entropy

MDPI

Article

Critical Analysis of Hypothesis Tests in Federal Information Processing Standard (140-2)

Elena Almaraz Luengo [1], Marcos Brian Leiva Cerna [1], Luis Javier García Villalba [1,*], Julio Hernandez-Castro [2] and Darren Hurley-Smith [3]

[1] Group of Analysis, Security and Systems (GASS), Universidad Complutense de Madrid, 28040 Madrid, Spain; ealmaraz@ucm.es (E.A.L.); marcolei@ucm.es (M.B.L.C.)
[2] School of Computing, University of Kent, Canterbury CT2 7NZ, UK; jch27@kent.ac.uk
[3] Information Security Group, Royal Holloway University of London, Egham TW20 0EX, UK; darren.hurley-smith@rhul.ac.uk
* Correspondence: javiergv@fdi.ucm.es

Abstract: This work presents an analysis of the existing dependencies between the tests of the FIPS 140-2 battery. Two main analytical approaches are utilized, the first being a study of correlations through the Pearson's correlation coefficient that detects linear dependencies, and the second one being a novel application of the mutual information measure that allows detecting possible non-linear relationships. In order to carry out this study, the FIPS 140-2 battery is reimplemented to allow the user to obtain *p*-values and statistics that are essential for more rigorous end-user analysis of random number generators (RNG).

Keywords: correlation; Dieharder; ENT; FIPS 140-2; independence; mutual information; NIST SP 800-22; *p*-value; randomness; statistic; TestU01

Citation: Almaraz Luengo, E.; Cerna, M.B.L.; Villalba, L.J.G.; Hernandez-Castro, J.; Hurley-Smith, D. Critical Analysis of Hypothesis Tests in Federal Information Processing Standard (140-2). *Entropy* **2022**, *24*, 613. https://doi.org/10.3390/e24050613

Academic Editor: Hector Zenil

Received: 16 March 2022
Accepted: 22 April 2022
Published: 27 April 2022

1. Introduction

The generation of random or pseudo-random sequences is crucial for scientific, cryptographic, and even entertainment purposes; from the generation of random variables [1,2], for mathematical or analytical purposes (for example, [3–6], and others) to applications in information and communication technologies [7–9], and image encryption [10] in medicine [11] among others.

The degree of randomness required by a given application can vary. Sequences may need to be reproducible (seeded RNGs produce these for simulations) or 'truly' random (e.g., cryptographic keys). The shared requirement is that one has some way of verifying that a RNG is sufficiently random for the target application. For the purpose of verifying the goodness of generated sequences, different statistical tests are used, which, to a given degree of confidence (α level), inform users whether the sequences can be used in such systems. Such tests may also be used to profile RNGs so that their flaws may be identified, reported, and rectified. Statistical tests of randomness are implemented in a variety of software languages, with some (FIPS 140-2 in particular) implemented in both FPGA and ASIC to provide rapid in-line testing of the RNG ouput (e.g., so-called total failure tests in hardware RNGs). Some of the best-known batteries in the literature are NIST SP 800-22 [12], TestU01 [13], Dieharder [14], ENT [15], and FIPS 140-2 [16] among others [17].

To successfully use a statistical test (especially a group of them, or battery), one must be aware of the attributes tested, rigor, and duration of the tests. The application of many different hypothesis tests, to ensure a thorough analysis of various traits of a sequence (such as independent and identical distribution if such is desired), may take significant computational time: hours or possibly days, even on high-end systems (e.g., 64-bit 3.6 GHz+ 8-core CPU 32 GB RAM DDR4 3066 MHz+, for the context of high-end at the time of writing). For this reason, one of the current lines of research is the

analysis of the interrelationship that may exist between the different tests that make up the batteries in order to, if there is one, discard any of the tests which duplicate results with little additional value. One of the most widely used approaches is the analysis of linear correlations between the different obtained p-values ([18–20], among others) or even between the statistics directly (see, for example, the reasoning about this issue in [21]). The most popular correlation measure used in this approach is Pearson's correlation coefficient. Given two random variables X and Y, this coefficient is defined as

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where σ_X and σ_Y are the standard deviations of X and Y, and $\text{cov}(X,Y)$ is the covariance between them. Its value belongs to the interval $[-1,1]$ and if $|\rho|$ has a value close to 1, it indicates a high linear dependence, and the sign informs if the dependence is direct or inverse. If $|\rho|$ has a value close to zero, it indicates the lower (linear) dependence between the variables. For random samples $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$, the sample correlation $r_{X,Y}$ is defined as

$$r_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where $\bar{x}$ and $\bar{y}$ are the arithmetic means of X and Y, respectively. The drawback of this approach is that the analysis focuses only on the study of linear relationships, omitting other possible (non-linear) relationships. Because of this, the analysis of interrelationships through mutual information (MI) [22] was recently proposed, as this measure allows for the detection of non-linear relationships between variables. Mutual information is always used to evaluate the "amount of information" obtained about one random variable when given the other random variable. If X and Y are continuous random variables with values in S_x and S_y, MI is defined as

$$MI = \int_{S_x}\int_{S_y} p(x,y)\ \log\left(\frac{p(x,y)}{p(x)p(y)}\right) dxdy$$

where $p(x,y)$ is the joint distribution of X and Y. If X and Y were discrete, then:

$$MI = \sum_{i=1}^{n}\sum_{j=1}^{m} p(x_i,y_j)\ \log\left(\frac{p(x_i,y_j)}{p(x_i)p(y_j)}\right)$$

where $p(x,y)$ is the discrete joint distribution of X and Y. Some of the areas in which MI has been applied are, for example, in lip reading [23], medical image segmentation [24], signal analysis [25], information theory [26] or clustering analysis [27], among others. For more details about mutual information, see [28].

In this research, we analyze the FIPS battery in detail. The design of this test battery, in terms of its output and analytical value, is one of the main issues when considering FIPS 140-2 as a means of determining whether an RNG is appropriate or safe to use in a given context. In this case, in contrast to NIST SP 800-22, for example, the implementation only provides the user with information on whether or not a sequence passes the applied test, but does not give more details. This prevents users from making judgments based on the statistical data generated by those tests, reducing a complex analysis to a Boolean pass/fail parameter that defies analysis without the use of more verbose tests. In this research, the battery is re-implemented to provide the user with a wider range of statistics, and the results are analyzed both from the point of view of Pearson's correlation and from the point of view of the mutual information measure.

With the new implementation (whose code we present in this paper), the user can apply the battery to real data and obtain the p-values associated to the different tests that form the battery. This allows the user to check the sequences obtained by different

generators and to decide if the generated sequences can be considered of good quality or if they need to be improved. The presentation of the p-values allows the user to have a statistical measure of the result obtained with each of the tests and to perform more in-depth studies related to the threshold that could be considered for the α level of a test that would change, for example, from rejecting a hypothesis in the test to having no evidence to do so with that α.

This paper is organized as follows: in Section 2, the re-implementation of FIPS test battery is explained; in Section 3, the materials and methods used in our analysis and an analysis of the independence of the test in the battery is performed; finally, in Section 4, the conclusions of the study are given.

2. FIPS Test Battery and the New Implementation

The FIPS 140-2 [16] battery is the successor of the FIPS 140-1 standard. It provides the same tests as 140-1, but with updated and stronger conditions for passing, with revised confidence intervals for all tests. It is a battery that, despite its limitations, is widely used by various manufacturers as the standard due to its speed and understandable (if cursory) output. An interesting study about this battery can be found in [29]. There is a new standard, FIPS 140-3 (https://csrc.nist.gov/publications/detail/fips/140/3/final, accessed on 15 March 2022), published in 2019. FIPS 140-3 does not implement statistical tests. However, as far as RNG testing is concerned, it focuses on entropy source modeling. Despite the existence of this version, FIPS 140-2 is still used by manufacturers when onboarding new RNGs, or as a start up/procedural check of sequences with expectations of randomness. FIPS 140-2 results are also frequently used in marketing materials for RNG hardware.

The rng-tools module for Linux includes an implementation of FIPS 140-2 (in rngtest). However, this implementation is not suitable for the purposes of this research for two reasons: (i) it works only with a fixed number of bits (20000 bits), and (ii) it only tells us if a sequence has passed the tests or not and it does not provide a statistic or a p-value. This work re-implemented these tests to overcome these limitations. The code was designed in Python and can be found in Appendix A.

With the new implementation, the tests work for any sequence size (though sequences smaller than 20,000-bits will not produce reliable results), and return both internal test statistics and the p-values calculated over those stats. The first three are broadly the same, except that instead of comparing the statistics with a range, statistical tests are performed (a binomial test in Monobit, and a Chi-square goodness-of-fit test in Poker and Runs). However, we need to make further changes to the last two tests for two reasons:

- In both cases, a value is not compared to a range, but directly fails the test if certain requirements are met (streaks of more than 25 bits in Long Run, two consecutive equal blocks in Continuous Run). This can be improved by calculating the probability of test-specific conditions occurring (runs of bits, repeat sequences, etc.), counting the number of times these cases occur, and performing a binomial test.
- The probability of fail conditions occurring is too low for the sequence size tested (at most 10^7). This makes the expected frequency of these conditions almost always 0 so the p-value is, most of the time, also 0. As the required sequence size is too large to run the battery in a reasonable time, we changed the original tests to increase the probability: the minimum length in the Long Run test goes from 26 bits to 8, and the block size in the Continuous Run test goes from 32 bits to 4. We leave it as future work to optimize the battery (or a possible C implementation) to repeat the tests with larger sequences and thus address the original tests.

2.1. Monobit Test

It consists of counting the number of ones, c_1, in a sequence. On the original battery, the test is passed if this number is between 9725 and 10,275. Adapting this test is quite simple: to measure the randomness of a sequence, a binomial test must be applied. The number

of data is n (length of the sequence), and the expected value is $c = \frac{n}{2}$. The resulting p-value will inform about how close to c the statistic c_1 is.

2.2. Poker Test

The sequence is divided into 4-bit blocks. For each block, there are 2^4 possible values; now, it is counted how many times each of them occurs. In the original implementation, the following formula is applied to the frequencies f_i:

$$X = \frac{16}{5000} \sum_{i=0}^{15} f_i^2 - 5000$$

and it is checked if $X \in [2.16, 46.17]$. This formula is actually a Chi-square goodness-of-fit test [30]. In the experimental case, the observed values are the frequencies f_i, and the expected values are the frequencies $e_i = e = 5000/16$, so

$$\sum_{i=0}^{15} \frac{(f_i - e)^2}{e} = \frac{16}{5000} \sum_{i=0}^{15} f_i^2 - 5000$$

Now it is possible to scale the test to sequences of size n, taking into account that there are $n/4$, so $e_i = e = (n/4)/16 = n/64$. Then, the statistic is

$$X = \sum_{i=0}^{15} \frac{(f_i - e)^2}{e} = \frac{64}{n} \sum_{i=0}^{15} f_i^2 - \frac{n}{4}$$

which follows a Chi-Square distribution and the p-value is: $p = P(z > X)$.

2.3. Runs Test

A run is a set of consecutive elements in the sequence (in this case consecutive 0 s or 1 s). This test calculates all the runs in a sequence and classifies them according to their element (0 or 1) and the size in bits: 1, 2, 3, 4, 5, and 6+ (6 or more). In the original battery, the sequence passes this test if the number of elements in each category is in the range shown in Table 1.

Table 1. Ranges and lengths in runs test.

Length	Range	Length	Range
1	2343–2657	4	251–373
2	1135–1365	5	111–201
3	542–708	6+	111–201

The ranges are the same for runs of 0 s and runs of 1 s. It is possible to perform a Chi-square goodness-of-fit test, where the observed values are s_i^j, with $i \in S = \{1, 2, 3, 4, 5, 6+\}$ and $j \in \{0, 1\}$. Under the hypothesis of randomness, the expected number of runs of zeros and ones must be the same for each size, that is, $e_i^0 = e_i^1 \; \forall i \in S$, so it will be considered, without loss of generality, only the streaks of 1 s. Let $k \in S$ and $n >> k$, with n being the sequence length.

- If the run does not appear at the beginning or at the end of a sequence, we must set $k + 2$ bits: the k ones of the run and the two zeros that delimit the beginning and end of the run.

$$\ldots 0 \overbrace{1 \ldots 1}^{k} 0 \ldots$$

- If the run appears at the beginning or end of the sequence, then we only need one zero to delimit the run, so we need to set $k + 1$ bits:

$$\overbrace{1 \ldots 1}^{k} 0 \ldots$$

$$\ldots 0\overbrace{1\ldots 1}^{k}$$

By linearity, the expected number of runs is $e_k^1 = (n-k-1)p^k(1-p)^2 + 2p^k(1-p)$, with p being the probability that a bit in the sequence is 1. In this case, $p = 1/2$, so $e_k^1 = (n-k+3)(1/2)^{k+2}$. Similarly, $e_k^0 = e_k^1$. In addition, the total expected number of runs is [31]: $e_T = (2n_0n_1/(n_0+n_1)) + 1$ where n_0 is the number of zeros and n_1 is the number of ones. Under the hypothesis of randomness, $n_0 = n_1 = n/2$ so $e_T = ((n^2/2)/n) + 1 = (n/2) + 1$ and $e_{6+} = e_T - \sum_{i=1}^{5}\sum_{j=0}^{1} e_i^j = \frac{n}{64} + \frac{7}{16}$. Then $e_{6+}^0 = e_{6+}^1 = e_6/2 = (n/128) + (7/32)$. Now it can be performed the goodness-of-fit test:

$$R = \sum_{i \in S}\sum_{j=0}^{1} \frac{(s_i^j - e_i^j)^2}{e_i^j}$$

2.4. Long Run Test

Originally, a sequence fails the test if it has a run of length greater than 25 bits. The simplest idea would be to perform a binomial test with the expected value number of runs of size 26 or more. However, this amount would be too small. As the size of the sequences in our experiments is not large, they have an effectively zero probability of 0 runs of this size or more (the minimum size in our experiments is $n = 1$ MB $= 8 \times 10^6$ bits, in that case $e_{26+} \simeq 0.619$). This problem was not significant in the original test (which only determined whether the sequence passed or not), but it affected our experiments, as this translates into almost always obtaining the same p-value for high-order runs. There are two solutions to address the problem: (i) working with much larger sequences, or (ii) altering the original test, causing it to fail with runs of less length. This solution was tested with runs of size 8 bits or more and gave a greater range of expected quantities. This is, therefore, not truly reflective of the original test, but is used to provide a representative and meaningful statistic for use in a subsequent binomial test. In this research, solution (ii) was taken. The number of runs of size 8 or more was used as the expected value for the binomial test. In that case, $e_{8+} = e_T - \sum_{i=1}^{7}\sum_{j=0}^{1} e_i^j = \frac{n+2^7-6}{2^8}$. If $n = 1$ MB, then $e_{8+} \simeq 31{,}250.48$.

2.5. Continuous Run Test

The original test divides the sequence u into N blocks of 32 bits, and associates to each block a real number in $(0,1)$, using the transformation

$$b_1 \ldots b_{32} \to \frac{\sum_{i=1}^{32} 2^{32-i} \cdot b_i}{2^{32}}$$

The test fails if a run is found. This is equivalent to say that two consecutive blocks are equal without the need to transform each block into a real number.

On this basis, the p-value is calculated. Again, the possibility of performing a binomial test (with expected value e being the number of times two consecutive blocks are equal) is presented. There are $N-1$ pairs of consecutive blocks, and the probability that two blocks are equal is $p = 1/2^{32}$ so $e = (N-1) \cdot p = (N-1)/2^{32}$. As in the previous test, this number is almost always 0 if the sequence is not sufficiently large. It is decided to alter the test (in this, 4-bit blocks are used, which offer a greater variety of equal-block pairings within a given sequence to analyze and characterize).

3. Analysis of the Independence of the Tests in FIPS Battery

Two machines are used for this analysis:

- Windows machine with specifications:
  - Operating System: Windows 10 (64-bit)
  - CPU: AMD Ryzen 3700XT (3.6 Ghz, 8 cores, 16 threads)
  - RAM: 64 GB

- Linux machine with specifications:
  - Operating System: Debian (64-bit)
  - CPU: Intel Core 6200U (2.3 Ghz, 2 cores, 4 threads)
  - RAM: 8 GB

In general, the Windows machine is used for the main calculations, as it is more powerful, reserving the Linux machine for the generation of sequences. The elements that are selected for the experimentation are shown in Table 2. As the results are not significantly different when changing the sequence size or the generator, it is shown the case of sequences of 10^7 bits, generated with dev/urandom. We work with Pearson's correlation coefficient and mutual information. For both measures, running a single experiment is not ideal: recall that the resulting p-values are uniformly distributed in $(0, 1)$ so that for an α significance, the probability of failing the test is α. Therefore, we carry out 100 different tests for each pair of tests (both Pearson's correlation and mutual information), and we execute a Kolmogorov–Smirnov (K-S) test on each set of 100 p-values, thus we have a reliable measure of their uniformity.

Table 2. Information and parameters of the experiments.

Number of Sequences	Size (Bits)	Generator	Significance
10^4	10^5 10^6 10^7	dev/urandom CryptGenRandom() python.secrets() qRNG	0.001

In Figure 1 are represented the results corresponding to Pearson's correlation of the obtained p-values. As can be seen, there is some correlation between (i) the Poker and Monobit tests, (ii) Monobit with Runs tests, (iii) Poker and Runs tests and (iv) Poker and Continuous Run tests. In all cases, the correlation is similar (around 0.2). The rest of the correlations are, in principle, low, although they are significantly lower between Long Run and Monobit, and between Continuous Run and Long Run.

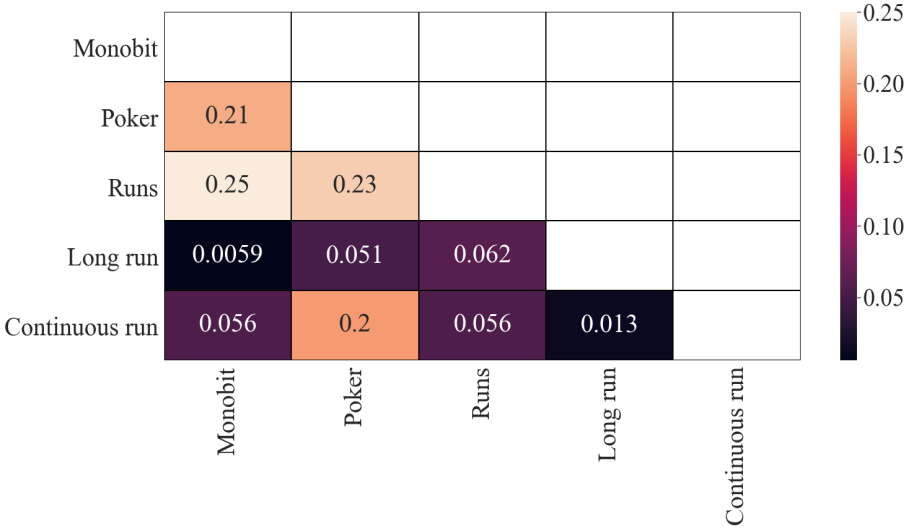

Figure 1. Pearson's correlation (p-values): results.

Let us turn to the significance matrix (Figure 2). The results of the K-S test give us interesting information; only the two pairs with the lowest correlation passed the test: Long Run and Monobit (0.41) and Continuous Run and Long Run (0.53). The other four pairs, which initially did not have a high correlation, did not pass the test: 1.2×10^{-10} for Continuous Run and Monobit, 2.4×10^{-10} for Long Run and Poker, 1.5×10^{-18} for Long

Run and Runs, and 2.2×10^{-7} for Continuous Run and Runs, which is an indicator that they share dependencies. Finally, the four most correlated pairs fail the test with a p-value of ϵ in all cases.

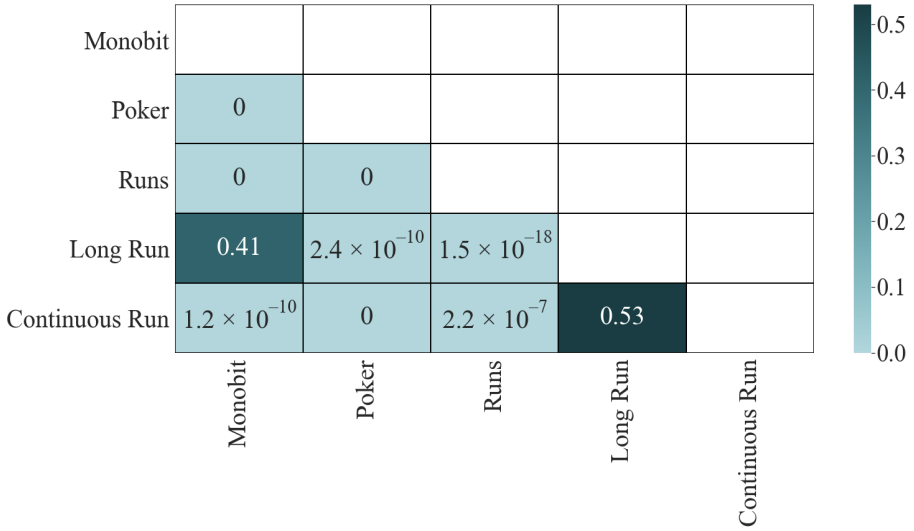

Figure 2. K-S test results.

With regard to the results of mutual information, the obtained results are similar (see Figure 3) although on a different scale. The previous four correlated pairs remain correlated, with values around 0.013, more than 10 times less than in Pearson's correlation results. That could be an indicator that the dependence in these pairs is mainly linear. In the same way, the two least-correlated pairs remain (around 0.001), although this time they are closer to the other values (around 0.0016).

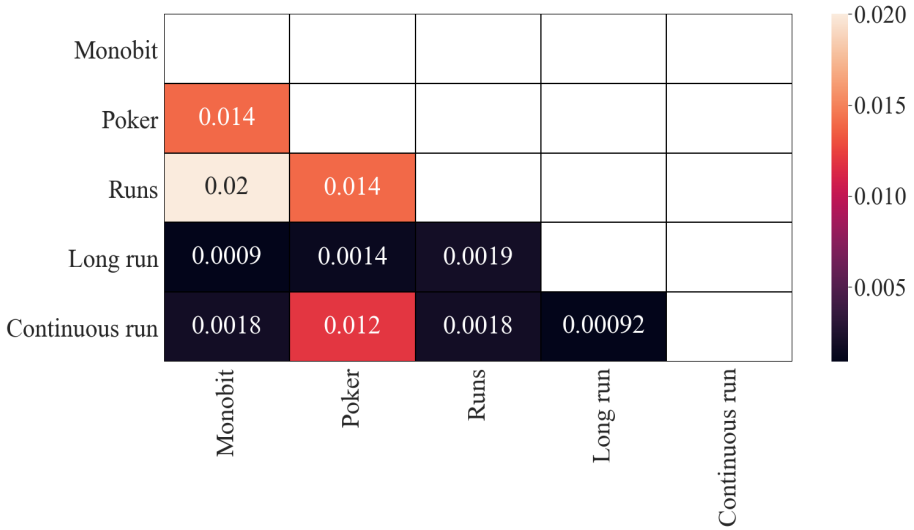

Figure 3. Mutual information (p-values): results.

The results of the K-S test do not vary concerning those of the correlation (see Figure 4): the four most dependent pairs have a p-value of ϵ, the two least correlated pass the test, and the other four fail it, although not in such an extraordinary way (between 2.4×10^{-5} and 9.9×10^{-11}).

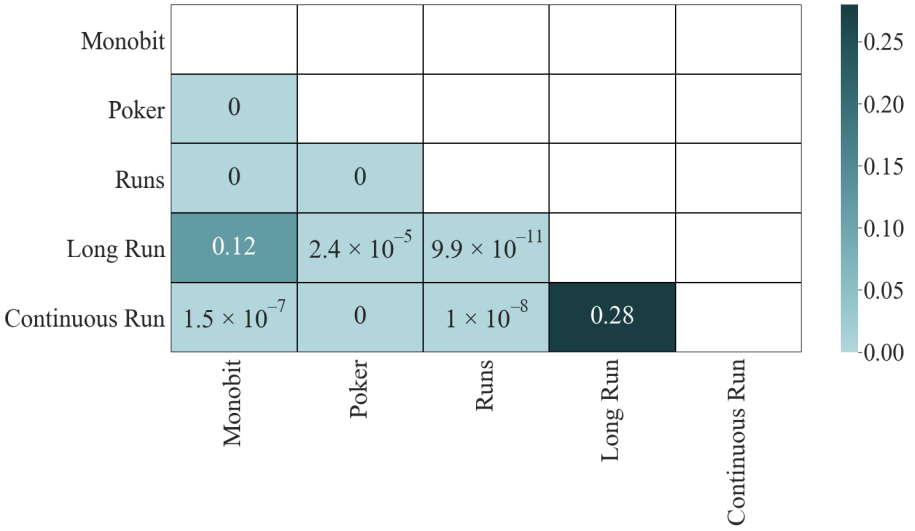

Figure 4. Mutual information (*p*-values): K-S.

Figure 5 shows the dispersion matrix of the *p*-values. It is possible to detect the largest correlations between tests. There is a little deviation in the lower right corner for the pairs Poker with Monobit, Runs with Monobit, and Runs with Poker, that is, there are no cases in which the first of the tests has a low *p*-value and the second has a high *p*-value. As for the pair between Continuous Run and Poker, this deviation is in the upper left corner. There are no cases in which the first has a high *p*-value and the second has a low *p*-value. In the rest of the pairs, it is not possible to detect dependencies.

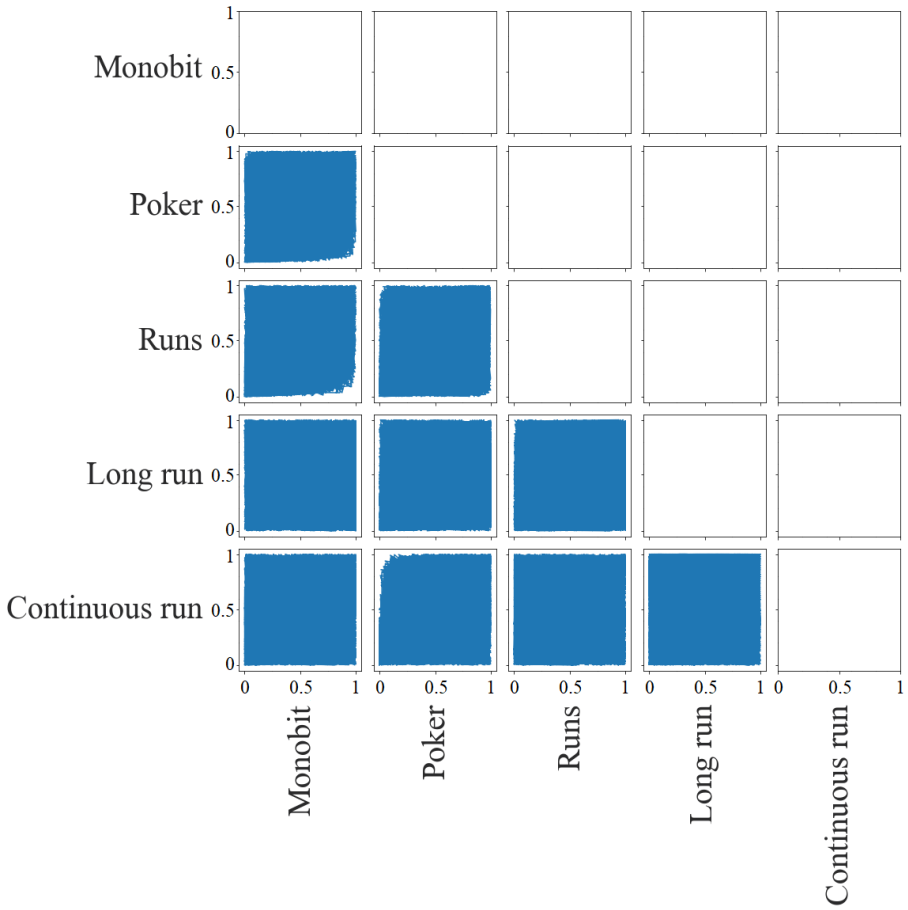

Figure 5. Dispersion matrix (*p*-values).

In addition, the study was developed on the values obtained for the statistics output by FIPS 140-2 tests for 100 random sequences. This approach was adopted to provide a more rigorous analysis than that provided by *p*-value comparisons alone. While *p*-values inform us of a test's result, given various input, and can allow one to judge whether two tests consistently report the same outputs, it is only a deeper analysis of the statistical basis for these *p*-values that allows us to determine if the correlation is due to test characteristics, or the influence of tested sequences. In Figure 6, the results with the Pearson's correlation applied to the statistics are represented. This matrix is very different from that of the *p*-values. Of the four most correlated pairs, only one remains (Runs with Poker) with a similar value (0.247889). Only one more pair has a prominent correlation: Continuous Run with Monobit (−0.124682). The rest of the correlations, in principle, do not seem to be very high (around 0.01).

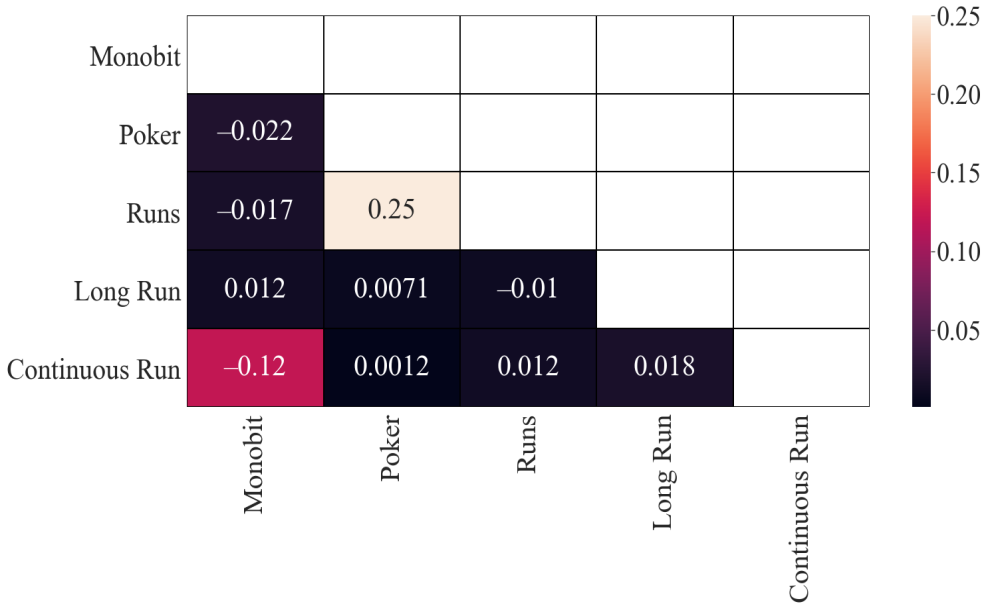

Figure 6. Pearson's correlation (statistics): results.

These results remain in the K-S test (Figure 7), where only these two pairings fail the test. The rest pass (although some were borderline, such as Poker with Monobit (0.0048)).

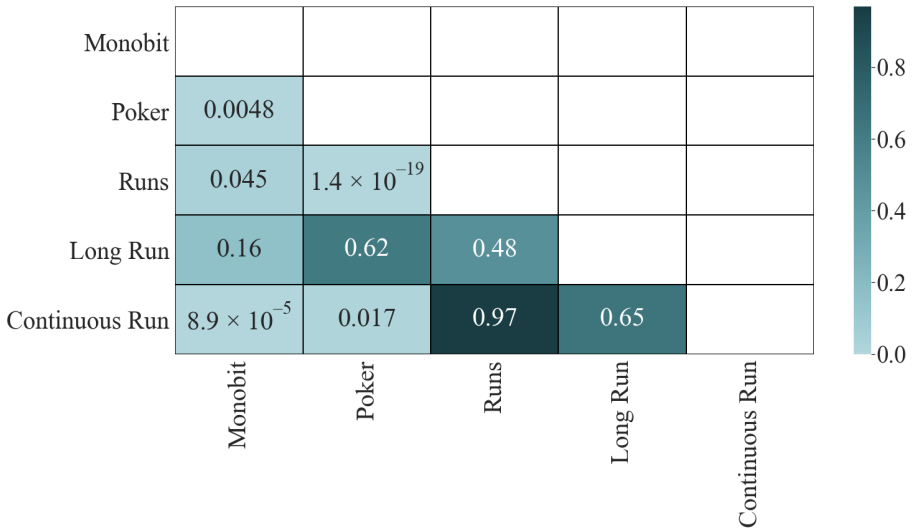

Figure 7. Pearson's correlation (statistics): K-S.

Results related to the mutual information measure can be seen in Figure 8. Unlike what we saw in Pearson's correlation, these results are more consistent with those obtained for the *p*-values. In general, the values obtained remain in the same line, distinguishing the three usual cases. Perhaps the most prominent variation is in the pair between Continuous Run and Monobit (from 0.001801 to 0.005572, three times more, but still low).

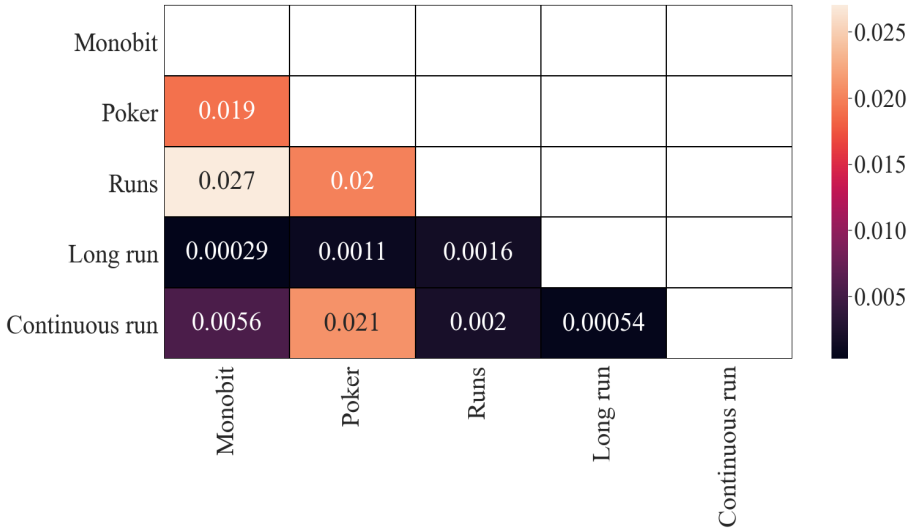

Figure 8. Mutual information (statistics): results.

The K-S matrix (Figure 9) corroborates these results. As with *p*-values, only two pairs pass the test. The main change concerning those results is that the pairs that do not pass the test have a higher *p*-value than before (for example, the four most correlated pairs go from ϵ to values around 10^{-70}), but they are still invalid. These results show us that mutual information is more resistant to changes and that it is capable of giving a general measure of independence between tests.

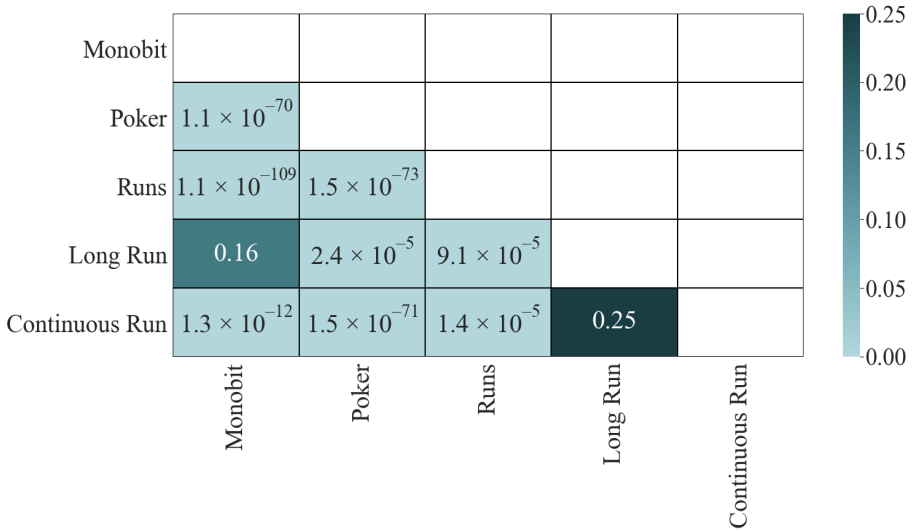

Figure 9. Mutual information (statistics): K-S.

Figure 10 provides the dispersion matrix for the statistics study. The most appreciable graphical dependencies are those between (i) Runs with Poker (they have a distribution with a deviation toward the lower right corner that is, they share low *p*-values); (ii) Poker and Runs with Monobit (their distribution has a deviation toward the left, so Monobit obtains low *p*-values when the others have *p*-values around the mean) and (iii) Continuous Run with Poker (its distribution has downward deviation, so Continuous Run obtains low *p*-values

when Poker has p-values around the mean). There is also a slight downward deviation, like the last case, between Long Run and Poker, Long Run and Runs, and Continuous Run and Runs.

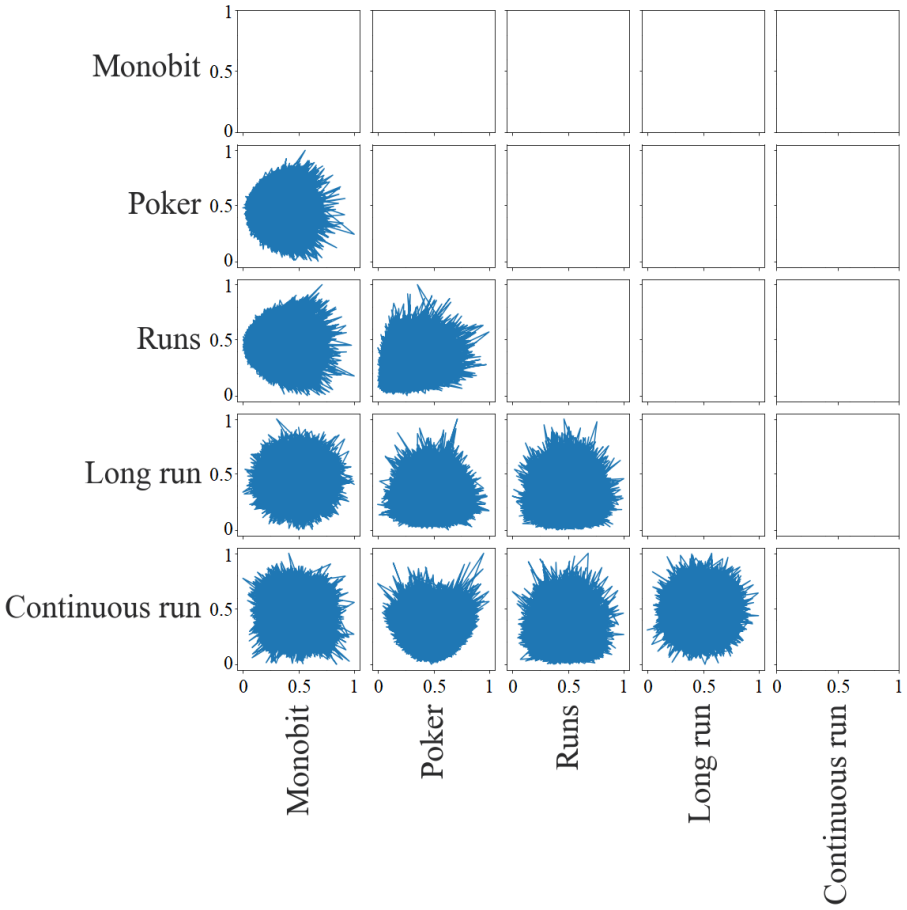

Figure 10. Dispersion matrix (statistics).

4. Conclusions

In this work, we carried out a study of the linear and non-linear dependencies between FIPS 140-2 battery tests. In order to carry out this analysis, it was necessary to re-implement the battery in such a way as to provide the user with the *p*-values and statistics resulting from the application of different hypothesis tests within the battery. The original tests (as implemented in the rngtools rngtest suite) only provide a Boolean pass or fail output, and so this re-implementation is vital for the success of this analytical work.

As for the analysis of *p*-values, we were able to verify that the results derived from the analysis using Pearson's correlations and mutual information are similar, although measured on different scales, with those relating to mutual information being more than 10 times lower than those of Pearson's correlation. This suggests that the existing relationships in this battery are fundamentally linear. With regard to the analysis of the statistics, we were able to verify that the mutual information measure is more resistant to changes and that it is capable of providing a general measure of independence between tests.

The most important interrelationships are between the Poker, Runs, and Monobit tests, with dependencies on each other. If required to select a single test (for the purposes of streamlining the test battery while retaining meaningful output), it would be Monobit, as Poker also has dependencies on Continuous Run, and the Runs test is somewhat more complex. In addition, of the three, it is the one who has the lowest correlation with Long Run.

We are left with three tests (Monobit, Long Run, and Continuous Run), but there is still a dependency (not as big as the first ones, but significant) between Continuous Run and Monobit. If the ultimate objective is the elimination of redundancies in a battery, some of these two tests should be eliminated, but we consider that more studies with the Continuous Run test should be done. Recall that this test is not the original one and depends on a parameter (block size). It is left as future work to test other versions of this test to see if any of them are independent of Monobit.

Author Contributions: Conceptualization, E.A.L., M.B.L.C., L.J.G.V. and J.H.-C.; methodology, E.A.L., M.B.L.C., L.J.G.V. and J.H.-C.; software, E.A.L., M.B.L.C., L.J.G.V. and J.H.-C.; validation, E.A.L., M.B.L.C., L.J.G.V. and J.H.-C.; formal analysis, E.A.L., M.B.L.C., L.J.G.V. and J.H.-C.; investigation, E.A.L., M.B.L.C., L.J.G.V. and J.H.-C.; data curation, E.A.L., M.B.L.C., L.J.G.V. and J.H.-C.; writing—original draft preparation, E.A.L., M.B.L.C., L.J.G.V. and J.H.-C.; writing—review and editing, E.A.L., M.B.L.C., L.J.G.V., D.H.-S. and J.H.-C.; visualization, E.A.L., L.G-V., M.B.L.C., D.H.-S. and J.H.-C. All authors have read and agreed to the published version of the manuscript.

Funding: This research work has received funding from UCM Projects THEIA (FEI-EU-19-04), THEIA I (FEI-EU-21-01) and THEIA II (FEI-22-01). This work has been partly funded by the EPSRC Quantum Communications Hub Project (EP/T001011/1). It is also supported by the InnovateUK funded AquRand project (106374-49229).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: We thank the reviewers for their comments, which have helped us to improve our work.

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

Appendix A. Code of the Re-Implementation of FIPS Battery

```
import numpy as np
import collections
from scipy import~stats

# Executes the Monobit test: it counts ones in the sequence
def monobit(u,n):
f = collections.Counter(u)
ones = f['1']
return (ones,stats.binom_test(ones, n))

# Divides the sequence into blocks and executes the Monobit test on each one
def monobit_blocks(u,n,m):
a = n//m
l = []
for i in range(a):
f = collections.Counter(u[i*m:(i+1)*m])
ones = f['1']
l.append(ones)
return stats.chisquare(l,a*[m/2])

# Executes the Poker test: it counts the number of occurrences of each
                                        possible 4-bit subsequence
def poker(u,n):
b = n//4
r = 4*b
mat = [int(u[start:start+4],2) for start in range(0, r, 4)]
oc_mat = sep(2,2,mat)
return chisquare(np.array(oc_mat))

# Returns a vector with the number of runs (consecutive 0s or 1s) of size 1,
                                        2, 3, 4, 5, +6 and + long
from itertools import~groupby

def runs_seq(u,n,long):
```

```
r = [(k, sum(1 for i in g)) for k,g in groupby(u)]
f = collections.Counter(r)
runs = [0 for i in range(13)]
cont = 0
l = len(r)
for i in range(1,6):
r1,r2 = f[('0',i)],f[('1',i)]
runs[2*i-2] = r1
runs[2*i-1] = r2
cont += r1 + r2
for i in range(6,long):
r1,r2 = f[('0',i)],f[('1',i)]
runs[10] += r1
runs[11] += r2
cont += r1 + r2
j = long
while(cont < l):
r1,r2 = f[('0',j)],f[('1',j)]
r3 = r1 + r2
runs[10] += r1
runs[11] += r2
runs[12] += r3
cont += r3
j += 1
return~runs

# Executes the Runs test: it counts the number of occurrences of runs (of 0s
                                        or 1s) of size k, with~k = 1, 2, 3, 4,
                                         5, +6 (6 and more)
long = 8
runs_freq = [1/4,1/4,1/8,1/8,1/16,1/16,1/32,1/32,1/64,1/64,1/64,1/64]

def runs(u,n,long):
runs = runs_seq(u,n,long)
s = sum(runs) - runs[12]
rf = [s*i for i in runs_freq]
return stats.chisquare(runs[:12],rf,1)

# Executes the Long Run test (modified): it counts the number of runs of size
                                         long or more
def long_run(u,n,long):
runs = runs_seq(u,n,long)
rl = runs[12]
s = sum(runs) - rl
return (rl,stats.binom_test(rl,s,1/(2**(long-1))))

# Divides the sequence into blocks and executes the Long Run test on each one
def long_run_blocks(u,n,m,long):
a = n//m
l = []
for i in range(a):
l.append(runs_seq(u[i*m:(i+1)*m],m,long)[12])
return stats.chisquare(l,a*[m/(2**long)])

# Executes the Long Run test (modified): it counts how many times two
                                        subsequences of size bits are equal
def cont_run(u,n,bits):
b = n//bits
r = bits*b
v = [u[start:start+bits] for start in range(0, r, bits)]
cont = 0
act = v[0]
for i in range(1,b):
if (v[i] == act):
cont += 1
v[i] = act
return (cont,stats.binom_test(cont, b-1, 1/(2**bits)))
```

```
# Divides the sequence into blocks and executes the Continuous test on each
                                        one
def cont_run_blocks(u,n,m,bits):
a = n//m
l = []
for i in range(a):
b = m//bits
r = bits*b
v = [u[start:start+bits] for start in range(i, i+r, bits)]
cont = 0
act = v[0]
for i in range(1,b):
if (v[i] == act):
cont += 1
v[i] = act
l.append(cont)
return stats.chisquare(l,a*[(b-1)/(2**bits)])

# Executes FIPS 140-2 battery tests in a sequence (faster than executing each
                                        test separately)
def fips(u,n,long):
runs_freq = [1/4,1/4,1/8,1/8,1/16,1/16,1/32,1/32,1/64,1/64,1/64,1/64]
aux = 1/(2**(long-1))
l = len(u)
b = n//4
r = 4*b
sp = [[] for i in range(l)]
for i in range(l):
# Monobit
f = collections.Counter(u[i])
ones = f['1']
sp[i].append((ones,stats.binom_test(ones, n)))
# Poker
mat = [int(u[i][start:start+4],2) for start in range(0, r, 4)]
oc_mat = sep(2,2,mat)
sp[i].append(stats.chisquare(np.array(oc_mat)))
# Runs
runs = runs_seq(u[i],n,long)
rl = runs[12]
s = sum(runs) - rl
rf = [s*i for i in runs_freq]
sp[i].append(stats.chisquare(runs[:12],rf,1))
# Long Run
sp[i].append((rl,stats.binom_test(rl,s,aux)))
# Continuous Run
if bits != 4:
mat = [int(u[i][start:start+bits],2) for start in range(0, r, bits)]
cont = 0
act = mat[0]
for j in range(1,b):
if (mat[j] == act):
cont += 1
mat[j] = act
sp[i].append((cont,stats.binom_test(cont, b-1, 1/16)))
return~sp

# Executes FIPS 140-2 battery tests in a set of sequences
def FIPS(u,n,long):
l = len(u)
f = fips(u,n,8)
s = [[] for i in range(l)]
p = [[] for i in range(l)]
for i in range(l):
(s[i], p[i]) = zip(*f[i])
return (s,p)
```

References

1. Figueroa-García, J.C.; Varón-Gaviria, C.A.; Barbosa-Fontecha, J.L. Fuzzy Random Variable Generation Using α-Cuts. *IEEE Trans. Fuzzy Syst.* **2021**, *29*, 539–548. [CrossRef]
2. Cotrina, G.; Peinado, A.; Ortiz, A. Gaussian Pseudorandom Number Generator Using Linear Feedback Shift Registers in Extended Fields. *Mathematics* **2021**, *9*, 556. [CrossRef]
3. Cogliatti, R.; de Souza, R.A.A.; Yacoub, M.D. Practical, Highly Efficient Algorithm for Generating κ-μ and η-μ Variates and a Near-100% Efficient Algorithm for Generating α-μ Variates. *IEEE Commun. Lett.* **2012**, *16*, 1768–1771. [CrossRef]
4. Hernández, J.A.; Sánchez, R.; Larrabeiti, D. Oversubscription Dimensioning of Next-Generation PONs With Different Service Levels. *IEEE Commun. Lett.* **2016**, *20*, 1341–1344. [CrossRef]
5. Rennó, V.M.; de Souza, R.A.A.; Yacoub, M.D. On the Generation of White Samples in Severe Fading Conditions. *IEEE Commun. Lett.* **2019**, *23*, 180–183. [CrossRef]
6. Wang, L.; Cheng, H. Pseudo-Random Number Generator Based on Logistic Chaotic System. *Entropy* **2019**, *21*, 960. [CrossRef]
7. Lee, K.; Lee, S.; Seo, C.; Yim, K. TRNG (True Random Number Generator) Method Using Visible Spectrum for Secure Communication on 5G Network. *IEEE Access* **2018**, *6*, 12838–12847. [CrossRef]
8. Xu, M.; Pan, W.; Yan, L.; Luo, B.; Zou, X.; Zhang, L.; Mu, P. An Explicit Non-Malleable Extraction Scheme for Quantum Randomness Amplification With Two Untrusted Devices. *IEEE Commun. Lett.* **2018**, *22*, 85–88. [CrossRef]
9. Sfeir, E.; Mitra, R.; Kaddoum, G.; Bhatia, V. RFF Based Detection for SCMA in Presence of PA Nonlinearity. *IEEE Commun. Lett.* **2020**, *24*, 2604–2608. [CrossRef]
10. Moysis, L.; Volos, C.; Jafari, S.; Munoz-Pacheco, J.M.; Kengne, J.; Rajagopal, K.; Stouboulos, I. Modification of the Logistic Map Using Fuzzy Numbers with Application to Pseudorandom Number Generation and Image Encryption. *Entropy* **2020**, *22*, 474. [CrossRef]
11. Lin, C.H.; Wu, J.X.; Chen, P.Y.; Li, C.M.; Pai, N.S.; Kuo, C.L. Symmetric Cryptography With a Chaotic Map and a Multilayer Machine Learning Network for Physiological Signal Infosecurity: Case Study in Electrocardiogram. *IEEE Access* **2021**, *9*, 26451–26467. [CrossRef]
12. Bassham, L.E.; 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*; Technical Report; National Institute of Standards & Technology: Gaithersburg, MD, USA, 2010.
13. L'ecuyer, P.; Simard, R. TestU01: AC library for empirical testing of random number generators. *ACM Trans. Math. Softw.* **2007**, *33*, 1–40. [CrossRef]
14. Brown, R.G.; Eddelbuettel, D.; Bauer, D. *Dieharder: A Random Number Test Suite (Version 3.31.1)*; Duke University Physics Department: Durham, NC, USA, 2014.
15. Walker, J. *ENT: A Pseudorandom Number Sequence Test Program*; 2008. Available online: https://www.fourmilab.ch/random/ (accessed on 15 March 2022).
16. FIPS PUB 140-2. Available online: https://csrc.nist.gov/publications/detail/fips/140/2/final (accessed on 15 March 2022).
17. Almaraz Luengo, E.; García Villalba, L.J. Recommendations on Statistical Randomness Test Batteries for Cryptographic Purposes. *ACM Comput. Surv.* **2021**, *54*, 1–34. [CrossRef]
18. Doğanaksoy, A.B.E.; Muş, K. Extended results for independence and sensitivity of NIST randomness tests. In Proceedings of the Information Security and Cryptography Conference, ISC Turkey, Ankara, Turkey, 25–27 December 2008; pp. 190–194.
19. Fan, L.; Chen, H.; Gao, S. A General Method to Evaluate the Correlation of Randomness Tests. In *Information Security Applications, WISA 2013, Lecture Notes in Computer Science, Jeju Island, Korea*; Springe: Cham, Switzerland, 2014; Volume 8267.
20. Sulak, F.; Uğuz, M.; Koçak, O.; Doğanaksoy, A. On the Independence of Statistical Randomness Tests Included in the NIST Test Suite. *Turk. J. Electr. Eng. Comput. Sci.* **2017**, *25*, 3673–3683. [CrossRef]
21. Hernandez-Castro, J.; Barrero, D.F. Evolutionary generation and degeneration of randomness to assess the indepedence of the Ent test battery. In Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain, 5–8 June 2017; pp. 1420–1427. [CrossRef]
22. 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]
23. Zhao, X.; Yang, S.; Shan, S.; Chen, X. Mutual Information Maximization for Effective Lip Reading. In Proceedings of the 2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020), Buenos Aires, Argentina, 16–20 November 2000; pp. 420–427. [CrossRef]
24. Sun, Y.; Yuan, P.; Sun, Y.; Zhai, Z. Hybrid Segmentation Algorithm for Medical Image Segmentation Based on Generating Adversarial Networks, Mutual Information and Multi-Scale Information. *IEEE Access* **2020**, *8*, 118957–118968. [CrossRef]
25. Ji, C.; Wang, J.; Zhang, G. Approximate Expression for the Mutual Information of Dense PAM. *IEEE Commun. Lett.* **2018**, *22*, 2182–2185. [CrossRef]
26. Ciganović, N.; Beaudry, N.J.; Renner, R. Smooth Max-Information as One-Shot Generalization for Mutual Information. *IEEE Trans. Inf. Theory* **2014**, *60*, 1573–1581. [CrossRef]
27. Maji, P. Mutual Information-Based Supervised Attribute Clustering for Microarray Sample Classification. *IEEE Trans. Knowl. Data Eng.* **2012**, *24*, 127–140. [CrossRef]
28. Kvålseth, T.O. On normalized mutual information: Measure derivations and properties. *Entropy* **2017**, *19*, 631. [CrossRef]

29. Hurley-Smith, D.; Patsakis, C.; Hernandez-Castro, J. On the unbearable lightness of FIPS 140-2 randomness tests. *IEEE Trans. Inf. Forensics Secur.* **2020**, 1. [CrossRef]
30. D'Agostino, R.B. (Ed.) *Goodness-of-Fit Techniques*; CRC Press: Boca Raton, FL, USA, 1986; Volume 68.
31. Mogull, R.G. Teacher's Corner: The One-Sample Runs Test: A Category of Exception. *J. Educ. Stat.* **1994**, *19*, 296–303. [CrossRef]