Iraqi Journal of Information and Communications Technology(**IJICT**)
Vol. 8, Issue 3, December 2025
www.ijict.edu.iq
ISSN:2222-758X
e-ISSN:2789-7362

RC4X: A NEW ENHANCEMENT OVER RC4 ENCRYPTION SECURITY

Rawan A. Fawzy Alsharida[1]
[1] Cybersecurity Department, College of Computer Science and Mathematics, University of Tikrit, Tikrit, Iraq
Rawan_adel@tu.edu.iq[1]
Corresponding Author: **Rawan A. Fawzy Alsharida**

Received:05/10/2025; Revised:16/11/2025; Accepted:22/12/2025
DOI:**10.31987/ijict.8.3.355**

***Abstract*- The RC4 stream cipher has known security weaknesses due to its weak keystream distribution and biased key scheduling. In this article, an extended RC4X is proposed to enhance the security of the original RC4 stream cipher by combining sophisticated mixing methods into its Key Scheduling Algorithm (KSA-X) and Pseudo-Random Generation Algorithm (PRGA-X) process. The proposed KSA-X uses two full-shuffle operations to eliminate predictable key-byte relationships: a first RC4-style permutation and a nonlinear mixing function that performs bitwise shifts and XOR operations. The internal state becomes more unpredictable through an additional permutation phase, which diffuses the state elements and increases the randomness. The PRGA-X update mechanism performs state-value rotations to reduce linear dependencies and minimize distribution-related weaknesses in the generated keystream. The experimental results show that RC4X produces keystreams with improved higher-order correlation properties, enhanced resistance to key-recovery attacks, and fast operation and efficient memory usage. The study proves the fact that RC4X is a lightweight, secure substitute of RC4, which is utilized where maximum efficiency and security are needed and minimum resources are at hand. The experimental findings indicate that RC4X provides security against many known security weaknesses and offers equivalent performance as the traditional stream ciphers. The distance-equalities statistical test will assist us in determining structural flaws in algorithmic keystreams created by similar algorithms such as RC4. This test has been done to identify the statistical bias in RC4. The largest RC4 bias is subsequently fixed by RC4X. Precisely, the most critical RC4 deviation (Event-2 bias) was narrowed down to -11.59X -0.14X in RC4X.**

***keywords:* RC4, Stream Cipher, Key Scheduling Algorithm, Pseudo-Random Generation Algorithm, Encryption Security.**

I. INTRODUCTION

Stream ciphers are the key encryption solutions to the digital communication systems as they offer safety to lightweight as well as energy-efficient applications. They include the lightweight security stream ciphers, fast stream ciphers [1], and power efficient encryption systems. Rivest Cipher 4 (RC4) was a popular encryption algorithm because it was a simple algorithm, and it was fast. It was invented in 1987 by Ron Rivest at the RSA Security [1]. It was applied in the protocols like WEP, WPA, TLS and SSL. It was very popular, however, after decades of cryptanalysis, security researchers found vulnerabilities with RC4. RC4 was found to have serious weaknesses in cryptanalysis. Such flaws were biases within the Key Scheduling Algorithm (KSA) and lack of uniformity. The failures comprised the malfunctioning of the keystream statistics, the key recovery and the failure to differentiate between various attacks. Consequently, RC4 has been officially phased out of modern cryptography standards, providing the final step in a 2-decade-long evolution. It is necessary to have better alternatives that offer greater security.

Nonetheless, block ciphers like the AES are usually utilized to encrypt communication. Resource-constrained environments (e.g., an embedded system) can be characterized by high computation overhead. There are also IoT systems in the legacy environments which have resource limitations. Stream ciphers are still appealing in these scenarios since they are efficient in generating the keystream and encrypt data at a low-latency. The difficulty is to come up with a stream cipher that has the lightweight characteristics of RC4 but no weaknesses. This research is motivated by the need for an enhanced RC4 variant that provides improved resistance to known attacks, generates keystreams with higher statistical uniformity, and can be deployed in real-time applications with limited computational power. Presently, encryption is essential to protecting the privacy of our personally identifiable information. The data must protected with encryption to prevent interception over public channels [2]. Using one of the encryption techniques protects the data from unauthorized attempts to expose sensitive information. Furthermore, the process of encrypting data so that unauthorized parties cannot decipher it is known as encryption in information security. Just a key for encryption that transforms the standard textual form into an encrypted format may be used to process data [3].

Encryption algorithms can be categorized into two primary types: symmetric and asymmetric. Two keys are used in asymmetric cryptography: one for encryption and one for decryption in symmetric key cryptography, and both must be created at random. Since it encompasses two different types of randomness, True Random Number Generators (TRNGs) and Pseudo-Random Number Generators (PRNGs), this category is distinguished by its resilience [4]. However, the type of randomness generation relies on unpredictable elements. This type is unsuitable for encryption due to the non-deterministic nature of the randomness source; consequently, the same chain cannot be regenerated. You can create random keys and use them for encryption (in a database). However, it is perfect for generating random keys and using them in cryptographic operations. The other class is the pseudorandom number generators (PRNGs), which produce a fixed pair of random numbers at each call [5]. Reliant on a repeatable computational cycle, these generators produce numbers through deterministic processes modeled by functions and mathematical algorithms, yielding a set of pseudorandom numbers. 4.2 Time Randomness Model series that appears random. It is used in encryption due to its speed, and because if the exact input string is used as a "seed," it can produce the same sequence [1]. The seed must be random and sufficiently long to increase the target sequence's randomness. RC4 is a compact, well-studied algorithm that has been implemented efficiently across various platforms. The main component of this algorithm is an N-entry (typically n = 256) table that serves as the S-box in the byte substitution process.

This paper proposes RC4X, a conceptual enhancement of the RC4 cipher. RC4X is the first algorithm to implement a Key Scheduling Algorithm (KSA-X) which performs two shuffle passes. The former is similar to RC4 but uses an index-dependent mixing and the latter uses a nonlinear permutation function that is a combination of bitwise shifts and XOR. Last permutation process will provide high diffusion in the state. Pseudo-Random Generation Algorithm (PRGA-X) is an algorithm that introduces rotational mixing to the index-update function which interferes with linear patterns and minimizes the bias of the keystream. The combination of these changes makes the keystream statistically much more resistant to corruption and ensures the cipher is more resistant to correlation and key-recovery attacks. Most of the industry and the academia have left the use of RC4 in security sensitive applications due to the seriousness of its known failures. However, it is relatively simple and efficient, which is why it can become an attractive candidate in case major vulnerabilities can be addressed. RC4X aims to:

- Retain the core advantages of RC4, including simplicity, high speed, and minimal memory requirements.

- Eliminate or significantly reduce known statistical biases and security vulnerabilities.
- Introduce new design elements that enhance both the key scheduling algorithm (KSA) and the pseudo-random generation algorithm (PRGA).

The sequence of the rest of the article is as follows: Section II: Review of the Literature. At the same time, Section III provides a detailed explanation of the RC4 algorithm. Section IV presents the design of the RC4X algorithm, including detailed descriptions of KSA-X and PRGA-X. Section V contains the performance evaluation of RC4X's keystream distribution. The work is finally concluded, and future research possibilities are outlined in Section VI.

II. LITERATURE REVIEW

The literature has proposed several RC4 variants including RC4A, VMPC and RC4+. These algorithms tried to correct the documented RC4 weaknesses by adding more S-boxes, non linear state updates or more diffusion steps. Although these newly added features made it more resistant to certain types of attacks, numerous of them did not address the statistical biases as a whole or were too complicated to be used as a practical alternative to RC4. There is still the need to have a balanced design that is capable of making it stronger in randomness and does not add too many complexities. In [5] suggested an attack to this algorithm to destroy interconnections between outputs. There are many modified algorithms that have been designed to enhance the original algorithm of RC4 and they have also accelerated the speed of implementation. On the other hand, others have aimed at making the algorithm faster resulting in less randomness [6-18]. In [7] created NeLEM, which is a lightweight, substitution based encryption technique of Internet of Things devices with low-resource usage, which replaces RC4 algorithm with DNA. NeLEM encryption technique employs one round of a lightweight encryption through a 16 x16 substitution S-box. The NeLEM encryption system uses RC4 high-performance augmented with the complexity of the DNA sequences to form a lightweight encryption system. The lightweight operation and high cryptographic level of S-box of NeLEM is the result of its unique combination of the ease of computation and the high level of protection. The nonlinearity and the natural randomness of DNA combined with the simplicity of RC4 in NeLEM generate S-boxes which maximize security at minimum resource consumption.

In [8] suggested a random RC4 (RRC4) technique to overcome the shortcoming of key scheduling in RC4. This method proved to be more unpredictable and more secretive of the encrypted information than the initial RC4 method. The code and decoding times of the improved algorithm had the same value as the one of the original algorithm. In [9] achieved a better technique by using mathematical operations, including factorials and other variables, as compared to the original RC4 algorithm that yielded a greater number of random results. The security of the algorithm was also augmented, but comparisons of the new and the original algorithm showed that there was a significant augmentation in the execution time and storage needs. This growth can be explained by the factorial process. Variably Modified Permutation Composition (VMPC) introduced by [10], is characterized by some modifications made to the initializing process, rounds performed and the way the output is generated. The vulnerability of the KSA of RC4 identified and addressed as presented by [5] proved to be popular in its detection and prevention. Pseudo-Random Generation Algorithm (PRGA) in VMPC is a more complex structure compared to the RC4 structure; thus, its resilience to attacks is improved.

In [11] proposed a better RC4A algorithm to enhance RC4 on learning that there is a statistical vulnerability in the first two key bytes of RC4 output. It was mentioned that the number of outputs to be required to tell the difference between the output of RC4 and truly random data is 2^{25}. RC4A is an efficient approach of dealing with numerous weaknesses that are related to the allocation of the initial two output bytes of the RC4 algorithm. In the next year, Maximov launched a new assault against VMPC and RC4A at the same time. Due to this, this particular attack will be able to distinguish between ciphertext and genuinely random data [12]. In [13] suggest the use of RC4 as a secure way of encrypting data in smart grids. To achieve the safety and integrity of the data, RC4 encryption is used throughout the data exchange process. The architecture states that all linked nodes that comprise the smart grid's data are encrypted using the RC4 cipher. Light nodes at the consumer level facilitate individual electricity generation and data handling of the energy supply, speed up transactions, and provide a secure foundation for managing the complexity of smart grid management. In contrast, complete nodes at distribution points validate data and ensure efficient electricity routing. Utilizing RC4 in smart grids is a groundbreaking approach to enhance electrical security, efficiency, and transparency. Enhancing security is the aim of this integration.
In [14] introduced an improvement to RC4 by implementing two-state tables, which enhance randomness, reduce the correlation between the output and the key, and increase encryption time. The index i is likewise a byte, but it is increased by 11 after each step in the author's method (RC4OK) [15]. Since 11 is a prime number, it gives access to every cell in S. However, a more "loose" updating of the array is provided here at the cost of a larger step. The j variable undergoes the most significant change. In contrast to RC4, only the least significant byte of the 32-bit number (4 bytes) represented by this variable is actually used for indexing. The variable j is defined as j0, j1, j2, and j3, with j0 being the least significant. The variable j can also be represented by two 16-bit words, jw0 and jw1, with jw1 being the most important. In [16], three enhancement algorithms are presented; among them, RC4-2S+ stands out as the most effective, using two state tables to generate four key bytes per round. This method enhances randomness without compromising speed. The effective ciphertext-only plaintext recovery attack on RC4 operated on the existence of biases in the RC4 keystream to recover the plaintext when using RC4 in Transport Layer Security (TLS). The results are supported by an experimental assessment of the probability of the attacks. This work also discuss countermeasures of [17]. The authors in [18] find a new two additional biases which are uncovered. Besides, their experiment has successfully produced and generalized a set of non-consecutive byte biases from RC4 keystream.

III. DESCRIPTION OF RC4

Linear Feedback Shift Registers (LFSRs) are important to numerous stream cipher techniques, notably when hardware is used. The use of LFSRs is however not applied in the RC4 design. The algorithm contains PRGA and KSA that are run one after another. RC4 is based on the principle of swapping to achieve the best randomness. It applies a variable during the encryption process which is permuted to allow multiple keys. The key has a length that goes between 0 and 255 bytes and it is used to construct an initial-state array of 256 bytes, S[0] to S[255]. This is to provide security of RC4 bytes. The minimum key length in RC4 must be 128 bits in length [1]. The KSA and a Pseudo-Random Number Generator (PRNG) produce the key using the starting value x [0]. Initiating PRNG is done by KSA and it generates pseudo-random numbers that are used at the PRGA. The pseudocode of the Algorithms I and II show the two sections of the RC4 algorithm. In this case, plaintext message length equals m, original key length equals L and PRNG refers to pseudo-random number generator, i and j are index pointers, N is the size of the array or state S and the key has a size of few bytes. The algorithm below produces a set of keys which may be concatenated with the ciphertext to obtain the plaintext or XORed with the plaintext to obtain the ciphertext.

Algorithm I: Key Scheduling Algorithm for RC4 [1]

```
Input:    Secret key $K = [k_1, k_2, \ldots, k_L]$, key length $L$
Output:   Permutation array $S[i]$, $i = 0, 1, \ldots, 255$
1         Initialize $S[i] = i$, where $i = 0, 1, 2, \ldots, 255$
2         Set $j = 0$
3         For $i = 0$ to $255$ do
3.1           $j = (j + S[i] + K[i \bmod L]) \bmod 256$
3.2           Swap $S[i]$ and $S[j]$
4         End For
5         Return $S$
```

Algorithm II: Pseudo-Random Generation for RC4 [1]

```
Input:    State array $S$
Output:   Key sequence $K_{seq}$
1         Initialize $i = 0,\ j = 0$
2         Until the end of the sequence, do
2.1           $i = (i + 1) \bmod 256$
2.2           $j = (j + S[i]) \bmod 256$
2.3           Swap $S[i]$ and $S[j]$
2.4           $K_{seq} = S[(S[i] + S[j]) \bmod 256]$
3         Return $K_{seq}$
```

IV. PROPOSED METHOD: RC4X

RC4X is a re-engineered version of the original RC4 stream cipher. Key schedule, non-linear index mixing, and rotational updates in the PRGA for enhancements to randomness and security. Similar to RC4, encryption and decryption are performed by XORing the plaintext or ciphertext with the keystream. RC4X is built on the principle of incrementally enhancing two core components of RC4:

1) Key Scheduling Algorithm (KSA-X): Introduce additional mixing rounds and incorporate non-linear transformations as shown in Algorithm III.
2) Pseudo-Random Generation Algorithm (PRGA-X): Reduce known output biases by diversifying how the internal state is updated each round, as shown in Algorithm IV.

A. *RC4X Key Scheduling Algorithm (KSA-X)*

1) **Initialization**
   - Similar to RC4, an array S of size 256 (for an 8-bit permutation) is initialized with values from 0 to 255.
   - A temporary key array K is constructed by repeating the user-supplied key if necessary to match the size of S.

- $S[i] = i$ for $i = 0, 1, \ldots, 255$.
- $K[i] =$ user-supplied key repeated.

2) **Multiple Shuffle Rounds**
   - Instead of a single pass of the standard RC4 shuffle, RC4X performs two full passes. In each pass:
     - $j = (j + S[i] + K[i]) \bmod 256$.
     - swap $(S[i], S[j])$.
   - A second pass with a different mixing function ensures that any leftover correlations are further randomized:
     - $j = (j + S[(i \ll 3) \oplus (i \gg 2)] + K[i]) \bmod 256$.
     - swap $(S[i], S[j])$.
3) By mixing the index i in a non-linear way using bitwise shifts and XOR operations, KSA-X disrupts predictable key-byte correlations.
4) **Additional Permutation Step**
   - After the two passes, a final permutation step is applied to further obfuscate the internal state $i = 0;\ j = 0$.
   - for $i = 0$ to 255:
   - $j = (j + S[i] + i) \bmod 256$.
   - swap $(S[i], S[j])$.

Algorithm III: Key Scheduling for RC4X

```
Input:   Key $K = [k_1, k_2, \ldots, k_L]$, key length $L$
Output:  State array $S = [S_0, S_1, \ldots, S_{255}]$
1        Initialize the state array
1.1          $S[i] = i$, for $i = 0, 1, 2, \ldots, 255$
2        First Shuffle Pass
2.1          Set $j = 0$
2.2          For $i = 0$ to 255, do
2.2.1            $j = (j + S[i] + K[i \bmod L]) \bmod 256$
2.2.2            Swap $S[i]$ and $S[j]$
3        Second Shuffle Pass (Non-Linear Mixing)
3.1          Reset $j = 0$
3.2          For $i = 0$ to 255, do
3.2.1            $idx = ((i \ll 3) \oplus (i \gg 2)) \bmod 256$
3.2.2            $j = (j + S[i] + S[idx] + K[i \bmod L]) \bmod 256$
3.2.3            Swap $S[i]$ and $S[j]$
4        Additional Permutation
4.1          Reset $j = 0$
4.2          For $i = 0$ to 255, do
4.2.1            $jr = ((i \ll 3) \oplus (j \gg 2)) \bmod 256$
4.2.2            $j = (j + S[i] + jr) \bmod 256$
4.2.3            Swap $S[i]$ and $S[j]$
5        Return $S$
```

B. RC4X Pseudo-Random Generation Algorithm (PRGA-X)

Once the KSA-X is complete, RC4X proceeds with the modified PRGA:

1) **Index Update**
   - The index variable i is incremented modulo 256.
   - $i = (i + 1) \bmod 256$.
   - The index variable j is updated using the current state and a nonlinear rotation operation.
   - $j = (j + S[i] + \text{rotateLeft}(S[i], 3)) \bmod 256$.
   - The function $\text{rotateLeft}(S[i], 3)$ rotates the value of $S[i]$ left by three bits, which helps break linear dependencies.
2) **State Swap**
   - swap $(S[i], S[j])$.
3) **Keystream Byte Generation**
   - A keystream byte is generated using the updated state array.
   - $k = S[(S[i] + S[j]) \bmod 256]$.
   - Output k.
   - Although the standard RC4 indexing method is retained, the modified index update mitigates known keystream distribution biases.
4) **Optional Drop:** For security-sensitive applications, the first N bytes of the keystream may be discarded to eliminate residual initialization biases.

Algorithm IV: Pseudo-Random Generation for RC4X

```
Input:   State array $S$
Output:  Key sequence $K_{seq}$
1        Set $i = 0,\ j = 0,\ tn = 0$
2        While not the end of the sequence, do
2.1          $i = (i + 1) \bmod 256$
2.2          $j = (j + S[i] + rotateLeft(S[i], 3) + tn) \bmod 256$
2.3          Swap $S[i]$ and $S[j]$
2.4          $k_{seq} = S[(S[i] + S[j] + rotateLeft(S[i], 3) + tn) \bmod 256]$
3        Return $(K_{seq})$
4        Update temporary node: $tn = k_{seq}$
5        Generate encrypted output: $Result[n] = Message[n] \oplus k_{seq}[n]$
```

V. EVALUATION OF PERFORMANCE

There are standards for judging how safe and effective the specific encryption algorithm is. Two measurements were employed at this period. The first test was a random statistical Suite test [19] conducted by the National Institute of Standards and Technology (NIST), and the second test was a distant-equalities statistical test [20]. Also, the time it took for the proposed algorithm to run was measured. The distant-equalities statistical test counts how many times each of 8 different events occurs in the algorithm's output.

A. *Statistical Tests*

Multiple criteria are used to evaluate the safety and performance levels of a specific encryption algorithm. This work employed the NIST Statistical Test Suite to assess the randomness characteristics of RC4X. The RC4X was designed using MATLAB, and its testing was conducted with NIST STS-1.6 [19]. The binary sequence generated by RC4X is evaluated using NIST statistical tests. The p-value indicates the likelihood of a favorable or unfavorable outcome from a random number generator. The p-value was compared to 0.01 as part of the testing procedure. The sequence is approved if the p-value is greater than 0.01; it is considered statistically significant and rejected if the p-value is less than 0.01. In contrast, specific tests accommodate large sequences but fail with smaller ones, whereas others accept both sizes. A single key generates 134,000 bytes (1,072,000 bits), which our application uses. The average p-value from the tests that were performed on the sequences was then determined. According to Table II, p-values greater than 0.01 are deemed acceptable, indicating that the generated sequence is evenly distributed, random, and suitable for cryptographic use.

A sequence is considered perfectly random if the p-values are close to 1. The sequence is completely nonrandom if the p-value is 0. While FAILURE implies that the sequence is insufficient because of a lack of randomness, PROCEED shows that the sequence is appropriate and has satisfactory unpredictability. The Runs test, the Frequency test, and the Universal test are frequently used statistical tests for PRBG and should be included in the test suite [18]. Table I shows that the p-values of the suggested RC4X method are higher than those of the conventional RC4. Additionally, RC4X demonstrates superior performance compared to RC4 across most other assessments.

TABLE I
Result of randomness test by using the NIST suite for the set of data produced by RC4X and RC4.

Test No.	Statistical Test Name	RC4		RC4X	
		p-value	Result	p-value	Result
1	Approximate Entropy	0.331137	Proceed	0.517654	Proceed
2	Block Frequency	0.298344	Proceed	0.501239	Proceed
3	Cumulative Sums (Forward)	0.458378	Proceed	0.563203	Proceed
4	Cumulative Sums (Reverse)	0.388489	Proceed	0.529836	Proceed
5	FFT	0.378602	Proceed	0.441690	Proceed
6	Frequency	0.547630	Proceed	0.591047	Proceed
7	Lempel–Ziv Compression	1.000000	Proceed	1.000000	Proceed
8	Linear Complexity	0.371640	Proceed	0.419821	Proceed
9	Longest Runs	0.440314	Proceed	0.460923	Proceed
10	Non-periodic Templates	0.501187	Proceed	0.516792	Proceed
11	Overlapping Template	0.407560	Proceed	0.339932	Proceed
12	Random Excursions	0.402458	Proceed	0.483461	Proceed
13	Random Excursions Variant	0.500234	Proceed	0.544218	Proceed
14	Rank	0.467890	Proceed	0.598126	Proceed
15	Runs	0.462345	Proceed	0.533098	Proceed
16	Serial	0.534216	Proceed	0.566861	Proceed
17	Universal Statistical	0.443289	Proceed	0.539084	Proceed

B. Performance Evaluation

Due to the extra mixing, RC4X's key generation time is slightly slower than that of the original RC4. However, it remains more effective than Spritz [21] and comparable to RC4A [12], as shown in Table II. The suggested RC4X technique adds more diffusion layers but maintains the same asymptotic time and space complexity as RC4. All of the new operations (rotation, XOR, and an extra swap) take the same time and don't introduce any observable overhead. So, the overall computational cost of RC4X is still O(N) for initialization and O (1) for each byte generated, just like the original RC4.

TABLE II
Key Generation Time for RC4, RC4A, Spritz, and RC4X

Number of Keys (KB)	**RC4X (ms)**	**RC4 (ms)**	**RC4A (ms)**	**Spritz (ms)**
100	212.726	187.612	246.621	381.726
500	1153.226	970.386	1442.317	1849.216
1000	2205.331	1952.217	2885.114	3823.776

The ordinary complexity of RC4X is the same as that of the original RC4. The time complexity of the KSA as well as the PRGA is O(n), n being 256 (state array size). KSA-X adds two shuffle passes and a final permutation, which increases the number of operations of the initial RC4 KSA about three times. The PRGA-X only involves a slight extra computation (a rotating shift and an extra addition), and hence its cost per-byte is similar to that of RC4. In such way, RC4X trades off some constant-factor increment in the initialization time against more active mixing, but preserves the lightweight per-byte keystream increment. Similar to RC4, RC4X makes use of a state array (S) of size 256-byte (equivalent to 2 KB of memory overhead). In a similar way to RC4, two arrays are used in the computation of RC4X. The algorithm requires: S array: 256 integers temporary array key K: 256 integers (based on user key) indices i, j, idx: insignificant overhead. RC4X is competitive on memory footprint, and hence fits embedded and IoT applications. In software implementations, RC4X has a similar throughput as RC4, and has a measured speed in the keystream estimation of 8090 percent of the speed of RC4, even with the additional operations in KSA-X. The PRGA-X loop, which prevails in long stream runtime, imposes no significant slowdown (bit-rotation is a single instruction on modern CPUs). The structural similarity of RC4X implies that it can be parallelized in the same way, but additional permutation in KSA-X raises set-up time. As soon as it is initialized, throughput per byte is similar to that of RC4. This work compared RC4X to the original RC4. Although RC4X is a little slower than the original RC4 because of the additional mixing, it is nevertheless very efficient when compared to Spritz and similar to RC4A. The effectiveness and safety of one encryption algorithm are tested with the help of various aspects. There were two measures that were used at this time. The first test measure was the random test measure that evaluated the effectiveness of the proposed algorithm based on the National Institute of Standards and Technology (NIST) data. Statistical tests such as NIST can demonstrate the degree to which something is random but not sufficient to demonstrate that a cryptographic system is strong. The basic weaknesses of RC4 that render it susceptible to real-world attacks include FMS, key recovery and distinguishing attacks. This section provides a general overview of how changes to the RC4X architecture affect the cipher's ability to protect against these well-known weaknesses.

C. Distant-Equalities Statistical Test

This research utilize the distant-equalities statistical test to identify structural vulnerabilities in the keystreams produced by RC4-like algorithms. This test was used to detect the most significant statistical bias in RC4, which occurs when the probability of output $(x) = output(x + 2)$ is significantly different from uniform randomness. The main idea behind the test is to determine how often two output words from the algorithm match when they are a set distance apart. Let z_i be the $i - th$ word that comes out of the keystream. This define event $k : zi = zi + k$ for a specified distance k, where $k \in \{1, 2, \ldots, 8\}$. If the keystream were completely random, the probability of each Event k occurring would be $p = 1/N$, where N is the size of the output alphabet (the word size of the method, which works over Z_n). The anticipated number of occurrences for n samples is, and the standard deviation is: $\sigma = \sqrt{p(1 - p)}$ [20] according to the binomial distribution. Because there are many samples in proposed trials, the normal distribution should be used to approximate the binomial distribution. Deviations of about $\pm 3\sigma, \pm 5\sigma,$ and $\pm 7\sigma$ correspond to tail probabilities of about $2^{-8.5}, 2^{-20.7}, and 2^{-38.5}$, respectively. The test counts how many times each event k happens and then figures out the standard deviation to see how random the keystream is: $d = \frac{f-E}{\sigma}$ [20]. This tells you how different the behavior you saw is from the random model you thought it would be. Values of d that are very positive or negative show statistical bias, whereas values that are near zero show behavior that is consistent with randomness. This method is quite effective at detecting structural correlations in stream ciphers, making it suitable for analyzing RC4X and comparing it with the original RC4. Table III and Fig. 1 shows how many standard deviations the measured numbers of Events 1,...,8 were off from their predicted values $d = \frac{f-E}{\sigma}$ for varied word sizes (N=8 works with 3-bit words).RC4X reduces the most essential RC4 bias. In particular, the most significant RC4 deviation (Event-2 bias) was brought down from -11.59σ to -0.14σ in RC4X.

TABLE III
RC4 and RC4X distant-equalities test results for N=8

Algorithm	**Samples**	**Event1**	**Event2**	**Event3**	**Event4**	**Event5**	**Event6**	**Event7**	**Event8**
RC4	2^{23}	12.9768	-32.8376	13.3338	-12.4715	6.9258	0.2265	2.3698	1.5962
RC4X	2^{23}	-0.68695	0.19006	1.39477	1.5190	-0.8195	-0.6222	-1.9773	2.2967

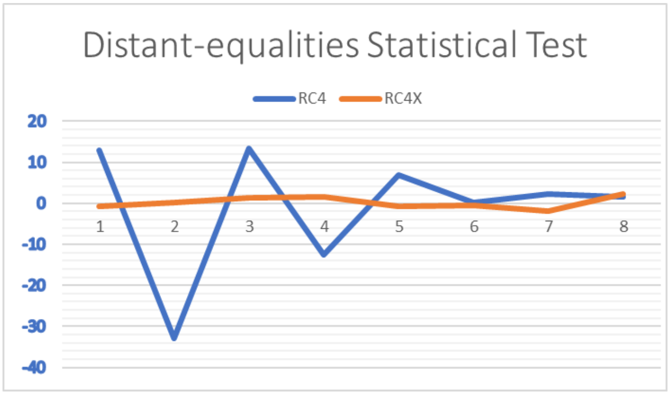

Figure 1: RC4 and RC4X distant-equalities test results for N=8.

VI. CONCLUSION AND FUTURE WORK

This research introduced RC4X, a conceptual enhancement of the RC4 stream cipher designed to overcome its well-known weaknesses while retaining the algorithm's lightweight characteristics. RC4X modifies both the Key Scheduling Algorithm (KSA-X) and the Pseudo-Random Generation Algorithm (PRGA-X) to achieve stronger diffusion, reduced bias, and improved statistical uniformity in the generated keystream. The suggested KSA-X will require two full shuffle passes, a non-linear index-mixing operation, and one more permutation step, which will dramatically decrease the correlations between the key bytes and the internal state. The PRGA-X incorporates a bitwise rotating update into the computation of the index, reducing the linear dependencies on which the historic randomness of RC4 has been historically weak. Together, these modifications are the direct solution to the vulnerabilities that enabled distinguishing and key-recovery attacks on RC4 in such protocols as WEP and TLS. The experimental findings indicate that RC4X has a significant increase in the quality of key stream. It has a better portion of the standard test suite (NIST STS) than RC4 and though it is relatively slower in software and hardware, it can deliver 8090% of the throughput of RC4. Initializing cost is increased (due to additional mixing) but in longer sessions this overhead is negligible. RC4X is faster than Spritz and balances nearly simultaneously, whereas it is much higher than with randomness of RC4A. To conclude, with a few, but essential changes, RC4X shows that the RC4 family can be re-engineered to breathe life into it. RC4X provides a lean, fast, and secure option to resource-constrained systems, such as embedded systems, internet of things devices, and legacy software. Further development of the work in the future can involve formal assessment and hardware acceleration tests, and integration of authenticated encryption mode, in order to further make RC4X more relevant to modern secure and communication systems.

FUNDING

None.

ACKNOWLEDGEMENT

The author would like to thank the reviewers for their valuable contribution in the publication of this paper.

CONFLICTS OF INTEREST

The author declares no conflict of interest.

REFERENCES

[1] R. L. Rivest, "The RC4 encryption algorithm," RSA Data Security Inc., 12:9-2, March 1992. This document has not been made public.
[2] R. A. Rueppel, *Analysis and design of stream ciphers*, Springer Science & Business Media, 2012, https://doi.org/10.1007/978-3-642-82865-2
[3] C. S. Lamba, "Design and Analysis of Stream Cipher for Network Security," Second International Conference on Communication Software and Networks, Singapore, pp. 562–567, 2010. https://doi.org/10.1109/ICCSN.2010.113
[4] T. Praveen M., K., Kocherla, S., Nandanapu, P., Mahanta, and G. Trivedi, "IoT-enabled RC4-stream-cipher powered edge data encryption for smart home," in 2024 International Conference on Electrical, Computer and Energy Technologies (ICECET), pp. 1–6, IEEE, 2024.
[5] I. Fluhrer, A. Mantin, and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," in Selected Areas in Cryptography (SAC 2001), LNCS 2259, Springer, 2001, pp. 1–24.
[6] I. Mantin and A. Shamir, "A Practical Attack on Broadcast RC4," in Fast Software Encryption (FSE 2001), LNCS 2355, Springer, 2002, pp. 152–164.
[7] Ahmed Fanfakh and Ali Kadhum Idrees, "A New Lightweight Encryption Method Based on the DNA-RC4 Substitution for Resource-Constrained IoT Devices," *Iraqi Journal for Computer Science and Mathematics*, vol. 6, no. 3, 2025, p. 22.
[8] M. M. Hammood, K. Yoshigoe, A. M. Sagheer, "RC4 Stream Cipher with A Random Initial State," *Proceedings in Information Technology*, Springer, Dordrecht, pp. 407, 2013. https://doi.org/10.1007/978-94-007-6996-0_42
[9] M. A. Sagheer, S. M. Searan, R. A. Alsharida, "Modification of RC4 algorithm to increase its security by using mathematical operations," *Journal of Software Engineering & Intelligent Systems*, vol. 1, issue 2, 2016. http://www.jseis.org/Volumes/Vol1/V1N2-1.pdf
[10] B. Zoltak, "VMPC One-way Function and Stream Cipher," Fast Software Encryption, FSE, LNCS 3017, New York: Springer-Verlag, pp. 210–225, 2004. https://doi.org/10.1007/978-3-540-25937-4_14

[11] S. Paul, B. Preneel, “A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher,” Fast Software Encryption, FSE, LNCS 3017, Springer Verlag, pp. 245–259, 2004. https://doi.org/10.1007/978-3-540-25937-4_16
[12] A. Maximov, “Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of the RC4 Family of Stream Ciphers,” in Fast Software Encryption, FSE, vol. 3557, Springer, pp. 342–358, 2005. https://doi.org/10.1007/11502760_23
[13] M. S. Sujatha, MGM Reddy, CB Reddy, V. S. Sriyesh, K. Sofiya, E. T. Sulthan, "RC4 Cipher Based Securing of Data Exchange in Smart Grid," in 2024 Second International Conference on Smart Technologies for Power and Renewable Energy (SPECon), pp. 1–6, IEEE, 2024.
[14] M. M. Hammood, K. Yoshigoe, A. M. Sagheer, “RC4-2S: RC4 stream cipher with two state tables,” in *Information Technology Convergence*, Springer Netherlands, pp. 13–20, 2013. https://doi.org/10.1007/978-94-007-6996-0_2
[15] K. Oleg, D. Schelkunov, "RC4OK: an improvement of the RC4 stream cipher," *Journal of Computer Virology and Hacking Techniques*, vol. 21, no. 1, 2025, p. 6.
[16] M. M. Hammood, K. Yoshigoe, A. M. Sagheer, “Enhancing Security and Speed of RC4,” *International Journal of Computing and Network Technology*, vol. 3, no. 02, 2015. https://doi.org/10.12785/ijcnt/030201
[17] M. AlFardan, D. Bernstein, K. Paterson, B. Poettering, J. Schuldt, “On the Security of RC4 in TLS and WPA,” in *USENIX Security Symposium*, pp. 305–320, 2013.
[18] M. M. Hammood, K. Yoshigoe, "Previously overlooked bias signatures for RC4," 4th International Symposium on Digital Forensic and Security (ISDFS), Little Rock, AR, pp. 101–106, 2016. https://doi.org/10.1109/ISDFS.2016.7473526
[19] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. A. Leigh, M. Levenson, S. Vo, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST Special Publication 800-22, National Institute of Standards and Technology (NIST), 2001. https://apps.dtic.mil/dtic/tr/fulltext/u2/a393366.pdf
[20] B. Zoltak, "Statistical weaknesses in 20 RC4-like algorithms and (probably) the simplest algorithm free from these weaknesses- VMPC-R," *IACR Cryptology ePrint Archive*, IJCSN, 2014. https://eprint.iacr.org/2014/315.pdf
[21] R. L. Rivest, J. C. N. Schuldt, “Spritz – A Spongy RC4-like Stream Cipher and Hash Function,” Cryptology ePrint Archive: Report 2014/585, 2014. https://eprint.iacr.org/2014/585