IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 1 September 2023, accepted 6 October 2023, date of publication 13 October 2023, date of current version 19 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3324612

RESEARCH ARTICLE

Extended-Chacha20 Stream Cipher With Enhanced Quarter Round Function

VICTOR R. KEBANDE, (Member, IEEE)
Secure Distributed Systems (SDS), Department of Computer Science (DIDA), Blekinge Institute of Technology, 371 79 Karlsksrona, Sweden
e-mail: victor.kebande@bth.se

This work was supported by Blekinge Institute of Technology, Sweden, through Grant Funded Research.

ABSTRACT Chacha20 is a widely used stream cipher known for using permutation functions to enhance resistance against cryptanalysis. Although the existing literature highlights its strengths, it is worth further exploring its potential susceptibility to differential attacks. This paper proposes an Extended Chacha20 (EChacha20) stream cipher, which offers a slight improvement of Chacha20. It incorporates enhanced Quarter Round Functions $QR - F$ with 32-bit input words and *Add*, *Rotate*, and *XOR* (*ARX*) operations on 16, 12, 8, 7, 4, and 2 constants. Using these improved $QR - Fs$, we expect EChacha20 to be more secure and effective against attacks than Chacha20. The threat model leveraged in this paper considers attacker assumptions based on the Bellare-Rogaway Model (B-RM) and the Chosen Plaintext Attack (CPA) to assess the potential security weaknesses. Then, the study analyzes the EChacha20 cipher using the NIST Statistical Test Suite (NSTS) and demonstrates its effectiveness against differential cryptanalysis. A differential attack addresses this challenge, where the study comprehensively analyses the differences between original and flipped bits. The NSTS has been used to statistically analyze the outcome for uniformity and evaluate the randomness of generating sequences of tests considering 1000 tests based on a range of [0, 1]. Uniformity is evaluated based on the $p - values$ test against a battery of *passing sequences*, and 100% is achieved from *Runs* and $Serial(2) : Test\ 1$, respectively. The performance evaluation metrics leveraged include encryption speed, decryption speed, and memory usage. Based on the test conducted, it has been observed that with increased $QR - F$, EChacha20 maintains a good balance in speed although slightly higher than Chacha20; however, with also slightly high memory usage compared to Chacha20. Despite that, a comparative study has been conducted against state-of-the-art studies, and the outcome has been reported to show the significance of the current study. Ultimately, the outcome indicates that the EChacha20 cipher has improved $QR - F$ and security properties compared to Chacha20 and may provide a more robust encryption solution for various applications.

INDEX TERMS Chacha20, Salsa20, stream cipher, Quater round function, EChacha20.

I. INTRODUCTION

The ancient secret messaging techniques date back to Scytale, which the Spartans leveraged during the Persian War in the Third Century BC, and communication in 1084 BC during the fall of Troy [1]. While secret messaging techniques are very old, cryptography has taken several years to be independently invented [2], [3], [4], [5]. For example, early inventions date back to Caesar's ciphers by Gaius Julius Caesar in epistolae (Letters from Caesar, which allowed messages to be transmitted based on the changed orders. It is also seen that it was difficult to distinguish an algorithm used for encryption from the secret key based on ancient Cryptology. This was never solved amicably until the Kerckhoffs principle highlighted that the security of an encrypted message did not have to rely on the protection of the encryption algorithm [6] but on the secrecy of the key.

The transformations ever since have seen Caesar ciphers being structured to modern-based ciphers through substitution with keyspaces of 26! which is also seen as quite an exhaustive search for the modern computer, and also it suffers from weaknesses that can allow it to be broken

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamad Afendee Mohamed.

when frequencies are analyzed. Further, these approaches need obfuscation techniques, as seen in homomorphic ciphers and Beale cipher [2], [7], -a poly-alphabetic cipher that uses simple substitutions for each letter dated from 1568, mainly using the Enigma Machine [8], [9], [10].

The objective of the above-mentioned ancient techniques is to guarantee security from the letter frequencies; however, the modern evolved ciphers are seen from a stream and block cipher perspective. Block ciphers allow encryption to be done to a data block [11]. In contrast, stream ciphers allow bit-by-bit encryption where they are seen as a periodic approach if a given keystream, k, can do a repetitive action for some given characters [12].

Chacha20 is a stream cipher and a variant of Salsa20, defined by Bernstein during the eSTREAM project at EU ECRYPT [13]. It uses 8-round cipher Salsa 20/8 256-bit keys. The main objective of this paper is to provide an Extended Provable Chacha20 Stream Cipher, which has the potential to be leveraged in secure communications in various applications based on its performance. Specifically, the study introduces a modified version of the Chacha20 quarter round function ($QR-Fs$), herein referred to as ''Extended Chacha20 Quarter Round'' (ECQR), that improves the diffusion of the cipher and strengthens its security against potential attacks. In its nature, Chacha20 can replace the Salsa20 round functions by allowing doubling elements within a single cycle [14]. Similarly, it is the authors' opinion that, based on the same approach of doubling the elements, EChacha20 could have a slightly perturbed diffusion during the process to strengthen the cipher.

A. CONTRIBUTIONS

The contributions that have been proposed in this paper can be summarized as follows:

- The author proposes an Extended Chacha20 (EChacha20) stream cipher that uses more rounds of the quarter-round function ($QR - Fs$) for improved security. Additional rounds of the $QR - Fs$ in the EChacha20 cipher can provide stronger resistance against known cryptographic attacks, such as differential and linear cryptanalysis, which could enhance the confidentiality, integrity, and authenticity of sensitive data.
- A test against the differential cryptanalysis attack was conducted to evaluate the security of the proposed extended Chacha20 cipher, and observations are documented based on the randomness and passing sequences using the NIST Statistical Test Suite (NSTS) where uniformity and randomness of tests are checked.
- The study has also evaluated performance by measuring the encryption speed, decryption speed, and memory usage of EChacha20 and compared it with the widely used Chacha20 cipher. This evaluation has been used to compare EChacha20 and Chacha20 and assess the effects of security enhancements.
- The study has been compared to state-of-the-art ciphers through a comprehensive comparative analysis, and a contextual evaluation of the proposed cipher has also been provided, including a test against the differential cryptanalysis attack. The results demonstrate the improved security and efficiency of the EChacha20 cipher, making it a promising candidate for secure communication applications.

The remainder of this paper is organized as follows. Section II provides Background, while Section III presents the Preliminaries needed to understand the proposed EChacha20 stream cipher. Section IV discusses the Problem Formulation, followed by an explanation of the proposed Extended Chacha20 Stream Cipher $QR - Fs$ in Section V. The test against the differential cryptanalysis attack is given in Section VI. Security Analysis is given in Section VII, while a comparison with existing studies is given in Section VIII. After this, a discussion of the results and implications of this study are presented in Section IX. Section X concludes the paper and suggests future research directions.

II. BACKGROUND

This section discusses modern stream ciphers, focusing on the differences between block and stream ciphers. In addition, this section provides an overview of the fundamental aspects of Salsa20 and Chacha20 ciphers. The preliminaries of this study are discussed in a later section of this paper.

A. MODERN STREAM CIPHERS

Communication systems mainly rely on cryptographic primitives for protection and security. For quite some time, symmetric-based ciphers have been leveraged, owing to the guarantee of data integrity, confidentiality, and hardened systems. Stream ciphers are essential to symmetric cryptosystems with faster encryption and decryption and easy hardware and software implementations [15]. Mostly, these primitives have been preferred because of the functions they use during cipher encryption. A stream cipher takes a given key of n bits with some length and stretches it to a long keystream. Using an Exclusive OR function, the generated keystream is XORed to the plaintext p to provide a ciphertext. To decrypt the ciphertext, the same keystream is generated and XORed with the ciphertext to give the original plaintext. Figure 1 shows a high-level representation of the stream cipher. The plaintext is combined with the keystream generated by a Pseudorandom Number Generator (PNG) to create a ciphertext in the encryption process. To decrypt, the keystream is XORed with the generated ciphertext to revert to the original plaintext.

In addition to the bit-by-bit XOR operation between the plaintext and the keystream, some stream ciphers employ a feedback mechanism that introduces a dependency on the ciphertext of previous blocks. This feedback mechanism, also known as self-synchronizing mode, enhances the security and complexity of the cipher [16]. In these stream ciphers, the plaintext of the current block is not only XORed with the keystream but also with the previous block's

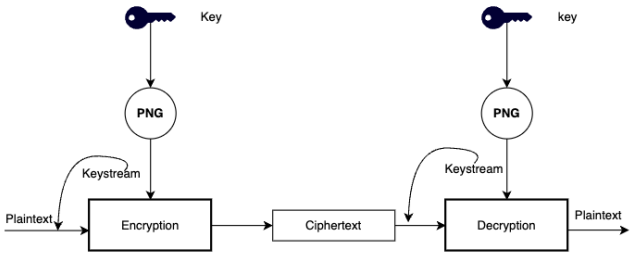

FIGURE 1. High-level representation of the stream cipher with a view of the Pseudo-number generator used to generate the keystream that encrypts bit by bit.

ciphertext [17], [18]. This feedback introduces diffusion, non-linearity, and non-linear feedback shift registers (NFSR), making the cipher more resistant to various attacks [18]. By incorporating the feedback mechanism, stream ciphers can achieve higher levels of security and can offer stronger resistance against known plaintext [19] and chosen plaintext attacks (CPA) [20].

Consequently, existing research has also shown the relevance of having simpler Substitution-Box (S-box) construction in ciphers, owing to them being the key ingredients that provide robustness and the enhancement of security of block ciphers because the S-box has the capabilities of confusing the potential attackers [21]. This is mainly because the strength of any cryptographic technique primarily relies on the S-box. As a result, research that focuses on a modular approach has shown the applicability of non-linear S-box construction using modular inverse and permutation, differential approximation, bit independence, and probabilities. These approaches have been leveraged to deliver the strength of block ciphers [22], [23], [24].

Stream ciphers have mainly been adopted by many international standards and other communication systems because of their effectiveness. For example, E0 stream cipher for Bluetooth, ZUC for 3GPP [25], RC4 for IEEE 802.11, A5/1 stream cipher for European GSM [26], SNOW 3G [27], also, other most recent include the stream ciphers from the ECRYPT project launched by Europe [28], where through the eSTREAM [29], 7 out of 34 algorithms were selected. This include Grain V1 [30], trivium [31], Mickey.V2 [32], Salsa20/12 [29], SOSEMANUK [33], rabbit [34] and HCI-128 [35].

1) SALSA20

Salsa20 belongs to a family of 256-bit stream ciphers, and it was designed as part of the ECRYPT stream cipher project in 2005 and was submitted to the eSTREAM project in Europe [29]. Being a 256-bit cipher, it allows smaller keys because larger ones are more expensive. Its encryption function conducts three operations on the 32-bit words, as shown in Table 1 [13]. Generally, Salsa20 rotations account for most integer operations where the 256-bit keys are expanded and the 64-bit nonce is expanded to 2^{70} byte stream. The general approach is as follows:

TABLE 1. Salsa20 operations on 32-bit word.

No	Action	Result on 32-bit word
1	32 bit addition	sum, $a + b\ mod 2^{32}\ a, b$
2	32 bit XOR	XOR, $a \oplus b$, a, b
3	Constant-distance 32 bit rotation	Rotation $a \lll b$, b is a constant

- A plaintext, P, is encrypted by being XORed with the first P bytes of the stream. The other bytes get discarded.
- The P byte ciphertext is then decrypted by XORing it with the first P byte of that stream.
- A stream of 64 bytes, a 516-bit block is generated, which is an independent hash of the utilized key, the nonce, and the 64-bit block number.
- The key and the nonce are not pre-processed before block generation.

Salsa20, as highlighted in Figure 2, is represented as a single 32-bit word as is shown in Table 1. In addition, Exclusive OR (XOR), bitwise AND, and left shift operators are used to compute the $QR - Fs$ and rotations, respectively. In addition, Salsa20 is built on a Pseudorandom function based on Add-Rotate-XOR (ARX) operations. As is shown in Algorithm 1, the initial round for processing Salsa20 is represented as a 4×4 matrix as is shown below and also in Equation 1;

$$M = \begin{Bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{Bmatrix} \tag{1}$$

In the context of this description, we use the notation x_i to represent values from matrix M, with $[x]_10 \leq i \leq 15$ where i ranges from 0 to 15. Additionally, the round function is represented as described in Equation 2, Equation 3, and Equation 4 respectively. This clarification highlights that x_i is drawn from the set $[M]$ within the specified range, and it confirms how the round function, $R(m)$ is presented in the in Equation 2.

$$R(m) = \left\{Q^{i4}(\mathrm{m})\right\}^T \tag{2}$$

and

$$Q^i = \begin{Bmatrix} x_5 & x_6 & x_7 & q_1 \\ x_9 & x_{10} & x_1 1 & q_2 \\ x_{13} & x_{14} & x_{15} & q_3 \\ x_1 & x_2 & x_3 & q_0 \end{Bmatrix} \tag{3}$$

$$q = \begin{Bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{Bmatrix} = Q = \begin{Bmatrix} m_0 \\ m_4 \\ m_8 \\ m_{12} \end{Bmatrix} \tag{4}$$

Four (4) words are then used for the input and output function Q that are represented by $y = y_0 \parallel y_1 \parallel y_2 \parallel y_3$ and $z = z_0 \parallel z_1 \parallel z_2 \parallel z_3$, where $||$ are represented as concatenations. q represents a vector or an array of four elements: q_0, q_1, q_2, and q_3. These elements are involved

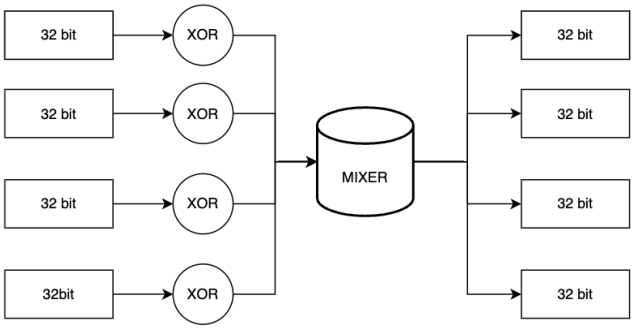

FIGURE 2. High-level view of Salsa20 representation based on an input of 32-bit words, XOR operation, a mixer, and a 32-bit word output.

Algorithm 1 Salsa20 Algorithm

Input: Matrix X, $r \in \mathbb{N}$
Output: $Z = X + X^r$

Begin QR**-Rounds from Matrix:**
for $l = 0$ *to* $r/2$ **do**
  Quaterround (x'_0, x'_1, x'_2, x'_3)
  Quaterround (x'_5, x'_6, x'_7, x'_4)
  Quaterround $(x'_{10}, x'_{11}, x'_8, x'_9)$
  Quaterround $(x'_{15}, x'_{12}, x'_{13}, x'_{14})$
  Quaterround $(x'_0, x'_4, x'_8, x'_{12})$
  Quaterround $(x'_5, x'_9, x'_{13}, x'_1)$
  Quaterround $(x'_{10}, x'_{14}, x'_2, x'_6)$
  Quaterround $(x'_{15}, x'_{13}, x'_7, x'_{11})$
end
return $X + X^r$

in the round function and play a role in the computation of Salsa20. They are derived from the input vector 'm' and used to construct the matrix Q for each round. x in this context represents the elements of the Salsa20 matrix and 'x' refers to each element within this grid, while m represents a vector or an array consisting of 16 elements: $m_0, m_1, m_2, \ldots, m_{15}$. $R(m)$ represents the round function in Salsa20 applied to the input vector 'm'. It involves multiple operations such as addition, rotation, and XOR. The result of applying the round function to 'm' is a modified version of the input vector. A keystream in Salsa20 is defined using a block that uses r rounds as X^r, i.e., the round function represented as $X^r = Round^r(x)$. In this case, the keystream with a block Z represents 20 Salsa rounds, as shown in Equation 5.

$$Z = X + X^{20} \tag{5}$$

The vector for the four words is recalculated as follows:

The variable 'z' represents intermediate values computed during the encryption process. These intermediate values are derived from the original elements of the Salsa20 matrix ('x') using bitwise operations such as XOR and left rotations. Each line in the provided equations calculates a new 'z' value based on previous 'x' and 'z' values.

$$z_1 = x_1 \oplus ((x_0 + x_3) \lll 7) \tag{6}$$
$$z_2 = x_2 \oplus ((z_1 + x_0) \lll 9) \tag{7}$$
$$z_3 = x_3 \oplus ((z_2 + z_1) \lll 13) \tag{8}$$
$$z_0 = x_0 \oplus ((z_3 + z_2) \lll 18) \tag{9}$$

In the context of the Salsa20 encryption algorithm, 'z' represents intermediate values computed during the encryption process. These intermediate values are derived from the original elements of the Salsa20 matrix ('x') and are used in various calculations to create the keystream and mix the key and input data.

As a result, the Salsa20 encryption is defined as shown below in Equation 10

$$Salsa20 = \begin{Bmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ i_0 & i_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{Bmatrix} \tag{10}$$

where c represents the constant values used in Salsa20 encryption, i represents the input values or nonce used in Salsa20 encryption and v represents the counter values used in Salsa20 encryption. In this matrix, the 'k' values represent parts of the encryption key that are mixed with other constants ('c'), input values or nonces ('i'), and counter values ('v') to generate the keystream used in the encryption process. The exact role of each 'k' value in the algorithm depends on the specific step and operation being performed.

2) CHACHA20

Chacha20 stream cipher is a descendant of Salsa20 cipher that was launched in the year 2008 [13] as a replacement for obsolete $RC4$ [36] and was adopted by Google to run over Chrome and Android-based devices. Chacha design principles are based on Salsa20's round functions; however, under the same level of security, Chacha outperforms Salsa in speed. While Chacha20 follows the design principles of Salsa20, it presents different operations when performing column transformations in round functions, where it can update each round twice rather than once. In addition, Chacha20 incorporates unique operations during column transformations that allow it to update each round twice, making it faster than Salsa20. It is represented as shown in Equation 11 and Algorithm 2:

$$X = \begin{Bmatrix} c_0 & 1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & v_0 & v_1 & v_2 \end{Bmatrix} \tag{11}$$

Characters can be updated in Chacha20 as shown below.

$$z_0 = z_0 + z_1, \quad z_3 = z_3 \oplus z_0, \quad z_3 = z_3 \lll 16, \tag{12}$$
$$z_2 = z_2 + z_3, \quad z_1 = z_1 \oplus z_2, \quad z_1 = z_1 \lll 12, \tag{13}$$
$$z_0 = z_0 + z_1, \quad z_3 = z_3 \oplus z_0, \quad z_3 = z_3 \lll 8, \tag{14}$$

Algorithm 2 Chacha20 Algorithm

Input: Matrix X, $r \in \mathbb{N}$
Output: $Z = X + X^r$

Begin QR**-Rounds from Matrix:**
for $l = 0$ *to* $r/2$ **do**
  Quaterround $(x'_0, x'_4, x'_8, x'_{12})$
  Quaterround $(x'_1, x'_5, x'_9, x'_{13})$
  Quaterround $(x'_2, x'_6, x'_{10}, x'_{14})$
  Quaterround $(x'_3, x'_7, x'_{11}, x'_{15})$
  Quaterround $(x'_0, x'_5, x'_{10}, x'_{15})$
  Quaterround $(x'_1, x'_6, x'_{11}, x'_{12})$
  Quaterround $(x'_2, x'_7, x'_8, x'_{13})$
  Quaterround $(x'_3, x'_4, x'_9, x'_{14})$
end
return $X + X^r$

$$z_2 = z_2 + z_3, \quad z_1 = z_1 \oplus z_2, \quad z_1 = z_1 \lll 7 \tag{15}$$

III. PRELIMINARIES

A. MOTIVATION

Developing a cryptographic algorithm plays a significant role in guaranteeing the security of data, information, privacy perspective, digital transactions, and secure communication in networks. Recently, Chacha20 has seen widespread adoption due to its efficiency and supposed security [37], [38], [39]. While it has gained this adoption, from a research perspective, there is a need for continuous exploration in cryptography. As a result, the author considers the following factors to be relevant as to why the need for exploring variant ciphers in the field of cryptography is essential:

- **Evolving Cyber-Threat Landscape:** The cyber threat landscape and demands are evolving [40], with constant advancements in cryptographic algorithms relevant in countering emerging key attack techniques. Based on this premise, proactiveness is essential for anticipatory vulnerabilities. As such, exploring variant ciphers like EChacha20 strengthens the cryptographic defenses against emerging threats while guaranteeing long-term resiliency.
- **Weakness Mitigation in other ciphers:** While to some extent the novel Chacha20 is considered to be robust, it is worth noting that other ciphers still exhibit some weaknesses and vulnerabilities [41]. Exploring EChacha20 is a step towards addressing some weaknesses by incorporating enhanced security features and strengthening the overall security posture. EChacha20 could provide alternative solutions by offering security guarantees based on specific use cases.
- **Future Security Proofing:** The rapid technological advancements mean it is also crucial to anticipate futuristic security challenges. Exploring or strengthening cryptographic algorithms like EChacha20, acts as foundations of secure communication and allows us to stay ahead of emerging security threats and face evolving attack vectors.
- **Continual Advancement of Cryptographic Science:** The development and research of novel cryptographic research is an ongoing ongoing endeavor focused on advancement in science and innovation. It is envisaged that EChacha20 is a valuable addition to the body of cryptographic algorithms that offers new insights, techniques, and possibilities that can benefit the broader community of security practitioners and researchers.

B. SCOPE AND RELEVANCE

The relevance of this study lies in the fact that it proposes an extension of the well-known Chacha20 stream cipher that offers an increased level of security against a selected type of attack. The extended Chacha20 uses a longer key and nonce to generate a larger keystream, which is then XORed with the plaintext to produce the ciphertext. The proposed cipher is highly relevant in today's digital age, where secure communication is critical to data transmission. With the increasing number of cyber-attacks and the growing demand for secure communication protocols, the development of ciphers that offer a higher level of security is of great importance. Therefore, the scope of this study is to investigate the performance of the extended Chacha20 algorithm, which uses a $QR - Fs$ with 36 rounds, in the context of differential cryptanalysis and its resilience against emerging attacks. Specifically, this study aims to analyze the algorithm's security by examining its resistance to differential attacks and efficiency. Also, this study aims to contribute to the ongoing efforts to improve cryptographic algorithms' security and efficiency and provide a deeper understanding of the strengths and limitations of an extended Chacha20 algorithm.

IV. PROBLEM FORMULATION

A. SYSTEM MODEL

A scenario that assesses the effectiveness of an extended Chacha20 is considered with 36 $QR - Fs$ in the context of differential cryptanalysis. This scenario involves an attacker attempting to obtain the secret key used by a sender to encrypt two different plaintexts, denoted as P and P', the secret key K used by the sender to encrypt the plaintexts, the ciphertexts C and C' generated by encrypting P and P' respectively, using the Chacha20 algorithm with the secret key K, and the attacker who is trying to obtain the secret key K. In the long run, the attacker/adversary will be tasked with changing the content of the message, and statistically, an assessment is made based on the altered or diverse plaintexts. The specific roles of each other identified entities are described below:

- **Plaintexts P and P':** These are the messages that the sender wants to transmit securely to the receiver. In the context of the differential cryptanalysis approach used in this paper, these plaintexts are carefully chosen to satisfy certain properties that make them vulnerable to attack.

- **Secret key K:** This is the key used by the sender to encrypt the plaintexts P and P' using the EChacha20 algorithm. The security of the algorithm depends on the secrecy of the key, which is assumed to be known only to the sender and the intended receiver.
- **Ciphertexts C and C':** These are the encrypted versions of the plaintexts P and P' respectively, generated using the Chacha20 algorithm with the secret key K. The attacker intercepts these ciphertexts and attempts to use them to obtain information about the secret key K.
- **Attacker:** This is an adversary who intercepts the ciphertexts C and C' and tries to obtain information about the secret key K. The attacker's goal is to be able to decrypt future messages encrypted using the same secret key K.

To formulate the problem, we consider the following factors which are critical to understanding the issue and developing a solution:

Let P and P' be two plaintexts that differ only in one block, denoted as P_i and P'_i, respectively, where i is the block index. Let C_i and C'_i be the corresponding ciphertext blocks obtained by encrypting P_i and P'_i, respectively, using the EChacha20 algorithm with the secret key K. The attacker tries to find a set of differential characteristics $(\Delta P, \Delta C)$ such that the difference between the output of the EChacha20 algorithm on P_i and P'_i is related to the difference between the inputs $P_i \oplus \Delta P$ and $(P'_i \oplus \Delta P)$ as follows in Equation 16:

$$C_i \oplus C'_i = (P_i \oplus P'_i) \oplus (\Delta C) \tag{16}$$

where ΔC is the difference between the output of the Chacha20 algorithm on $(P_i \oplus \Delta P)$ and $(P'_i \oplus \Delta P)$. In this scenario, we assume P_i and P'_i are two plaintexts, each with a length of X bytes. ΔP is the difference between the two plaintexts, i.e., $\Delta P = P_i \oplus P'_i$. C_i and C'_i are the corresponding ciphertexts generated by the Extended Chacha20 algorithm with 36 rounds using P_i and P'_i, respectively. ΔC is the difference between the two ciphertexts, i.e., $\Delta C = C_i \oplus C'_i$. The goal is to analyze the security of the Extended Chacha20 algorithm by assessing its resistance to differential cryptanalysis attacks.

It is essential to clarify that the attacker's goals may vary depending on their capabilities and objectives. The formulation considers both possibilities, acknowledging that attackers may aim to find the correct key or exploit differential characteristics.

B. THREAT MODEL

The Bellare-Rogaway model (BR-M) is a widely used security model for analyzing symmetric key cryptographic algorithms [42]. In this model, the attacker is assumed to have access to a public encryption oracle, which can be used to encrypt messages of the attacker's choice based on the foundations of the BR-M. In the context of our study, we assume that the attacker is honest but curious [43], and has access to the plaintexts and ciphertexts generated by the EChacha20 algorithm with 36 rounds. The attacker is interested in obtaining information about the secret key used by the algorithm. We model the attacker as having access to an encryption oracle that can encrypt messages of the attacker's choice.

Nevertheless, we also consider the chosen-plaintext attack (CPA) attack model [20], where the attacker can choose plaintexts and obtain their corresponding ciphertexts from the encryption oracle. The attacker's goal is to use this information to gain information about the secret key and other operations. We assume that the attacker has no knowledge of the secret key used by the algorithm and that the only information available to the attacker is the set of plaintext-ciphertext pairs generated by the algorithm. Our objective is to analyze the security of the EChacha20 algorithm against this CPA attack model with a test of differential cryptanalysis attack. Therefore, we represent the attacker's capabilities as follows:

- Attacker's encryption oracle: $E(m)$, where m is the message to be encrypted.
- Set of plaintext-ciphertext pairs available to the attacker: (P_i, C_i), (P'_i, C'_i), where P_i and P'_i are two plaintexts, C_i and C'_i are the corresponding ciphertexts generated by the Extended Chacha20 algorithm with 36 rounds.
- Attacker's goal: Recover the secret key used by the Extended Chacha20 algorithm for purposes of alteration or manipulation (active and passive attacks).

C. SECURITY REQUIREMENTS

We assess the security concerns and requirements based on the honesty and curiosity of the attacker based on the Bellare-Rogaway model (BR-M) [42], and the chosen-plaintext attack (CPA) attack model [20], where the attacker has access to the encryption oracle $E(m)$ and a set of plaintext-ciphertext pairs (P_i, C_i), (P'_i, C'_i). The attacker's goal is to recover the secret key in order to conduct some manipulations. The security requirements are to prevent the attacker from obtaining any information about the secret key from the given information. In this model, the honest but curious attacker can perform chosen plaintext attacks and obtain ciphertexts for arbitrary plaintexts of their choice with an unknown goal once the key is obtained. The security requirements for this scenario include ensuring that the attacker cannot obtain the secret key even if they have access to multiple plaintext-ciphertext pairs. This requirement can be expressed as follows:

For all polynomial-time attackers A, the advantage Adv_A of the attacker in distinguishing between the encryption of a random message m and the encryption of a message m' that was not previously queried to the encryption oracle should be negligible, as is shown in Equation 17.

$$Adv_A = \left|Pr[A_{E(m)} = 1] - Pr[A_{E(m')} = 1]\right| \leq negl(n) \tag{17}$$

where Adv_A is the advantage of the adversary A in distinguishing the encryptions. It measures how well A can differentiate between the encryption of m and m', $Pr[A(E(m)) = 1]$ is

the probability that the adversary A outputs 1 (indicating a successful distinction) when given the encryption of plaintext m, and $Pr[A(E(m')) = 1]$ is the probability that the adversary A outputs 1 (indicating a successful distinction) when given the encryption of plaintext m'. The inequality " $\leq''$ indicates that the adversary's advantage must be negligible, meaning that the probability of the adversary successfully distinguishing between the encryptions approaches zero as the security parameter increases. n is the security parameter and negl(n) is a negligible function. A function is considered negligible if it becomes arbitrarily small as n (security parameter) grows.

The attacker should not be able to recover the secret key even if they have access to multiple plaintext-ciphertext pairs. The equation expresses the security requirement that the advantage of the attacker A in distinguishing between the encryption of two messages m and m' using the encryption oracle. We, therefore, assess the security of the system based on the following security requirements:

- **Confidentiality:** The attacker should not be able to learn any information about the plaintext or the secret key from the ciphertext alone as follows in Equation 18:

$$\Pr\left[\begin{array}{c}(P_i, P_i') \leftarrow \text{Gen};, C_i \leftarrow \text{E}(K, P_i);, \\ C_i' \leftarrow \text{E}(K, P_i');, b \leftarrow 0, 1;, (P_b, C_b)\end{array}\right] \approx \frac{1}{2}, \tag{18}$$

  where Gen is the key generation algorithm, E is the encryption algorithm, K is the secret key, and $\approx$ denotes statistical indistinguishability.
- **Key Recovery:** The attacker should not be able to recover the secret key used by the algorithm. This requirement can be expressed as follows in Equation 19.

$$\Pr[K' \leftarrow A(C_1, C_2, \ldots, C_n)] \approx 0, \tag{19}$$

  where A is the attacker's algorithm, and n is the number of plaintext-ciphertext pairs available to the attacker.
- **Resistance to Chosen Plaintext Attacks:** An attacker should not be able to learn any information about the secret key from the ciphertexts generated by the algorithm, even when the attacker is allowed to choose the plaintexts; this can be expressed as follows in Equation 20:

$$Pr[c \leftarrow E(k, m_1) : c = E(k, m_2)] \tag{20}$$

  where Pr denotes the probability, E is the encryption function with key k, m_1 and m_2 are two distinct plaintext messages, and c is the corresponding ciphertext. The equation measures the probability that an attacker, who has the ability to choose plaintext messages and observe their corresponding ciphertexts, can determine any useful information about the encryption key k that would allow them to decrypt other ciphertexts encrypted with the same key. A higher probability indicates a weaker resistance to CPA.

TABLE 2. List of notations.

Notations	Descriptions
n	Security parameter
$negl(n)$	Negligible function
A	Attacker
$E(m)$	Encryption oracle
m	Plaintext
(P_i, C_i)	Plaintext-ciphertext pair
(P_i', C_i')	Plaintext-ciphertext pair
K	Secret key
Gen	Key generation algorithm
E	Encryption algorithm
$QR - F$	Quarter Round Function
$NSTS$	NIST Statistical Test Suite
XOR	Exclusive-OR
ARX	Add Rotate XOR
CPA	Chosen Plaintext Attack

V. EXTENDED CHACHA20 STREAM CIPHER

This section describes the proposed Extended Chacha20 (EChacha20) Cipher with an increased Quater Round Function ($QR - Fs$) as a contribution.

A. OVERVIEW

EChacha20 improves the Quater Round Function ($QR - Fs$) used in Chacha20. Essentially, EChacha20 is a pseudorandom function (PRF) that takes a 256-bit key, a 64-bit nonce, and a 64-bit counter as inputs to generate a stream of keystream bytes, which is expressed as shown in Equation 21.

$$Chacha20(k, n, c) = PRF_k(n \parallel c) \tag{21}$$

where k is the 256-bit key, n is the 64-bit nonce, c is the 64-bit counter, PRF is the pseudorandom function, and || denotes concatenation.

The EChacha20 cipher, as proposed, employs six $QR - Fs$, in contrast to the four $QR - Fs$ used in the Chacha20 cipher. Each $QR - Fs$ takes four 32-bit words as input and generates four 32-bit words as output. The six $QR - Fs$ operate on a $[6 \times 6]$ matrix of 32-bit words, each row of the matrix being processed by a distinct $QR - Fs$.

Understanding the Chacha20 stream cipher requires familiarity with its key component, the $QR - F$. This function is crucial to the encryption process and the overall security of the cipher. It combines four 32-bit input words (a, b, c, and d), expressed as 8-digit hexadecimal numbers, into a single 32-bit output word using an ARX sequence. The resulting output word creates a secure stream cipher that ensures the encrypted data can't be decrypted without the correct key. In the Chacha20 algorithm, this quarter-round function is repeated multiple times to strengthen the encryption further.

B. ECHACHA20 STEPS

The $QR - F$ used in the Chacha20 stream cipher involves several steps. First, the values of two input words, a and b, are added together, and the result is stored in a as is shown below in Algorithm 3. Then, the values of two other input words,

c and d, are XORed together, and the result is stored in d. Next, the value of a is rotated 16 bits to the left and stored back in a. The values of a and d are then added together, and the result is stored in d. The same addition of a and d is performed again, but the result is not stored this time.

The values of b and d are XORed together, and the result is stored in b. The value of b is then rotated 12 bits to the left and stored back in b. The values of b and c are added together, and the result is stored in c. The values of a and c are XORed together, and the result is stored in a. The value of a is rotated 8 bits to the left and stored back in a. The values of a and d are added together again, and the result is stored in d. The values of b and d are XORed together again, and the result is stored in b. Finally, the value of b is rotated 7 bits to the left and stored back in b as is shown below in Algorithm 3.

Algorithm 3 QR-F Steps

$a \leftarrow a + b\ d \leftarrow c \oplus d\ a \leftarrow$ left_rotate$(a, 16)\ d \leftarrow a + d$
$d \leftarrow a + d\ b \leftarrow b \oplus d\ b \leftarrow$ left_rotate$(b, 12)\ c \leftarrow b + c$
$a \leftarrow a \oplus c\ a \leftarrow$ left_rotate$(a, 8)\ d \leftarrow a + d\ b \leftarrow b \oplus d$
$b \leftarrow$ left_rotate$(b, 7)$

C. ECHACHA20 QR – FS: HIGH-LEVEL VIEW

Given that the significant operations on ChaCha are performed on matrices, we can assume a matrix, M, with 32-bit words and an extended six (6) $QR - Fs$ with 32 words as output. Specifically, the $QR - Fs$ round (column) takes in six 32-bit words and generates six 32-bit words as output, as shown in Figure 2. The proposed EChacha20 approach leverages six (32-bit) words as input and mixes them to generate 32-bit words as output. The $QR - Fs$ works on four 32-bit words of input and output and is used in each of the four rounds. It works by taking two 32-bit words (A and B) and combining them with two other 32-bit words (C and D) to create two new 32-bit words (A′ and B′). The quarter-round works by first adding A and B together, then XORing C with the result, and finally adding D to the result. The output words of the quarter-round function are A′ and B′.

Taking EChacha20 input for $x = (x_0, x_1, x_2, x_3, x_4, x_5)$, then the $QR - Fs$ will be given by $y = (y_0, y_1, y_2, y_3, y_4, y_5)$, where x and y are six-word input and output, respectively. Therefore, the extended $QR - Fs$ is given as shown in the word computation.

$$z_1 = y_1 \oplus ((y_0 + y_5) \lll 2, z_2 = y_2 \oplus ((z_1 + y_3) \lll 4,$$
$$z_3 = y_3 \oplus ((z_2 + y_4) \lll 7, z_4 = y_4 \oplus ((z_3 + y_0) \lll 8,$$
$$z_5 = y_3 \oplus ((z_4 + z_1) \lll 12, z_0 = y_0 \oplus ((z_5 + z_2) \lll 16$$

In the context of six-word round input, the following is true: The EChacha20 $QR - Fs$ generates a 32-bit output for each of the 32-bit input words. Additionally, the rotation constants used in the algorithm have been extended from the original Chacha20 constants of 16, 12, and 8 to include 7, 4, and 2, respectively, as is shown next.

$$y_0, y_1, y_2, y_3, y_4, y_5 = QR - Fs(x_0, x_1, x_2, x_3, x_4, x_5)$$
$$y_6, y_7, y_8, y_9, y_{10}, y_{11} = QR - Fs(x_6, x_7, x_8, x_9, x_{10}, x_{11})$$
$$y_{12}, y_{13}, y_{14}, y_{15}, y_{16}, y_{17} = QR - Fs(x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17})$$
$$y_{18}, y_{19}, y_{20}, y_{21}, y_{22}, y_{23} = QR - Fs(x_{18}, x_{19}, x_{20}, x_{21}, x_{22}, x_{23})$$
$$y_{24}, y_{25}, y_{26}, y_{27}, y_{28}, y_{29} = QR - Fs(x_{24}, x_{25}, x_{26}, x_{27}, x_{28}, x_{29})$$
$$y_{30}, y_{31}, y_{32}, y_{33}, y_{34}, y_{35} = QR - Fs(x_{30}, x_{31}, x_{32}, x_{33}, x_{34}, x_{35})$$

D. ECHACHA20 COLUMN ROUND

The EChacha20 column round selects six 32-bit input words and sequentially generates the respective parameters based on the $QR - Fs$ shown above. This results in 36 inputs arranged in a 6×6 matrix and 36 outputs denoted as $x = (x_0, x_1, x_2, \ldots\ldots x_{35})$ and $y = (y_0, y_1, \ldots\ldots y_{35})$ respectively. The final EChacha20 column round is then obtained from these outputs.

The Echacha $QR - Fs$ is designed as a 32-bit block cipher that can process arbitrary message sizes. It employs six rotation constants ranging from 16 to 2, which are used in a round function shown in Figure 2. The $QR - Fs$ operates on six input 32-bit words using Addition, Rotation, and Exclusive-OR (ARX) operations. At each input block, the 32-bit words are processed sequentially, triggering a new round of the QR process. When the last ARX operation is performed, a termination signal is sent, and the resulting 32-bit block is truncated and outputted.

The architecture of the $QR - Fs$ involves pairs of rotational constants that allow randomized permutations to resist potential attacks. The rotational constants are applied using XOR pairs, denoted by y, z, and q, as shown in Equation 22.

$$y + z \bigoplus \langle\langle q = (y \langle\langle q) \bigoplus (z \langle\langle q) \tag{22}$$

The EChacha20 is represented as a 6×6 matrix shown in Equation 23

$$MX = \begin{Bmatrix} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 \\ M_6 & M_7 & M_8 & M_9 & M_{10} & M_{11} \\ M_{12} & M_{13} & M_{14} & M_{15} & M_{16} & M_{17} \\ M_{18} & M_{19} & M_{20} & M_{21} & M_{22} & M_{23} \\ M_{24} & M_{25} & M_{26} & M_{27} & M_{28} & M_{29} \\ M_{30} & M_{31} & M_{32} & M_{33} & M_{34} & M_{35} \end{Bmatrix} \tag{23}$$

The $QR - Fs$ is represented using Equation 24 below

$$QR{-}Fs = [MX_i^k]_{6\times6} = QR - F[(MX_i^{k-1})]_{6\times6} \tag{24}$$

Based on equation 24, the Quarter Round Function $QR{-}Fs$ is given as shown in Equation 24 -30, as shown at the bottom of the next page. 'MX' is a matrix used to compute the Quarter Round Function $QR - Fs$. For example, 'MX_i^k' represents a 6×6 matrix derived from elements of the 'x' matrix (Salsa20 matrix) starting from index 'i'. The matrix 'MX_i^k' is used to compute the Quarter Round Function.

To improve the security of Chacha20, a few steps can be taken. First, the key size should be increased to 256 bits,

which is the maximum the cipher allows. This is done by increasing the number of rounds, currently set to 20. Increasing the number of rounds increases the cipher's security and makes it more resistant to brute-force attacks. Second, the nonce size should be increased to 64 bits, the maximum the cipher allows. This is done by increasing the

$$(QR-F)_1 = \begin{Bmatrix} MX_1^k \\ MX_2^k \\ MX_3^k \\ MX_4^k \\ MX_5^k \\ MX_0^k \end{Bmatrix} = \begin{Bmatrix} MX_6^{k-1} \oplus ((MX_0^{k-1} + MX_5^{k-1}) \lll 2) \\ MX_{12}^{k-1} \oplus ((MX_1^{k-1} + MX_3^{k-1}) \lll 4) \\ MX_{18}^{k-1} \oplus ((MX_2^{k-1} + MX_4^{k-1}) \lll 7) \\ MX_{24}^{k-1} \oplus ((MX_3^{k-1} + MX_0^{k-1}) \lll 8) \\ MX_{30}^{k-1} \oplus ((MX_4^{k-1} + MX_2^{k-1}) \lll 12) \\ MX_0^{k-1} \oplus ((MX_5^{k-1} + MX_2^{k-1}) \lll 16) \end{Bmatrix} \tag{25}$$

$$(QR-F)_2 = \begin{Bmatrix} MX_6^k \\ MX_8^k \\ MX_7^k \\ MX_9^k \\ MX_{11}^k \\ MX_{10}^k \end{Bmatrix} = \begin{Bmatrix} MX_1^{k-1} \oplus ((MX_1^{k-1} + MX_5^{k-1}) \lll 2) \\ MX_7^{k-1} \oplus ((MX_{11}^{k-1} + MX_3^{k-1}) \lll 4) \\ MX_{13}^{k-1} \oplus ((MX_{10}^{k-1} + MX_9^{k-1}) \lll 7) \\ MX_{19}^{k-1} \oplus ((MX_6^{k-1} + MX_0^{k-1}) \lll 8) \\ MX_{25}^{k-1} \oplus ((MX_9^{k-1} + MX_8^{k-1}) \lll 12) \\ MX_{31}^{k-1} \oplus ((MX_{10}^{k-1} + MX_6^{k-1}) \lll 16) \end{Bmatrix} \tag{26}$$

$$(QR-F)_3 = \begin{Bmatrix} MX_{15}^k \\ MX_{14}^k \\ MX_{13}^k \\ MX_{12}^k \\ MX_{16}^k \\ MX_{17}^k \end{Bmatrix} = \begin{Bmatrix} MX_2^{k-1} \oplus ((MX_1^{k-1}5 + MX_{12}^{k-1}) \lll 2) \\ MX_8^{k-1} \oplus ((MX_{15}^{k-1} + MX_{17}^{k-1}) \lll 4) \\ MX_{14}^{k-1} \oplus ((MX_{17}^{k-1} + MX_{12}^{k-1}) \lll 7) \\ MX_{20}^{k-1} \oplus ((MX_{16}^{k-1} + MX_{13}^{k-1}) \lll 8) \\ MX_{26}^{k-1} \oplus ((MX_{16}^{k-1} + MX_{17}^{k-1}) \lll 12) \\ MX_{32}^{k-1} \oplus ((MX_{14}^{k-1} + MX_{14}^{k-1}) \lll 16) \end{Bmatrix} \tag{27}$$

$$(QR-F)_4 = \begin{Bmatrix} MX_{18}^k \\ MX_{19}^k \\ MX_{20}^k \\ MX_{22}^k \\ MX_{21}^k \\ MX_{23}^k \end{Bmatrix} = \begin{Bmatrix} MX_3^{k-1} \oplus ((MX_{10}^{k-1} + MX_{21}^{k-1}) \lll 2) \\ MX_9^{k-1} \oplus ((MX_{19}^{k-1} + MX_{23}^{k-1}) \lll 4) \\ MX_{15}^{k-1} \oplus ((MX_{10}^{k-1} + MX_{12}^{k-1}) \lll 7) \\ MX_{21}^{k-1} \oplus ((MX_{21}^{k-1} + MX_{23}^{k-1}) \lll 8) \\ MX_{27}^{k-1} \oplus ((MX_{19}^{k-1} + MX_{22}^{k-1}) \lll 12) \\ MX_{33}^{k-1} \oplus ((MX_{20}^{k-1} + MX_{19}^{k-1}) \lll 16) \end{Bmatrix} \tag{28}$$

$$(QR-F)_5 = \begin{Bmatrix} MX_{24}^k \\ MX_{25}^k \\ MX_{26}^k \\ MX_{27}^k \\ MX_{28}^k \\ MX_{29}^k \end{Bmatrix} = \begin{Bmatrix} MX_4^{k-1} \oplus ((MX_{19}^{k-1} + MX_{25}^{k-1}) \lll 2) \\ MX_1^{k-1}0 \oplus ((MX_{26}^{k-1} + MX_{25}^{k-1}) \lll 4) \\ MX_{16}^{k-1} \oplus ((MX_{29}^{k-1} + MX_{28}^{k-1}) \lll 7) \\ MX_{22}^{k-1} \oplus ((MX_{25}^{k-1} + MX_{25}^{k-1}) \lll 8) \\ MX_{28}^{k-1} \oplus ((MX_{19}^{k-1} + MX_{24}^{k-1}) \lll 12) \\ MX_{29}^{k-1} \oplus ((MX_{24}^{k-1} + MX_{23}^{k-1}) \lll 16) \end{Bmatrix} \tag{29}$$

$$(QR-F)_6 = \begin{Bmatrix} MX_{30}^k \\ MX_{31}^k \\ MX_{32}^k \\ MX_{33}^k \\ MX_{34}^k \\ MX_{35}^k \end{Bmatrix} = \begin{Bmatrix} MX_5^{k-1} \oplus ((MX_{30}^{k-1} + MX_{31}^{k-1}) \lll 2) \\ MX_1^{k-1}1 \oplus ((MX_{32}^{k-1} + MX_{32}^{k-1}) \lll 4) \\ MX_{17}^{k-1} \oplus ((MX_{31}^{k-1} + MX_{33}^{k-1}) \lll 7) \\ MX_{23}^{k-1} \oplus ((MX_{30}^{k-1} + MX_{35}^{k-1}) \lll 8) \\ MX_{29}^{k-1} \oplus ((MX_{30}^{k-1} + MX_{35}^{k-1}) \lll 12) \\ MX_{35}^{k-1} \oplus ((MX_{24}^{k-1} + MX_{35}^{k-1}) \lll 16) \end{Bmatrix} \tag{30}$$

number of blocks currently set to 8. Increasing the number of blocks increases the cipher's security and makes it more resistant to replay attacks.

Based on the premise mentioned above, EChacha20 utilizes an increased key size,, making it difficult to break. This modification increases the round size and allows faster encryption/decryption speeds. This increases the number of rounds and multiplications, making the cipher more secure. Additionally, the Echacha20 uses a larger key size and a more secure nonce size. The larger key size helps to increase the cipher's security, while the larger nonce size helps ensure that each message is encrypted with a different key.

Consequently, the EChacha20 stream cipher includes additional rounds and operations to enhance its resistance to attack. Furthermore, using a [6 × 6] matrix has a more significant influence on the memory, which slightly reduces memory usage, which can be advantageous for memory-constrained applications. Also the [6 × 6] matrix provides higher security than the [8 × 8] matrix due to its increased diffusion and additional encryption rounds. This improved version of Chacha20 offers superior security and performance while reducing memory usage. It provides faster encryption and decryption times due to the increased number of rounds and longer time per round with the [6 × 6] matrix, further increasing security.

E. ECHACHA20 ROW ROUND

The EChacha20 row round is the second step in the EChacha20 cipher after the column round. The output of the column round is used as the input for the row round. The row round consists of six $QR - Fs$, which are applied to the input data row-wise. The row round is designed to provide diffusion in the horizontal direction, while the column round provides diffusion in the vertical direction.

The EChacha20 row round is similar to the column round, employing the ARX operations of Addition, Rotation, and XOR. However, the row round has different rotational constants for the ARX operations and takes six 32-bit inputs. Specifically, the rotational constants used in the row round are (13, 16, 17, 21, 5, 8).

The output of the row round is a 6 × 6 matrix of 32-bit words that undergoes shuffling within each row to provide additional diffusion. To achieve this, a permutation matrix is constructed using the Fibonacci numbers to generate a cyclic shift pattern for each row. The following equations can represent the EChacha20 row round:

$$y_{4i} = y_{4i} \oplus ((y_{4i+1} + y_{4i+2}) \lll r_1) \quad (31)$$

$$y_{4i+3} = y_{4i+3} \oplus ((y_{4i} + y_{4i+1}) \lll r_2) \quad (32)$$

$$y_{4i+2} = y_{4i+2} \oplus ((y_{4i+3} + y_{4i}) \lll r_3) \quad (33)$$

$$y_{4i+1} = y_{4i+1} \oplus ((y_{4i+2} + y_{4i+3}) \lll r_4) \quad (34)$$

Equations shown above operate on a 128-bit data block and consist of four QR-F (Quarter Round - Function) operations. The input block is divided into 16, 32-bit words, processed in four sets of four words each. For each set, the equation computes four new values for the output block by performing the following operations:

- The first output word is obtained by XORing the fourth input word with the left shift of the addition of the second and third input words by a constant value r_1.
- The second output word is obtained by XORing the first input word with the left shift of the addition of the third and fourth input words by a constant value r_2.
- The third output word is obtained by XORing the second input word with the left shift of the addition of the fourth and first input words by a constant value r_3.
- The fourth output word is obtained by XORing the third input word with the left shift of the addition of the first and second input words by a constant value r_4.

VI. TEST AGAINST DIFFERENTIAL ANALYSIS ATTACK

Differential analysis is a cryptanalysis technique commonly used to identify the statistical differences between pairs of inputs and outputs of a cipher. To achieve this, a large number of pairs of plaintexts are generated that differ in bits; they are then encrypted with the EChacha20 cipher to obtain the corresponding ciphertext. The following approaches have been used as shown in Figure 3:

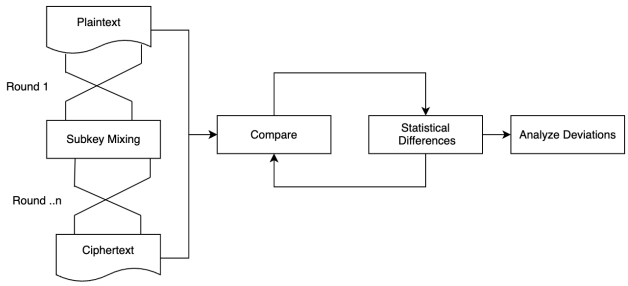

FIGURE 3. Differential cryptanalysis approach.

- Generate a large number of pairs of plaintext that differ by a few bits
- Encrypt the plain text with EChacha20 to obtain the corresponding ciphertext pair
- Compare the ciphertext pairs (Look for statistical differences), employ NIST Statistical Test Suite
- Compare results of the test with outcomes and analyze deviations.
- Statistical differences show weakness in the cipher
- Interpret the results and draw conclusions on the security of EChacha20

To ascertain the effectiveness of the EChacha20 cipher, the differences between the two ciphertexts are computed,. A comparison of their differences is expected if the cipher is perfectly secure. When or if the distribution of the differences between the ciphertext is significantly different from the desired outcome, then it indicates that the cipher is vulnerable to differential analysis. As a result, further analysis of the

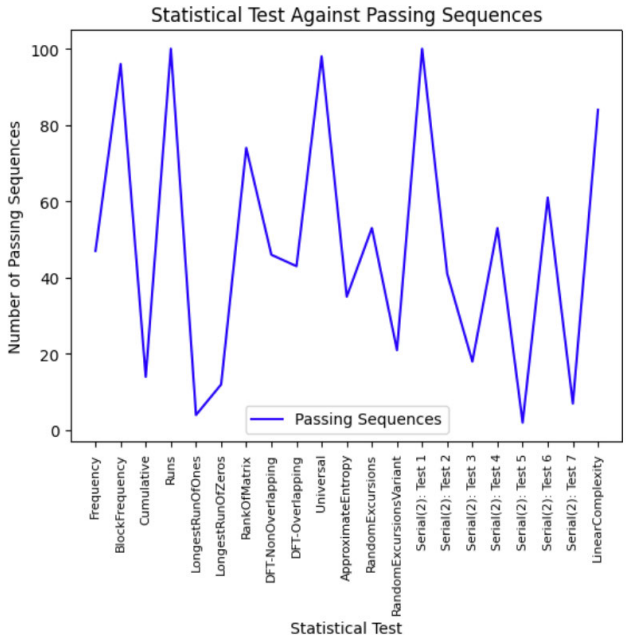

FIGURE 4. Passing sequences obtained from a battery of statistical tests used in hypothesis testing, including frequency, block frequency, runs, and longest run of ones and zeros, among others.

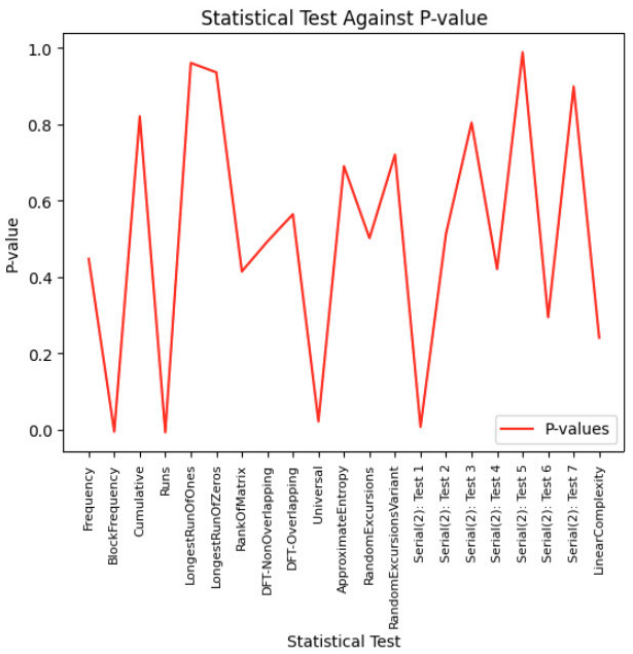

FIGURE 5. P-values obtained from a battery of statistical tests used in hypothesis testing, including frequency, block frequency, runs, and longest run of ones and zeros, among others.

cipher becomes necessary to identify the source of the vulnerability.

A. RANDOM PLAINTEXT GENERATION & BIT FLIPPING

This study has leveraged the NIST Statistical Test Suite (NSTS) for conducting differential analysis for EChacha20, owing to its ability to evaluate the randomness and the quality of random number generators, which makes it suitable for the proposed EChacha20. In addition, the NSTS test is designed to detect a variety of deviations from randomness like correlation, biases, and other patterns that are likely to exhibit weaknesses or vulnerabilities in the cipher. In this study, we have generated 1000 pairs of 32-byte plaintexts that differ only in one bit, as is shown in Algorithm 4 where bitflips are realized. In this approach, Algorithm 4 takes an n input that specifies the number of plaintexts and outputs a list of plaintexts, each of 32-byte plaintext. Random bytes are generated using the *os.urandom* function for the plaintext, and the byte-array is used in the mutation of $p1$, which allows the random *index* j to be flipped between 0 and 31 using a bitwise XOR operation.

This eventually is followed by appending the tuple where the process repeats for iteration purposes, as is shown in Algorithm 4.

Algorithm 4 Generating Plaintexts With Bitflips

Input: Number of plaintexts to generate: n
Output: List of plaintext tuples: *plaintexts*
initialize *plaintexts* as an empty list
for $i \leftarrow 1$ **to** n **do**
  $p1 \leftarrow$ generate 32-byte random string using *os.urandom*()
  $p2 \leftarrow$ convert $p1$ to a bytearray
  $j \leftarrow$ random integer between 0 and 31
  $p2[j] \leftarrow$ bitwise XOR of $p2[j]$ and 1
  append ($p1$, *bytes*($p2$)) to *plaintexts*
end
return *plaintexts*

B. ENCRYPTING FLIPPED BIT WITH ECHACHA20

In flipping and encrypting the plaintext, the plaintext is first XORed with the key stream generated by the EChacha20 algorithm. The result of the XOR operation is the ciphertext. The key stream is generated by running the EChacha20 algorithm with the secret key and the nonce as inputs. However, before running the encryption process, a random bit in the plaintext is flipped to introduce an error. This bit flip is intended to simulate an error in transmitting or storing the encrypted data. The $QR - F$ is a fundamental building block of the Chacha20 cipher and by extension, the EChacha20 cipher. The $QR - F$ itself involves four basic operations: addition, XOR, left rotation, and right rotation. These operations are performed on the four input elements in a specific order, with the output elements permutations of the input elements. Whereas the original Chacha20 cipher uses 20 rounds, the EChacha20 cipher uses 36 rounds. This increased number of rounds is intended to provide greater

Algorithm 5 EChacha20 Encryption

Input: Number of iterations: n
Output: EChacha20-ciphertext and nonce
for $i \leftarrow 1$ **to** n **do**
  key ← os.urandom(32);
  nonce ← os.urandom(16);
  rounds ← 36;
  backend ← default-backend();
  cipher ← Cipher(algorithms.Chacha20(key, nonce, rounds), mode=None, backend=backend);
  plaintext ← b"This is a secret message";
  encryptor ← cipher.encryptor();
  ciphertext ← encryptor.update(plaintext) + encryptor.finalize();
  Print "Ciphertext:" + ciphertext.hex() + " Nonce:" + nonce.hex();
end
return *ciphertext and nonce*

TABLE 3. Results of the NIST statistical suite tests for the EChacha20 cipher with 36 Q-Rounds.

Statistical Test	P-Value/Sequence	Proportion of Passing Sequences
Frequency	0.499132	47/1000
BlockFrequency	0.074648	96/1000
Cumulative	0.834615	14/1000
Runs	0.000006	1000/1000
LongestRunOfOnes	0.983454	4/1000
LongestRunOfZeros	0.912505	12/1000
RankOfMatrix	0.384058	74/1000
DFT-NonOverlapping	0.504032	46/1000
DFT-Overlapping	0.527005	43/1000
Universal	0.068926	98/1000
ApproximateEntropy	0.619587	35/1000
RandomExcursions	0.453155	53/1000
RandomExcursionsVariant	0.790207	21/1000
Serial(2): Test 1	0.008748	1000/1000
Serial(2): Test 2	0.563070	41/1000
Serial(2): Test 3	0.819699	18/100
Serial(2): Test 4	0.458212	53/1000
Serial(2): Test 5	0.934396	2/1000
Serial(2): Test 6	0.357650	61/1000
Serial(2): Test 7	0.899161	7/1000
LinearComplexity	0.256915	84/1000

Results for the uniformity of p-values and the proportion of passing sequences (samples=1000, blocks in each sequence=1)

security against attacks on the cipher while still maintaining a high level of performance. Encrypting the flipped bit with EChacha20 based on 36 QR-F is shown in Algorithm 5.

C. MEASURE WITH NIST STATISTICAL SUITE

Using the NIST Statistical Test Suite (NSTS), EChacha20's quality is evaluated by being subjected to 36 rounds with a battery of tests using the approach shown in Algorithm 6. This study considers 1000 pairs of plaintexts that only differed by a few bits, which had been flipped before being encrypted by the EChacha20 algorithm. The resulting ciphertext has been analyzed using NSTS to determine its randomness and the quality of encryption. It is worth noting that the NSTS consists of diverse tests used to assess the randomness and the quality of randomized number generators and respective

Algorithm 6 Running the NIST Statistical Test Suite

Input: Ciphertexts *ciphertext*1 and *ciphertext*2
Output: NIST Statistical Test Suite results
Write *ciphertext*1 to a binary file named "ciphertext1.bin"
Write *ciphertext*2 to a binary file named "ciphertext2.bin"
for *i from 1 to N* **do**
  Execute the NIST Statistical Test Suite with the command "Compute ciphertext1.bin & ciphertext2.bin" Obtain the NIST Statistical Test Suite results for iteration *i*
end

cryptographic algorithms. Uniformity of the p-values is measured or evaluated based on uniform distribution with an interval of [0, 1].

After applying NSTS to the ciphertexts, the p-values, statistical tests, and the measure of randomness were assessed. Table 3 shows the statistical test results after analysis for the p-value for the sequence and the proportion of passing sequences. As shown, the frequency test showed a p-value of 0.499132. In addition, the Run-test exhibited a p-value of 0.000006, and all the 1000 sequences passed the test. This was followed by The LongestRunOfOnes test, which had a p-value of 0.983454, a selected performance of the sequences is shown out of the 1,000 sequences passed the test as is shown in Figures 5 and 4 respectively. Figure 5 displays the p-values obtained from a battery of statistical tests used in hypothesis testing, including frequency, block frequency, runs, and longest run of ones and zeros, among others, while Figure 4 displays the passing rates against the battery of tests that are shown in Table 3.

VII. SECURITY ANALYSIS

This section focuses on evaluating the security properties of the proposed EChacha20 encryption scheme. The ultimate aim is to comprehensively analyze the security guarantees offered by EChacha20 and compare them to the well-established Chacha20 cipher. This is relevant because it enriches the field of cryptographic research, providing a thorough assessment of the security properties of EChacha20 and offering insights into its strengths and potential vulnerabilities.

A. ECHACHA20 SECURITY MODEL ANALYSIS

It is vital to establish the foundational concepts, assumptions, and adversary models essential for evaluating the security guarantees of the proposed cipher. This analysis explores the security notions and properties that EChacha20 satisfies by identifying the capabilities and constraints of the adversaries, considering their access to various resources, such as ciphertexts, plaintexts, and encryption keys, as was initially shown in the descriptions of EChacha20.

The notions that have been considered significant while assessing EChacha20 in the context of adversarial capabilities are as follows:

- ***Indistinguishability under Chosen-Plaintext Attack (IND-CPA):*** This property ensures that an adversary cannot distinguish between encryptions of different plaintexts with a non-negligible advantage [44]. The proof for IND-CPA typically demonstrates that the probability of successfully distinguishing between the ciphertexts in EChacha20 is negligible.
  The security of EChacha20 is shown in this context by proving that no efficient adversary can distinguish the encryptions with non-negligible advantage. This is achieved by carefully analyzing the encryption process of EChacha20 and the computational hardness assumptions underlying the cipher.
- ***Non-malleability:*** This property guarantees that an adversary cannot modify ciphertexts meaningfully without knowledge of the corresponding plaintext [45]. The proof for non-malleability involves demonstrating that any modification to the ciphertext results in an unpredictable change in the corresponding decrypted plaintext. This is established by analyzing the cryptographic properties of EChacha20, such as the integrity protection mechanisms and the resistance against chosen-ciphertext attacks.
- ***Randomness Preservation:*** This property ensures that EChacha20 preserves the randomness properties of the underlying Chacha20 cipher. The proof for randomness preservation typically involves analyzing the modifications introduced in EChacha20 and verifying that they do not compromise the underlying randomness properties of Chacha20.

B. PERFORMANCE EVALUATION AND COMPARISON

Performance evaluation and comparison show the effect of implementing EChacha20 over Chacha20, coupled with an analysis of the effectiveness of the suggested security improvements.

For this performance evaluation, data were collected on the encryption and decryption time metrics for Chacha20 and EChacha20 using a data size of 1048576 bytes, respectively. The selection of the appropriate metrics is important during performance evaluation to assess the effectiveness and efficiency of the cryptographic algorithm. In this evaluation, the following metrics that are relevant for this study have been chosen owing to them providing their crucial performance characteristics:

- **Encryption time:** This metric measures the duration required to encrypt a given amount of data. The importance of this metric lies in the fact that it is directly used to reflect the speed at which the cipher can process and protect sensitive information. Normally, fast encryption time is preferred because it facilitates efficient and timely data encryption.
- **Decryption time:** This metric measures the time taken to decrypt encrypted data. It is similar to encryption time. Faster decryption time shows the efficiency of the cipher in recovering the original plaintext from the ciphertext. When rapid access to encrypted information is essential, then quick decryption is necessary in such scenarios.
- **Memory Usage:** Memory usage refers to the amount of computer memory the cipher requires during its operation. It is an essential metric as it affects the overall resource consumption of the cipher. Lower memory usage is desirable as it minimizes the impact on system resources and allows for more efficient execution in resource-constrained environments.

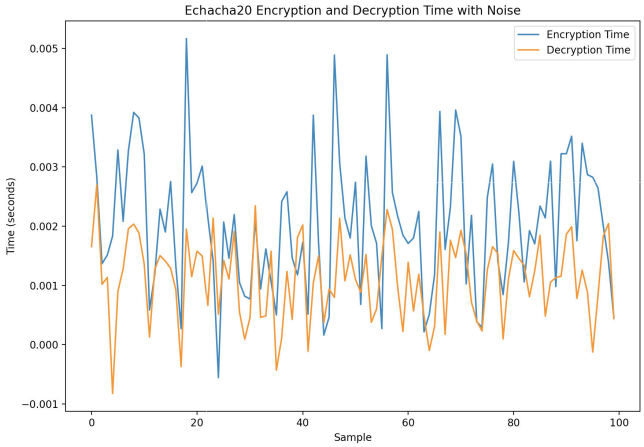

FIGURE 6. Performance evaluation results of EChacha20. The graph illustrates the encryption and decryption time measurements with the inclusion of noise, providing a visual representation of the performance characteristics. The x-axis represents the data size in bytes, while the y-axis denotes the time in seconds. The plotted data points show varying data sizes' encryption and decryption time values.

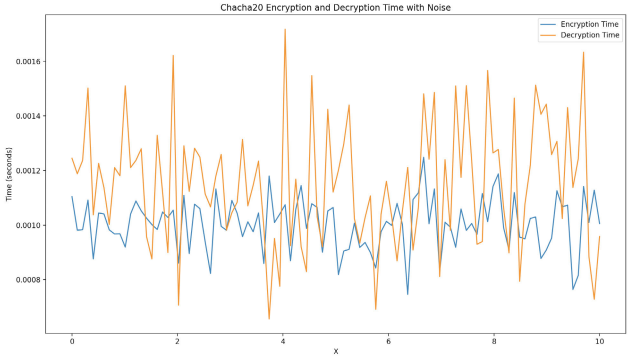

FIGURE 7. Performance evaluation results of Chacha20. The graph illustrates the encryption and decryption time measurements with the inclusion of noise, providing a visual representation of the performance characteristics. The x-axis represents the data size in bytes, while the y-axis denotes the time in seconds. The plotted data points show varying data sizes' encryption and decryption time values.

Based on the results that have been obtained from this evaluation, as is shown in Table 4, it is observed that EChacha20 demonstrates comparable performance to Chacha20 in terms of encryption and decryption time. It is worth noting that the slight differences in the measured times can be attributed to various factors such as system variations and measurement noise, as is shown in Figures 7 and 6 respectively.

TABLE 4. Performance evaluation results.

Cipher	Encryption Time (s)	Decryption Time (s)	Memory Usage (B)
Chacha20	0.0019118786	0.0013480186	26976256
EChacha20	0.0018911362	0.0012269020	27551712

One of the key improvements introduced in EChacha20 while conducting this evaluation is the increased number of $QR - F$ from 20 to 36. This enhancement is to improve the security property of the EChacha20 cipher. In addition, despite the additional computational complexity introduced by the extra $QR - F$ in this process, our evaluation has shown that the impact on the overall performance is relatively minimal.

Moreover, the line graphs with noise generated, as is shown in Figures 7 and 6 respectively, are based on the collected data, and they provide more insights into the performance characteristics of Chacha20 and EChacha20. It is observed that these graphs exhibit similar trends, indicating that the additional $QR - F$ in EChacha20 does not significantly affect the overall performance in terms of encryption and decryption time, however, with a small margin. This suggests that the proposed modifications in EChacha20 maintain a good balance between security and performance.

On the other hand, it has been observed that the memory usage of EChacha20 is higher than that of Chacha20, as is shown in Figure 8. This suggests that EChacha20 requires more memory resources to perform its operations effectively. High memory usage can have implications for systems with limited memory capacity. This may lead to increased resource consumption and potential performance degradation. Consequently, the graph in Figure 8 highlights the trade-off between computational efficiency and memory usage in the performance of Chacha20 and EChacha20. While EChacha20 offers improved encryption and decryption times, it comes at the cost of higher memory usage. Ultimately, the choice between the two ciphers depends on the specific requirements and constraints of the system or application where they are used.

It is important to note that the performance evaluation conducted in this paper is just one aspect of cipher analysis that is relevant for this study, and a comprehensive evaluation that is positioned as an avenue for future work should consider other factors such as throughput and latency. However, our evaluation demonstrates that EChacha20 can provide comparable performance to Chacha20 while incorporating the proposed improvements for the specific metrics of encryption and decryption time.

The results of our performance evaluation support the effectiveness of the EChacha20 cipher as a potential alternative to Chacha20 based on the security enhancements. The enhancements made in EChacha20, particularly the increased $QR - F$, contribute to its security without compromising its performance. The ability to balance security and performance is a crucial factor in the design of modern encryption algorithms, as seen in EChacha20.

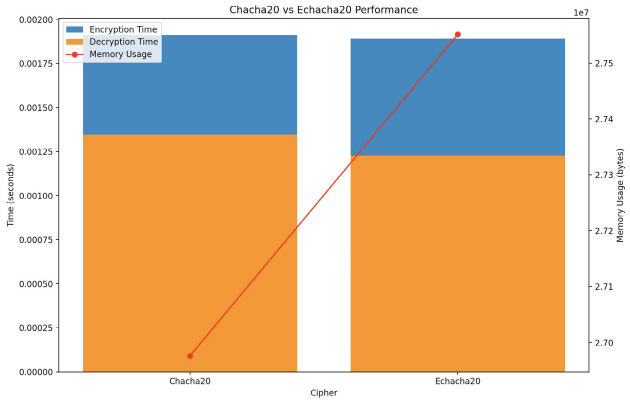

FIGURE 8. Performance comparison between Chacha20 and EChacha20 in terms of encryption and decryption time, along with memory usage. The bars represent each cipher's encryption and decryption time, while the red curve depicts the memory usage.

VIII. COMPARISON WITH EXISTING STUDIES

This section concentrates on giving comparative studies on the proposed extended $QR - Fs$ and other studies that have somewhat been used as baselines of the propositions identified in this paper. These studies have been explored to show relevance and how Chacha20 performs against other algorithms.

The primary difference between Chacha20 and other block ciphers is that it uses a different type of key scheduling algorithm. Specifically, Chacha20 uses a 20-round variant of the well-known salsa20 algorithm, designed for high-speed encryption. Chacha20 was designed to be more secure and efficient than existing block ciphers like AES. The main benefit of Chacha20 is its increased speed: it can encrypt and decrypt data up to four times faster than AES. In addition, Chacha20 is more secure than AES since it has a larger number of rounds, and its rounds are more complex than those used in AES.

Nevertheless, Chacha20 offers some other advantages over older block ciphers, such as the ability to use nonce-based encryption. This allows for more secure and unique encryption keys for each session, improving security by making it more difficult for attackers to guess the encryption key. It also supports a variety of different modes of operation, allowing it to be used in a variety of applications.

The need for combining Chacha20 and Poly1305, that IETF protocols have suggested, is necessitated by the fact that Chacha20 and Poly1305 primitives are seen to be secure. Security analysis shows that adversaries are given access to the encrypting oracles where the decryption oracle takes a chance to query the oracles; however, adversaries are treated as nonce-respecting adversaries [46]. Also, side-channel attacks on Chacha Quarter Round show initial diverse levels of resistance; however, a divide and conquer approach (Brick-layer attack approach) shows the efficiency when attacking the reverse Quarter Round from the keystream as opposed to the attack from the input block [47]. Consequently, an analysis of Chacha and Salsa Quarter

Round that was aimed at exploring alternative designs for maximizing diffusion showed that for each algorithm, more than a million diffusion matrices were generated, and the findings also show that there exists a higher number of rotational constants which in the long run generates more diffusion [48].

Another study has explored rotational analysis of Chacha permutation where it is seen that Chacha20 stream cipher hardly acts as a random permutation when 17 rounds are observed during cryptanalysis. Furthermore, it is seen that the probability of finding a rotational collision that is parallel is estimated to be less than 2^{400} and (88117) rounds are observed as opposed to 2^{-511} for a random permutation [49]. Other research on vectorization approaches for Chacha stream cipher has shown the need for speeding encryption and decryption of X86-64 processors through AVX-2 implementation. Through this, there is a widening of the vector from 128 bits to 256/512 bits,, showing an improved throughput and an increase in speed. However, with an improved double Quarter-Round, there is a need for changing the initialization vectors [50].

Consequently, a study focused on analyzing the non-randomness of estream cipher and the possibility of recovering the key after five (5) rounds of 128 bits with 281 operations and six (6) rounds of 256 bits with 2^{177} rounds respectively. In this study, the weakness of this approach is extensively highlighted in the seven rounds [51].

Another research that analyses Probabilistic Neutral Bits (PNB) using the input and output pairs has suggested the PNB-based differential attacks concentrate on the reduced round Chacha. It was observed that the extended attack can work for up to 7.25 rounds. However, with a time complexity of $2^{555.62}$, $2^{48.36}$ with the success of 0.5. While this outcome seems feasible, the suggested attack seems less effective as compared to a brute force attack [52]. Another research by [53], on the rotational cryptanalysis of Chacha cipher, that checks how to obtain rotational XOR pairs shows that in the first quarter round, the probability is seen to be $2^{-251.7875}$, which ultimately is expected based on a random permutation. The potential weakness is seen to be the rotational XOR probability, which increases in other Chacha20 variants. An analysis for fault in Chacha and Salsa stream cipher by [54] shows commonality in differential attacks. From this study, fault model attacks are suggested, and it is seen that when the attacker knows the plaintext and the ciphertext, then the fault mode can easily be exploited; however, the low complexity of this study showed that it is practical in nature [54].

From the above studies that have offered more insights, it is seen that Chacha20 is a cipher widely regarded as one of the most secure and efficient encryption algorithms currently available. Compared to other ciphers, such as AES, Chacha20 has been more efficient at high-speed encryption. Additionally, Chacha20 is more secure than other ciphers due to its increased number of rounds and more complex round structure.

It is important to note that the motivation for exploring the extended rounds for Chacha20 is based on the limitations identified in other ciphers, as shown in Table 5. These limitations include issues such as weaknesses in the key schedule and the susceptibility of some ciphers to side-channel attacks [47], attacking $QR-Fs$ from keystream [48], $QR-F$ modification, weaknesses in rotational XOR in Chacha20 variants, etc [49], [50], [51], [54]. EChacha20 has been designed to address the specific limitation of improved $QR-F$ and stands to provide a more secure and efficient encryption algorithm.

IX. DISCUSSIONS AND IMPLICATIONS

Chacha20 is a round-based cryptographic construction that processes data through a series of rounds, each consisting of various operations. It differs from Salsa20 in the number of rounds used, with Chacha20 employing 10 or 20 rounds depending on the application, compared to Salsa20's 20 rounds. EChacha20 is an improved version of the Chacha20 cipher. Its goal is to enhance the cipher's security by implementing additional encryption rounds. Compared to the original Chacha20, the EChacha20 cipher uses more rounds, making it more secure and resilient to attacks. Moreover, the EChacha20 quarter round is an extension of the standard Chacha20 quarter round, utilizing the same algorithm but with larger rounds. Nevertheless, the QR-F in EChacha20 increases the security of the Chacha20 algorithm by implementing additional rounds. Its purpose is to provide extra protection against brute force and cryptographic attacks that exploit known plaintext. The QR-F takes two input vectors, a state array and a round constant, and performs four rounds of the Chacha20 algorithm. Each round comprises several operations, as Section V and Equations 1-22 shows.

By adding additional rounds to the Chacha20 algorithm, the EChacha20 cipher provides enhanced protection against known plaintext and brute force attacks based on the test conducted for differential cryptanalysis. The $QR-F$ operation is a crucial component of the EChacha20 QR-F, designed to ensure that the same plaintext does not produce the same ciphertext when using the same key. However, the EChacha20 quarter round is similar to the standard quarter round but includes four rounds instead of two, providing even more excellent protection. The four rounds consist of adding and XORing 32-bit word blocks, rotating the blocks left or right by a specific number of bits, and performing additional XOR operations.

Based on the need to assess the statistical suitability of the EChacha20 with the enhanced $QR-F$ as is shown in Section VI of this paper, the study went the extra mile to test the effectiveness and randomness of EChacha20 cipher using the NIST Statistical Test Suite [55], designed to detect various deviations from randomness that can lead to vulnerabilities in the cipher. The study leveraged 1000 pairs of plaintexts that differed only in one bit and then flipped a bit in the plaintext before encrypting with EChacha20. The resulting ciphertext

TABLE 5. Comparing the study with the state of the art that closely matches.

REF	Focus	Observation	Limitations
[47]	Side channel attack on Chacha Quarter Round	Brick layer attack based on Divide and Conquer	Attacking $QR - Fs$ from keystream is efficient than from the input block
[48]	Quarter Round function analysis	Comparison of Quarter Round function with diffusion matrices	Quarter Round function of modified Chacha produces more diffusion
[49]	Rotational analysis of Chacha permutation	Chacha2o stream cipher hardly acts as a random permutation on covering 17 rounds during rotational cryptanalysis	Less probability when it comes to finding parallel rotation permutation for 17 rounds is less than 2^{-488}
[50]	Vectorizing Chacha cipher	Wide vectorization of Chacha in its speed in encryption and decryption-widening 128 bit to 256/512 with AVX2 extensions	Improved double Quarter Round needs changes in the initialization Vector
[51]	Non-Randomness in eastream ciphers (Salsa20 and Chacha20)	Observing the non-randomness on six(6) rounds and portray a key receiving attack	Key can easily be recovered after five(5) rounds of 128 bit 2^{81} operations coupled with six rounds of 256-bit with 2^{177} operations
[52]	Cryptanalysis of Chacha cipher	Probabilistic neutral Bits (PNB)-focused on the differential attack on reduced-round Chacha for all single bits, and it works on 7.25 rounds	Attack is less efficient than a brute-force attack
[53]	Rotational cryptanalysis of Chacha cipher	Technique of obtaining rotational XOR pairs after first Quarter Round with non-zero probability. The probability is $2^{-251.7857}$ from one permutation	Rotational XOR probability increases-which shows potential weaknesses in Chacha20 variants
[54]	Fault analysis of Chacha cipher	Suggests fault attack against Chacha cipher. With the knowledge of plaintext and ciphertext, fault modes are exploited	The fault attack model's low complexity shows they are practical but can be exploited

Consideration for relevant studies that aligns to the proposed extended $QR-F$

was analyzed using the NIST Statistical Suite to evaluate its randomness and quality of encryption.

The frequency test showed a p-value of 0.499132, and only 47 out of the 1000 sequences passed the test. The BlockFrequency test showed a p-value of 0.074648, and 96 out of the 1000 sequences passed the test. The Runs test showed a p-value of 0.000006, and all the 1000 sequences passed the test. The *LongestRunOfOnes* test showed a p-value of 0.983454, and 2 sequences passed the test based on the current sequences in Table 3. The *LongestRunOfZeros* test showed a p-value of 0.912505, and 12 out of the 1000 sequences passed the test.

Based on the outcome, it was seen that EChacha20 performs well in terms of the Runs test, but it may not perform well in terms of other tests. Specifically, the frequency and block frequency tests suggest that the EChacha20 cipher may not be as random as desired. This could be because the cipher uses only 36 rounds, which may not be enough to provide sufficient security against attacks, or it could give room for lengthy processes that tamper with effective diffusion. However, the cipher is still considered secure for most applications, as it has passed many other tests in the NIST Statistical Suite, such as the Universal test, which showed a p-value of 0.068926, and 98 out of the 1000 sequences passed the test.

The choice of using more than 20 rounds is necessitated by the fact that, in some cases, it may be desirable to use more than 20 rounds of Chacha20 encryption to increase protection. For example, if an application needs to protect susceptible data, it may be necessary to use additional rounds of Chacha20 encryption to ensure the data remains secure. Additionally, if a system is being attacked, it may be beneficial to use additional rounds of Chacha20 encryption to make it more difficult for an attacker to gain access to the data. Adding additional rounds to the algorithm will also increase the time it takes for an attacker to crack the encryption, thus providing additional layers of protection. For example, an attacker who knows the structure of Salsa20 will not necessarily be able to apply the same techniques to Chacha20. For example, increasing the number of rounds from 20 to 36, or 40, increases the cipher's security by a factor of 2^{20}, or over one million. However, increasing the number of rounds also increases the cipher's complexity, as each additional round requires additional operations and memory. Consequently, When considering additional rounds, it is essential to understand the trade-off between security and performance. The additional rounds may provide higher protection but at the cost of slower performance. As such, it is essential. Overall, the additional rounds in Chacha20 make it a more secure option than Salsa20, especially for applications requiring higher levels of security; however, it is essential to note that additional rounds will also add additional processing time, so it should only be used when necessary.

In analyzing the security posture of EChacha20, it demonstrates promising characteristics. The analysis suggests that the increased number of $QR - F$ rounds, from 20 to 36, enhances the resistance against differential attacks and strengthens the overall security of the cipher. Additionally, utilizing the extended key schedule contributes to the mitigation of related-key attacks, further improving the cipher's security profile.

While assessing the performance, the quantitative evaluation shows the comparable efficiency of EChacha20 and Chacha20. Both ciphers' encryption and decryption times exhibit minimal differences, indicating that the additional computational complexity introduced by EChacha20's

enhancements does not significantly impact its overall performance. In the author's opinion, this finding is crucial as it demonstrates that the proposed modifications in EChacha20 can be integrated into existing systems without sacrificing computational efficiency.

It has also been observed that EChacha20 exhibits higher memory usage compared to Chacha20. This trade-off between computational efficiency and memory consumption should be carefully considered when deploying the cipher in resource-constrained environments. While EChacha20 offers improved encryption and decryption times, it has been observed that the slightly higher memory requirements may impact system resources. In the authors' opinion, it is worth mentioning that based on this outcome, the selection of the appropriate cipher between EChacha20 and Chacha20 depends on the specific requirements and constraints of the system or application.

X. CONCLUSION AND FUTURE WORK

The author of this paper has suggested an extended EChacha20 stream cipher that utilizes ARX with six rotational constants. The Chacha20 quarter-round function is an essential component of the ChaCha stream cipher. Also, the proposed quarter-round function is a non-linear transformation that operates on four 32-bit words at a time. It is designed to be efficient, fast, and secure and is well-suited for use in embedded systems. The proposed Echacha is a substitution-permutation network that consists of four mixing functions, each operating on a 4-word (16-byte) state. The $QR - F$ is used to mix the state in a pseudo-random manner, and it operates on four 32-bit words at a time.

Consequently, this study will explore comprehensive performance evaluation in the future to consider other factors, such as throughput and latency, further to assess the performance of EChacha20 against the Chacha20 cipher.

Also, future work is aimed at studying how the efficiency of the $QR - F$ can be increased based on the existing number of operations. Also, the security analysis of the $QR - F$ will also be studied to see the degree and whether the proposed $QR - F$ is resistant to known attacks in a complex environment with massive data. Also, while the suggested approach is more secure, future research will focus on reducing the number of *XOR* operations on the $QR - F$.

ACKNOWLEDGMENT

The author would like to thank anonymous reviewers for their valuable insights, the Secure Distributed Systems (SDS), and the Department of Computer Science (DIDA) at Blekinge Institute of Technology, BTH, Karlskrona, Sweden, for their support while coming up with this research. The author also acknowledges that the opinions, findings, and conclusions expressed in this paper are purely of the authors.

REFERENCES

[1] W. Butler and L. D. Keeney, *Secret Messages*. London, U.K.: Simon & Schuster, 2001.
[2] D. Kahn, *The Codebreakers: The Comprehensive History of Secret Communication From Ancient Times to the Internet*. New York, NY, USA: Simon and Schuster, 1996.
[3] A. M. Qadir and N. Varol, "A review paper on cryptography," in *Proc. 7th Int. Symp. Digit. Forensics Secur. (ISDFS)*, Jun. 2019, pp. 1–6.
[4] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 6, Nov. 1976.
[5] D. Pandya, K. Ram, S. Thakkar, T. Madhekar, and B. S. Thakare, "Brief history of encryption," *Int. J. Comput. Appl.*, vol. 131, no. 9, pp. 28–31, Dec. 2015.
[6] A. Kerckhoffs, "La cryptographie militaire," *J. Des. Sci. Militaries*, vol. 4, no. 38, p. 5, 1883.
[7] L. Kruh, "The Beale cipher as a bamboozlement—Part II," *Cryptologia*, vol. 12, no. 4, pp. 241–246, Oct. 1988.
[8] N. P. Smart and N. P. Smart, "The enigma machine," in *Cryptography Made Simple*, 2016, pp. 133–161.
[9] C. A. Deavours and J. Reeds, "The enigma part I historical perspectives," *Cryptologia*, vol. 1, no. 4, pp. 381–391, Oct. 1977.
[10] C. Ellis, *Exploring the Enigma*. Cambridge, U.K.: Univ. Cambridge, 2005.
[11] J. Daemen and V. Rijmen, "The block cipher Rijndael," in *Smart Card Research and Applications*. Cham, Switzerland: Springer, 2000, pp. 277–284.
[12] T. W. Cusick, C. Ding, and A. R. Renvall, *Stream Ciphers and Number Theory*. Amsterdam, The Netherlands: Elsevier, 2004.
[13] D. J. Bernstein, "The Salsa20 family of stream ciphers," in *New Stream Cipher Designs*. Cham, Switzerland: Springer, 2008, pp. 84–97.
[14] R. Velea, F. Gurzau, L. Margarit, I. Bica, and V.-V. Patriciu, "Performance of parallel ChaCha20 stream cipher," in *Proc. IEEE 11th Int. Symp. Appl. Comput. Intell. Informat. (SACI)*, May 2016, pp. 391–396.
[15] L. Jiao, Y. Hao, and D. Feng, "Stream cipher designs: A review," *Sci. China Inf. Sci.*, vol. 63, no. 3, pp. 1–25, Mar. 2020.
[16] J. Daemen and P. Kitsos, "The self-synchronizing stream cipher MOUSTIQUE," in *New Stream Cipher Designs*. Cham, Switzerland: Springer, 2008, pp. 210–223.
[17] J. D. Golić, "Modes of operation of stream ciphers," in *Selected Areas in Cryptography*. Cham, Switzerland: Springer, 2001, pp. 233–247.
[18] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and Y. Papaefstathiou, "A survey of lightweight stream ciphers for embedded systems," *Secur. Commun. Netw.*, vol. 9, no. 10, pp. 1226–1246, Jul. 2016.
[19] P. C. van Oorschot and M. J. Wiener, "A known-plaintext attack on two-key triple encryption," in *Advances in Cryptology—EUROCRYPT '90*. Cham, Switzerland: Springer, 1991, pp. 318–325.
[20] Y. Liu, L. Y. Zhang, J. Wang, Y. Zhang, and K.-W. Wong, "Chosen-plaintext attack of an image encryption scheme based on modified permutation–diffusion structure," *Nonlinear Dyn.*, vol. 84, no. 4, pp. 2241–2250, Jun. 2016.
[21] A. H. Zahid, E. Al-Solami, and M. Ahmad, "A novel modular approach based substitution-box design for image encryption," *IEEE Access*, vol. 8, pp. 150326–150340, 2020.
[22] A. H. Zahid, M. Ahmad, A. Alkhayyat, M. J. Arshad, M. M. U. Shaban, N. F. Soliman, and A. D. Algarni, "Construction of optimized dynamic S-boxes based on a cubic modular transform and the sine function," *IEEE Access*, vol. 9, pp. 131273–131285, 2021.
[23] A. H. Zahid, A. M. Iliyasu, M. Ahmad, M. M. U. Shaban, M. J. Arshad, H. S. Alhadawi, and A. A. A. El-Latif, "A novel construction of dynamic S-box with high nonlinearity using heuristic evolution," *IEEE Access*, vol. 9, pp. 67797–67812, 2021.
[24] A. Zahid, M. Arshad, and M. Ahmad, "A novel construction of efficient substitution-boxes using cubic fractional transformation," *Entropy*, vol. 21, no. 3, p. 245, Mar. 2019.
[25] F. Xiu-Tao, "ZUC algorithm: 3GPP LTE international encryption standard," *Inf. Secur. Commun. Privacy*, vol. 19, no. 12, pp. 45–46, 2011.
[26] D. P. Anderson and R. G. Herrtwich, "Internet communication with end-to-end performance guarantees," in *Telekommunikation und Multimediale Anwendungen der Informatik*. Cham, Switzerland: Springer, 1991, pp. 246–258.
[27] G. Orhanou, S. E. Hajji, Y. Bentaleb, and J. Laassiri, "EPS confidentiality and integrity mechanisms algorithmic approach," 2011, *arXiv:1102.5191*.
[28] D. J. Bernstein. *Notes on the Ecrypt Stream Cipher Project (eSTREAM)*. Accessed: Jul. 2023. [Online]. Available: http://cr.yp.to/streamciphers/#timings
[29] M. Robshaw, "The eSTREAM project," in *New Stream Cipher Designs*. Cham, Switzerland: Springer, 2008, pp. 1–6.

[30] M. Hell, T. Johansson, and W. Meier, "Grain: A stream cipher for constrained environments," *Int. J. Wireless Mobile Comput.*, vol. 2, no. 1, pp. 86–93, 2007.
[31] C. De Cannière, "TRIVIUM: A stream cipher construction inspired by block cipher design principles," in *Proc. Int. Conf. Inf. Secur.* Cham, Switzerland: Springer, 2006, pp. 171–186.
[32] S. Babbage and M. Dodd, "The stream cipher Mickey 2.0. eSTREAM," Project Dates 2004 to 2008, EU, Eur. Project, ECRYPT Stream Cipher Project, Tech. Rep. 2, Sep. 2009.
[33] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, and M. Minier, "SOSEMANUK, a fast software-oriented stream cipher," in *New Stream Cipher Designs*. Cham, Switzerland: Springer, 2008, pp. 98–118.
[34] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scavenius, "Rabbit: A new high-performance stream cipher," in *Proc. Int. Workshop Fast Softw. Encryption*. Cham, Switzerland: Springer, 2003, pp. 307–329.
[35] H. Wu, "The stream cipher HC-128," in *New Stream Cipher Designs*. Cham, Switzerland: Springer, 2008, pp. 39–47.
[36] A. Klein, "Attacks on the RC4 stream cipher," *Des., Codes Cryptogr.*, vol. 48, no. 3, pp. 269–286, Sep. 2008.
[37] W. Cai, H. Chen, Z. Wang, and X. Zhang, "Implementation and optimization of ChaCha20 stream cipher on sunway taihuLight supercomputer," *J. Supercomput.*, vol. 78, no. 3, pp. 4199–4216, Feb. 2022.
[38] A. T. Maolood, E. K. Gbashi, and E. S. Mahmood, "Novel lightweight video encryption method based on ChaCha20 stream cipher and hybrid chaotic map," *Int. J. Electr. Comput. Eng.*, vol. 12, no. 5, p. 4988, Oct. 2022.
[39] Z. Wang, H. Chen, and W. Cai, "A hybrid CPU/GPU scheme for optimizing ChaCha20 stream cipher," in *Proc. IEEE Int. Conf Parallel Distrib. Process. Appl., Big Data Cloud Comput., Sustain. Comput. Commun., Social Comput. Netw.*, Sep. 2021, pp. 1171–1178.
[40] B. Ang, "Mitigating challenges in an evolving cyber threat landscape," *Cyber Secur., Peer-Reviewed J.*, vol. 6, no. 2, pp. 168–177, 2022.
[41] W. Stone, D. Kim, V. Y. Kemmoe, M. Kang, and J. Son, "Rethinking the weakness of stream ciphers and its application to encrypted malware detection," *IEEE Access*, vol. 8, pp. 191602–191616, 2020.
[42] C. Brzuska, M. Fischlin, B. Warinschi, and S. C. Williams, "Composability of Bellare–Rogaway key exchange protocols," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, Oct. 2011, pp. 51–62.
[43] L. E. Olson, M. J. Rosulek, and M. Winslett, "Harvesting credentials in trust negotiation as an honest-but-curious adversary," in *Proc. ACM Workshop Privacy Electron. Soc.*, Oct. 2007, pp. 64–67.
[44] T. V. Carstens, E. Ebrahimi, G. N. Tabia, and D. Unruh, "Relationships between quantum IND-CPA notions," in *Proc. Theory Cryptogr. Conf.* Cham, Switzerland: Springer, 2021, pp. 240–272.
[45] G. Alagic and C. Majenz, "Quantum non-malleability and authentication," in *Advances in Cryptology—CRYPTO 2017*. Cham, Switzerland: Springer, 2017, pp. 310–341.
[46] G. Procter, "A security analysis of the composition of ChaCha20 and Poly1305," *IACR Cryptol. ePrint Arch.*, vol. 2014, p. 613, 2014.
[47] A. Adomnicai, J. J. A. Fournier, and L. Masson, "Bricklayer attack: A side-channel analysis on the ChaCha quarter round," in *Proc. Int. Conf. Cryptol. India*. Cham, Switzerland: Springer, 2017, pp. 65–84.
[48] R. Sobti and G. Ganesan, "Analysis of quarter rounds of salsa and ChaCha core and proposal of an alternative design to maximize diffusion," *Indian J. Sci. Technol.*, vol. 9, no. 3, pp. 1–10, Jan. 2016.
[49] S. Barbero, E. Bellini, and R. Makarim, "Rotational analysis of ChaCha permutation," 2020, *arXiv:2008.13406*.
[50] M. Goll and S. Gueron, "Vectorization on ChaCha stream cipher," in *Proc. 11th Int. Conf. Inf. Technol., New Generat.*, Apr. 2014, pp. 612–615.
[51] S. Fischer, W. Meier, C. Berbain, J.-F. Biasse, and M. J. B. Robshaw, "Non-randomness in eSTREAM Candidates Salsa20 and TSC-4," in *Proc. Int. Conf. Cryptol. India*. Cham, Switzerland: Springer, 2006, pp. 2–16.
[52] S. Miyashita, R. Ito, and A. Miyaji, "PNB-focused differential cryptanalysis of ChaCha stream cipher," in *Information Security and Privacy*. Cham, Switzerland: Springer, 2022, pp. 46–66.
[53] S. Barbero, D. Bazzanella, and E. Bellini, "Rotational cryptanalysis on ChaCha stream cipher," *Symmetry*, vol. 14, no. 6, p. 1087, May 2022.
[54] A. Beckers, B. Gierlichs, and I. Verbauwhede, "Fault analysis of the ChaCha and Salsa families of stream ciphers," in *Smart Card Research and Advanced Applications*. Cham, Switzerland: Springer, 2018, pp. 196–212.
[55] A. M. Zubkov and A. A. Serov, "Testing the NIST statistical test suite on artificial pseudorandom sequences," *Math. Aspects Cryptogr.*, vol. 10, no. 2, pp. 89–96, 2019.
[56] K. Marton and A. Suciu, "On the interpretation of results from the NIST statistical test suite," *Romanian J. Inf. Sci. Technol.*, vol. 18, no. 1, pp. 18–32, 2015.
[57] M. Sýs and Z. Ríha, "Faster randomness testing with the NIST statistical test suite," in *Security, Privacy, and Applied Cryptography Engineering*. Cham, Switzerland: Springer, 2014, pp. 272–284.

VICTOR R. KEBANDE (Member, IEEE) received the Ph.D. degree in computer science (information and computer security architectures and digital forensics) from the University of Pretoria, Hatfield, South Africa. He was a Researcher with the Information and Computer Security Architectures (ICSA) and the DIgiFORS Research Groups, University of Pretoria, and he was a Postdoctoral Researcher with the Internet of Things and People (IOTAP) Center, Department of Computer Science, Malmö University, Malmö, Sweden. He was also a Postdoctoral Researcher of cyber and information security in information systems research subject with the Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Luleå, Sweden. He is currently an Assistant Professor in IT security with the Department of Computer Science (DIDA), Blekinge Institute of Technology (BTH), Karlskrona, Sweden. His research interests include cyber, information security, and digital forensics in the IoT, the IoT security, digital forensics-incident response, cyber-physical system protection, critical infrastructure protection, cloud computing security, computer systems, distributed system security, threat hunting and modeling, cyber-security risk assessment, blockchain technologies, and privacy-preserving techniques. He is also an Editorial Board Member of *Forensic Science International: Reports* journal.