Article

Measuring Avalanche Properties on RC4 Stream Cipher Variants

Evaristo José Madarro-Capó ¹ , Carlos Miguel Legón-Pérez ¹ , Omar Rojas ^{2,3} and Guillermo Sosa-Gómez ^{2,*}

¹ Instituto de Criptografía, Facultad de Matemática y Computación, Universidad de la Habana, Habana 10400, Cuba; ejmcapo@gmail.com (E.J.M.-C.); clegon58@gmail.com (C.M.L.-P.)

² Facultad de Ciencias Económicas y Empresariales, Universidad Panamericana, Álvaro del Portillo 49, Zapopan 45010, Jalisco, Mexico; orojas@up.edu.mx

³ Faculty of Economics and Business, Universitas Airlangga, Surabaya 60285, Indonesia

* Correspondence: gsosag@up.edu.mx; Tel.: +52-3313682200

Abstract: In the last three decades, the RC4 has been the most cited stream cipher, due to a large amount of research carried out on its operation. In this sense, dissimilar works have been presented on its performance, security, and usability. One of the distinguishing features that stand out the most is the sheer number of RC4 variants proposed. Recently, a weakness has been reported regarding the existence of statistical dependence between the inputs and outputs of the RC4, based on the use of the strict avalanche criterion and the bit independence criterion. This work analyzes the influence of this weakness in some of its variants concerning RC4. The five best-known variants of RC4 were compared experimentally and classified into two groups according to the presence or absence of such a weakness.

Keywords: SAC; BIC; RC4 variants; stream cipher; statistical dependence

Citation: Madarro-Capó, E.J.; Legón-Pérez, C.M.; Rojas, O.; Sosa-Gómez, G. Measuring Avalanche Properties on RC4 Stream Cipher Variants. *Appl. Sci.* **2021**, *11*, 9646. <https://doi.org/10.3390/app11209646>

Academic Editor: Seokhie Hong

Received: 24 August 2021

Accepted: 11 October 2021

Published: 15 October 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

A stream cipher is a method widely used to cipher large volumes of information per unit of time. Thus, scenarios, such as mobile telephony, wireless networks, and cloud computing use this method as an encryption option [1]. Examples such as the encryption E0 in Bluetooth [2], A5 in GSM [3], and RC4 in wired equivalent privacy (WEP), Wi-Fi protected access (WPA), WPA2 [4] have been reported in the literature.

In this respect, it is impossible not to highlight the RC4 stream cipher, which has been mentioned on several occasions as the most used stream cipher in practice [4], mostly due to its inclusion in other scenarios such as Secure Sockets Layer (SSL), Transport Layer Security (TLS), Microsoft Windows, Lotus Notes, Apple Open Collaboration Environment (AOCE), and Oracle Secure SQL [4].

However, in the last decade, some applications [5,6] avoided the RC4 encryption, given some weaknesses found [7]. Although it is not considered safe [8], RC4 continues to motivate a lot of research [8–10], and it is cited to measure the effectiveness of methods that analyze weaknesses in stream ciphers [11–13] or the performance of applications that make use of cryptography [14–16].

Many of these investigations consist of proposals for modifications to the RC4 stream cipher, which has resulted in a considerable number of variants of this cipher being reported; without much effort, it is possible to find more than 20 variants in the literature [17–40]. All of these variants seek to increase the performance, usability, or security of the encryptor.

However, not a few have included reports on the weaknesses of the RC4 that remain in these variants or other new ones reported [4].

While the studies on the RC4 stream cipher continue, research on the proposed variants have also increased, to verify the safety of these variants concerning RC4. This

work analyzes the influence of one of the weaknesses reported in the literature on RC4 in the five best known (and most referenced) variants.

2. Motivation

In international literature, up until the first decade of the 21st century, the cryptographic analysis of stream ciphers or random number generators has become customary by applying statistical tests to their outputs to measure their degrees of randomness [41–43]. There are numerous statistics among those grouped in the batteries of NIST [44], Diehard [45], TestU01 [46], and Knuth [47], among others [48].

These tests have become standard, but since the first decade of this century, there has been a considerable increase in studies that show the inability of these tests in detecting other weaknesses that lead to the appearance of related keys, considerably weak keys due to the provocation of patterns in outputs, statistical dependence between inputs and outputs, statistical dependence between internal state and outputs, etc. Weaknesses that can cause attacks [49,50], which are practical or not, are considered to maintain a cipher in the suite of some computer applications.

In this way, it is essential to avoid the previous weakness and to have methods to detect it in the design and evaluation stage of the algorithm. In particular, it is necessary to have statistical tests that are capable of detecting the existence of significant statistical dependencies between the inputs and outputs of stream ciphers. In general, there are very few statistical test reports to detect the existence of statistical dependencies between the outputs and inputs of a stream cipher. Therefore, the design of statistical tests that allow for their evaluation in this sense is highly important in cryptography. In [49,50], Algorithms 1 and 2 were presented to extend the strict avalanche and bit independence criteria stream ciphers, respectively.

Algorithm 1 SAC stream ciphers algorithm

Input: $f, n, m, \alpha_1, \alpha_2, D$.

Output: If f satisfies the SAC

```

1:  $T = 0$ 
2: for  $i = 1 \rightarrow n$  do
3:   for  $r = 1 \rightarrow l$  do
4:     Compute  $V_r^i = Y_r \oplus Y_r^i$ 
5:   end for
6:   for  $j = 1 \rightarrow m$  do
7:     if  $\chi_{v^j}^2 > \chi_{\alpha_1, 1}^2$  then
8:        $T = T + 1$ 
9:     end if
10:  end for
11: end for
12: if  $Z_T > Z_{1-\alpha_2}$  then  $f$  does not satisfy the SAC
13: else  $f$  satisfies the SAC
14: end if
```

Algorithm 2 BIC stream ciphers algorithm**Input:** $f, n, m, \alpha_1, \alpha_2, D$.**Output:** If f satisfies the BIC

```

1:  $T = 0$ 
2: for  $i = 1 \rightarrow n$  do
3:   for  $r = 1 \rightarrow l$  do
4:     Compute  $V_r^i = Y_r \oplus Y_r^i$ 
5:   end for
6:   for each  $(j, k)$  do
7:     if  $\chi_{s_{jk}^i}^2 > \chi_{\alpha_1, 1}^2$  then
8:        $T = T + 1$ 
9:     end if
10:  end for
11: end for
12: if  $Z_T > Z_{1-\alpha_2}$  then  $f$  does not satisfy the BIC
13: else  $f$  satisfies the BIC
14: end if

```

In applying both criteria, the RC4 stream cipher was used as a test case to evaluate the effectiveness of the proposed algorithms experimentally [49,50]. As a result, it was shown that the RC4 presents a weakness of statistical dependence when large input keys are used. Although this weakness is not new, it has been reported in previous works. It is interesting to highlight the proposed evaluation using these two algorithms to determine its presence. As mentioned, there are numerous modification proposals on the RC4 stream cipher. The main motivation of this work is to evaluate whether some of the RC4 variants present this weakness. For this, both algorithms will be applied to the five most well-known RC4 variants RC4+ [27], VMPC [25], RC4A [26], NGG [23], and GGHN [24]. The results will be compared experimentally with those obtained for RC4.

3. SAC and BIC

Let $D = \{X_r | 1 \leq r \leq l\}$ the set of n -bit inputs X_r of a function (cipher) $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ and $Y_r = f(X_r)$ the output of m bits corresponding to the input X_r for each $1 \leq r \leq l$, where $n, m \in \mathbb{N}$. In [51] the difference between the outputs $Y_r = f(X_r)$ and $Y_r^i = f(X_r^i)$ corresponding to the inputs X_r and X_r^i is called the avalanche vector $V_r^i = Y_r \oplus Y_r^i$, where $X_r^i = X_r \oplus e_i$, with e_i , $1 \leq i \leq n$, denoting the unit vectors $e_1 = (1, 0, \dots, 0, 0, 0), \dots, e_n = (0, 0, \dots, 0, 0, 1)$.

Each $V_r^i = Y_r \oplus Y_r^i = (v_{r1}^i, v_{r2}^i, \dots, v_{rm}^i)$, where $v_{rj}^i \in \mathbb{F}_2$, with $1 \leq j \leq m$ and $1 \leq i \leq n$, is called an avalanche variable, represents the row r of the matrix H^i defined in Table 1 in [50].

Based on its definition, the strict avalanche criterion (SAC) [51] checks whether changing any input bit implies changing approximately half of the output bits. It is said that f satisfies the SAC if for all i , every avalanche variable v_{rj}^i with $1 \leq j \leq m$ and $1 \leq i \leq n$, follows a binomial distribution with parameters l and $1/2$, that is, $v_{rj}^i \sim B(l, 1/2)$. On the other hand, the bit independence criterion (BIC) [51] measures the degree of independence between each pair v_{rj}^i, v_{rk}^i of avalanche variables, with $1 \leq j, k \leq m$.

In this way, the two criteria measure a different characteristic of the effect of changing one bit of the input on the output bits. The SAC verifies uniformity in the distribution of each output bit, while the BIC measures the degree of independence between the output

bits. One important characteristic is to measure whether there is a correlation between these two criteria [52]. In [53], it was concluded that these two tests are quite uncorrelated by using the absolute correlation coefficient.

Both algorithms have in common the construction of the avalanche matrix. This matrix constitutes the basis for the detection of statistical dependency between the inputs and outputs of f . It can be seen that the BIC has a longer execution time when comparing all pairs of random variables.

In SAC and BIC, the random variable T measures the number of failures of the null hypothesis H_0 presented in [49,50]. In the case of the SAC criterion

$$T = \sum_{i=1}^n \sum_{j=1}^m t(v_{.j}^i, \alpha_1),$$

and for the BIC criterion,

$$T = \sum_{i=1}^n \sum_{j=1}^{m-1} \sum_{k>j}^m t(v_{.j}^i, v_{.k}^i, \alpha_1),$$

T can be approximated by the normal distribution with the mean and variance presented in [50] by,

$$Z_T = \frac{T - E(T|H_0)}{\sqrt{\sigma^2(T|H_0)}} \sim N(0, 1).$$

where $t(v_{.j}^i, \alpha_1)$ and $t(v_{.j}^i, v_{.k}^i, \alpha_1)$ take value 1 if H_0 is rejected with significance α_1 or 0 otherwise.

To compare the Z_T value with the $N(0, 1)$ distribution, a significance level α_2 is selected. Then, it is tested if f does not satisfy the BIC if $Z_T > Z_{1-\alpha_2}$ where $Z_{1-\alpha_2}$ is the critical value of the normal distribution table with a significance level α_2 .

4. RC4 Stream Cipher Description

Ron Rivest designed the RC4 algorithm in 1987 for the company RSA Data Security [4]. Its implementation is straightforward and fast and aims to generate sequences in units of one byte and allow keys of different lengths. The internal state of RC4 consists of a permutation S of the numbers $0, \dots, N-1$ and two indices $i, j \in \{0, \dots, N-1\}$. The index i is known, while j and the permutation S remain secret.

The RC4 comprises two components: the key scheduling algorithm (KSA) and the pseudo-random generator algorithm (PRGA). The KSA generates an initial state from the input parameter K . This starts with an array $\{0, 1, \dots, N-1\}$ where $N = 256$ by default. In the end, an initial state S_N is obtained, see Algorithm 3.

Algorithm 3 RC4 key-scheduling

```

1: for  $i = 0 \rightarrow 255$  do
2:    $S[i] \leftarrow i$ 
3: end for
4:  $j \leftarrow 0$ 
5: for  $i = 0 \rightarrow 255$  do
6:    $j \leftarrow (j + S[i] + K[i \bmod \text{keyLength}]) \bmod 256$ 
7:   Swap  $S[i]$  and  $S[j]$ 
8: end for

```

Once the initial state is obtained, it is used by the PRGA. The purpose of the PRGA is to generate the output sequence, in this case in bytes (Algorithm 4).

Algorithm 4 RC4 pseudo-random generator

```

1:  $i \leftarrow 0$ 
2:  $j \leftarrow 0$ 
3: while Generating Output do
4:    $i \leftarrow (i + 1) \bmod 256$ 
5:    $j \leftarrow (j + S[i]) \bmod 256$ 
6:   Swap  $S[i]$  and  $S[j]$ 
7:   Output  $S[(S[i] + S[j]) \bmod 256]$ 
8: end while

```

There are a variety of results on the weaknesses of the RC4 cipher [4], especially in the non-pseudorandomness of the permutation resulting from the KSA [11,54,55], and the reflection of input patterns in the outputs and the permutation [12,56,57].

5. RC4 Stream Cipher Variants Description

Numerous variants of the RC4 stream cipher algorithm have been proposed in the literature, many poorly referenced. Each variant seeks to solve or strengthen the RC4 algorithm against reported weaknesses or improve its practice performance. In this section, we present the most well-known and referenced variants.

5.1. RC4+ Algorithm

The stream cipher RC4+ was proposed in [27] to exploit the qualities of the RC4 algorithm and provide additional tools, to obtain a greater margin of safety by adding some operations to its structure. Modifications in both algorithms of the RC4 cipher are proposed and are called KSA+ and PRGA+ (see Algorithms 5 and 6).

The KSA+ consists of three basic scramblings, scrambling with IV, and zigzag scrambling. The initialization and the first layer basic scrambling are the same as in the original RC4 KSA. The second layer shuffles the permutation using IVs and the secret key K, and in the third layer, more shuffling is achieved in a zigzag way (see Algorithm 5).

Algorithm 5 KSA+**Input:** Secret key $K[0 \dots N - 1]$, initialization vector $IV[0 \dots N - 1]$.**Output:** Shuffled permutation $S[0 \dots N - 1]$.

```

1: for  $i = 0 \rightarrow N - 1$  do                                     ▷ Initialization
2:    $S[i] = i$ 
3: end for
4:  $j = 0$ 
5: for  $i = 0 \rightarrow N - 1$  do                                     ▷ Layer 1 Basic Scrambling
6:    $j = (j + S[i] + K[i \bmod \text{keyLength}]) \bmod N$ 
7:    $\text{Swap}(S[i], S[j])$ 
8: end for
9: for  $i = N/2 - 1$  down to  $0$  do                             ▷ Layer 2: Scrambling with IV
10:   $j = ((j + S[i]) \bmod N) \oplus ((K[i \bmod \text{keyLength}] + IV[i \bmod \text{IVLength}]) \bmod N)$ 
11:   $\text{Swap}(S[i], S[j])$ 
12: end for
13: for  $i = N/2 \rightarrow N - 1$  do
14:   $j = ((j + S[i]) \bmod N) \oplus ((K[i \bmod \text{keyLength}] + IV[i \bmod \text{IVLength}]) \bmod N)$ 
15:   $\text{Swap}(S[i], S[j])$ 
16: end for
17: for  $y = 0 \rightarrow N - 1$  do                                     ▷ Layer 3: Zigzag Scrambling
18:  if  $y \equiv 0 \bmod 2$  then
19:     $i = y/2$ 
20:  else
21:     $i = N - (y + 1)/2$ 
22:     $j = (j + S[i] + K[i \bmod \text{keyLength}]) \bmod N$ 
23:     $\text{Swap}(S[i], S[j])$ 
24:  end if
25: end for

```

KSA+ was designed in order to avoid secret key correlations from the permutation bytes and to guarantee that the keystream output bytes cannot have any correlation with the secret key. In the design of the PRGA+ algorithm (see Algorithm 6), the main cryptanalytic results about RC4 PRGA was taken into consideration.

Algorithm 6 PRGA+**Input:** KSA shuffled permutation $S[0 \dots N - 1]$.**Output:** Pseudo-random keystream bytes z .

```

1:  $j = 0$ 
2: for  $i = 0 \rightarrow L$  do
3:    $i = (i + 1) \bmod N$ 
4:    $j = (j + S[i]) \bmod N$ 
5:    $Swap(S[i], S[j])$ 
6:    $t = (S[i] + S[j]) \bmod N$ 
7:    $t' = ((S[(i \gg 3) \oplus (j \ll 5)] + S[(i \ll 5) \oplus (j \gg 3)]) \bmod N) \oplus 0xAA$ 
8:    $t'' = (j + S[j]) \bmod N$ 
9:   Output  $z = (S[t] + S[t']) \oplus S[t'']$ 
10: end for

```

Where $\gg$ and $\ll$ mean right shifting and left shifting, respectively, and L is the number of output bytes. The proposed design is based on avoiding that each output byte depends only on one input of the permutation. In this sense, they incorporate the exclusive or operation $\oplus$ and two new indices t' , and t'' .

5.2. RC4A Algorithm

The stream cipher RC4A was proposed in [26] with the objective to reduce the correlations between output bytes and internal variables by increasing the internal state (see Algorithms 7 and 8).

Algorithm 7 RC4A KSA**Input:** Secret keys $K_1[0 \dots N - 1]$ and $K_2[0 \dots N - 1]$.**Output:** Shuffled permutation $S_1[0 \dots N - 1]$ and $S_2[0 \dots N - 1]$.

```

1: for  $i = 0 \rightarrow N - 1$  do
2:    $S_1[i] = i$ 
3:    $S_2[i] = i$ 
4: end for
5:  $j_1 = 0$ 
6:  $j_2 = 0$ 
7: for  $i = 0 \rightarrow N - 1$  do
8:    $j_1 = (j_1 + S_1[i] + K_1[i \bmod keyLength]) \bmod N$ 
9:    $Swap(S_1[i], S_1[j_1])$ 
10: end for
11: for  $i = 0 \rightarrow N - 1$  do
12:    $j_2 = (j_2 + S_2[i] + K_2[i \bmod keyLength]) \bmod N$ 
13:    $Swap(S_2[i], S_2[j_2])$ 
14: end for

```

The difference between the algorithms RC4A KSA (see Algorithm 7) and KSA RC4 is that the first generates two state table by using two different keys. Thus, secret internal state of RC4A consists of two permutations S_1, S_2 and three variables i, j_1, j_2 .

Algorithm 8 RC4A PRGA**Input:** KSA shuffled permutations $S_1[0 \dots N - 1]$ and $S_2[0 \dots N - 1]$.**Output:** Pseudo-random keystream bytes z .

```

1:  $j_1 = 0$ 
2:  $j_2 = 0$ 
3: for  $i = 0 \rightarrow L$  do
4:    $i = (i + 1) \bmod N$ 
5:    $j_1 = (j_1 + S_1[i]) \bmod N$ 
6:    $Swap(S_1[i], S_1[j_1])$ 
7:   Output  $z_1 = S_2[(S_1[i] + S_1[j_1]) \bmod N]$ 
8:    $j_2 = (j_2 + S_2[i]) \bmod N$ 
9:    $Swap(S_2[i], S_2[j_2])$ 
10:  Output  $z_2 = S_1[(S_2[i] + S_2[j_2]) \bmod N]$ 
11: end for

```

In each round two output bytes are generated, the first byte is generated from S_2 and the second byte is generated from S_1 . Thus, this cipher speed up the RC4 encryption.

5.3. VMPC Algorithm

In [25] was proposed the algorithm VMPC (see Algorithms 9 and 10), whose name comes from “Variably Modified Permutation Composition”, which is based in the transformation $z = S[S_k[S_{k-1}[\dots[S_1[S[x]] \dots]]], 0 \leq x \leq N - 1$, where $x \in \{0, 1, \dots, N\}$ and $S_k[x] = (S[x] + k) \bmod N$. Such transformation is called a k-level VMPC function.

Algorithm 9 VMPC KSA**Input:** Secret keys $K[0 \dots N - 1]$ and $IV[0 \dots N - 1]$.**Output:** Shuffled permutation $S[0 \dots N - 1]$.

```

1: for  $i = 0 \rightarrow N$  do
2:    $S[i] = i$ 
3: end for
4:  $j = 0$ 
5: for  $k = 0 \rightarrow 3N - 1$  do
6:    $i = k \bmod N$ 
7:    $j = S[(j + S[i] + K[k \bmod keyLength]) \bmod N]$ 
8:    $Swap(S[i], S[j])$ 
9: end for
10: for  $k = 0 \rightarrow 3N - 1$  do
11:   $i = k \bmod N$ 
12:   $j = S[(j + S[i] + IV[i \bmod IVLength]) \bmod N]$ 
13:   $Swap(S[i], S[j])$ 
14: end for

```

In the KSA, the main differences from RC4 are that the length of loop where permutation S is updated six times more than RC4—firstly, three times with the key, and after

three times more with the initialization vector IV. The second update is an optional loop in KSA used by the designer to improve the diffusion using an initialization vector. However, designer statements show that it is possible to obtain good diffusion without using the initialization vector.

Algorithm 10 VMPC PRGA

Input: KSA Shuffled permutation $S[0 \dots N - 1]$, the last key dependence value of the index j .

Output: Pseudo-random keystream bytes z .

```

1: for  $i = 0 \rightarrow L$  do
2:    $j = S[(j + S[i]) \bmod N]$ 
3:   Output  $z = S[(S[j] + 1) \bmod N]$ 
4:    $Swap(S[i], S[j])$ 
5:    $i = (i + 1) \bmod N$ 
6: end for
  
```

5.4. NGG Algorithm

In [24], the authors proposed some modifications to the RC4 algorithm to expand RC4 to 32/64 bits with a table size significantly smaller than 2^{32} or 2^{64} . The proposed algorithm used different word and table sizes. The authors tried to keep the original structure of RC4 as much as possible, but the proposed changes affect some underlying design principles in which the security of RC4 is based (see Algorithms 11 and 12).

Algorithm 11 NGG KSA

Input: Secret keys $K[0 \dots N - 1]$ and a pre-computed random array $a[0 \dots N - 1]$.

Output: Shuffled permutation $S[0 \dots N - 1]$.

```

1: for  $i = 0 \rightarrow N - 1$  do
2:    $S[i] = a[i]$ 
3: end for
4:  $j = 0$ 
5: for  $i = 0 \rightarrow N - 1$  do
6:    $j = S[(j + S[i] + K[i \bmod keyLength]) \bmod N]$ 
7:    $Swap(S[i], S[j])$ 
8:    $S[i] = S[i] + S[j] \bmod M$ 
9: end for
  
```

The new algorithm was denoted by $RC4(n, m)$ where $N = 2^n$ is the size of the array S in word, m is the word size in bits, and $n \leq m$. It can be noted that the contents of the array S do not constitute a complete permutation of 32-bit or 64-bit words.

Algorithm 12 NGG PRGA**Input:** KSA Shuffled array $S[0 \dots N - 1]$.**Output:** Pseudo-random keystream bytes z .

```

1:  $j = 0$ 
2: for  $i = 0 \rightarrow L$  do
3:    $i = (i + 1) \bmod N$ 
4:    $j = (j + S[i]) \bmod N$ 
5:    $Swap(S[i], S[j])$ 
6:   Output  $z = S[(S[i] + S[j]) \bmod M] \bmod N$ 
7:    $S[(S[i] + S[j]) \bmod M] \bmod N = (S[i] + S[j]) \bmod M$ 
8: end for

```

Where $N = 2^n, M = 2^m$.

5.5. GGHN Algorithm

A version of NGG was introduced in [23], named GGHN cipher. A third variable k is used in KSA and PRGA to increase the security of the cipher, k is initialized in the KSA and is key dependent (see Algorithms 13 and 14).

Algorithm 13 GGHN KSA**Input:** Secret keys $K[0 \dots N - 1]$ and a pre-computed random array $a[0 \dots N - 1]$.**Output:** Shuffled array $S[0 \dots N - 1]$.

```

1: for  $i = 0 \rightarrow N - 1$  do
2:    $S[i] = a[i]$ 
3: end for
4:  $j = 0, k = 0$ 
5: repeat
6:   for  $i = 0 \rightarrow N - 1$  do
7:      $j = S[(j + S[i] + K[i \bmod keyLength]) \bmod N]$ 
8:      $Swap(S[i], S[j])$ 
9:      $S[i] = S[i] + S[j] \bmod M$ 
10:     $k = (k + S[i]) \bmod M$ 
11:   end for
12: until  $r$  times

```

The value of r can be computed for different values of n and m , see Table 1 in [23]. For example, for $n = 8$ and $m = 32, r = 20$.

Algorithm 14 GGHN PRGA**Input:** KSA Shuffled array $S[0 \dots N - 1]$.**Output:** Pseudo-random keystream bytes z .

```

1:  $j = 0$ 
2: for  $i = 0 \rightarrow L$  do
3:    $i = (i + 1) \bmod N$ 
4:    $j = (j + S[i]) \bmod N$ 
5:    $k = (k + S[j]) \bmod M$ 
6:   Output  $z = (S[(S[i] + S[j]) \bmod N] + k) \bmod M$ 
7:    $S[(S[i] + S[j]) \bmod N] = (k + S[j]) \bmod M$ 
8: end for

```

6. Measuring Avalanche Properties in RC4 Variants

Both statistical tests depend on the parameters n size of inputs, m size of outputs, and l number of inputs. In [50], we discuss and give recommendations for selecting these parameters to guarantee the effectiveness of the tests. The values of n , m , and l in this work were selected in such a way that they conform to these recommendations based on the significance levels α_1 and α_2 chosen.

6.1. Selecting Parameters n, m, l for Experiments

It is known [11,12,56] that, when using large input keys in the RC4, it is possible to find the so-called “related keys”, which provide outputs whose values are correlated with high probability. Therefore, the maximum possible value of the size of the inputs $n = 2048$ bits was selected. According to [50], the increase of the m parameter has the most significant influence on the increase in the execution time of the tests.

Furthermore, it is known [54,55] that the first four RC4 output values have the most significant bias. Thus, from both details, $m = 32$ bits was selected. For the number l of entries, the same value used in [50] was chosen, that is, $l = 65,538$. In the case of the two significance levels α_1 and α_2 , values that minimize errors were selected in both cases, type I and type II. According to the results in [50], for $\alpha_1 = 0.01$ and $\alpha_2 = 0.001$, observed values are obtained that satisfy the theoretically expected values; in this way, these were the chosen values.

6.2. Experiments on the SAC Criterion

When evaluating the SAC criterion, $E(T_i) = m \cdot \alpha_1 = 32 \times 0.01 = 0.32$ avalanche variable uniformity rejections for each change of bit i , with $1 \leq i \leq n$, and $E(T) = n \cdot m \cdot \alpha_1 = 2048 \times 32 \times 0.01 \approx 656$ rejects on all changes. Table 1 shows the expected value and the observed values for each cipher, while Figure 1 shows the distribution of the fails per bit changed when evaluating the SAC.

Table 1. Expected value and observed values for each cipher in SAC.

	Expected	RC4	RC4A	NGG	VMPC	RC4+	GGHN
Number of failures T	656	26,251	53,168	36,522	679	664	719

Figure 1. Distribution of the fails per bit changed in SAC for $n = 2048$, $m = 32$, and $l = 16,384$.

It is observed that the RC4A and NGG ciphers do not satisfy the SAC; they even have worse performance than the RC4 stream cipher. This result is not surprising since both variants do not add greater randomness to the operation of the RC4. In the case of the RC4A, the internal state is increased by using two permutation tables instead of one. While in the NGG, the internal state is also increased, but this time, the size in bits of each component of the permutation is increasing. Based on the results achieved, none of these two modifications eliminate this weakness of the RC4; it could even be said that they make it more detectable.

On the other hand, the RC4+, VMPC and GGHN ciphers do satisfy the SAC criterion since they have an observed value of failures very close to the expected value, for the significance level α_2 chosen. Table 2 shows the Z_T values for each cipher against the critical value $Z_{1-\alpha_2} = 3.09$.

Table 2. Values of Z_T for each cipher.

	RC4	RC4A	NGG	VMPC	RC4+	GGHN
Z_T	1004.87	2061.61	1408.10	0.90	0.34	2.50

Remarkably, the NGG and GGHN algorithms are very similar. The difference is that GGHN adds a variable k to the input-dependent internal state. To measure the implication of this variable, the same a arrangement of initial random values was used in both cases for the permutation that is used in the KSA. From the results obtained, it is experimentally demonstrated that this variable makes the difference in the behavior between both variants of RC4.

6.3. Experiments on the BIC Criterion

In the application of the BIC to the ciphers, $E(T_i) = C_2^m \cdot \alpha_1 = 496 \times 0.01 = 4.96$ avalanche variable uniformity rejections are expected for each bit change i , with $1 \leq i \leq n$, and $E(T) = n \cdot C_2^m \cdot \alpha_1 = 2048 \times 496 \times 0.01 \approx 10,158$ rejects on all you change. Table 3 shows the expected value and the observed values for each cipher, while Figure 2 shows the distribution of the fails per bit changed when evaluating the BIC.

Table 3. Expected value and observed values for each encryptor in BIC.

	Expected	RC4	RC4A	NGG	VMPC	RC4+	GGHN
Number of failures T	10,158	281,345	564,254	503,561	10,410	10,381	10,261

Figure 2. Distribution of the fails per bit changed in BIC for $n = 2048$, $m = 32$, and $l = 16,384$.

In this case, RC4A and NGG do not satisfy the BIC and maintain a worse behavior than the RC4 stream cipher. However, it is important to highlight how the RC4A from bit $i = 1150$, approximately, has a slightly better performance than the NGG. This is due to the rapid trend of the NGG towards the maximum value of possible failures.

The RC4+, VMPC, and GGHN ciphers satisfy the BIC criterion showing stable behavior, with the significance level α_2 chosen. Table 4 shows the Z_T values for each cipher against the critical value $Z_{1-\alpha_2} = 3.09$.

Table 4. Values of Z_T for each cipher.

	RC4	RC4A	NGG	VMPC	RC4+	GGHN
Z_T	2690.68	5497.67	4895.48	2.50	2.21	1.02

These results show that it is possible to obtain different results in both criteria. The GGHN cipher exhibits a better behavior in the BIC than the VMPC and RC4+ ciphers, the opposite of the SAC criterion.

Using the SAC and BIC criteria extended to stream ciphers, one of the weaknesses in RC4 was the existence of statistical dependence between the outputs and inputs of RC4 (avalanche weaknesses). The RC4A and NGG variants were experimentally shown not to satisfy the SAC and BIC tests. Its behavior is even worse than that of the RC4; these variants do not eliminate the statistical dependence between the outputs and inputs of the RC4; on the contrary, they increase it. The VMPC, RC4+, and GGHN variants substantially reduce the statistical dependency between the RC4 inputs and outputs and meet the SAC and BIC tests. It is recommended not to use the RC4A and NGG variants as they do not eliminate the avalanche type weaknesses present in the RC4.

7. Conclusions

RC4 has been one of the most studied stream ciphers in the literature in recent decades, resulting in dozens of variant proposals through various modifications. In this work, five variants of RC4 were analyzed using the SAC and BIC criteria extended to stream ciphers. The VMPC, RC4+, and GGHN variants meet both criteria, while the RC4A and NGG variants do not meet both criteria. Even the NGG variant had worse results than RC4 itself. Future work should apply these criteria to other stream ciphers outside the RC4 scheme.

Author Contributions: Conceptualization, E.J.M.-C. and C.M.L.-P.; methodology, E.J.M.-C., C.M.L.-P., O.R. and G.S.-G.; software, E.J.M.-C.; validation, E.J.M.-C., C.M.L.-P., O.R. and G.S.-G.; formal analysis, E.J.M.-C., C.M.L.-P., O.R. and G.S.-G.; investigation, E.J.M.-C., C.M.L.-P. and G.S.-G.; writing—original draft preparation, E.J.M.-C.; writing—review and editing, E.J.M.-C., C.M.L.-P., O.R. and G.S.-G.; visualization, E.J.M.-C.; supervision, E.J.M.-C., C.M.L.-P., O.R. and G.S.-G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Jiao, L.; Hao, Y.; Feng, D. Stream cipher designs: A review. *Sci. China Inf. Sci.* **2020**, *63*, 1–25. [CrossRef]
- Specification, B.C. Specifications List. 2003. Available online: <https://www.bluetooth.com/specifications/specs/> (accessed on 16 June 2021).
- Briceno, M.; Goldberg, I.; Wagner, D. A Pedagogical Implementation of the GSM A5/1 and A5/2 “Voice Privacy” Encryption Algorithms. 1999; Volume 26, p. 25. Available online: <http://cryptome.org/gsm-a512.htm> (accessed on 5 June 2021).
- Paul, G.; Maitra, S. *RC4 Stream Cipher and Its Variants*; CRC Press: Boca Raton, FL, USA, 2011.
- RC4 Cipher is No Longer Supported in Internet Explorer 11 or Microsoft Edge. Available online: <https://support.microsoft.com/en-us/help/3151631/rc4-cipher-is-no-longer-supported-in-internet-explorer-11-or-microsoft> (accessed on 23 May 2021).
- SSL Configuration Required to Secure Oracle HTTP Server After Applying Security Patch Updates. Available online: https://support.oracle.com/knowledge/Middleware/2314658_1.html (accessed on 24 May 2021).
- Satapathy, A.; Livingston, J. A Comprehensive Survey on SSL/ TLS and their Vulnerabilities. *Int. J. Comput. Appl.* **2016**, *153*, 31–38. [CrossRef]
- Soundararajan, E.; Kumar, N.; Sivasankar, V.; Rajeswari, S. Performance analysis of security algorithms. In *Advances in Communication Systems and Networks*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 465–476.
- Alsharida, R.; Hammood, M.; Ahmed, M.A.; Thamer, B.; Shakir, M. RC4D: A New Development of RC4 Encryption Algorithm. In *Lecture Notes in Networks and Systems, Proceedings of the 12th International Networking Conference, INC, Rhodes, Greece, 19–21 September 2020*; Ghita, B., Shialeles, S., Eds.; Springer: Cham, Switzerland, 2021; Volume 180.
- Parah, S.A.; Sheikh, J.A.; Akhoon, J.A.; Loan, N.A.; Bhat, G.M. Information hiding in edges: A high capacity information hiding technique using hybrid edge detection. *Multimed. Tools Appl.* **2018**, *77*, 185–207. [CrossRef]
- Grosul, A.L.; Wallach, D.S. *A Related-Key Cryptanalysis of RC4*; Technical Report; Department of Computer Science, Rice University: Houston, TX, USA, 2000.
- Matsui, M. Key collisions of the RC4 stream cipher. *Lect. Notes Comput. Sci.* **2009**, *5665 LNCS*, 38–50. [CrossRef]
- Chen, J.; Miyaji, A. How to find short RC4 colliding key pairs. In *International Conference on Information Security, Proceedings of the 14th International Conference, ISC 2011, Xi'an, China, 26–29 October 2011*; Springer: Cham, Switzerland, 2011; pp. 32–46.
- Tyagi, M.; Manoria, M.; Mishra, B. Effective data storage security with efficient computing in cloud. In *International Conference on Communication, Networks and Computing, Proceedings of the First International Conference, CNC 2018, Gwalior, India, 22–24 March 2018*; Springer: Singapore, 2018; pp. 153–164.
- Dhiman, A.; Gupta, V.; Singh, D. Secure portable storage drive: Secure information storage. In *International Conference on Communication, Networks and Computing, Proceedings of the First International Conference, CNC 2018, Gwalior, India, 22–24 March 2018*; Springer: Singapore, 2018; pp. 308–316.
- Nita, S.L.; Mihailescu, M.I.; Pau, V.C. Security and cryptographic challenges for authentication based on biometrics data. *Cryptography* **2018**, *2*, 39. [CrossRef]
- Sagheer, A.M.; Searan, S.M.; Salih, S.S. Developing RC4 Algorithm Using S-Box of Advanced Encryption Standard Cipher. *Int. J. Comput. Digit. Syst.* **2018**, *7*, 207–214. [CrossRef]
- Weerasinghe, T. An effective RC4 stream cipher. In *Proceedings of the IEEE 8th International Conference on Industrial and Information Systems, Peradeniya, Sri Lanka, 17–20 December 2013*; pp. 69–74.
- Xie, J.; Pan, X. An improved RC4 stream cipher. In *Proceedings of the International Conference on Computer Application and System Modeling (ICCSM 2010), Taiyuan, China, 22–24 October 2010*; Volume 7, pp. V7–156.
- Crainicu, B.; Boian, F.M. KSAm—An Improved RC4 Key-Scheduling Algorithm for Securing WEP. In *Novel Algorithms and Techniques in Telecommunications and Networking*; Springer: Dordrecht, The Netherlands, 2010; pp. 391–396.
- Hameed, S.M.; Mahmood, I.N. A modified key scheduling algorithm for RC4. *Iraqi J. Sci.* **2015**, *57*, 262–267.
- Jindal, P.; Singh, B. Performance analysis of modified RC4 encryption algorithm. In *Proceedings of the International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014), Jaipur, India, 9–11 May 2014*; pp. 1–5.
- Nawaz, Y.; Gupta, K.C.; Gong, G. A 32-bit RC4-like Keystream Generator. *IACR Cryptol. ePrint Arch.* **2005**, *2005*, 175.
- Gong, G.; Gupta, K.C.; Hell, M.; Nawaz, Y. Towards a general RC4-like keystream generator. In *International Conference on Information Security and Cryptology, Proceedings of the First SKLOIS Conference, CISC 2005, Beijing, China, 15–17 December 2005*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 162–174.

25. Zoltak, B. VMPC one-way function and stream cipher. In *International Workshop on Fast Software Encryption, Proceedings of the 11th International Workshop, FSE 2004, Delhi, India, 5–7 February 2004*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 210–225.
26. Paul, S.; Preneel, B. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. In *International Workshop on Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, 5–7 February 2004*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 245–259.
27. Maitra, S.; Paul, G. Analysis of RC4 and proposal of additional layers for better security margin. In *International Conference on Cryptology in India, 9th International Conference on Cryptology in India, Kharagpur, India, 14–17 December 2008*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 27–39.
28. Pardeep, P.; Pateriya, P. PC 1-RC4 and PC 2-RC4 algorithms: Pragmatic enrichment algorithms to enhance RC4 stream cipher algorithm. *Int. J. Comput. Sci. Netw.* **2012**, *1*, 2277–5420.
29. Paul, G.; Maitra, S.; Chattopadhyay, A. Quad-RC4: Merging Four RC4 States towards a 32-bit Stream Cipher. *IACR Cryptol. ePrint Arch.* **2013**, *2013*, 572.
30. Khine, L.L. A new variant of RC4 stream cipher. *World Acad. Sci. Eng. Technol.* **2009**, *50*, 958–961.
31. Hashem, S.H.; Jasim, A.B. A proposed modification on RC4 algorithm by increasing its randomness. *Al-Rafidain Univ. Coll. Sci.* **2017**, *39*, 349–372.
32. Hammood, M.M.; Yoshigoe, K.; Sagheer, A.M. RC4-2S: RC4 Stream Cipher with Two State Tables. In *Information Technology Convergence*; Springer: Dordrecht, The Netherlands, 2013; Volume 253 LNEE, pp. 13–20. [\[CrossRef\]](#)
33. McKague, M. Design and analysis of RC4-like stream ciphers. Master's Thesis, University of Waterloo, Waterloo, ON, USA, 2005.
34. Lv, J.; Zhang, B.; Lin, D. Distinguishing Attacks on RC4 and A New Improvement of the Cipher. *IACR Cryptol. ePrint Arch.* **2013**, *2013*, 176.
35. Jindal, P.; Makkar, S. Modified RC4 variants and their performance analysis. In *Microelectronics, Electromagnetics and Telecommunications*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 367–374.
36. Zhang, W.; Tang, S.; Zhang, L.; Ma, Z.; Song, J. Chaotic stream cipher-based secure data communications over intelligent transportation network. *Int. J. Antennas Propag.* **2015**, *2015*. [\[CrossRef\]](#) [\[PubMed\]](#)
37. Ma, H.; Zhang, Z. A New Private Information Encryption Method in Internet of Things under Cloud Computing Environment. *Wirel. Commun. Mob. Comput.* **2020**, *2020*, 8810987. [\[CrossRef\]](#)
38. Sadiq, A.T.; Farhan, A.K.; Hassan, S.A. A proposal to improve RC4 algorithm based on hybrid chaotic maps. *J. Adv. Comput. Sci. Technol. Res* **2016**, *6*, 74–81.
39. Searan, S.M.; Sagheer, A. Modification of RC4 Algorithm by using Two State Tables and Initial State Factorial. *Int. J. Comput. Netw. Inf. Secur.* **2016**, *8*, 1–8. [\[CrossRef\]](#)
40. Chang, D.; Gupta, K.C.; Nandi, M. RC4-hash: A new hash function based on RC4 (extended abstract). In *International Conference on Cryptology in India*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4329 LNCS, pp. 80–94. [\[CrossRef\]](#)
41. Yerukala, N.; Prasad, V.K.; Apparao, A. Performance and statistical analysis of stream ciphers in GSM communications. *J. Commun. Softw. Syst.* **2020**, *16*, 11–18. [\[CrossRef\]](#)
42. Gorbenko, I.; Kuznetsov, A.; Lutsenko, M.; Ivanenko, D. The research of modern stream ciphers. In *Proceedings of the 4th International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T)*, Kharkov, Ukraine, 10–13 October 2017; pp. 207–210.
43. Upadhyay, D.; Gandhi, S. Randomness evaluation of ZUC, SNOW and GRAIN stream ciphers. In *Proceedings of the International Conference on Communication and Networks*; Springer: Singapore, 2017; pp. 55–63.
44. Bassham, L.E., III; Rukhin, A.L.; Soto, J.; Nechvatal, J.R.; Smid, M.E.; Barker, E.B.; Leigh, S.D.; Levenson, M.; Vangel, M.; Banks, D.L.; et al. Sp 800-22 rev. 1a. a Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. 2010. Available online: <https://www.nist.gov/publications/statistical-test-suite-random-and-pseudorandom-number-generators-cryptographic> (accessed on 23 April 2021).
45. Marsaglia, G. The Marsaglia Random Number CDROM, with the DIEHARD Battery of Tests of Randomness. 2008. Available online <http://www.cs.hku.hk/diehard> (accessed on 3 April 2021).
46. L'ecuyer, P.; Simard, R. TestU01: AC library for empirical testing of random number generators. *ACM Trans. Math. Softw. (TOMS)* **2007**, *33*, 1–40. [\[CrossRef\]](#)
47. McClellan, M.T.; Minker, J.; Knuth, D.E. *The Art of Computer Programming, Vol. 3: Sorting and Searching*; Addison-Wesley Professional: Boston, MA, USA, 1974; Volume 28, p. 1175. [\[CrossRef\]](#)
48. Demirhan, H.; Bitirim, N. Statistical testing of cryptographic randomness. *İstatistikçiler Dergisi İstatistik Aktüerya* **2016**, *9*, 1–11.
49. Capó, E.J.M.; Cuellar, O.J.; Pérez, C.M.L.; Gómez, G.S. Evaluation of input-output statistical dependence PRNGs by SAC. In *Proceedings of the International Conference on Software Process Improvement (CIMPS)*, Aguascalientes, Mexico, 12–14 October 2016; pp. 1–6. [\[CrossRef\]](#)
50. Madarro-Capó, E.J.; Legón-Pérez, C.M.; Rojas, O.; Sosa-Gómez, G.; Socorro-Llanes, R. Bit independence criterion extended to stream ciphers. *Appl. Sci.* **2020**, *10*, 7668. [\[CrossRef\]](#)
51. Webster, A.; Tavares, S.E. On the design of S-boxes. In *Conference on the Theory and Application of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 1985; pp. 523–534.
52. Karell-Albo, J.A.; Legón-Pérez, C.M.; Madarro-Capó, E.J.; Rojas, O.; Sosa-Gómez, G. Measuring independence between statistical randomness tests by mutual information. *Entropy* **2020**, *22*, 741. [\[CrossRef\]](#) [\[PubMed\]](#)

-
53. Vergili, I.; Yücel, M.D. Avalanche and bit independence properties for the ensembles of randomly chosen $n \times n$ s-boxes. *Turk. J. Electr. Eng. Comput. Sci.* **2001**, *9*, 137–145.
 54. Martin, I. Analysis of the Stream Cipher RC4. Master's Thesis, The Weizmann Institute of Science, Rehovot, Israel, 2001.
 55. Mironov, I. (Not so) random shuffles of RC4. In *Annual International Cryptology Conference, Proceedings of the 22nd Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2002*; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2442, pp. 304–319. [[CrossRef](#)]
 56. Chen, J.; Miyaji, A. Generalized analysis on key collisions of stream cipher RC4. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **2011**, *E94-A*, 2194–2206. [[CrossRef](#)]
 57. Roos, A. A Class of Weak Keys in the RC4 Stream Cipher. 1995. Available online: <https://www.semanticscholar.org/paper/A-Class-of-Weak-Keys-in-the-RC-4-Stream-Cipher-Roos-Vironix/788c9474b2f74ff56253cdf799e71a3ef4d5f77c> (accessed on 24 August 2021).