Article

Practical Improvement in the Implementation of Two Avalanche Tests to Measure Statistical Independence in Stream Ciphers

Evaristo José Madarro-Capó [1], Eziel Christians Ramos Piñón [1], Guillermo Sosa-Gómez [2,*] and Omar Rojas [2]

[1] Instituto de Criptografía, Facultad de Matemática y Computación, Universidad de la Habana, Habana 10400, Cuba; evaristo.madarro@matcom.uh.cu (E.J.M.-C.)
[2] Facultad de Ciencias Económicas y Empresariales, Universidad Panamericana, Álvaro del Portillo 49, Zapopan 45010, Jalisco, Mexico; orojas@up.edu.mx
* Correspondence: gsosag@up.edu.mx; Tel.: +52-3313682200

Abstract: This study describes the implementation of two algorithms in a parallel environment. These algorithms correspond to two statistical tests based on the bit's independence criterion and the strict avalanche criterion. They are utilized to measure avalanche properties in stream ciphers. These criteria allow for the statistical independence between the outputs and the internal state of a bit-level cipher to be determined. Both tests require extensive input parameters to assess the performance of current stream ciphers, leading to longer execution times. The presented implementation significantly reduces the execution time of both tests, making them suitable for evaluating ciphers in practical applications. The evaluation results compare the performance of the RC4 and HC256 stream ciphers in both sequential and parallel environments.

Keywords: stream cipher; avalanche criterion; bit independence; parallel implementation; statistical test

Citation: Madarro-Capó, E.J.; Ramos Piñón, E.C.; Sosa-Gómez, G.; Rojas, O. Practical Improvement in the Implementation of Two Avalanche Tests to Measure Statistical Independence in Stream Ciphers. *Computation* **2024**, *12*, 60. https://doi.org/10.3390/computation12030060

Academic Editors: Francesco Cauteruccio and Yudong Zhang

Received: 17 January 2024
Revised: 3 March 2024
Accepted: 8 March 2024
Published: 19 March 2024

1. Introduction

Advancements in wireless communications have made protecting privacy between communicating parties a very important issue [1]. Stream ciphers are widely used in practice as an essential solution for protecting the information transmitted in communications. Currently, there are proposals for secure and efficient stream ciphers in various communication protocols for encrypting and decrypting messages in networks [2–5]. One of the encryption schemes in symmetric cryptography is stream ciphers [6,7]. These designs are based on the specification of a pseudorandom number generator (PRNG) algorithm. The term "pseudorandom" denotes a sequence of deterministic numbers that mimic an independent and uniformly distributed variable. The idea of this type of scheme is to generate a sequence of pseudorandom numbers with the same length as the text to be encrypted, represented, for example, in ASCII code or binary, and then combine them element by element using a simple and reversible function (e.g., Exclusive-OR, XOR) [8–10].

Nowadays [11–14], the suitability of these schemes in cryptographic environments is typically assessed in three ways. One approach is to estimate the numerical characteristics of the statistical distribution of the sequences generated by the PRNG that defines the stream cipher [15,16]. The second method involves determining the existence of a statistical correlation between the internal state and the outputs of the cipher algorithm [17–20]. Meanwhile, the third method involves measuring the resistance of the cipher algorithm against known attacks [21,22].

A wealth of information on statistical tests designed to assess the randomness of sequences of pseudorandom numbers is available in the literature. Some statistical test sets (called "battery" or "suite") are considered international standards, such as NIST [23], Dieharder [24], Knuth [25], and Marsaglia [26], among others [27,28]. However, this approach has the limitation of omitting the evaluation of other essential elements in determining the security of a stream cipher. One of these factors is the statistical independence between the internal state (comprising all the variables that determine the operation of the PRNG, including the input) of the cipher and its outputs [29]. Their determination is essential to avoid powerful cryptographic attacks such as the correlation attack [30,31], the differential attack [32], or the attack on neutral bits [33]. However, there is a lack of comprehensive bibliographies on assessments of statistical independence between the values of the internal state variables of a PRNG and its outputs.

New ways of increasing the effectiveness of these attacks are constantly emerging, and new attacks are also being developed due to the challenge faced by cryptographic algorithm designers in identifying any statistical dependencies [34,35]. Thus, to this day, this topic remains a subject of ongoing study and, simultaneously, a persistent challenge for developers of stream cipher algorithms.

The avalanche property is conceptualized as the possibility of measuring the impact of a specific change in a variable X on another variable Y, related in such a way that $Y = f(X)$ with f, an arbitrary function. The impact of altering certain bit(s) of X on Y has been studied in [36]. Assuming that f is a stream cipher, this property enables us to determine, at the bit level, the existence of statistical independence between the outputs and inputs of the cipher. There are specific criteria for evaluating avalanche properties in cryptography, including the strict avalanche criterion (SAC) and the bit independence criterion (BIC). However, both criteria were introduced to evaluate substitution boxes used in the design of block cipher algorithms [36]. However, an extension of both criteria was recently proposed for the evaluation of stream ciphers [37–39].

The proposed statistical tests play a fundamental role in evaluating stream ciphers for cryptographic purposes. Suppose the output elements in a specific cipher depend on only a few bits of the input. In that case, an attacker can detect these behaviors and use them to obtain information about the internal state parameters. For a transformation to exhibit the avalanche effect, on average, half of the output bits should change when a single input bit is complemented [36]. Both tests require extensive input parameters, which leads to longer execution times. This study's main result is the development of an implementation that substantially reduces the execution time of both tests, making it practical to evaluate stream ciphers.

2. Motivation of the Research

In cryptography, the avalanche effect is a term associated with a specific behavior of the mathematical elements used for encryption. It is considered one of the desirable properties when designing a strong cipher or cryptographic hash function. It is basically based on the fact that a slight change in the key or the cleartext should result in a significant change in the ciphertext. In practice, flipping a single bit in the key or cleartext should cause at least half of the bits in the output value of the cipher or hash to change.

This definition can be used to abstract the more general concept of independence in stream ciphers and, therefore, a proposal to measure the quality of a PRNG. Total independence is impossible to achieve since, given a PRNG, the same output is always obtained for the same input. However, in practice, an ideal PRNG will resemble a random function where the inputs and internal state do not cause statistical patterns in the outputs. That is, any cryptographically valid PRNG would satisfy the avalanche effect.

In [37,38], two statistical tests are proposed based on two avalanche properties: the strict and bit independence criterion. Both tests are presented as an extension of both criteria for evaluating independence in stream ciphers. In practice, applying either test over the entire space of possible keys is not feasible. This way, it is applied for a randomly generated subset of keys of prefixed size l. If both tests are satisfied for all keys in the subset, then it is assumed that the generator could meet both avalanche criteria.

Given l keys of n bits each, for each i, with $1 \leq i \leq n$, l sequences of m bits are generated corresponding to the l differences (in this case, the XOR) between the original output of each key and the output of the same key with the i bit inverted. The above process results in a matrix of $l \times m$, and from this matrix, both criteria, SAC and BIC, require m and C_2^m Hamming weight calculations, respectively. It can be noted that, with the increase in the parameters n, m, and l, an increase in the execution time of the algorithms corresponding to each test is obtained. This increase is given, in the worst case, by an approximate cubic and quartic factor for SAC and BIC, respectively.

Executing both tests can constitute a practical problem, given that high parameter values are usually used for statistical evaluation in cryptography. Examples of these values can be seen in [37–39]. To increase the applicability of the tests in this work, a practical improvement is presented in implementing both tests through the use of a parallel architecture.

3. Statistical Independence in Stream Ciphers

As previously mentioned, the design of statistical tests for measuring statistical independence in stream ciphers is crucial in cryptography. In [37,38], two algorithms are proposed to expand the bit independence criterion and the strict avalanche criterion to stream ciphers to achieve this goal. Both tests are described below, following some definitions.

The avalanche matrix shown in Table 1 forms the foundation for assessing the statistical independence between the inputs and outputs of a stream cipher f. Both algorithms are constructed in the same way. Through the SAC, it can be determined if the change in each input bit uniformly influences each output bit; meanwhile, through the BIC, it can be determined if this influence implies any correlation between the output bits.

Table 1. SAC matrix $H^i = (v_{rj}^i)$ defined in [37–39].

Avalanche Vectors	**Avalanche Variables**							
	$v_{\cdot 1}^i$	$v_{\cdot 2}^i$	$\ldots$	$v_{\cdot j}^i$	$\ldots$	$v_{\cdot k}^i$	$\ldots$	$v_{\cdot m}^i$
V_1^i	v_{11}^i	v_{12}^i	$\ldots$	v_{1j}^i	$\ldots$	v_{1k}^i	$\ldots$	v_{1m}^i
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
V_r^i	v_{r1}^i	v_{r2}^i	$\ldots$	v_{rj}^i	$\ldots$	v_{rk}^i	$\ldots$	v_{rm}^i
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
V_l^i	v_{l1}^i	v_{l2}^i	$\ldots$	v_{lj}^i	$\ldots$	v_{lk}^i	$\ldots$	v_{lm}^i

Let $V_r^i = Y_r \oplus Y_r^i = (v_{r1}^i, \ldots, v_{rm}^i)$ be each avalanche vector obtained between the outputs $Y_r = f(X_r)$ and $Y_r^i = f(X_r^i)$, with $X_r^i = X_r \oplus e_i$, $1 \leq i \leq n$, $1 \leq r \leq l$. Here, $f : F_2^n \to F_2^m$ is the cipher function to be evaluated, n is the input size, m is the output length, l is the number of inputs X to evaluate, and e_i denotes each canonical vector of dimension n with unit in bit i. It is known that f satisfies the SAC if each variable $v_{\cdot j}^i \sim B(l, 1/2)$; that is, $v_{\cdot j}^i$ follows a Binomial distribution, with $1 \leq j \leq m$. Meanwhile, the BIC is fulfilled based on the degree of independence between each pair of variables $(v_{\cdot j}^i, v_{\cdot k}^i)$ with $1 \leq j, k \leq m$.

3.1. Test of Independence Based on SAC

In the case of the SAC, the null hypothesis H_0 of the test is given by:

$$H_0 : HW(v_{\cdot j}^i) \sim B\left(l, \frac{1}{2}\right),$$
$$H_1 : HW(v_{\cdot j}^i) \nsim B\left(l, \frac{1}{2}\right).$$

The test statistic used under the null hypothesis is:

$$\chi^2{}_{v^i_{\cdot j}} = \frac{\left(HW(v^i_{\cdot j}) - \frac{l}{2}\right)^2}{\frac{l}{4}}. \tag{1}$$

The random variable T is defined by

$$T = \sum_{i=1}^{n}\sum_{j=1}^{m} t(v^i_{\cdot j}, \alpha_1), \tag{2}$$

with

$$t(v^i_{\cdot j}, \alpha_1) = \begin{cases} 1, & \text{If } H_0 \text{ is rejected for } v^i_{\cdot j} \\ & \text{ with significance } \alpha_1 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The expected value and the variance of T under the null hypothesis is given by

$$E(T|H_0) = \alpha_1 \cdot n \cdot m,\ \sigma^2(T|H_0) = \alpha_1 \cdot n \cdot m \cdot (1-\alpha_1). \tag{4}$$

Thus

$$Z_T = \frac{T - E(T|H_0)}{\sqrt{\sigma^2(T|H_0)}} \sim N(0,1). \tag{5}$$

The decision criterion compares the obtained value Z_T with the value $Z_{1-\alpha_2}$ so that if $Z_T > Z_{1-\alpha_2}$, the stream cipher does not satisfy the SAC. The SAC test is summarized in Algorithm 1.

Algorithm 1 SAC test's algorithm.

```
Input: $f, n, m, \alpha_1, \alpha_2, D$.
Output: If $f$ satisfies the SAC
1:  $T = 0$
2:  for $i = 1 \to n$ do
3:      for $r = 1 \to l$ do
4:          Compute $V^i_r = Y_r \oplus Y^i_r$
5:      end for
6:      for $j = 1 \to m$ do
7:          if $\chi^2_{v^i_{\cdot j}} > \chi^2_{\alpha_1,1}$ 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
```

3.2. Test of Independence Based on BIC

In the BIC test, the independence check between each pair $(v^i_{\cdot j}, v^i_{\cdot k})$, with $1 \le j,k \le m$, is based on verifying whether $s^i_{jk} = v^i_{\cdot j} \oplus v^i_{\cdot k}$ is balanced. The fact that s^i_{jk} is balanced implies that $HW(s^i_{jk}) \sim B(l, 1/2)$, where $HW(\cdot)$ is the Hamming weight. This adjustment is the basis of the BIC hypothesis test, given by:

$$H_0 : HW(s^i_{jk}) \sim B\left(l, \frac{1}{2}\right),$$

$$H_1 : HW(s^i_{jk}) \nsim B\left(l, \frac{1}{2}\right).$$

The test statistic is

$$\chi^2_{s^i_{jk}} = \frac{\left(HW(s^i_{jk}) - \frac{l}{2}\right)^2}{\frac{l}{4}}. \tag{6}$$

If $\chi^2_{s^i_{jk}} > \chi^2_{\alpha_1,1}$, then the null hypothesis H_0 is rejected with α_1 as the significance level.

Based on the random variable T, with:

$$T = T(n, m, \alpha_1) = \sum_{i=1}^{n} T^i(m, \alpha_1), \tag{7}$$

where

$$T^i(m, \alpha_1) = T^i = \sum_{j=1}^{m-1} \sum_{k>j}^{m} t\left(v^i_{\cdot j},\ v^i_{\cdot k}, \alpha_1\right), \tag{8}$$

and

$$t(v^i_{\cdot j},\ v^i_{\cdot k}, \alpha_1) = \begin{cases} 1 & \text{If } H_0 \text{ is rejected for } v^i_{\cdot j} \text{ and } v^i_{\cdot k} \\ & \text{ with significance } \alpha_1 \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

It is decided whether f satisfies the BIC, taking into account the number of rejections of H_0. With a significance level of α_1, the expected number of rejections of H_0 is:

$$E(T|H_0) = \alpha_1 \cdot n \cdot C^m_2,\ \sigma^2(T|H_0) = \alpha_1 \cdot n \cdot C^m_2 \cdot (1 - \alpha_1). \tag{10}$$

Thus

$$Z_T = \frac{T - E(T|H_0)}{\sqrt{\sigma^2(T|H_0)}} \sim N(0, 1). \tag{11}$$

Then, f satisfy the BIC if $Z_T > Z_{1-\alpha_2}$, with a significance level of α_2. Algorithm 2 implements the BIC test.

Both tests rely on large input parameters, meaning the execution time increases as the value of these parameters increases. In practice, one must find an implementation variation that reduces execution time. One solution is implementing the algorithms corresponding to each test in a parallel environment [40].

Algorithm 2 BIC test's algorithm.

```
Input: $f, n, m, \alpha_1, \alpha_2, D$.
Output: If $f$ satisfies the BIC
 1: $T = 0$
 2: for $i = 1 \to n$ do
 3:     for $r = 1 \to l$ do
 4:         Compute $V^i_r = Y_r \oplus Y^i_r$
 5:     end for
 6:     for each $(j, k)$ do
 7:         if $\chi^2_{s^i_{jk}} > \chi^2_{\alpha_1,1}$ 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
```

4. Implementation in the Parallel Environment of the SAC and BIC Algorithms

Multi-core processors have become the standard processor architecture, with dual-core, quad-core, and octa-core processors commonly available for personal computing.

The number of threads that can be utilized depends on the available hardware. Thanks to hyper-threading technology, two threads can be executed simultaneously per processor core. This practice is recommended [41]. In this manner, parallel programming is commonly implemented using multi-threading. In this implementation, parallelism is achieved by creating asynchronous threads that can run in parallel, usually with synchronization points during execution. A critical aspect of executing multi-threaded programs is scheduling, which determines the allocation of threads to processors and their execution sequence. The execution schedule of a multi-threaded program can be integrated into the program itself, implemented by an external scheduler, or a combination of both.

C++ incorporates support for concurrent operations through the thread library [42], designed for parallel execution. This library was selected because it enables the simultaneous execution of multiple functions, allows the passing of multiple parameters to a thread, and facilitates the writing of multi-threaded C++ programs without relying on platform-specific extensions. This will enable the writing of portable multi-threaded code [43]. The C++ programming language was used to implement both algorithms. The developer creates and manages threads using the C++ threading library. The SAC and BIC algorithms perform n iterations in the outer for loop to compute the n avalanche matrices and statistics. For both algorithms, threads were utilized to create a loop in parallel, with each thread being controlled. To implement a parallel for loop in C++11, the following function Listing 1 is defined:

Listing 1. Function parallelFor to create a loop in parallel

```
 void parallelFor(unsigned n,
 std ::function < void( int begin ,int end)>func)
 {
     unsigned n_threads = thread ::hardware_ concurrency();

     unsigned chunk _ size = n/n_ threads;
     unsigned chunk_ remainder = n % n_threads ;

     vector<thread>my _ threads(n_threads);

// Starting threads
for(unsigned i = 0; i < n_threads; ++i)
{
 int start = i *     chunk_ size ;
 my_threads[i]= thread(func,start,start+chunk_size);
 }

     // Remaining elements
 int start = n_threads *  chunk_size ;
     func (start,start + chunk_remainder);

     // Execution of tasks in parallel until finished
     for_each( my_threads. begin( ), my_threads. end(),
     std ::mem _fn(&  std ::thread ::join));
}
```

The above function can be used to parallelize a **for** Listing 2 , as follows in Listing 3:

Listing 2. Sequential for loop

```
for(int i=0; i<n;i++)
{
    compute(i);
}
```

Listing 3. Parallel for loop

```
parallelFor(n, [&](int start,int end)
{
    for(int i=start; i<end; i++)
        compute(i);
});
```

The parallelFor function splits a for loop into multiple smaller loops assigned to a thread and creates $2k$ threads (hyper-threading-based), where k is the number of cores in the machine where the algorithm is running. The parallelFor function uses the parameter n, which is the number of iterations of the cycle, equivalent to the number of input bits of the cipher and $std :: function < void(int\ begin, int\ end) > func$, where $func$ is a class that contains a function and in line 23 the for each is used to execute the $join()$ function of each thread as shown in [42,43].

Applying the above, the parallel implementation of Algorithms 1 and 2 can be written as:

In Algorithm 3, the function $ComputeSAC(i, f)$ is the implementation of steps 3 to 7 in Algorithm 1; in other words, it computes the i-th avalanche matrix and counts the number of rejections in the m columns of the null hypothesis in this matrix, this i-th value is stored in an array to avoid race conditions [44]. In Algorithm 4, the function $ComputeBIC(i, f)$ counts the number T_i of rejections complementing the i-th bit in the input among the C_2^m pairs of columns in the avalanche matrix H^i, which corresponds to steps 3 to 7 in Algorithm 2. The parallel code section in Algorithms 3 and 4 corresponds to steps 3 and 4.

Algorithm 3 Parallel SAC.

```
Input: $f, n, m, \alpha_1, \alpha_2, D$.
Output: If $f$ satisfies the SAC or not,
 1: $T = 0$
 2: $rejections = [1, n]$;
 3: parallelFor, $i = 1, 2, \ldots, n$;
 4:   $rejections[i] = ComputeSAC(i, f)$;
 5: for $i = 1 \to n$ do
 6:     $T = T + rejections[i]$;
 7: end for
 8: if $Z_T > Z_{1-\alpha_2}$ then $f$ does not satisfy the SAC
 9: else $f$ satisfies the SAC
10: end if
```

Algorithm 4 Parallel BIC.

```
Input: $f, n, m, \alpha_1, \alpha_2, D$.
Output: If $f$ satisfies the BIC or not,
 1: $T = 0$
 2: $rejections = [1, n]$;
 3: parallelFor $i = 1, 2, \ldots, n$;
 4:   $rejections[i] = ComputeBIC(i, f)$;
 5: for $i = 1 \to n$ do
 6:     $T = T + rejections[i]$;
 7: end for
 8: if $Z_T > Z_{1-\alpha_2}$ then $f$ does not satisfy the BIC
 9: else $f$ satisfies the BIC
10: end if
```

Once the threads are created, the work is assigned to each thread and divided equally among them. For instance, in the case of the SAC algorithm, if one had $n = 256$ and eight threads, then each thread would need to execute 32 operations. It is essential to ensure proper memory distribution because assigning a block to more than one stream can lead to race conditions [44]. The parallel implementation of the for loop handles each thread's creation, initialization, and termination. Due to the cost of dynamic memory management in both algorithms, selecting a dynamic memory allocator can play an important role in reducing wall clock time. For this purpose, GCC's dynamic memory manager works very well, but alternative options can be found in [45]. In this work, the implementation utilized the GCC's default dynamic memory allocator.

5. Experiments and Discussion

The increase in speed that was obtained was calculated by comparing the time between the sequential and parallel implementations of the SAC and BIC algorithms [46], independently of each other, and is given by

$$A(n,m,L) = \frac{T_s(n,m,L)}{T_p(n,m,L)},$$

where $T_s(n,m,L)$ is the execution time of the sequential algorithm, and $T_p(n,m,L)$ is the execution time of the parallel algorithm.

However, to accurately evaluate the effectiveness of the parallel and sequential algorithms, the execution time was measured in both cases using the same method and accuracy to ensure valid comparisons. This is because the main objective of this work is to improve the practical application of each algorithm. In [37,38], the computational complexity of each algorithm is specified, respectively, providing a basis for comparing them. The computer used for the experiments has the following technical specifications: an 11th Gen Intel(R) Core(TM) i5-1135G7 2.40 GHz (Intel, Santa Clara, CA, USA), with four cores, and 8 GB of RAM. Additionally, the computer has hyper-threading enabled. With four cores, hyper-threading enables the CPU to support two threads on a single core, allowing for the simultaneous execution of eight threads. The experiments have been conducted for the following scenarios of the problem.

5.1. Experiments in RC4 Stream Cipher

The suggestions in [39] were considered to choose the parameters. It is known that in the RC4 stream cipher [47], there is a statistical dependency between the first 4 bytes of the output and long keys. It was used as a parameter $n = 2048, m = 32$, and for the size of the set of pseudorandom vectors input D, the following values were used: $|D| = L$, where $L \in \{1000, 4096, 5000, 8192, 10{,}000, 15{,}000, 16{,}384\}$. Table 2 shows the improvement in the execution time of the parallel algorithm compared to the sequential algorithm, with an average execution time that was approximately 2.3 and 2.4 times faster in each case for BIC and SAC, respectively.

Table 2. Runtime results in RC4 for $n = 2048, m = 32$. Time in seconds.

L	**BIC**			**SAC**		
	Parallel	**Sequential**	**Increase in Speed**	**Parallel**	**Sequential**	**Increase in Speed**
1000	48	127	2.645833	40	106	2.650000
4096	225	533	2.368889	183	460	2.513661
5000	280	635	2.267857	229	585	2.554585
8192	469	1071	2.283582	389	940	2.416452
10,000	576	1307	2.269097	491	1134	2.309572
15,000	884	1965	2.222851	759	1721	2.267457
16,384	950	2135	2.247368	794	1891	2.381612

Figure 1 shows the execution times of the sequential and parallel algorithms for both algorithms, respectively, with the parameters specified applied to the RC4 cipher, where the improvement in execution time is observed.

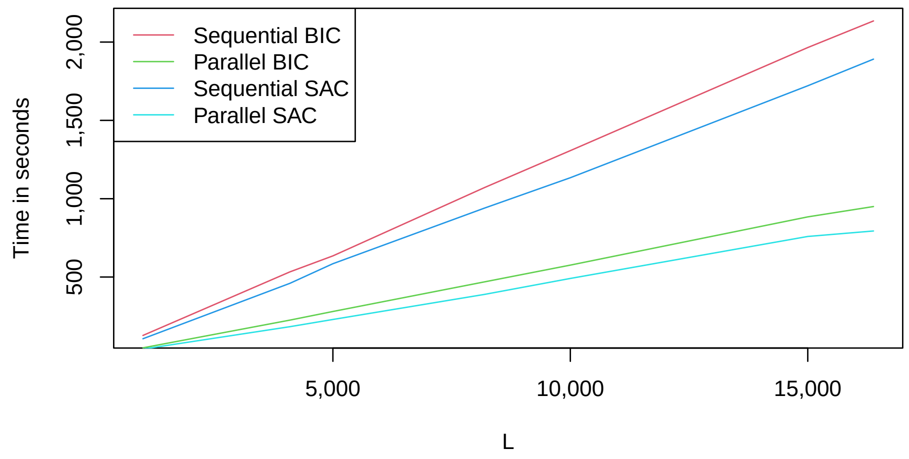

Figure 1. Comparison in time of parallel and sequential execution of SAC and BIC algorithms, for $n = 256, m = 32$, in RC4 with L in {1000, 4096, 5000, 8192, 10,000, 15,000, 16,384}.

In the results achieved for RC4, it can be seen in Table 2, for the specified parameters, that the execution time of Algorithm 4 is close to $L/1000$ min; meanwhile, Algorithm 3 remains below this value. As reported in [37–39], the RC4 cipher does not satisfy either of the two statistical tests.

5.2. Experiments in HC256 Stream Cipher

The HC256 [48] stream cipher generates a stream of bits from a 256-bit secret key and a 256-bit initialization vector. For the experiments, 256-bit keys were used, and as the initialization vector iv, the null vector, so that $iv[i] = 0$. The parameters used in this case were $n = 256, m = 32$. For the size of the set of pseudorandom vectors input D, the following values were used: $|D| = L$ in {1000, 4096, 5000, 8192, 10,000, 15,000, 16,384}. As seen in Table 3, for these parameters, the execution time of the parallel algorithm was, on average, 3 and 3.3 times faster than those in the sequential for BIC and SAC, respectively.

It is highlighted that the HC256 stream cipher satisfies both tests based on the strict avalanche criterion and the bit independence criterion. This cipher reported very few failures, including zero rejections of the null hypothesis in complementing some input bits.

Figure 2 shows another example of the execution time graphs of the SAC and BIC algorithms in parallel and sequential, using the HC256 cipher.

Table 3. Runtime results in HC256 for $n = 2048, m = 32$. Time in seconds.

L	BIC			SAC		
	Parallel	**Sequential**	**Increase in Speed**	**Parallel**	**Sequential**	**Increase in Speed**
1000	4	12	3.000000	3	11	3.666667
4096	16	55	3.437500	12	46	3.833333
5000	20	68	3.400000	15	56	3.733333
8192	34	103	3.029412	26	87	3.346154
10,000	44	133	3.022727	33	105	3.181818
15,000	71	198	2.788732	54	163	3.018519
16,384	78	227	2.910256	59	172	2.915254

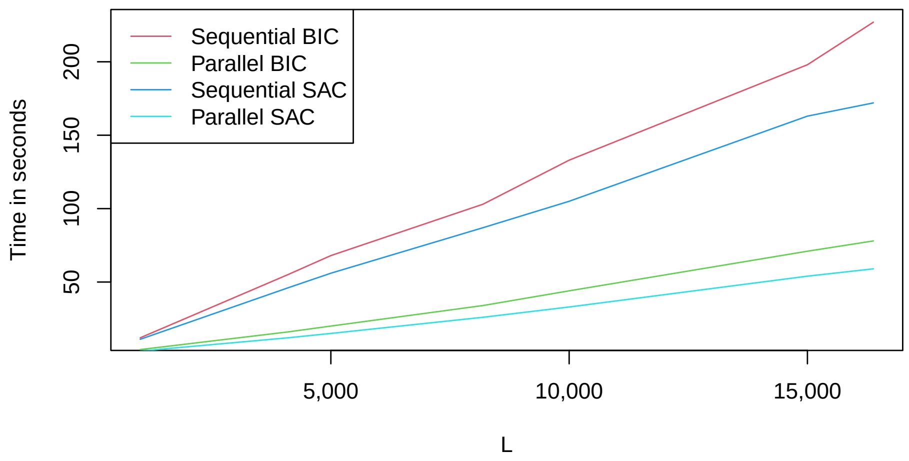

Figure 2. Comparison in time of parallel and sequential execution of SAC and BIC algorithms, for $n = 256, m = 32$, in HC256 with L in {1000, 4096, 5000, 8192, 10,000, 15,000, 16,384}.

The application of both tests on the HC256 cipher showed greater increases in speed than for the RC4 cipher. This could be attributed to the value of parameter n in the HC256 key initialization scheme.

5.3. Comparison between Both Tests

Another interesting comparison is that between the increase in speed measurements for both tests. We know from [37,38] that the BIC test has greater computational complexity than the SAC test. However, this comparison allows us to see the behavior of both tests separately. Figures 3 and 4 show the performance of each test on each cipher and each implementation.

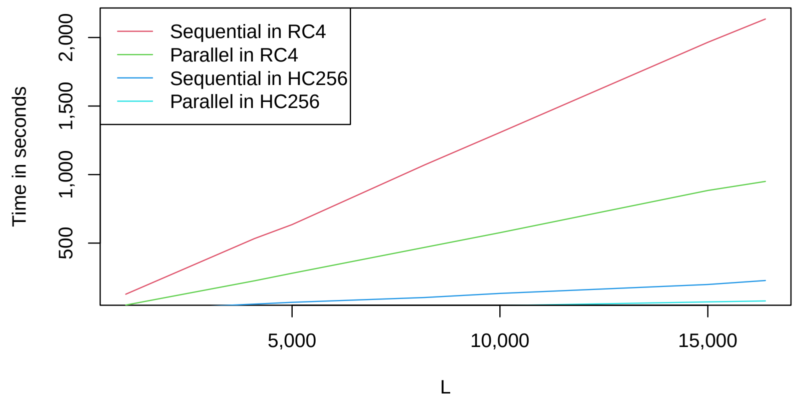

Figure 3. Comparison in time of parallel and sequential execution of BIC algorithm, for the same parameters described above in HC256 and RC4.

As can be seen in Figures 3 and 4, the parallel approach is more efficient and saves a considerable amount of time during execution for both tests. The difference in the execution of both tests is dominated by the value of parameter n. It should be noted that, for the RC4, the value of $n = 2048$ is 8 times greater than the value of $n = 256$ for the HC256. For this reason, it is clear that even the parallel time of the RC4 is greater than that obtained in the sequential application of the test for the HC256 cipher.

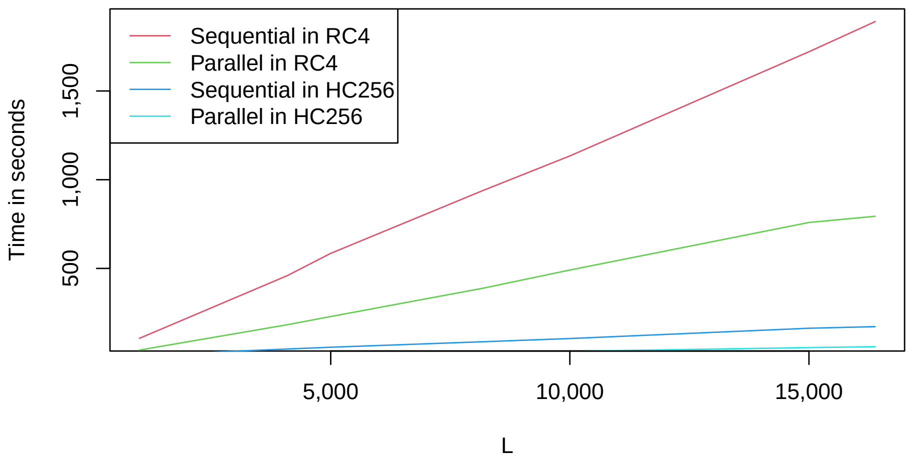

Figure 4. Comparison in time of parallel and sequential execution of the SAC algorithm, for the same parameters described above in HC256 and RC4.

As can be seen in Table 4, the implementation allows a slightly greater reduction in the execution time of the SAC algorithm for both ciphers. This may be because the SAC algorithm performs fewer operations than the BIC algorithm. However, the acceleration obtained for the BIC test is very close to that obtained in the SAC test.

Table 4. The average increase in speed of each test in seconds.

Test	RC4	HC256
SAC	2.441906	3.385011
BIC	2.329354	3.084090

In this sense, when the parallelized algorithms of both tests are run on a four-core processor, an acceleration that is around two–three times faster is obtained for the SAC and BIC, respectively. Using a computer with a higher number of cores or GPUs, for example employing CUDA [46], both algorithms could derive a significantly improved performance.

6. Conclusions

The experiments were conducted on portable computing devices with features that are currently available to everyone and easy to acquire, demonstrating the tests' applicability. In addition, libraries that can run on these devices, such as the C++11 threads library, were used. The parallel implementation of the SAC and BIC algorithms reduces the execution time compared to the sequential implementation. This allows for the practical evaluation of stream ciphers for higher values of the parameters n, m, and l in less execution time. On the other hand, it is essential to specify that the execution time is influenced by the encryption algorithm's execution time. The experiments revealed that the RC4 stream cipher does not pass either of the two tests, while the HC256 cipher passes both tests. In addition, a greater increase in speed was achieved in the SAC test for both ciphers. Meanwhile, on the other hand, both tests showed a faster execution in parallel when using the HC256 cipher.

Author Contributions: Conceptualization, E.J.M.-C.; Formal analysis, E.J.M.-C., E.C.R.P. and G.S.-G.; Investigation, E.J.M.-C., E.C.R.P., G.S.-G. and O.R.; Methodology, E.J.M.-C., E.C.R.P. and G.S.-G.; Project administration, O.R.; Supervision, G.S.-G. and O.R.; Validation, E.J.M.-C. and G.S.-G.; Writing—original draft preparation, E.J.M.-C. and E.C.R.P.; Writing—review and editing, O.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Acknowledgments: This work was (partially) supported by the Ibero-American Program of Science and Technology for Development (CYTED) through red522RT0131.

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

References

1. Venčkauskas, A.; Morkevicius, N.; Bagdonas, K.; Damaševičius, R.; Maskeliūnas, R. A lightweight protocol for secure video streaming. *Sensors* **2018**, *18*, 1554. [CrossRef]
2. 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]
3. Kaushik, N.; Bagga, T.; Aggarwal, R. Comparative study on IoT technologies-short & long range. *Int. J. Eng. Trends Technol.* **2020**, *68*, 37–42.
4. Mohammed, E.A.; Areed, N.F.; Takieldeen, A.; El-Awady, R.M. Hybrid Cryptographic Algorithm for LTE DataConfidentiality. *Int. J. Eng. Res. Technol. (IJERT)* **2016**, *5*, 12.
5. Fatma, N. Comparative Study of Vulnerabilities in Lte Cryptographic Algorithm. *Univ. Res. Symp.* **2018**, *50*, 19–25.
6. Kuznetsov, A.A.; Potii, O.V.; Poluyanenko, N.A.; Gorbenko, Y.I.; Kryvinska, N. *Stream Ciphers in Modern Real-Time IT Systems: Analysis, Design and Comparative Studies*; Studies in Systems, Decision and Control; Springer International Publishing: Cham, Switzerland, 2022; Volume 375. [CrossRef]
7. Jassim, S.A.; Farhan, A.K. A Survey on Stream Ciphers for Constrained Environments. In Proceedings of the 2021 1st Babylon International Conference on Information Technology and Science (BICITS), Babil, Iraq, 28–29 April 2021; pp. 228–233. [CrossRef]
8. Jiao, L.; Hao, Y.; Feng, D. Stream cipher designs: A review. *Sci. China Inf. Sci.* **2020**, *63*, 131101. [CrossRef]
9. Klein, A. *Stream Ciphers*; Springer: London, UK, 2013. [CrossRef]
10. Robshaw, M.; Billet, O.; Hutchison, D.; Kanade, T.; Kittler, J.; Kleinberg, J.M.; Mattern, F.; Mitchell, J.C.; Naor, M.; Nierstrasz, O.; et al. (Eds.) *New Stream Cipher Designs: The eSTREAM Finalists*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2008; Volume 4986. [CrossRef]
11. Bellini, E.; Huang, Y.J.; Rachidi, M. Statistical Tests for Symmetric Primitives: An Application to NIST Lightweight Finalists. In *Innovative Security Solutions for Information Technology and Communications*; Bella, G., Doinea, M., Janicke, H., Eds.; Lecture Notes in Computer Science; Springer Nature: Cham, Switzerland, 2023; Volume 13809, pp. 133–152. [CrossRef]
12. Afzal, S.; Yousaf, M.; Afzal, H.; Alharbe, N.; Mufti, M.R. Cryptographic Strength Evaluation of Key Schedule Algorithms. *Secur. Commun. Netw.* **2020**, *2020*, e3189601. [CrossRef]
13. Thabit, F.; Alhomdy, S.; Jagtap, S. Security analysis and performance evaluation of a new lightweight cryptographic algorithm for cloud computing. *Glob. Transit. Proc.* **2021**, *2*, 100–110. [CrossRef]
14. Silva, C.; Cunha, V.A.; Barraca, J.P.; Aguiar, R.L. Analysis of the Cryptographic Algorithms in IoT Communications. *Inf. Syst. Front.* **2023**, 1–18. [CrossRef]
15. Demirhan, H.; Bitirim, N. Statistical testing of cryptographic randomness. *İstatistikçiler Derg. İstatistik Ve Aktüerya* **2016**, *9*, 1–11.
16. Gorbenko, I.; Kuznetsov, A.; Gorbenko, Y.; Vdovenko, S.; Tymchenko, V.; Lutsenko, M. Studies on statistical analysis and performance evaluation for some stream ciphers. *Int. J. Comput.* **2019**, *18*, 82–88. [CrossRef]
17. Turan, M.S.; Doganaksoy, A.; Calik, C. Detailed statistical analysis of synchronous stream ciphers. In Proceedings of the ECRYPT Workshop on the State of the Art of Stream Ciphers (SASC'06), Leuven, Belgium, 2–3 February 2006.
18. Englund, H.; Johansson, T.; Sönmez Turan, M. A framework for chosen IV statistical analysis of stream ciphers. In Proceedings of the International Conference on Cryptology in India, Chennai, India, 9–13 December 2007; Springer: Berlin/Heidelberg, Germany, 2007; pp. 268–281.
19. Mishra, P.R.; Arvind Kumar, S.K.P.; Ojjela, O. Bit inclusion test: An enhanced avalanche test for stream ciphers. *Cryptologia* **2023**, *48*, 1–20. [CrossRef]
20. Shrivastava, S.; Lakshmy, K.V.; Srinivasan, C. On the Statistical Analysis of ZUC, Espresso and Grain v1. *Int. J. Comput.* **2021**, *20*, 384–390. [CrossRef]
21. Nandi, S.; Krishnaswamy, S.; Mitra, P.; Nandi, S.; Krishnaswamy, S.; Mitra, P. Recent Results on Some Word Oriented Stream Ciphers: SNOW 1.0, SNOW 2.0 and SNOW 3G. In *Information Security and Privacy in the Digital World—Some Selected Topics*; IntechOpen: London, UK, 2022. [CrossRef]
22. Moch, A. Provable security against generic attacks on stream ciphers. *J. Math. Cryptol.* **2023**, *17*, 20220033. [CrossRef]
23. Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Barker, E.; Leigh, S.; Levenson, M.; Vangel, M.; Banks, D.; Heckert, N.; et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; Technical Report NIST Special Publication (SP) 800-22 Rev. 1; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2010. [CrossRef]
24. Dieharder: A Random Number Test Suite. Available online: http://webhome.phy.duke.edu/~rgb/General/dieharder.php (accessed on 16 December 2023).
25. Knuth, D.E. The Art of Computer Programming. In *Seminumerical Algorithms*; Addition-Wesley: Reading, MA, USA, 1981; Volume 2.
26. Marsaglia, G.; Tsang, W.W. Some Difficult-to-pass Tests of Randomness. *J. Stat. Softw.* **2002**, *7*, 1–9. [CrossRef]

27. L'Ecuyer, P.; Simard, R. TestU01: A C library for empirical testing of random number generators. *ACM Trans. Math. Softw.* **2007**, *33*, 1–40. [CrossRef]
28. Ent—A Pseudorandom Number Sequence Test Program. Available online: https://www.fourmilab.ch/random/ (accessed on 16 December 2023).
29. Alamer, A.; Soh, B. Design and Implementation of a Statistical Testing Framework for a Lightweight Stream Cipher. *Eng. Technol. Appl. Sci. Res.* **2020**, *10*, 5132–5141. [CrossRef]
30. Penzhorn, W. Correlation attacks on stream ciphers. In Proceedings of the IEEE, AFRICON '96, Stellenbosch, South Africa, 25–27 September 1996; Volume 2, pp. 1093–1098. [CrossRef]
31. Meier, W.; Staffelbach, O. Fast Correlation Attacks on Stream Ciphers: Extended Abstract. In *Advances in Cryptology—EUROCRYPT '88*; Goos, G., Hartmanis, J., Barstow, D., Brauer, W., Brinch Hansen, P., Gries, D., Luckham, D., Moler, C., Pnueli, A., Seegmüller, G., et al., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1988; Volume 330, pp. 301–314. [CrossRef]
32. Biham, E.; Dunkelman, O. *Differential Cryptanalysis in Stream Ciphers*; DBLP Team: Trier, Germany, 2007.
33. Alekseychuk, A.N.; Konyushok, S.N. On the Efficiency of the Probabilistic Neutral Bits Method in Statistical Cryptanalysis of Synchronous Stream Ciphers. *Cybern. Syst. Anal.* **2016**, *52*, 503–508. [CrossRef]
34. Palit, S.; Roy, B.K.; De, A. A Fast Correlation Attack for LFSR-Based Stream Ciphers. In *Applied Cryptography and Network Security*; Goos, G., Hartmanis, J., Van Leeuwen, J., Zhou, J., Yung, M., Han, Y., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2846, pp. 331–342. [CrossRef]
35. Knellwolf, S.; Meier, W. High order differential attacks on stream ciphers. *Cryptogr. Commun.* **2012**, *4*, 203–215. [CrossRef]
36. Webster, A.F.; Tavares, S.E. On the Design of S-Boxes. In *Advances in Cryptology—CRYPTO '85 Proceedings*; Williams, H.C., Ed.; Springer: Berlin/Heidelberg, Germany, 1986; pp. 523–534.
37. 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 2016 International Conference on Software Process Improvement (CIMPS), Aguascalientes, Mexico, 12–14 October 2016; pp. 1–6. [CrossRef]
38. 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]
39. 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. [CrossRef]
40. Suciu, A.; Nagy, I.; Marton, K.; Pinca, I. Parallel implementation of the NIST Statistical Test Suite. In Proceedings of the 2010 IEEE 6th International Conference on Intelligent Computer Communication and Processing, Cluj-Napoca, Romania, 26–28 August 2010; pp. 363–368. [CrossRef]
41. Mochurad, L.; Shchur, G. Parallelization of Cryptographic Algorithm Based on Different Parallel Computing Technologies. In Proceedings of the IT&AS, Bratislava, Slovakia, 5 March 2021; pp. 20–29.
42. std::thread—cppreference.com. Available online: https://en.cppreference.com/w/cpp/thread/thread (accessed on 11 December 2023).
43. Williams, A. *C++ Concurrency in Action*; Simon and Schuster: New York, NY, USA, 2019.
44. C++ Tutorial: C++11/C++14 8. Race Conditions. 2017. Available online: https://www.bogotobogo.com/cplusplus/C11/8_C11_Race_Conditions.php (accessed on 16 December 2023).
45. Aparicio, G.; Salmerón, J.M.G.; Casado, L.G.; Asenjo, R.; Hendrix, E.M.T. Parallel algorithms for computing the smallest binary tree size in unit simplex refinement. *J. Parallel Distrib. Comput.* **2018**, *112*, 166–178. [CrossRef]
46. Bikov, D.; Bouyukliev, I.; Dzhumalieva-Stoeva, M. BooLSPLG: A Library with Parallel Algorithms for Boolean Functions and S-Boxes for GPU. *Mathematics* **2023**, *11*, 1864. [CrossRef]
47. Paul, G.; Maitra, S. *RC4 Stream Cipher and Its Variants*; CRC Press: Boca Raton, FL, USA, 2011.
48. Wu, H. A New Stream Cipher HC-256. In *Fast Software Encryption*; Roy, B., Meier, W., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 226–244.