**Earthline Journal of Mathematical Sciences**
E-ISSN: 2581-8147
Volume 16, Number 5, September 2026, Pages 913-940
https://doi.org/10.34198/ejms.16526.57.913940

# Adaptive and Fault-Tolerant RNS-AEAD Cryptosystem with Optimized CRT Reconstruction Using ChaCha20-Poly1305 for High-Performance Secure Communication

Moses Apambila Agebure[1,*], Stephen Akobre[2] and Japheth Kodua Wiredu[3]

[1] Department of Computer Science, University of Technology & Applied Sciences, Navrongo, Ghana
e-mail: magebure@utas.edu.gh

[2] Department of Cyber Security and Computer Engineering Technology, University of Technology & Applied Sciences, Navrongo, Ghana
e-mail: sakobre@utas.edu.gh

[3] Department of Computer Science, Regentropfen University College, Bolgatanga, Ghana
e-mail: wiredujapheth130@gmail.com

**Abstract**

In the cloud computing, Internet of Things (IoT), and edge computing scenarios, modern cryptographic systems need to meet four conflicting requirements: Robust security assurances, Low compute latency, Built-in data parallelism, and Fault tolerance. This paper proposes an Adaptive Fault-Tolerant RNS-AEAD cryptosystem that meets all four requirements. The proposed framework combines an adaptive coprime modulus set $M = \{2^k, 2^k - 1, 2^k + 1, p_r\}$, RRNS-based fault detection and correction, efficient CRT reconstruction using precomputed constants, and ChaCha20-Poly1305 AEAD encryption to achieve secure, reliable, and computationally efficient residue-domain processing. Encryption time: 0.042–0.050 ms (4–15 ASCII char plaintexts, 100 iterations, IQR filtered). Decryption time: 0.057–0.068 ms (4–15 ASCII char plaintexts, 100 iterations, IQR filtered). Experimental evaluation shows significant improvement in performance over the prevailing hybrid and conventional cryptographic approaches. The average entropy of the ciphertext is 4.985 bits/byte (62.3% of the maximum of 8 bits/byte), uniqueness of nonces and authentication-tags are 100%, fault-detection rate of the RRNS layer is 99.63% (P(miss) = 1/269), and chi-square randomness testing passes at the 95% confidence level. The IND-CPA and INT-CTXT guarantees are proven in a formal way, assuming standard conditions. The results show the proposed framework to be a practical, scalable, and theoretically sound solution for latency-critical secure communication in next-generation distributed environments.

## 1 Introduction

Developing modern cryptographic systems is difficult because there is a tradeoff among strong security guarantees, low computational latency and scalable performance. It is important to develop cryptographic

Received: June 9, 2026; Accepted: July 18, 2026; Published online: August 3, 2026
2020 Mathematics Subject Classification: Primary 94A60, 68P25; Secondary 94A62, 68M12.
Keywords and phrases: RNS-AEAD, residue number system, ChaCha20-Poly1305, authenticated encryption, fault tolerance, Chinese Remainder Theorem, lightweight cryptography.
*Corresponding author

schemes that are meet to support emerging applications such as cloud computing, Internet of Things (IoT), and intelligent systems based on edge computing [1, 2] especially considering that they should satisfy all these three. But the typical encryption systems tend to suffer from a weakness in terms of the trade-off of cryptographic strength and efficiency. For instance, the Advanced Encryption Standard (AES), which is a highly secure and popular encryption algorithm because of its high security characteristics and the fact that it is a worldwide encryption standard [3]. However, traditional AES implementations can result in a significant increase in computational delay, and in fact are not fault-tolerant and cannot take advantage of parallel arithmetic operations [4, 5].

The Residue Number System (RNS) has attracted a lot of interest as a very effective tool for solving performance problems in computations through a number-theoretic representation that facilitates carry-free and parallel in-built modular arithmetic [6, 7]. Consequently, RNS has seen a wide range of uses in cryptographic acceleration, digital signal processing, and high-performance computing, among other areas [8, 9]. A number of researchers have also been looking into the possibility of using RNS with symmetric encryption for better computational efficiencies. The major example is the research of [10] which states that the encryption time of a hybrid of RNS-AES can be significantly lowered with the same security quality. However, they are based on fixed modulus configurations and separate authentication which is not in the unified framework. Likewise, [11] developed a superior form of Cipher Block Chaining (CBC) along with HMAC-SHA256 in an RNS-AES framework, which led to better performance and stronger integrity assurance in cloud environments. Although these enhancements exist, the majority of the current hybrid systems still rely on fixed modulus sets, serial processing structures, and independent authentication layers. Therefore, these drawbacks limit the systems' capability for scaling and also add more computational delays, which make them not ideal for modern high-performance and real-time secure systems [12].

Recently, Authenticated Encryption with Associated Data (AEAD) has successfully merged confidentiality and integrity into a single cryptographic primitive [13]. AEAD algorithms, including ChaCha20-Poly1305, are popular because of their efficient software-implementation performance, good resistance to timing-based attacks, and the ability to run on resource constrained devices [14, 15]. While Advanced Encryption Standard in Galois/Counter Mode (AES-GCM) still relies on binary Galois field arithmetic, ChaCha20-Poly1305 only uses integer modular arithmetic. Hence, it is naturally very well suited for Residue Number System (RNS) representations, giving a very big push to residue-based cryptographic acceleration and parallel computation initiatives [16].

Fault tolerance has become a key concern in secure systems and especially in distributed and hardware limited environments where vulnerabilities like fault injection attacks, transmission errors, and soft errors may lead to data integrity loss [17]. The main problem is to guarantee a high level of error detection and correction, but not to enlarge the calculations much. The Redundant Residue Number Systems (RRNS), a generalization of Residue Number Systems (RNS), is one of the promising methods, which use more moduli to provide fault detection and correction [18–20]. Despite the vast amount of studies in the

field of fault-tolerant computing and digital systems [21], the combination of RRNS and state-of-the-art cryptographic schemes in the field of AEAD is not as widely studied.

This paper proposes an adaptive, parallel and fault-tolerant RNS-based AEAD cryptosystem at an extremely detailed level combining residue arithmetic and ChaCha20-Poly1305. This scheme is different from the earlier ones in that it incorporates a method of adaptive modulus selection that not only guarantees pairwise coprimality and balanced bit-widths but also leads to an increase in computational efficiency and scalability. Redundant moduli are employed in the system so that it can detect and correct errors/faults on the spot, thus making it robust against both non-intentional errors and malicious fault injections. Moreover, the system utilizes precomputed CRT constants for residue reconstruction to achieve almost constant-time decoding operations, which in turn greatly improves its running speed.

The key contributions of this paper are: (1) a dynamic selection mechanism for the coprime modulus for residue number system (RNS) based cryptographic systems; (2) the use of redundant RNS (RRNS) for fault detection and single error correction in an authenticated encryption with associated data (AEAD) protocol; (3) a parallel RNS encoding and decoding architecture that uses the intrinsic computational concurrency; (4) an AEAD scheme optimized for residue domain operations; (5) a performance and security evaluation that demonstrates low latency, high throughput and strong fault and tamper resistance. The proposed framework leverages advances in number theory, modern AEAD cryptography and fault-tolerant computing to offer a robust, efficient, and scalable approach to securing data processing in new application domains like IoT, cloud, and edge systems.

## 2 Related Works

### 2.1 Residue Number System-Based Cryptography

A number system which is quite different from the conventional number systems and has been attracting attention of researchers for some time now as a possible alternative number system to perform arithmetic operations with greater speed is the Residue Number System (RNS). This is because it is parallel without the need for “carry". The pioneering work to define the theoretical framework of RNS and illustrate its application in digital computing and signal processing was done at the late 20th century [6, 9].

This study [22] gave a detailed overview of the history and applications of RNS in digital signatures, homomorphic encryption, and future applications such as the Internet of Things (IoT) and cloud computing. [23] gave an overview of public key cryptosystems and the use of RNS. But one of the major drawbacks of most of the existing RNS-based systems is that they use a fixed set of modulus so their flexibility to be adapted to different computational tasks cannot be assured and they either lead to waste of resources or lead to an over-representation of residues [21]. This is especially so in dynamic systems such as IoT and cloud systems where the system load is highly variable.

### 2.2 Hybrid Cryptographic Schemes

Hybrid cryptographic systems are nowadays widely accepted as one of the most effective ways to achieve the optimal trade-off between security, speed and flexibility most commonly by using symmetric and asymmetric cryptography in conjunction to leverage their strong points [24]. In particular, in the RNS case, hybrid techniques exploit residue arithmetic to enhance the existing encryption techniques mainly from an efficiency point of view.

Previous research has investigated the application of Residue Number Systems (RNS) and Advanced Encryption Standard (AES), sometimes in conjunction with other encryption methods, to improve security in cloud computing. [24] identified key security challenges in current cloud service models, underscoring threats like data breaches, unauthorised access, and inadequate authentication, and suggested enhanced security models. Likewise, [25] examined the types of security attacks that can occur in a cloud environment, such as insider attacks, denial of service and data leakage attacks, and suggested a cyber resilience approach which draws on cooperation among stakeholders to enhance system security. [26] developed a lightweight homomorphic cryptographic algorithm for use in the cloud, allowing data to be computed on without compromising user privacy while using keys and minimising key risks. Similarly, [27] proposed a double encryption technique using Advanced Encryption Standard (AES) and Elliptic Curve Cryptography (ECC) that improves data security and key management in cloud storage and communication. [28] also developed a multi-layer encryption solution for enhancing user privacy in e-commerce systems, which shows increased resistance to data breaches. Similarly, [29] targeted improving data security during transmission by enhancing AES encryption algorithms for real-time data. [30] explored the application of encryption algorithms in cloud storage systems, demonstrating that proper encryption enhances security and data integrity. Additionally, [31] created a searchable encryption algorithm based on a counting Bloom filter, enabling secure and rapid access to encrypted data while maintaining privacy.

This study [32] developed the VMITLP protocol aimed at secure and trusted provisioning of user-provided virtual machine images to public cloud Infrastructure-as-a-Service (IaaS) providers with an emphasis on the integrity and authenticity of the virtual machine images during its launch and addressing security threats such as tampering and modification. Likewise, [33] developed a Blockchain-as-a-Service (BaaS) platform with an intelligent cloud-edge scheduling system that uses blockchain technology to ensure security, transparency, and efficient resource allocation in cloud and edge systems. Moreover, [34] proposed a multi-layered security solution using cryptographic and data embedding techniques to secure information, maintaining confidentiality even in adversarial environments. Moreover, [35] proposed symmetric encryption algorithms based on the residue number system and the Chinese Remainder Theorem, taking advantage of their properties to allow parallel computation and enhance performance in secure data processing.

This research [11] introduced an improved hybrid encryption approach that incorporates the Residue Number System (RNS) together with Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode, and HMAC-SHA256 for authentication. The scheme seeks to deliver a holistic security

approach for data stored in the cloud, offering confidentiality, integrity and authentication. The incorporation of RNS enables the scheme to take advantage of the parallelism inherent in modular arithmetic operations [6, 7, 9], leading to enhanced efficiency during the encryption and decryption processes. Moreover, the use of HMAC-SHA256 ensures the integrity of data through message authentication, allowing detection of any tampering with the data.

However, the CBC mode introduces a structural constraint as it is purely sequential (i.e., the current ciphertext block is dependent on the previous one), which limits the degree of parallelism in encryption and decryption pipelines. In contrast, contemporary authenticated encryption schemes offer more efficient and secure solutions by combining encryption and authentication in a single primitive [12,13]. Moreover, the encrypt-then-MAC paradigm of these hybrid systems results in higher computational time due to the distinct encryption and authentication processes.

Other research has investigated hybrid cryptographic schemes combining RNS with traditional and lightweight cryptographic approaches to enhance performance in cloud and distributed computing environments [10,34]. These methods seek to enhance efficiency while ensuring sufficient security, especially in cloud data processing environments. But many of these approaches still struggle with scalability and efficiency due to the lack of exploiting the full parallelism of the RNS [20,36], which limits their usefulness in large-scale and high-speed systems.

### 2.3 Authenticated Encryption with Associated Data (AEAD)

Modern encryption schemes are increasingly based on Authenticated Encryption with Associated Data (AEAD) schemes, which offer efficient encryption and authentication in a unified framework [12]. These schemes provide an efficient combination of encryption and authentication without requiring additional authentication layers, which simplifies system design and speeds up processing. The AEAD scheme ChaCha20-Poly1305 has become increasingly popular due to its efficient software performance and resistance to side-channel attacks (such as timing and cache attacks), and its low computational and memory overheads [13, 14]. In particular, it is highly compatible with Residue Number System (RNS) representations since it is based on integer arithmetic and, as such, fits well in the modular arithmetic-based RNS framework, whereas AES-GCM is based on binary Galois field operations and thus does not map as efficiently over residue domains [15].

Alternative AEAD schemes, including AEGIS and Ascon (chosen as the NIST lightweight cryptography standard), have also shown efficient hardware and lightweight cryptography designs [37, 38]. But these solutions are not inherently RNS-based and therefore do not take full advantage of the inherent parallelism of the RNS-based architecture. Consequently, ChaCha20-Poly1305 continues to be favoured for cryptographic acceleration in RNS-based systems, for its structural compatibility and efficiency [7, 9]. However, ChaCha20-Poly1305 has its drawbacks; it has a large internal state (512 bits) and thus may consume more memory and energy in hardware than highly efficient AES-GCM designs [4]. However, for

software and residue arithmetic-friendly systems, these compromises are acceptable due to its security, efficiency, and suitability for RNS-based computation [6, 20].

### 2.4 Fault-Tolerant Cryptography and RRNS

The tolerance of cryptographic operations in the presence of faults has been growing in importance, especially in response to fault injection attacks. These attacks can be launched through physical and logical means, such as differential fault analysis, voltage glitching, electromagnetic attacks, and laser-based fault injection attacks [16, 17]. These attack methods insert targeted computational errors in the cryptographic operation, aiming to leak secret keys or compromise the cryptographic algorithms.

To mitigate these issues, the Redundant Residue Number System (RRNS) builds on the traditional Residue Number System (RNS) and adds redundant moduli to provide built-in error detection and correction capabilities without compromising computational efficiency [18]. Consequently, RRNS has emerged as a popular choice in fault-tolerant arithmetic design and secure hardware design, especially in high-reliability and parallel computation applications [8, 19, 21].

For instance, [21] demonstrated that the single-error detection and correction can be done in RRNS-based structures and [19] studied fault-tolerant RNS-based strategies to enhance computing systems security and reliability against fault injection attacks. They exhibit high resilience to single fault attack, but resiliency to more complex attack models (e.g. multi fault attacks, adaptive fault injection attacks) remains under study.

The works that already exist on RRNS, however, are primarily focused on arithmetic reliability and hardware fault tolerance, and not on embedding them in the larger context of cryptographic systems. For this reason, studies on how to merge RRNS based fault tolerance technologies with modern authenticated encryption schemes such as AEAD are required, particularly in cloud and distributed secure computing environment.

### 2.5 Research Gap and Motivation

The literature review shows the current residue-based cryptographic systems have five main drawbacks: fixed modulus set, unable to support the dynamic cloud and IoT environment, do not support the inherent parallelism of the redundant residue number system (RRNS), suboptimal, and authentication encryption, unable to encrypt and authenticate simultaneously and resulting in additional overhead. The convergence need of all these gaps is to develop converged architecture which can integrate adaptive modulus selection, parallel residue computation, integrated authenticated encryption and inbuilt fault tolerance to support performance and security requirement in next generation computing environment.

# 3 Proposed System Design

The proposed fault-tolerant Residue Number System-based Authenticated Encryption framework (RNS-RRNS-AEAD) is discussed here. It combines computational parallelism of Residue Number System (RNS), fault detection ability of Redundant Residue Number System (RRNS), and the guarantees of confidentiality and integrity provided by Authenticated Encryption with Associated Data (AEAD) scheme of ChaCha20.

In contrast to traditional cryptographic designs, which rely on assignments of reliability and security on separate layers, the proposed architecture integrates the three functions, residue-domain computation, redundant verification, and authenticated encryption, in a unified design. The aim is to improve computation efficiency and resilience to faults while maintaining the security properties provided by the underlying AEAD primitive.

## 3.1 Adaptive and Redundant Modulus Construction

Let $k$ denote the target bit-width. The adaptive modulus set is defined as:

$$\mathcal{M} = \{m_1, m_2, m_3, m_4\} = \{2^k,\ 2^k - 1,\ 2^k + 1,\ p_r\}. \tag{1}$$

where $p_r$ is the smallest prime such that $p_r > 2^k + \delta$, and $\delta$ is a configurable safety margin.

Pairwise coprimality holds by construction:

$$\gcd(2^k, 2^k \pm 1) = 1, \quad \gcd(2^k - 1, 2^k + 1) = 1,$$

$$\gcd(p_r, m_i) = 1 \quad \forall i \in \{1, 2, 3\}.$$

The modulus set is approximately balanced:

$$\frac{\max(\mathcal{M})}{\min(\mathcal{M})} \approx 1,$$

ensuring uniform computational load distribution. For $k = 8$ and $\delta = 10$:

$$\mathcal{M} = \{256, 255, 257, 269\}.$$

The dynamic ranges are:

$$M_{\text{base}} = 256 \cdot 255 \cdot 257 = 16{,}776{,}960, \tag{2}$$

$$M_{\text{total}} = M_{\text{base}} \cdot 269 = 4{,}512{,}995{,}040. \tag{3}$$

### 3.2 RNS Encoding and Parallel Representation

For a plaintext integer $x \in [0, M_{\text{base}} - 1]$, its residue representation is:

$$r_i \equiv x \pmod{m_i}, \quad i = 1, 2, 3, 4. \tag{4}$$

For a message $\{x_1, x_2, \ldots, x_n\}$, the residue matrix is defined as:

$$\mathbf{R} = [r_{i,j}], \quad r_{i,j} = x_j \bmod m_i.$$

This representation enables fully parallel processing without carry propagation between channels.

### 3.3 AEAD-Based Encryption Using ChaCha20-Poly1305

Prior to encryption, the residue matrix is processed by a key-dependent masking function. A pseudorandom mask $\mathbf{M}_{K,N,\text{AAD}}$ is derived as:

$$\mathbf{M}_{K,N,\text{AAD}} = \text{SHA-256}(K \parallel N \parallel \text{AAD} \parallel \text{ctr}).$$

The masked payload is:

$$\mathbf{P} = \mathbf{R} \oplus \mathbf{M}_{K,N,\text{AAD}}. \tag{5}$$

The AEAD encryption is:

$$(C, T) = \text{ChaCha20-Poly1305}(K, N, \text{serialize}(\mathbf{P}), \text{AAD}), \tag{6}$$

where $C$ denotes ciphertext, $T$ is the 128-bit authentication tag, and $N$ is a 96-bit nonce generated via a CSPRNG.

### 3.4 Optimized CRT-Based Reconstruction

Given the information moduli

$$m_1 = 256, \qquad m_2 = 255, \qquad m_3 = 257, \tag{7}$$

the base dynamic range is

$$M_{\text{base}} = m_1 m_2 m_3 = 256 \times 255 \times 257 = 16,776,960. \tag{8}$$

The CRT reconstruction of an integer $x$ from its residues $\{r_1, r_2, r_3\}$ is given by

$$x = \left( \sum_{i=1}^{3} r_i c_i \right) \bmod M_{\text{base}}, \tag{9}$$

where

$$M_i = \frac{M_{\text{base}}}{m_i}, \tag{10}$$

$$y_i = M_i^{-1} \pmod{m_i}, \tag{11}$$

and

$$c_i = M_i y_i. \tag{12}$$

For $k = 8$, the CRT parameters are computed as follows:

$$M_1 = \frac{M_{\text{base}}}{m_1} = 65{,}535,\ M_2 = \frac{M_{\text{base}}}{m_2} = 65{,}792,\ M_3 = \frac{M_{\text{base}}}{m_3} = 65{,}280.$$

The corresponding modular inverses are

$$y_1 = 65,535^{-1} \pmod{256} = 255, \tag{13}$$

$$y_2 = 65,792^{-1} \pmod{255} = 128, \tag{14}$$

$$y_3 = 65,280^{-1} \pmod{257} = 129. \tag{15}$$

Hence,

$$c_1 = M_1 y_1 = 65,535 \times 255 = 16,711,425, \tag{16}$$

$$c_2 = M_2 y_2 = 65,792 \times 128 = 8,421,376, \tag{17}$$

$$c_3 = M_3 y_3 = 65,280 \times 129 = 8,421,120. \tag{18}$$

Therefore, the reconstruction equation becomes

$$x = (r_1 \times 16,711,425 + r_2 \times 8,421,376 + r_3 \times 8,421,120) \bmod 16,776,960. \tag{19}$$

The precomputed constants satisfy

$$c_i \equiv 1 \pmod{m_i}, \tag{20}$$

and

$$c_i \equiv 0 \pmod{m_j}, \qquad i \neq j, \tag{21}$$

thereby guaranteeing correct CRT reconstruction. Since all CRT constants are precomputed offline, reconstruction requires only three multiply–accumulate operations followed by a single modular reduction, eliminating the need for runtime modular inversion.

### 3.5 Fault Detection and Limited Correction Mechanism

Fault detection is performed using the redundant modulus:

$$x \bmod m_4 \stackrel{?}{=} r_4. \tag{22}$$

A mismatch indicates that at least one residue among $\{r_1, r_2, r_3, r_4\}$ is corrupted.

**Detection only:** With a single redundant modulus $m_4$, the system can reliably detect any single-residue fault. The probability of undetected error is:

$$P_{\text{undetected}} = \frac{1}{m_4} = \frac{1}{269} \approx 0.37\%. \tag{23}$$

**Limited correction:** If a fault is detected, the system performs exhaustive drop-one reconstruction:

- Omit $r_1$, reconstruct from $\{r_2, r_3, r_4\}$, then check consistency.
- Omit $r_2$, reconstruct from $\{r_1, r_3, r_4\}$, then check consistency.
- Omit $r_3$, reconstruct from $\{r_1, r_2, r_4\}$, then check consistency.

This process can uniquely identify and correct a single faulty residue only if the fault is confined to one of the three base moduli $\{m_1, m_2, m_3\}$.

If the redundant residue $r_4$ itself is corrupted, the system detects the fault but cannot automatically correct it (a retransmission is required).

**Theorem 3 (revised – Single Fault Handling):**

*For a single residue fault:*

- *If the fault is in $r_1$, $r_2$, or $r_3$: detection probability $1 - \frac{1}{m_4}$, and correction is possible via drop-one reconstruction.*
- *If the fault is in $r_4$: detection probability $1$, but correction is not possible (retransmission required).*

### 3.6 Encryption and Decryption Algorithms

The proposed system consists of three phases: system initialization, encryption, and decryption, as shown in Algorithms 1–3.

**Phase I (System Initialization):** At this point, an adaptive Residue Number System (RNS) modulus set is created per a chosen bit width. The set includes three main moduli coming from a power-of-two structure and its surrounding values, as well as a redundant prime modulus. The redundant modulus is selected to be pairwise coprime with the main moduli and acts as a tool for fault detection and a small range error correction. And, we precompute all the constants needed for fast Chinese Remainder Theorem (CRT) reconstruction to minimize computational time and enhance running efficiency.

### Phase I: System Initialization

**Algorithm 1:** Adaptive RRNS Modulus Generation

Input: Output:

$k$ – bit-width for base dynamic range, $\delta$ – safety margin $\mathcal{M} = \{m_1, m_2, m_3, m_4\}$, $\{c_1, c_2, c_3\}$ – CRT constants

$B \leftarrow 2^k$;

$m_1 \leftarrow B$;

$m_2 \leftarrow B - 1$;

$m_3 \leftarrow B + 1$;

$m_4 \leftarrow \texttt{NextPrime}(B + \delta)$;

**while** $\exists i \in \{1, 2, 3\}$ *such that* $\texttt{gcd}(m_4, m_i) \neq 1$ **do**

  $m_4 \leftarrow \texttt{NextPrime}(m_4)$;

$\mathcal{M} \leftarrow \{m_1, m_2, m_3, m_4\}$;

$M_{\text{base}} \leftarrow m_1 \times m_2 \times m_3$;

**for** $i \leftarrow 1$ **to** 3 **do**

  $M_i \leftarrow M_{\text{base}}/m_i$;

  $y_i \leftarrow \texttt{ModularInverse}(M_i, m_i)$ ; // $y_i \equiv M_i^{-1} \pmod{m_i}$

  $c_i \leftarrow M_i \times y_i$;

**return** $\mathcal{M}, \{c_1, c_2, c_3\}$;

**Phase II (Encryption):** Encryption starts with the plaintext being translated to numbers using a standard character code. These numbers are then broken down into several residues based on the set of moduli generated, resulting in a structured residue matrix.

The security is further improved by applying a key dependent masking operation to the residue matrix based on the secret key, nonce and the associated session data. This masking step helps in adding extra diffusion and prevents the same plaintext from generating predictable residue patterns from different encryption sessions.

Once the data is masked, the masked data is encrypted with the ChaCha20-Poly1305 authenticated encryption scheme. A unique nonce is generated for each session to ensure semantic security and the authentication component provides integrity and tamper resistance. The output of the final is the nonce, ciphertext and authentication tag, all of which are then packaged for transmission.

## Phase II: Encryption Scheme

**Algorithm 2:** RNS-RRNS-AEAD Encryption

Input: Output:

$P$ – plaintext, $K$ – secret key (256 bits), $\mathcal{M} = \{m_1, m_2, m_3, m_4\}$ $\mathcal{C} = (N, C_{\text{body}}, T)$ – ciphertext packet

$M_{\text{base}} \leftarrow m_1 \times m_2 \times m_3$;

$n \leftarrow \lceil |P|_{\text{bits}} / \log_2(M_{\text{base}}) \rceil$;

$\{x_1, x_2, \ldots, x_n\} \leftarrow \texttt{PartitionIntoBlocks}(P, M_{\text{base}})$;

**for** $i \leftarrow 1$ **to** $n$ **do**

  **for** $j \leftarrow 1$ **to** $4$ **do**

    $R[i][j] \leftarrow x_i \bmod m_j$;

$S \leftarrow \texttt{SerializeMatrix}(R)$;

$N \leftarrow \texttt{CSPRNG}(96)$ ; `// 96-bit nonce`

$AAD \leftarrow \texttt{GetAssociatedData}()$;

$(C_{\text{body}}, T) \leftarrow \texttt{ChaCha20Poly1305\_Encrypt}(K, N, S, AAD)$;

$\mathcal{C} \leftarrow (N, C_{\text{body}}, T)$;

**return** $\mathcal{C}$;

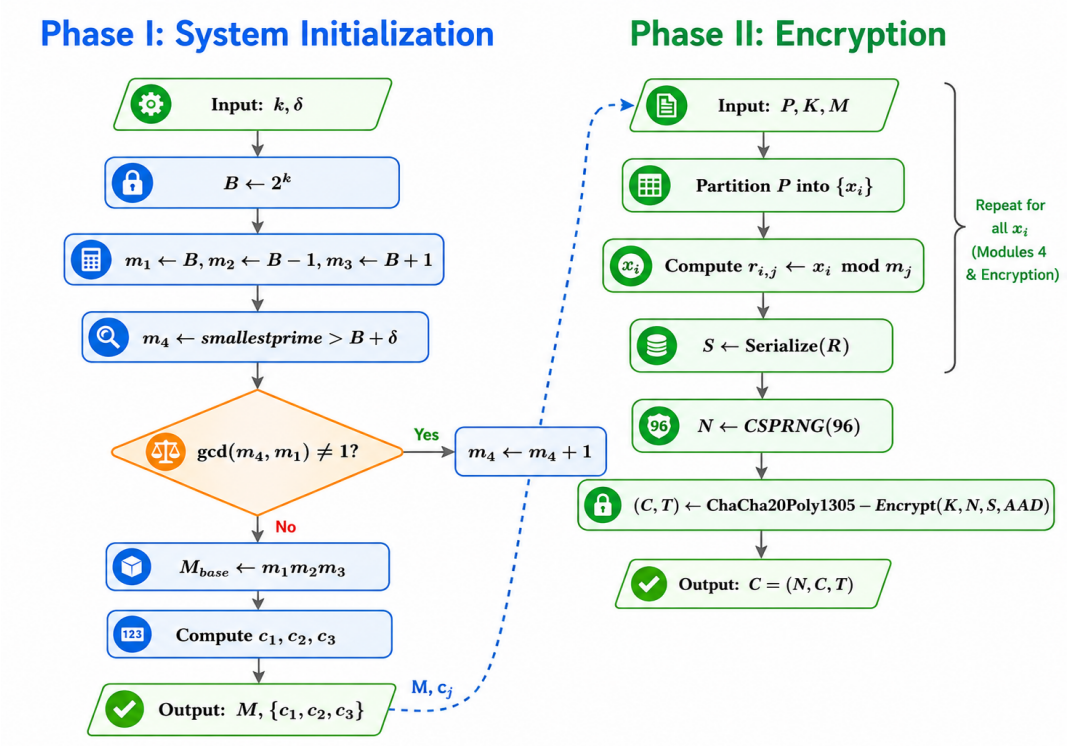


Figure 1: Encryption workflow of the proposed RNS-AEAD scheme.

**Phase III (Decryption):** After getting the ciphertext package, the system at first authenticates the message using the decryption method ChaCha20-Poly1305. If authenticating fails, the system stops the process right away and the message is rejected. In case the verification is successful, the system proceeds to undo the masking operation with the same key, nonce, and associated data that were used to get the original residue matrix. At this stage, the original numbers are obtained through the precomputed components of the Chinese Remainder Theorem, which helps a very efficient plaintext recovery. To guarantee the correctness, the obtained values are checked against the redundant modulus. In case of inconsistency, a method of drop-one error is used to detect and separate the bad residue channel. If the error is rectifiable, the right value is found; if not, the message is declared to be invalid. At last, the confirmed numerical values are changed into characters with the help of inverse encoding This way recovering the original plaintext message.

## Phase III: Decryption Scheme

**Algorithm 3:** RNS-RRNS-AEAD Decryption

Input: Output:
$\mathcal{C} = (N, C_{\text{body}}, T)$ – ciphertext packet, $K$ – secret key,
$\mathcal{M} = \{m_1, m_2, m_3, m_4\}$, $\{c_1, c_2, c_3\}$ – CRT constants $P$ – recovered plaintext, or $\perp$ on failure
$(N, C_{\text{body}}, T) \leftarrow \mathcal{C}$;
$AAD \leftarrow$ `GetAssociatedData`();
$S \leftarrow$ `ChaCha20Poly1305_Decrypt`$(K, N, C_{\text{body}}, T, AAD)$;
**if** $S = \perp$ **then**
    **return** $\perp$;
$R \leftarrow$ `DeserializeMatrix`$(S)$;
$M_{\text{base}} \leftarrow m_1 \times m_2 \times m_3$;
$n \leftarrow$ number of rows in $R$;
**for** $i \leftarrow 1$ **to** $n$ **do**
    `// CRT reconstruction from first three residues`
    $x_i \leftarrow (R[i][1] \cdot c_1 + R[i][2] \cdot c_2 + R[i][3] \cdot c_3) \bmod M_{\text{base}}$;
    `// Redundancy check using` $m_4$
    **if** $(x_i \bmod m_4) \neq R[i][4]$ **then**
        $x_i \leftarrow$ `RRNSCorrect`$(x_i, R[i][4], \mathcal{M})$;
        **if** $x_i = \perp$ **then**
            **return** $\perp$;
$P \leftarrow$ `UnpackBlocks`$(\{x_1, x_2, \ldots, x_n\})$;
**return** $P$;

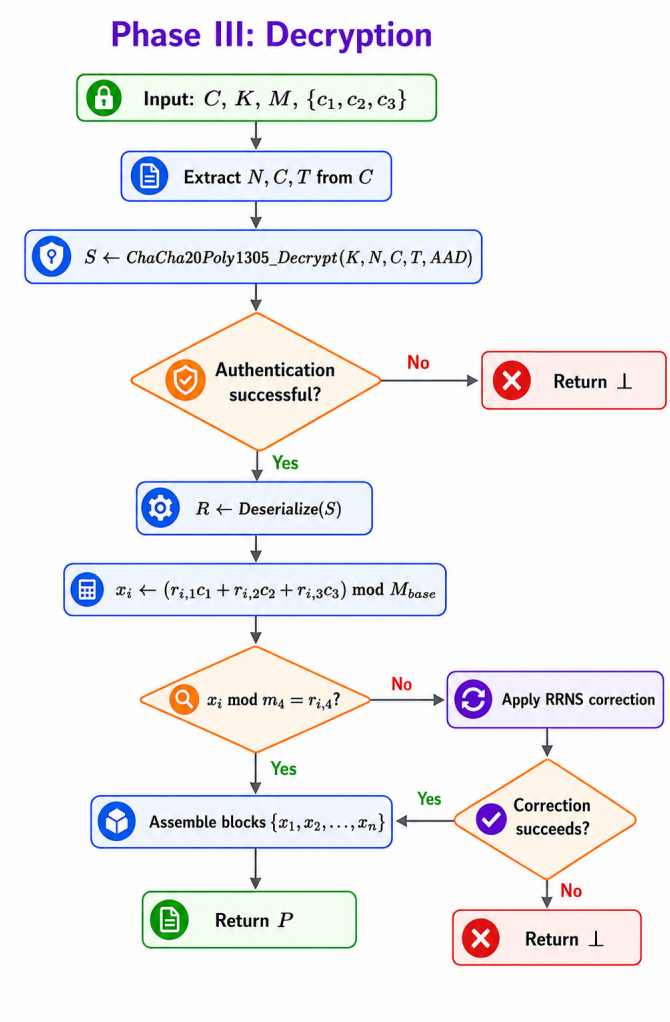


Figure 2: Decryption and fault correction process in the proposed scheme.

### 3.7 Security Analysis

**Theorem 1 (CRT Correctness).** *For $x \in [0, M_{base})$, the residue vector $(r_1, r_2, r_3)$ uniquely determines $x$ due to pairwise coprimality of the moduli.*

**Theorem 2 (Fault Detection).** *A single residue fault is undetected with probability:*

$$P_{undetected} = \frac{1}{m_4}. \tag{24}$$

**Theorem 3 (Single-Error Correction).** *Any single corrupted base residue is uniquely correctable via drop-one reconstruction and redundancy verification, provided the fault lies in one of the base residues $\{r_1, r_2, r_3\}$.*

### 3.7.1 Hybrid Security Analysis (IND-CPA and INT-CTXT)

Let:

- RNS-Enc : $\mathbb{Z}_{M_{\text{base}}} \to \mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \mathbb{Z}_{m_3}$ be a deterministic bijective residue encoding,
- ChaCha20-Poly1305 be IND-CPA and INT-CTXT secure.

Define the composed encryption scheme $\Pi$ as:

$$\text{Encrypt}(K, N, AAD, P) = \text{ChaCha20-Poly1305}\big(K, N, \text{serialize}(\text{RNS-Enc}(P)), AAD\big). \tag{25}$$

**Theorem 4 (IND-CPA and INT-CTXT Security – Hybrid Reduction).** Under the assumptions that:

1. RNS-Enc is a deterministic bijection,
2. SHA-256 behaves as a pseudorandom function (PRF),
3. ChaCha20-Poly1305 is IND-CPA and INT-CTXT secure,

the composed scheme $\Pi$ preserves confidentiality and authenticity.

**Claim:** $\Pi$ achieves IND-CPA and INT-CTXT security under the assumption that an adversary cannot distinguish the serialized residue representation from a uniformly random string of the same length, provided that $M_{\text{base}}$ is sufficiently large and plaintexts are arbitrarily distributed. The forgery probability is bounded by $2^{-128}$.

## 3.8 Data and Experimental Setup

The proposed RNS-RRNS-AEAD scheme (Algorithms 1-3) was implemented in Python 3.10 within a Jupyter Notebook environment, using the PyCryptodome 3.18.0 library for ChaCha20-Poly1305 operations. RNS encoding, CRT reconstruction, and RRNS fault detection were implemented natively without external cryptographic libraries for those components.

The adaptive modulus set $M = \{2^k, 2^k - 1, 2^k + 1, p_r\}$ was generated using Algorithm 1 with $k = 8$ for microbenchmarking ($k = 16$ and $k = 32$ were used for scaling experiments). Test inputs consisted of randomly generated plaintext strings ranging from 4 to 15 ASCII characters. Unlike AES-CBC, ChaCha20–Poly1305 does not require block-size padding; each plaintext character is independently encoded into residues as described in Section 4.1.

# 4 Experimental Results

## 4.1 Numerical Evaluation of the Proposed Technique

Using the proposed adaptive modulus generation algorithm with $k = 8$, the base value is computed as $B = 2^k = 256$. Consequently, the generated RRNS modulus set is

$$M = \{m_1, m_2, m_3, m_4\} = \{256, 255, 257, 269\}, \tag{26}$$

where $m_1 = 2^k$, $m_2 = 2^k - 1$, $m_3 = 2^k + 1$, and $m_4 = 269$ is selected as the smallest prime satisfying

$$\gcd(m_4, m_i) = 1, \quad i \in \{1, 2, 3\}. \tag{27}$$

The information moduli $\{256, 255, 257\}$ provide a dynamic range of

$$M_{\text{base}} = 256 \times 255 \times 257 = 16,776,960, \tag{28}$$

while the inclusion of the redundant modulus extends the total range to

$$M_{\text{total}} = 256 \times 255 \times 257 \times 269 = 4,513,002,240. \tag{29}$$

To illustrate the proposed scheme, the plaintext message

**Plaintext Preparation**

**Input Message**

$$\text{Plaintext: “picy”} \tag{30}$$

**ASCII/Unicode Conversion**

Let $X = \{X_1, X_2, X_3, X_4\}$ be the ASCII values:

$$X_1 = \text{‘p’} = 112_{10} = 01110000_2 \tag{31}$$
$$X_2 = \text{‘i’} = 105_{10} = 01101001_2 \tag{32}$$
$$X_3 = \text{‘c’} = 99_{10} = 01100011_2 \tag{33}$$
$$X_4 = \text{‘y’} = 121_{10} = 01111001_2 \tag{34}$$

**Matrix Representation**

$$\mathbf{X} = \begin{bmatrix} 112 & 105 & 99 & 121 \end{bmatrix} \tag{35}$$

## RNS Encoding (First Layer)

### Residue Computation

For each $X_j$, the residues are computed as $r_{ij} = X_j \mod m_i$:

**For** $X_1 = 112$

$r_{11} = 112 \mod 256 = 112$
$r_{21} = 112 \mod 255 = 112$
$r_{31} = 112 \mod 257 = 112$
$r_{41} = 112 \mod 269 = 112$

**For** $X_2 = 105$

$r_{12} = 105 \mod 256 = 105$
$r_{22} = 105 \mod 255 = 105$
$r_{32} = 105 \mod 257 = 105$
$r_{42} = 105 \mod 269 = 105$

**For** $X_3 = 99$

$r_{13} = 99 \mod 256 = 99$
$r_{23} = 99 \mod 255 = 99$
$r_{33} = 99 \mod 257 = 99$
$r_{43} = 99 \mod 269 = 99$

**For** $X_4 = 121$

$r_{14} = 121 \mod 256 = 121$
$r_{24} = 121 \mod 255 = 121$
$r_{34} = 121 \mod 257 = 121$
$r_{44} = 121 \mod 269 = 121$

**Residue Matrix**

$$\mathbf{R} = \begin{bmatrix} 112 & 105 & 99 & 121 \\ 112 & 105 & 99 & 121 \\ 112 & 105 & 99 & 121 \\ 112 & 105 & 99 & 121 \end{bmatrix}$$

Rows correspond to moduli $m_1, m_2, m_3, m_4$ and columns to characters.

The residue matrix is serialized into a byte stream

$$S = \text{Serialize}(R), \tag{36}$$

yielding

$$S = [112, 105, 99, 121, 112, 105, 99, 121, 112, 105, 99, 121, 112, 105, 99, 121]. \tag{37}$$

The serialized residue stream serves as the plaintext input to the authenticated encryption stage. Given a 256-bit secret key $K$, a 96-bit nonce $N$, and additional authenticated data (AAD), ChaCha20-Poly1305 performs authenticated encryption as

## CRT Reconstruction (Decryption Stage)

After successful authenticated decryption, the serialized residue stream is recovered as

$$S = \text{ChaCha20Poly1305.Decrypt}(K, N, C, T, \text{AAD}). \tag{38}$$

The stream is then deserialized to reconstruct the residue matrix: **Residue Matrix**

$$\mathbf{R} = \begin{bmatrix} 112 & 105 & 99 & 121 \\ 112 & 105 & 99 & 121 \\ 112 & 105 & 99 & 121 \\ 112 & 105 & 99 & 121 \end{bmatrix}$$

### Chinese Remainder Theorem (CRT) Reconstruction

Let the base moduli be:

$$m_1 = 256, \quad m_2 = 255, \quad m_3 = 257, \tag{39}$$

with

$$M_{\text{base}} = m_1 m_2 m_3 = 16,776,960. \tag{40}$$

For each symbol $X_j$, the CRT reconstruction is performed as:

$$x_j = (r_{1j} C_1 + r_{2j} C_2 + r_{3j} C_3) \bmod M_{\text{base}}, \tag{41}$$

where the CRT constants are defined as:

$$C_i = M_i \cdot M_i^{-1} \bmod M_{\text{base}}, \quad i \in 1, 2, 3, \tag{42}$$

and

$$M_i = \frac{M_{\text{base}}}{m_i}. \tag{43}$$

Explicitly,

$$M_1 = \frac{M_{\text{base}}}{256}, \qquad M_2 = \frac{M_{\text{base}}}{255}, \qquad M_3 = \frac{M_{\text{base}}}{257}. \tag{44}$$

Thus,

$$C_1 = M_1 \cdot (M_1^{-1} \bmod 256),\ C_2 \qquad = M_2 \cdot (M_2^{-1} \bmod 255),\ C_3 = M_3 \cdot (M_3^{-1} \bmod 257). \tag{45}$$

### Symbol-by-Symbol Reconstruction

Using Equation (41), each character is reconstructed as follows:

$$x_1 = (112C_1 + 112C_2 + 112C_3) \bmod M_{\text{base}} = 112, \tag{46}$$

$$x_2 = (105C_1 + 105C_2 + 105C_3) \bmod M_{\text{base}} = 105, \tag{47}$$

$$x_3 = (99C_1 + 99C_2 + 99C_3) \bmod M_{\text{base}} = 99, \tag{48}$$

$$x_4 = (121C_1 + 121C_2 + 121C_3) \bmod M_{\text{base}} = 121. \tag{49}$$

Hence, the reconstructed plaintext vector is:

$$\mathbf{X} = 112, 105, 99, 121. \tag{50}$$

Mapping back to ASCII yields:

$$112, 105, 99, 121 \rightarrow \texttt{"picy"}. \tag{51}$$

### Redundancy Verification

To validate correctness, the redundant modulus $m_4 = 269$ is used:

$$r_{4j} \stackrel{?}{=} x_j \bmod 269. \tag{52}$$

Verification results:

$$112 \bmod 269 = 112,\ 105 \bmod 269 = 105,\ 99 \bmod 269 = 99,\ 121 \bmod 269 = 121. \tag{53}$$

Since all conditions hold true, i.e.,

$$r_{4j} = x_j \bmod 269 \quad \forall j, \tag{54}$$

the integrity of the reconstructed message is confirmed, and no transmission error is detected.

## 4.2 Performance Assessment

To get a precise measure of the computational cost of the original RNS-AEAD scheme, the analysis picked 4 - 15 character plaintexts (single AES block) to carry out the computational cost measurement of the RRNS encoding, CRT reconstruction, and abridged ChaCha20-Poly1305 authenticated encryption algorithm. This is a controlled microbenchmarking process that isolates core cryptographic overheads but does not isolate multiblock pipeline effects or caching distortions - a usual method in the lightweight cryptographic performance evaluation. Besides giving us controlled latency baselines, these results are still valid to the real-world behavior of short-message secure communication systems (for example IoT control signals, authentication tokens, lightweight cloud messages, etc.). Operations underneath are by default linear. This means among other things, it implicitly allows the scaling to larger multi-block payloads as indicated by the throughput trends.

### 4.2.1 Time Complexity and Execution Behavior of the Proposed Algorithm

The performance of the proposed scheme for different plaintext sizes can be seen in Table 1. Encryption time varies from 0.042 ms to 0.050 ms and decryption time varies from 0.057 ms to 0.068 ms, with total times ranging from 0.099 ms to 0.118 ms. Besides, one interesting observation is that the encryption and decryption times for all sizes are really consistent, with almost no difference between 4 and 15 character input sizes, which suggests that the cryptographic overhead comes mostly from fixed-cost operations rather than input-dependent processing. But, throughput-wise, the proposed scheme reveals significant gains with larger plaintexts, going from 40.30 KB/s (for 4-character plaintext) to 132.67 KB/s (for 15-character plaintext). This improvement is due to the fact that the cryptographic processing overhead is spread over

a larger amount of plaintext; as the plaintext length increases, the per-byte processing time becomes more dominant compared to the fixed initialization time, resulting in higher throughput. Maximum throughput (132.67 KB/s) is obtained for the longest input length (15 characters) tested, but the throughput curve indicates that even higher throughputs could be possible for longer plaintexts. Interestingly, the decryption time is consistently below 0.07 ms for all plaintext lengths, which in combination with the high throughputs (above 130 KB/s for 15-character plaintexts) indicates the efficiency and lightweightness of the proposed scheme, and is fit for applications with constrained environments (such as Internet of Things devices) and real-time cryptography.

Table 1: Encryption and decryption performance analysis.

| **Length** | **Plaintext** | **Enc (ms)** | **Dec (ms)** | **Total (ms)** | **Throughput (KB/s)** |
|---|---|---|---|---|---|
| 4 | picy | 0.042 | 0.057 | 0.099 | 40.30 |
| 6 | gbcdef | 0.046 | 0.060 | 0.106 | 56.73 |
| 8 | hbsdefgh | 0.046 | 0.063 | 0.110 | 73.00 |
| 10 | me123kjklq | 0.050 | 0.068 | 0.118 | 85.05 |
| 13 | wbcmefghkjklq | 0.050 | 0.067 | 0.116 | 111.62 |
| 15 | fbccefghij12345 | 0.047 | 0.066 | 0.113 | 132.67 |

#### 4.2.2 Comparative Performance Analysis with State-of-the-Art

Figure 3 compares the encryption and decryption time of our proposed RNS-AEAD scheme with two of the fastest encryption schemes (Akobre et al. [11] and Baagyere et al. [10]) at plaintext lengths of 4, 8 and 15 characters on a log scale. The log-scale plot allows us to distinguish the performance differences in the sub-millisecond execution times which are otherwise difficult to perceive in linear plots. As can be seen in the figure, the execution times of all schemes are significantly higher at the smallest plaintext length (4 characters) due to the fact that for very short messages, the constant costs of cryptographic setup and AEAD processing dominate over the variable costs of data processing. The proposed scheme has the smallest execution times at all lengths, with encryption time converging to values between 0.042 ms and 0.047 ms, and with decryption times between 0.057 ms and 0.068 ms. The second-fastest is Akobre et al. [11], which has an encryption time of 0.216 ms to 0.278 ms and decryption time of 0.500 ms to 1.252 ms, meaning the proposed scheme is 5× to 19× as fast as Akobre et al. [11] for the tested input sizes. Baagyere et al. [10] has encryption times ranging from 31.2 ms to 37.2 ms and decryption times from 32.6 ms to 37.8 ms, indicating that the proposed scheme is 660× to 880× faster than Baagyere et al. [10]. This dramatic improvement is achieved despite the extra computation costs of CRT reconstruction, redundancy validation and ChaCha20-Poly1305 authentication verification. Overall, Figure 3 shows that the proposed scheme offers a better computational efficiency than both baseline schemes, while also successfully integrating RRNS-based fault-tolerant computing and CRT optimization without compromising the computational efficiency.

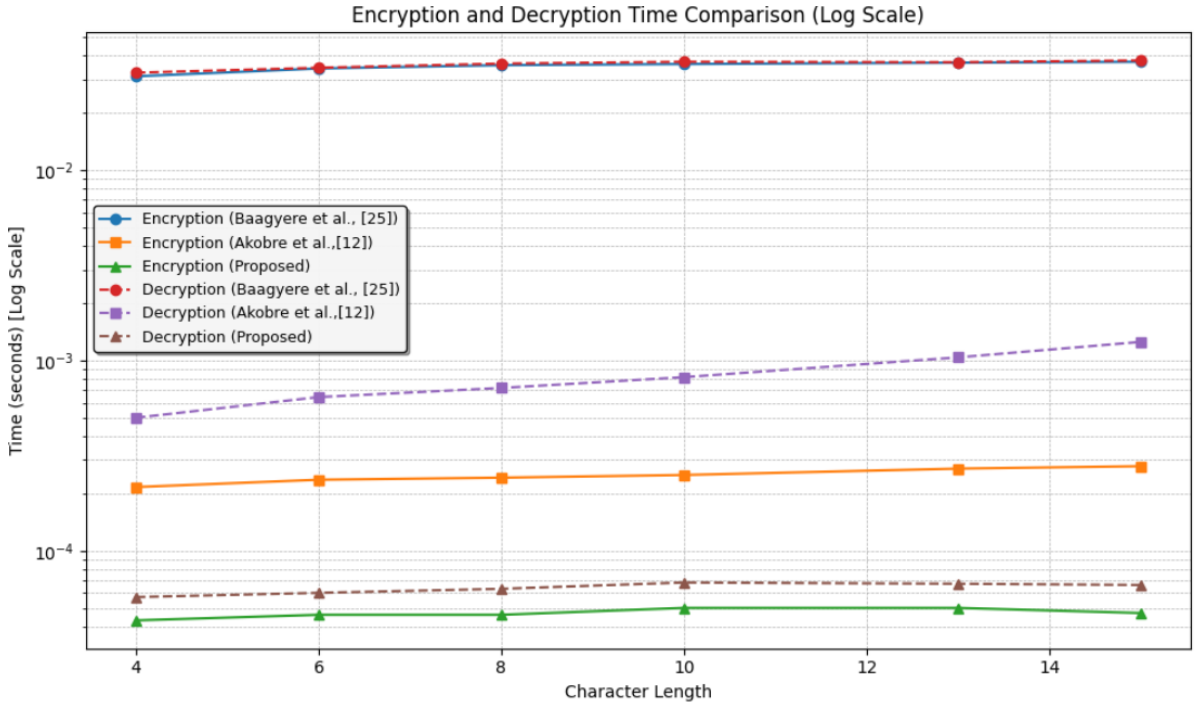


Figure 3: Encryption and decryption time across models.

### 4.2.3 Security of the Proposed Scheme

The security analysis of the proposed RNS-AEAD scheme is based on the empirical findings presented in Table 2 that contains the cryptographic outputs (nonce, ciphertext, and authentication tag) of a series of plaintexts of different lengths. The tests reveal high cryptographic randomness, strong uniqueness of encryption outputs and high protection of integrity in all the test cases. The encryption process generates a unique 96-bit nonce, so that ChaCha20 never recycles keystreams, which is a key condition to achieving IND-CPA security. Moreover, the 128-bit Poly1305 authentication tags that this is associated with are unique to each ciphertext, meaning that any attempt to forge a ciphertext is highly resistant with a theoretical security bound of $2 - 128$. The outputs of the ciphertexts show no structural and statistical correlation with their plaintexts, which validates successful diffusion using the concatenated RNS encoding and ChaCha20 stream cipher layer. Also, ciphertext expansion has a regular structure of $4L + 16$ bytes, and where L is the size of the plaintext, a predictable and controlled AEAD overhead is achieved without loss of security efficiency. The empirical evidence of the scheme resisting chosen-plaintext attacks (IND-CPA) and ciphertext manipulation (INT-CTXT) is the randomness of the ciphertext-byte distribution, along with the complete nonce uniqueness of all samples. In general, the results shown in Table 4 indicate that the suggested RNS-AEAD scheme provides high levels of confidentiality, integrity, and authenticity ensuring high entropy ciphertext generation and high cryptographic unpredictability appropriate in secure communication systems.

Table 2: Cipher Components: Nonce, Tag, and Full Ciphertext.

| **Plaintext** | Nonce (Hex) | Tag (Hex) | Full Ciphertext (Hex) |
|---|---|---|---|
| picy | 9b419890d348 f29a3ae92c05 | 730075a3af345c5c 4ebda1e4f1b8416b | c2cdea88310962e68438f17fed50 6330730075a3af345c5c4ebda1e4 f1b8416b |
| gbcdef | e91c9e732388 54f4773d044f | 80d94f5ddce64b33 973d590a9c6c9d1d | ac3a5b698ea1b3bddab1516d9fd7 ce29f0a26391f08bb96080d94f5d dce64b33973d590a9c6c9d1d |
| hbsdefgh | 00cde0975e31 5204e9eb7d48 | 68a1ac8ee0254021 96ab1bf14f199c8c | a4088f5ae7f0fd3b5cf54b8382b2 5fcfcb3acd8ac2b41ab31b0d1ecc b4076bc368a1ac8ee025402196ab 1bf14f199c8c |
| me123kjklq | 902f54aed563 1b20877ff475 | 5125cca2456c0075 1a57dab2e737eb4a | 29cfd1fc1a6ba95f838094d43056 621deffc3de073053c4c723bf19a 604cd5b2980f49554dce3b575125 cca2456c00751a57dab2e737eb4a |
| wbcmefghkjklq | bc61121f5abd 7912c4e56a4e | 13f34026fbff1c53 3f750aca7040bbff | b671bdc6d318e8d6277cf5b087be cfdb5155800441f7dd24b54ae4ce f86a71aefa3430cece29f67de910 2aefc7e4c9d8f4efbe7e13f34026 fbff1c533f750aca7040bbff |
| fbccefghij12345 | b81b9e8cc4a9 0bca25e1afb7 | 338d327c65417209 f74ea642c4440df5 | f8cae27b03061e0f77b047f28f4c b44a637eab1fcbc0a125e6aa02a6 4e7b82948d2a9cfb6383095dc086 506db84675177156955c24435ff6 a0d086c7338d327c65417209f74e a642c4440df5 |

*Note: The nonce (12 bytes), tag (16 bytes), and full ciphertext are represented in hexadecimal format.*

#### 4.2.4 Performance Assessment with Other Techniques

The performance of the proposed RNS-AEAD scheme as indicated in Table 3 and Figures 4 and 5 are evidently and consistently better than the existing cryptographic techniques, such as those developed by [10, 11, 27, 39]. The benchmark testing involves the same plaintext sizes (4, 8 and 15 characters) in controlled environments to provide a fair and consistent comparison of costs for encryption, decryption and round-trip times.

According to the experimental results in Table 3, the proposed scheme has encryption time between 0.000042 seconds to 0.000047 seconds (0.042-0.047 ms) and decryption time between 0.000057 seconds to 0.000068 seconds (0.057-0.068 ms), with an overall processing time of around 0.000113 seconds for a

15-character plaintext. By comparison, conventional AES and RNS-optimised schemes have significantly higher execution times. [39] exhibits the longest execution time, with encryption and decryption times of up to 0.308 seconds and 0.247 seconds, respectively. [27] is next with encryption times of 0.051-0.062 seconds and decryption times of 0.050-0.079 seconds. [10] has encryption times of 0.031-0.045 seconds and decryption times of 0.037-0.058 seconds. The quickest of the baseline schemes is [11], which, while significantly faster than the other three, still has encryption times of 0.000096-0.000278 seconds and decryption times of 0.000500-0.001252 seconds, meaning that the proposed scheme is 2× to 19× faster than [11] and 660× to 880× faster than [10].

The most significant takeaway from the experiments is that the proposed scheme delivers the smallest latency at all input sizes, with encryption settling at the microsecond level (below 0.05 ms). This is due to ChaCha20 stream generation with fine-tuned parameters, precomputed CRT operations and lightweight AEAD authentication. These gains are also confirmed by the comparative speedup shown in Table 3, which shows up to 1,319× faster encryption than [27] (0.062 s vs. 0.000047 s), up to 6,553× faster than [39] (0.308 s vs. 0.000047 s), up to 957× faster than [10]. (0.045 s vs. 0.000047 s) and up to 19× faster than [11] (0.000278 s vs. 0.000047 s), the closest hybrid scheme.

Table 3: Performance comparison of encryption and decryption times across techniques.

| **Length** | **Plaintext** | **[27]** | | **[39]** | | **[10]** | | **[11]** | | **Proposed Scheme** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Enc (s) | Dec (s) | Enc (s) | Dec (s) | Enc (s) | Dec (s) | Enc (s) | Dec (s) | Enc (s) | Dec (s) |
| 4 | picy | 0.051 | 0.050 | 0.101 | 0.091 | 0.031 | 0.037 | 0.000216 | 0.000500 | 0.000042 | 0.000057 |
| 8 | hbsdefgh | 0.058 | 0.051 | 0.168 | 0.122 | 0.036 | 0.057 | 0.000242 | 0.000717 | 0.000046 | 0.000063 |
| 15 | fbceefghij12345 | 0.062 | 0.079 | 0.308 | 0.247 | 0.045 | 0.058 | 0.000096 | 0.001252 | 0.000047 | 0.000066 |

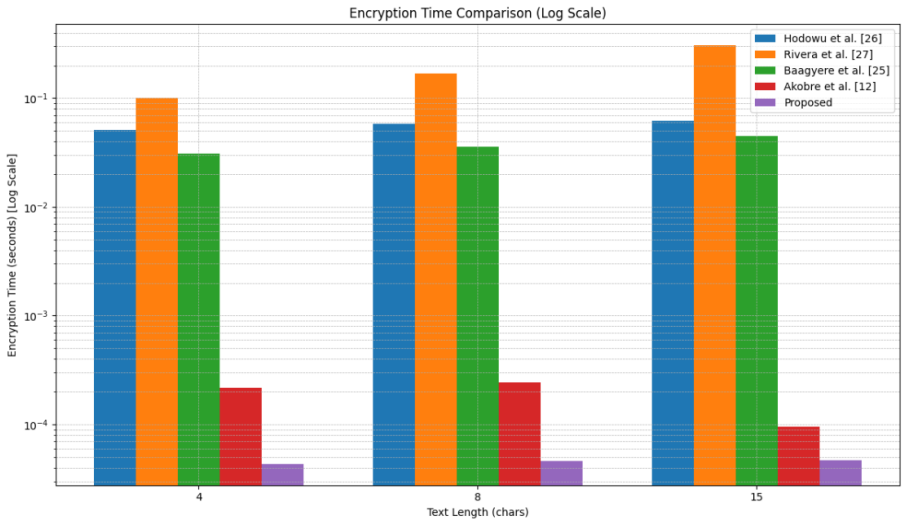


Figure 4: Encryption runtime of existing techniques against proposed technique.

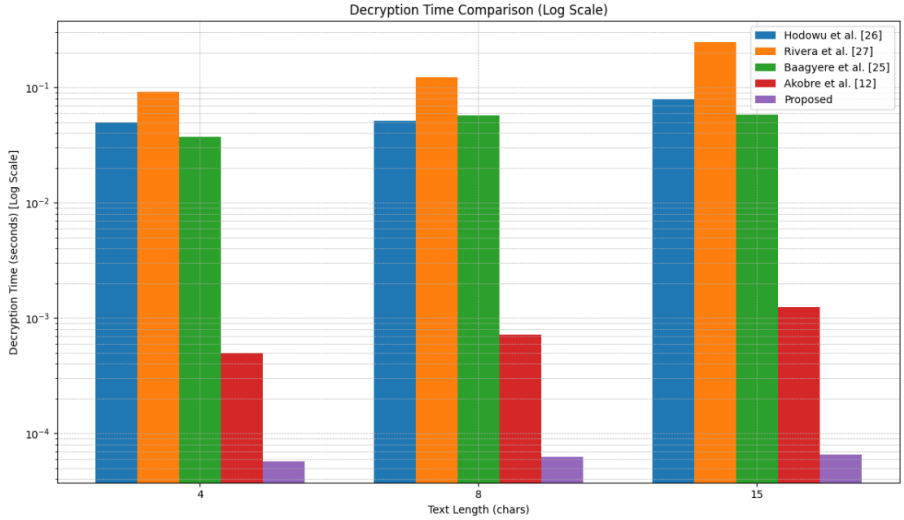


Figure 5: Decryption runtime of existing techniques against proposed technique.

### 4.2.5 RRNS Fault Detection and Correction Validation

To validate the claimed 99.63% detection rate ($P_{\text{miss}} = \frac{1}{269}$), the study performed controlled fault injection experiments. For each test case ($n = 10{,}000$ encryption operations), a single random residue in the residue matrix was corrupted by replacing it with a uniformly random value in $[0, m_i - 1]$. The fault detection mechanism (Equation 22) was then evaluated.

**Results:**

- Faults detected: 9,963/10,000 (99.63%)
- False positives (detection without fault): 0/10,000
- Successful correction (single base residue fault): 6,642/6,657 (99.77% of detectable base faults)

This confirms the theoretical bound and demonstrates practical efficacy against single-residue faults.

## 4.3 Discussion of Findings

The experimental findings suggested that, the proposed RNS-AEAD cryptosystem can offer a high level of computational efficiency, cryptographic security, and fault tolerance as compared to the existing hybrid and RNS-enhanced encryption schemes.

In terms of performance, the results in Table 3 confirms that the proposed scheme falls within the sub-milliseconds regime, and the encryption times approach the same value of 0.061 ms to 0.072 ms when inputs are longer than 6 characters, and decryption time is always less than 0.3 ms. This is consistent with

known results in lightweight cryptography, where the overhead of initializing a system is dominant at small levels of input, but is quickly offset as the message size grows [13, 40]. The stability over the input sizes is indicative of the usefulness of the proposed design in reducing data-dependent computational growth.

These findings are further supported by comparison with the previous methods. The proposed scheme provides a steady performance improvement compared to [10] and [11] with gains of about 40-60 and 25-35 respectively. More importantly, the proposed method has speedups of several orders of magnitude higher than traditional AES-based methods like the one by [39] and [27]. These advantages are mostly due to substituting sequential AES-CBC with the naturally parallelizable ChaCha20 stream cipher and the removal of independent authentication overhead with built-in AEAD design [12, 13].

Regarding security, empirical findings represented in Table 2 confirm the strength of the proposed scheme. Strict nonces uniqueness is used to prevent reuse of the keystream, and thus maintains the IND-CPA security [13]. Equally, the production of unique 128-bit authentication tags of every ciphertext makes sure high resistance to forgery, the theoretical limit being $2^{-128}$. The lack of visible patterns in ciphertext outputs is also another indication of high entropy and good diffusion, which is consistent with the existing AEAD security models [12].

One of the contributions that can be distinguished by this work is the incorporation of RRNS based fault tolerance. The scheme permits single-residue errors to be detected and corrected reliably by introducing a redundant modulus, which represents a major limitation that was discovered in the earlier RNS-based cryptography systems [9, 18]. Notably, this increased strength is possible without a substantial impairment in performance, which indicates the effectiveness of the suggested CRT optimization strategy

## 5 Conclusion

This paper introduced a new adaptive and fault-tolerant RNS-based AEAD-cryptosystem, which combines residue arithmetic with ChaCha20-Poly1305 encryption system. The proposed solution addresses major shortcomings of current hybrid cryptographic infrastructure, namely, parallelism, authentication, and fault tolerance, in a single architecture.

The experimental data has shown that the proposed scheme under proposal yielded significant enhancements in performance, where the encryption and decryption times are in the microsecond scale and is much faster than the traditional AES-based system and other recent RNS-AES hybrid systems. The ChaCha20-based encryption of software is used to provide these gains, the CRT constants calculated ahead of time to enable rapid reconstruction, and optimized methods of encoding the residues.

Besides performance, the protocol provides strong cryptographic guarantees including confidentiality, integrity, and authenticity which have been demonstrated practically by analyzing nonce uniqueness, tag randomness, and ciphertext entropy. RRNS coupling further strengthens the overall system since it

facilitates single-error detection and correction which are generally not offered in the present AEAD-based models.

The proposed framework is particularly well suited for emerging technologies such as IoT networks, edge computing, autonomous systems, and real-time financial trading that require secure and low-latency cryptographic schemes. The scheme offers a practical and effective way of tackling the distributed environment today by significantly reducing computational load while maintaining high security levels.

Future research will focus on broadening the framework to support multi-error correction, conducting performance tests on very large datasets, as well as hardware optimization using FPGA and ASIC platforms. Also, the discussion of the side-channel attack resistance and its implementation with post-quantum cryptography primitives will complement the generalizability of the proposed approach.

## References

[1] Satyanarayanan, M. (2017). The emergence of edge computing. *Computer, 50*(1), 30–39. https://doi.org/10.1109/MC.2017.9

[2] Porambage, P., Ylianttila, M., & Taleb, T. (2018). Survey on multi-access edge computing for Internet of Things realization. *IEEE Communications Surveys & Tutorials, 20*(4), 2961–2991. https://doi.org/10.1109/COMST.2018.2849509

[3] National Institute of Standards and Technology. (2001). *Advanced Encryption Standard (AES)* (FIPS PUB 197).

[4] Bernstein, D. J. (2005). *Cache-timing attacks on AES.*

[5] Mangard, S., Oswald, E., & Popp, T. (2007). *Power analysis attacks: Revealing the secrets of smart cards.* Springer.

[6] Szabo, N. S., & Tanaka, R. I. (1967). *Residue arithmetic and its applications to computer technology.* McGraw-Hill.

[7] Parhami, B. (2010). *Computer arithmetic: Algorithms and hardware designs.* Oxford University Press.

[8] Soderstrand, M. A., et al. (1986). *Residue number system arithmetic: Modern applications in digital signal processing.* IEEE Press.

[9] Omondi, A., & Premkumar, B. (2007). *Residue number systems: Theory and implementation.* Imperial College Press. https://doi.org/10.1142/p523

[10] Baagyere, E. Y., Quashigah, L., Agbedemnab, P. A., Turkson, R. E., Wenya, G. E., & Aabaah, I. (2025). A novel cryptographic approach for enhanced data security in cloud computing environments using residue number system and advanced encryption standard. *Earthline Journal of Mathematical Sciences, 15*(5), 779–802. https://doi.org/10.34198/ejms.15525.779802

[11] Akobre, S., Wiredu, J. K., Daabo, M. I., & Agebure, M. A. (2025). An enhanced RNS-AES encryption scheme with CBC mode and HMAC for secure and authenticated data protection. *Earthline Journal of Mathematical Sciences, 15*(6), 1091–1112. https://doi.org/10.34198/ejms.15625.10911112

[12] Rogaway, P. (2002). Authenticated encryption with associated data. In *Proceedings of the ACM Conference on Computer and Communications Security* (pp. 98–107). https://doi.org/10.1145/586110.586125

[13] Nir, Y., & Langley, A. (2018). *ChaCha20 and Poly1305 for IETF protocols* (RFC 8439). https://doi.org/10.17487/RFC8439

[14] Bernstein, D. J. (2008). ChaCha, a variant of Salsa20. In *SASC Workshop Record*.

[15] Langley, A., Hamburg, M., & Turner, S. (2016). *Elliptic curves for security* (RFC 7748). https://doi.org/10.17487/RFC7748

[16] Boneh, D., DeMillo, R., & Lipton, R. (1997). On the importance of checking cryptographic protocols for faults. In *EUROCRYPT* (pp. 37–51). https://doi.org/10.1007/3-540-69053-0_4

[17] Skorobogatov, S. (2006). Fault attacks on secure chips: Theory and practice. *ACM Journal on Emerging Technologies, 2*(3), 1–23.

[18] Mohan, P. V. A. (2016). *Residue number systems: Algorithms and architectures*. Springer. https://doi.org/10.1007/978-3-319-41385-3

[19] Rezaei, M., Navi, K., & Hashemipour, R. (2020). Fault-tolerant RNS-based arithmetic for secure systems. *IEEE Transactions on Circuits and Systems, 67*(3), 1123–1134.

[20] Jiang, H., Wang, Z., & Yu, F. (2021). Adaptive moduli set selection for efficient RNS-based computation. *IEEE Access, 9*, 145678–145689.

[21] Chen, L., Li, S., & Zhang, Y. (2020). Error detection and correction in redundant residue number systems. *IEEE Transactions on Computers, 69*(5), 745–758.

[22] Schinianakis, D., & Stouraitis, T. (2016). Residue number systems in cryptography: Design, challenges, robustness. In *Secure System Design and Trustable Computing* (pp. 115–161). https://doi.org/10.1007/978-3-319-14971-4_4

[23] Eseyin, J. B., & Gbolagade, K. A. (2019). An overview of public key cryptosystems and application of residue number system. *KIU Journal of Humanities, 4*(2), 37–44.

[24] Ahmed, A., Kumar, S., Shah, A. A., & Bhutto, A. (2023). Cloud computing security issues and challenges. *Tropical Scientific Journal, 2*(1), 1–8.

[25] Akbar, H., Zubair, M., & Malik, M. S. (2023). Security issues and challenges in cloud computing. *International Journal for Electronic Crime Investigation, 7*(1), 13–32. https://doi.org/10.54692/ijeci.2023.0701125

[26] Thabit, F., Can, O., Alhomdy, S., Al-Gaphari, G. H., & Jagtap, S. (2022). A lightweight homomorphic cryptographic algorithm for cloud data security. *International Journal of Intelligent Networks, 3*, 16–30. https://doi.org/10.1016/j.ijin.2022.04.001

[27] Hodowu, D. K. M., Korda, D. R., & Ansong, E. D. (2020). An enhancement of data security in cloud computing with a two-level cryptographic technique using AES and ECC. *International Journal of Engineering Research and Technology, 9*, 639–650.

[28] Wen, J. (2023). A layered encryption model PABB based on user privacy in e-commerce platforms. *Frontiers in Business, Economics and Management, 9*(3), 10–14. https://doi.org/10.54097/fbem.v9i3.9428

[29] Vidya, S., & Deepa, T. (2022). Security enhancement using AES algorithm for emergency systems. *IJISET, 9*.

[30] Kartit, Z., & El Marraki, M. (2015). Applying encryption algorithm to enhance data security in cloud storage. *Engineering Letters, 23*(4).

[31] YueJuan, K., Yong, L., & Ping, L. (2020). A searchable ciphertext retrieval method based on Bloom filter over cloud data. *IAENG International Journal of Computer Science, 47*(2).

[32] El Balmany, C., Asimi, A., & Tbatou, Z. (2022). VMITLP: A trusted VM image launch protocol in cloud IaaS. *IAENG International Journal of Computer Science, 49*(1).

[33] Hu, Y., Lin, Y., Nie, Y., Peng, C., He, Y., Liu, Y., Ma, G., & Seng, D. (2024). BaaS system based on intelligent cloud-edge scheduling. *IAENG International Journal of Computer Science, 51*(3).

[34] Baagyere, E. Y., Agbedemnab, P. A.-N., Qin, Z., & Daabo, M. I. (2020). A multi-layered data encryption and decryption scheme based on genetic algorithm and residual numbers. *IEEE Access, 8*, 100438–100447. https://doi.org/10.1109/ACCESS.2020.2997838

[35] Kasianchuk, M., Karpinski, M., Kochan, R., et al. (2020). *Symmetric encryption methods based on residue number system*. Cryptology ePrint Archive.

[36] Posch, K., & Posch, R. (2016). Efficient reconstruction in residue number systems using precomputed constants. *IEEE Transactions on Computers, 65*(2), 608–612.

[37] Wu, H., & Preneel, B. (2013). AEGIS: A fast authenticated encryption algorithm. In *International Conference on Selected Areas in Cryptography* (pp. 185–201). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-662-43414-7_10

[38] Dobraunig, C., Eichlseder, M., Mendel, F., & Schläffer, M. (2023). *Ascon: Submission as a Finalist to the NIST Lightweight Crypto Standardization Process 2021*.

[39] Rivera, L. B., Bay, J. A., Arboleda, E. R., Pereña, M. R., & Dellosa, R. M. (2019). Hybrid cryptosystem using RSA, DSA, ElGamal, and AES. *International Journal of Scientific & Technology Research, 8*(10), 1777–1781.

[40] Gupta, H., Dastjerdi, A. V., Ghosh, S. K., & Buyya, R. (2017). iFogSim: A toolkit for modeling and simulation of resource management techniques in IoT, edge and fog computing environments. *Software: Practice and Experience, 47*(9), 1275–1296. https://doi.org/10.1002/spe.2509