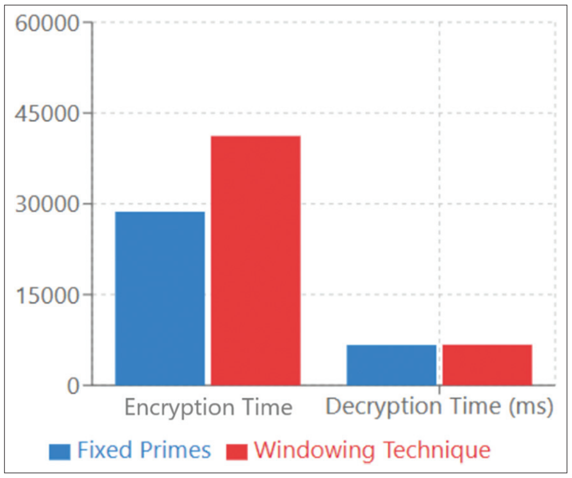


Fig. 10. Encryption and decryption time comparison between fixed primes (without windowing) and windowing approaches for the 10 MB file using the enhanced *n* Primes scheme.

The analysis was performed on a dataset of 42989 medication records, with 131 unique medication descriptions. Cost values were represented in decimals with up to two decimal places, ranging from \$0.99 to \$7015.8 per medication. Since our homomorphic encryption scheme operates on integers, we had to scale the data as a preprocessing step by converting the decimal cost values to integers by multiplying them by 100 (e.g., \$10.25 ← 1025 cents). Then, after applying encryption to these integers and performing homomorphic operations over them, we scale back the results by dividing them by 100 (Fig. 11).

We computed the total spending cost grouping per description by applying homomorphic addition to the encrypted values. Then, we computed the average by plaintext division post-decryption. We compared the results of the homomorphic operations with calculations on plain text data to verify accuracy. As shown in Table 5, our implementation achieved perfect accuracy. As for operational efficiency, performance metrics were recorded with an encryption time of 657 ms for 42,989 records and a decryption time of 7 ms for statistical results.

8.2. CHADS2 Score Calculation

The CHADS2 score is used as a validated tool for predicting stroke risk in patients with Atrial Fibrillation (AF). Five risk factors are assessed with a score ranging from 0 to 6: Congestive heart failure (1 point), hypertension (1 point), age ≥75 years (1 point), diabetes (1 point), and prior stroke/transient ischemic attack (2 points) [32]. Scores of ≥ 3 indicate an increased risk of stroke, which may require anticoagulation [33].

$$CHADS2Score = C+H+A+D+(S\times 2) \tag{26}$$

Patients with AF were identified in the conditions dataset by searching case descriptions (34 out of 1,171 patients). Risk factors (CHADS2 components) were then extracted for each AF patient. We encoded each risk factor as 1 if present and 0 if absent, then homomorphically encrypted the values. Finally, we calculated the score using E(C), E(H), E(A), E(D), and E(S).

The results of calculating encrypted CHADS2 scores showed 100% accuracy compared to plaintext calculations for 34 AF patients identified. 85.3% of these patients were classified as high risk. The total processing time was 309 ms, with 98.4% spent on encryption, 1.3% on decryption, and <1% on calculation.

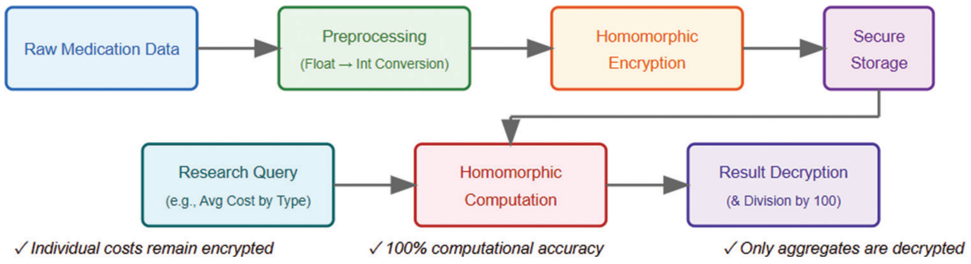
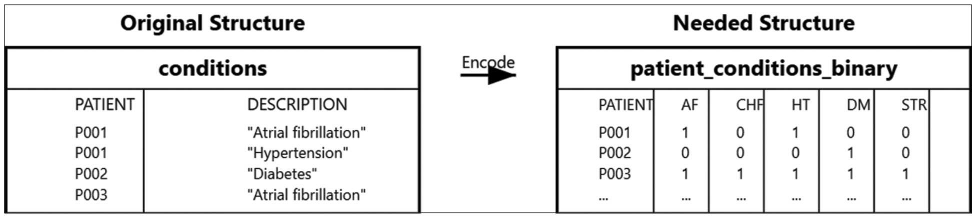


Fig. 11. Medication cost analysis workflow.


Original Structure
Needed Structure
Encode

conditions

PATIENT	DESCRIPTION
P001	"Atrial fibrillation"
P001	"Hypertension"
P002	"Diabetes"
P003	"Atrial fibrillation"

patient_conditions_binary

PATIENT	AF	CHF	HT	DM	STR
P001	1	0	1	0	0
P002	0	0	0	1	0
P003	1	1	1	1	1
...	...	...	...	...	...

Fig. 12. Data structure required to apply homomorphic operations over binary encoded categorical data to ensure full privacy.

TABLE 5: Accuracy comparison for selected medications

Medication description	Average after decryption	Plaintext average	Error %
Insulin Lispro 100 UNT/ML	$1,410.89	$1,410.89	0.000000
Simvastatin 10 MG	$5,891.26	$5,891.26	0.000000
Acetaminophen 325 MG	$7.05	$7.05	0.000000
Hydrochlorothiazide 25 MG	$2,448.78	$2,448.78	0.000000

However, the current technique faces challenges as related math operations require dealing with categorical data. For instance, identifying a patient with AF requires decrypting this information during comparison and encoding, which compromises privacy. Similarly, identifying descriptive risk factors required matching for "hypertension," "diabetes," etc., which required displaying the data unencrypted.

Binary encoding can ensure complete privacy through additional specific structured storage (Fig. 12). One drawback of this method is its storage cost, as it requires as many columns as the expected number of descriptions—about 131 in the current conditions' dataset. For example, a patient with diabetes would have a value of 1, while the remaining 130 descriptions would have a value of 0. All these values would be homomorphically encrypted, which would greatly increase storage requirements.

9. LIMITATIONS

Integer-Only Operations: Converting floating-point values to integers requires a specific system-level scale value; otherwise, metadata for each numerical field must be retained for use in the scale-back process. Therefore, the results need careful conversion back to the original units after decryption.

Limited Operations: Supporting homomorphic addition and multiplication is insufficient for performing other statistical operations (e.g., division and ≥ in the applied use cases). Therefore, complex calculations (such as correlation, regression, variance, etc.) cannot be performed directly on the encrypted data.

Grouping Impact on Security: When operations are required on groups (e.g., aggregating and calculating the average for each medication), metadata such as descriptions should remain unencrypted. Alternatively, it should be encrypted with the same parameters and random numbers to obtain the same ciphertext for each description to enable grouping based on ciphertexts.

10. FUTURE WORK

After implementing the technique, we introduced floating-point numbers to observe how it handles them. The encryption and decryption were successful, but the decimal parts remained visible in the ciphertext. For example, 2.24 as plaintext becomes 860955343.24 ciphertext, with the fractional value "0.24" still intact. Therefore, it is important to investigate this issue further and to establish mathematical methods for securely encrypting the fractional parts in future work.

We also carried out an initial investigation into the homomorphic properties using floating-point numbers, finding that they work for addition but not for multiplication. Therefore, further mathematical analysis is necessary to determine if the current encryption formula needs specific adjustments.

When decrypting 860955343.24, the expected value was 2.24, but the actual result was 2.24000000954, with a slight difference in precision. However, identifying the cause of this discrepancy and assessing the potential for significant deviations are essential for defining acceptable error limits.

11. CONCLUSIONS

This study addressed a key vulnerability that allows using factors for unauthorized decryption instead of the assigned secret key in integer-based homomorphic encryption schemes. Security analyses demonstrated that our improved encryption formula mitigates this vulnerability by ensuring that decryption relies solely on the assigned secret key while preserving additive and multiplicative homomorphic properties. NIST analyses also demonstrated that the random window technique strengthens the scheme against pattern detection attacks by introducing variability and random properties into the ciphertext generation process. Performance evaluations showed that the enhanced scheme maintains close decryption times and identical ciphertext expansion ratios to the original scheme, with slightly improved decryption performance for larger files despite the additional security measures. The medication cost analysis use case achieved optimal accuracy in homomorphic processes with reasonable performance metrics. Future research should focus on expanding the system to support more complex statistical operations directly on encrypted data, improving computational efficiency for real-time applications, and developing support for handling floating-point values. Despite these limitations, our research contributes significantly to creating more secure homomorphic cryptosystems for healthcare applications.

REFERENCES

[1] J. R. Paragas. "*An Enhanced Cryptographic Algorithm in Securing Healthcare Medical Records. In: 2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*". IEEE, Piscataway, pp. 1-6, 2020.

[2] K. Munjal and R. Bhatia. "A systematic review of homomorphic encryption and its contributions in healthcare industry". *Complex and Intelligent Systems*, vol. 9, no. 4, pp. 3759-3786, 2023.

[3] J. Scheibner, M. Ienca, S. Kechagia, J. R. Troncoso-Pastoriza, J. L. Raisaro, J. P. Hubaux, J. Fellay, E and Vayena. "Data protection and ethics requirements for multisite research with health data: A comparative examination of legislative governance frameworks and the role of data protection technologies". *Journal of Law and the Bioscience*, vol. 7, no. 1, p. lsaa010, 2020.

[4] C. S. Kruse, B. Smith, H. Vanderlinden and A. Nealand. "Security techniques for the electronic health records". *Journal of Medical System*, vol. 41, no. 8, p. 127, 2017.

[5] C. Brall, C. Berlin, M. Zwahlen, K. E. Ormond, M. Egger and E. Vayena. "Public willingness to participate in personalized health research and biobanking: A large-scale Swiss survey". *PLoS One*, vol. 16, no. 4, p. e0249141, 2021.

[6] C. K. Yee and M. F. Zolkipli. "Review on confidentiality, integrity and availability in information security". *Journal of ICT Education*, vol. 8, no. 2, pp. 34-42, 2021.

[7] A. A. Abdullah, R. Khalaf and M. Riza. "A realizable quantum three-pass protocol authentication based on hill-cipher algorithm". *Mathematical Problems in Engineering*, vol. 2015, p. 481824, 2015.

[8] S. Murthy and C. R. Kavitha. "*Preserving Data Privacy in Cloud Using Homomorphic Encryption. In: 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*". IEEE, Piscataway, pp. 1131-1135, 2019.

[9] S. Carpov, T. H. Nguyen, R. Sirdey, G. Constantino and F. Martinelli. "*Practical Privacy-Preserving Medical Diagnosis Using Homomorphic Encryption. In: 2016 IEEE 9th International Conference on Cloud Computing (Cloud)*". IEEE, Piscataway, pp. 593-599, 2016.

[10] A. Acar, H. Aksu, A. S. Uluagac and M. Conti. "A survey on homomorphic encryption schemes: Theory and implementation". *ACM Computing Surveys*, vol. 51, no. 4, pp. 1-35, 2019.

[11] A. V. Kumar, M. S. Sujith, K. T. Sai, G. Rajesh and D. J. S. Yashwanth. "*Secure Multiparty Computation Enabled E-Healthcare System with Homomorphic Encryption. In: IOP Conference Series: Materials Science and Engineering*". IOP Publishing, United Kingdom, p. 022079, 2020.

[12] W. Tang, J. Ren, K. Deng and Y. Zhang. "Secure data aggregation of lightweight E-healthcare IoT devices with fair incentives". *IEEE Internet Things Journal*, vol. 6, no. 5, pp. 8714-8726, 2019.

[13] J. Scheibner, J. L. Raisaro, J. R. Troncoso-Pastoriza, M. Ienca, J. Fellay, E. Vayena and J. P. Hubaux. "Revolutionizing medical data sharing using advanced privacy-enhancing technologies: Technical, legal, and ethical synthesis". *Journal Medical Internet Research*, vol. 23, no. 2, p. e25120, 2021.

[14] W. Guo, J. Shao, R. Lu, Y. Liu and A. A. Ghorbani. "A privacy-preserving online medical prediagnosis scheme for cloud environment," *IEEE Access*, vol. 6, pp. 48946-48957, 2018.

[15] M. M. Anwar and A. F. Salman. "A new fully homomorphic cryptosystem based on a super-increasing sequence". *Telfor Journal*, vol. 12, no. 1, pp. 50-55, 2020.

[16] K. Sinha, P. Majumder and S. K. Ghosh. "*Fully Homomorphic Encryption Based Privacy-Preserving Data Acquisition and Computation for Contact Tracing. In: 2020 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*". IEEE, Piscataway, pp. 1-6, 2020.

[17] M. A. Mohammed and F. S. Abed. "An improved fully homomorphic encryption model based on N-primes". *Kurdistan Journal of Applied Research*, vol. 4, no. 2, pp. 40-49, 2019.

[18] O. Mendelevitch and M. D. Lesh. “Fidelity and privacy of synthetic medical data”. 2021. *arXiv*: arXiv:2101.08658.

[19] M. Giuffrè and D. L. Shung. “Harnessing the power of synthetic data in healthcare: Innovation, application, and privacy”. *NPJ Digital Medicine*, vol. 6, no. 1, p. 186, 2023.

[20] E. Mollakuqe, A. Parduzi, S. R. Orcid, V. Dimitrova, S. J. R. Muharremi, M. H. Orcid and J. Qarkaxhija. “Applications of homomorphic encryption in secure computation”. *Open Research Europe*, vol. 4, no. 158, p. 158, 2024.

[21] J. Scheibner, M. Ienca and E. Vayena. “Health data privacy through homomorphic encryption and distributed ledger computing: An ethical-legal qualitative expert assessment study”. *BMC Medical Ethics*, vol. 23, no. 1, p. 121, 2022.

[22] M. A. Mohammed and F. S. Abed. “Cloud storage protection scheme based on fully homomorphic encryption”. *ARO- Science Journal Koya Univercity*, vol. 8, no. 2, pp. 40-47, 2020.

[23] T. Al Attar and M. A. Mohammed. “Fully homomorphic encryption scheme for securing cloud data”. *UHD Journal Science Technology*, vol. 7, no. 2, pp. 40-49, 2023.

[24] M. Hernandez, G. Epelde, A. Alberdi, R. Cilla and D. Rankin, “Synthetic data generation for tabular health records: A systematic review”. *Neurocomputing*, vol. 493, pp. 28-45, 2022.

[25] J. Walonoski, M. Kramer, J. Nichols, A. Quina, C. Moesel, D. Hall, C. Duffett, K. Dube, T. Gallagher and S. McLachlan. “Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record”. *Journal of the American Medical Information Association*, vol. 25, no. 3, pp. 230-238, 2018.

[26] “*RSA Algorithm in Cryptography*”. Available from: https://www.geeksforgeeks.org/rsa-algorithm-cryptography [Last accessed on 2025 Jul 17].

[27] L. E. Bassham 3rd, A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray and S. Vo. "*A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*". NIST Special Publication 800-22 Rev. 1a. National Institute of Standards and Technology, Gaithersburg, MD, 2010.

[28] E. B. Barker and J. M. Kelsey. “*Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)”*. US Department of Commerce, Technology Administration, National Institute, United States, 2007.

[29] “*SecureRandom (Java Platform SE 8)*”. Available from: https://docs.oracle.com [Last accessed on 2025 Jul 25].

[30] “MSC63-J. *Ensure that SecureRandom is properly seeded - SEI CERT Oracle Coding Standard for Java - Confluence*”. Available from: https://wiki.sei.cmu.edu/confluence [Last accessed on 2025 Jul 25].

[31] “*GitHub - Synthetichealth/Synthea: Synthetic Patient Population Simulator*”. Available from: https://github.com/synthetichealth/synthea [Last accessed on 2025 Jun 16].

[32] B. F. Gage, A. D. Waterman, W. Shannon, M. Boechler, M. W. Rich and M. J. Radford. “Validation of clinical classification schemes for predicting stroke: Results from the national registry of atrial fibrillation”. *JAMA*, vol. 285, no. 22, pp. 2864-2870, 2001.

[33] P. S. Goldman and M. D. Ezekowitz. “*Anticoagulation in atrial arrhythmias: Current therapy and new therapeutic options. In: Electrophysiological Disorders of the Heart”*. Elsevier, Netherlands, pp. 1175-1180, 2012.