Journal of Cryptographic Engineering (2023) 13:235–249
https://doi.org/10.1007/s13389-023-00313-5

REGULAR PAPER

StringENT test suite: ENT battery revisited for efficient *P* value computation

Elena Almaraz Luengo[1] · Bittor Alaña Olivares[1] · Luis Javier García Villalba[1] · Julio Hernandez-Castro[2] · Darren Hurley-Smith[3]

Received: 14 March 2022 / Accepted: 25 February 2023 / Published online: 1 April 2023

Abstract
Random numbers play a key role in a wide variety of applications, ranging from mathematical simulation to cryptography. Generating random or pseudo-random numbers is not an easy task, especially when hardware, time and energy constraints are considered. In order to assess whether generators behave in a random fashion, there are several statistical test batteries. ENT is one of the simplest and most popular, at least in part due to its efficacy and speed. Nonetheless, only one of the tests of this suite provides a *p* value, which is the most useful and standard way to determine whether the randomness hypothesis holds, for a certain significance level. As a consequence of this, rather arbitrary and at times misleading bounds are set in order to decide which intervals are acceptable for its results. This paper introduces an extension of the battery, named StringENT, which, while sticking to the fast speed that makes ENT popular and useful, still succeeds in providing *p* values with which sound decisions can be made about the randomness of a sequence. It also highlights a flagrant randomness flaw that the classical ENT battery is not capable of detecting but the new StringENT notices, and introduces two additional tests.

Keywords Cryptography · ENT · Hypothesis testing · Pseudo-random number generator (PRNG) · *P* values · Statistical tests suite · True random number generator (TRNG)

1 Introduction

Random numbers are ubiquitous, and their use is widespread in technology. They have a myriad of applications: mathematical simulation and generation [1–6], initialization of encryption algorithms [7–12], key generation [13–15], video games [16,17], etc. Very often, even for cryptography use, we seek to generate random numbers with very limited hardware (and sometimes energy) resources available, in constrained devices such as RFID tags or Internet of Things chips. In order to meet these demands, True Random Number Generators (TRNGs) and Pseudo-Random Number Generators (PRNGs) are used.

1. *True random number generators* use what we call entropy sources, which are sources of intrinsic natural randomness, in order to draw random bits. Examples of such natural phenomena include radioactive decay, thermal noise, clock jitter, meteorological changes... However, extracting this sort of information is usually rather costly, and hence the speed of such generators cannot generally match the demand of bitrate that is often required in many applications. Moreover, usually the data has to be processed through a so-called whitening process to get rid of biases.
2. *Pseudo-random number generators* produce a seemingly random output, which is usually obtained through deterministic methods. They are initially fed with a seed,

✉ Luis Javier García Villalba
javiergv@fdi.ucm.es

Elena Almaraz Luengo
ealmaraz@ucm.es

Bittor Alaña Olivares
balana@ucm.es

Julio Hernandez-Castro
jch27@kent.ac.uk

Darren Hurley-Smith
Darren.Hurley-Smith@rhul.ac.uk

[1] Group of Analysis, Security and Systems (GASS), Universidad Complutense de Madrid, Madrid, Spain

[2] School of Computing, University of Kent, Canterbury, UK

[3] Information Security Group, Royal Holloway, University of London, London, UK

which ideally emanates from a TRNG or some high entropy source.

There are numerous proposals that implement lightweight PRNGs and TRNGs, so that they can be fit in small embedded systems. Since these random numbers are frequently used for security purposes, it is extremely important that they actually behave as randomly as possible and that in particular there is no way to predict their output by just looking at previously generated numbers.

Hence, randomness tests play a major role in assessing and designing new random number generator proposals.

On this paper, we will treat number sequences one byte at a time. Therefore, for an n-byte sequence $x = x_0 x_1 \dots x_{n-1}$, to be considered a random enough sequence we expect that $x_0, x_1, \dots, x_{n-1}$ to be indistinguishable from independent, identically distributed samples (i.i.d.) from a discrete uniform distribution $\mathcal{U}\{0, 1, \dots, 255\}$. If we instead worked with the sequence at the bit level, the equivalent definition would be that their output bits cannot be efficiently and consistently distinguished with a given test battery from i.i.d. samples coming from a $Bernoulli(\frac{1}{2})$ distribution.

This paper is organized as follows: in Sect. 2, we describe the theoretical foundation on which our work is based. In Sect. 3, we analyse the (in)dependence of the tests under study. In Sect. 4, we describe a degenerate generator we have designed that is able to fool the ENT battery, thus highlighting its limitations. In Sect. 5, we carry out a detailed study of the statistics underlying the ENT tests and define the p values associated with them. In Sect. 6, we describe two tests that we will incorporate into the new battery to extend its analysis capacity and address the problem detected in Sect. 4. In Sects. 7 and 8, we focus, respectively, on the measurement of computational time and existing correlations. Finally, in Sect. 9, we draw the main conclusions of our work and point out future research lines.

2 Randomness tests

We have established what we expect from a random sequence. Therefore, we now have a hypothesis that we can try and contrast with a given byte sequence. This can be done through hypothesis testing, by computing statistics from which we derive p values. Suppose we have a sequence $x = x_0 \dots x_{n-1}$, with each $x_i \in \{0, 1, \dots, 255\}$.

2.1 Statistics

A statistic is a function of the sequence x, $f(x)$. It can represent any kind of operation, for instance, the sequence mean, the number of zeros that x has in binary representation, etc. If we can derive the distribution of the statistic under the assumption that x is random, this enables us to perform hypothesis testing.

2.2 Hypothesis testing

Hypothesis testing is a statistical inference procedure that allows us to check whether a certain hypothesis holds. We call the hypothesis we want to test the null hypothesis, H_0. In our case, the null hypothesis is that a given sequence is random. The alternative hypothesis is the opposite of the null hypothesis, and is denoted H_1. In our case, this means the sequence is not random.

A statistic whose distribution under the null hypothesis is known is computed. Since the distribution is known, we can compute the probability of obtaining statistics as extreme or more extreme than the ones observed. This is called a p value.

We fix a number α, usually between 0.001 and 0.05, and use it as a discriminating threshold. If the observed p value is smaller than α, we deem the statistic to be too extremely unlikely, and so we reject the null hypothesis. Otherwise, we do not reject the null hypothesis.

Therefore, even when the null hypothesis is true, there is a probability α that we will reject the null hypothesis. This error is referred to as a Type I error. On the other hand, in the scenario where the null hypothesis does not hold, but we fail to obtain a p value smaller than α and hence we do not reject the null hypothesis, we incur in a Type II error.

$$\alpha = P(\text{Reject } H_0 \mid H_0 \text{ is true})$$
$$\beta = P(\text{Fail to reject } H_0 \mid H_0 \text{ is not true})$$

It is important to note that statistical tests are necessary but will never be sufficient to completely evaluate the quality of a generator. Designers and practitioners should never put blind faith in their results, as only negative results are set in stone. It is important to constantly keep in mind that many poor and fully deterministic random number generators can fool most if not all randomness tests easily, a case in point been AES in counter mode.

That said, there are many randomness test suites which compute various useful statistics in order to assess whether a sequence *looks* random. To name a few, we can highlight the following (for more details about batteries see [18]):

1. ENT (for Entropy) [19]. It is made of five simple tests, which are extremely fast. Only one of the tests outputs an approximate p value, and the rest just produce statistics that can yield useful information with respect to the apparent randomness of the sequence.
2. NIST SP 800-22 [20]. It is one of the most widely used due to it being backed by the *National Institute of Stan-*

dards and Technology. It is a quite demanding suite with 15 tests, and a relatively high computing cost, specially compared to the former suite.
3. TESTU01 [21]. It treats the sequence as a collection of floating point numbers in [0, 1). It has three sets of tests, composed of 10, 96 and 100 tests each.
4. Dieharder [22]. This suite is built upon the famous suite Diehard, and adds a number of tests from other sources as well.

We will focus on ENT because it is very widely used due to its speed. However, we will prove that this speed comes with some caveats.

2.3 ENT

The ENT test suite [19] consists of the following statistics: entropy, ideal compression rate, Chi-square test, arithmetic mean, Monte Carlo estimation of π and serial correlation.

The Chi-square test is the only one which outputs an approximate p value, which helps determine whether the sequence is significantly non-random. We will now explain what the tests compute.

2.3.1 Entropy

This test finds the entropy as classically defined in Information Theory [23]. We denote the sequence $X = \{x_k\}_{k=0}^{N-1}$. The number of occurrences of values i between 0 and 255 is counted. We then denote $P_X(i) = \frac{Ap_i}{n}$, where $Ap_i = \#\{0 \leq k \leq n-1 : x_k = i\}$. The entropy of X is then computed by the following formula [23]:

$$H(X) = -\sum_{i=0}^{i=255} P_X(i) \log_2 P_X(i). \tag{1}$$

Intuitively, the entropy conveys the amount of bits per byte of in-compressible information the sequence holds. Therefore, it yields a value between 0 and 8. A value of 0 means the sequence contains no information, and 8 means the sequence is in-compressible, at the byte level. The ideal compression percentage is also given, which is computed as:

$$100 \cdot \frac{8 - \text{entropy}}{8}.$$

Therefore, it is really important to bear in mind that this is simply one additional visualization of the entropy's meaning, and does not actually constitute a different independent test.

Note that perfect entropy is achieved when all values i have $P_X(x_i) = 2^{-8}$, as one can see that Eq. (1) is then equivalent to 8.

2.3.2 Chi-square test

The Chi-square test measures the goodness of fit for the observed number of occurrences with respect to a uniform distribution. We draw a statistic known as X^2, computed in the following fashion:

$$X^2 = \sum_{0 \leq i \leq 255} \frac{(Ap_i - EAp_i)^2}{EAp_i} \tag{2}$$

where Ap_i again denotes the occurrences of value $i \in \{0, 1, \ldots, 255\}$ in the sequence, $Ap_i = \#\{0 \leq k \leq n-1 : x_k = i\}$, and EAp_i are the expected appearances under the perfect randomness hypothesis, that is $\frac{n}{256}$.

After computing this value, the test displays an approximate p value in the form of a percentage. That is, it shows what the probability is (under the randomness hypothesis) to get a statistic X^2 as the one observed. That is, the probability of getting a worse fit. This p value is given by

$$p - \text{value} = P(\chi^2_{255} \geq X^2),$$

where χ^2_{255} is the Chi-square distribution with 255 degrees of freedom. Usually, as a rule of thumb, we require that the output size be large enough to expect at least around 5 appearances per bin. Otherwise, the distribution of X^2 might not be close enough to the χ^2_{255} distribution.

This test is also focused on measuring uniformity between the values $\{0, 1, \ldots, 255\}$ in the sequence.

2.3.3 Arithmetic mean

Since we interpret bytes as integers between 0 and 255, this is simply the arithmetic mean of all observed bytes, which has an expected value of 127.5 under randomness hypothesis. We will write

$$\bar{x} = \frac{1}{n} \sum_{i=0}^{n-1} x_i. \tag{3}$$

2.3.4 Monte Carlo Pi estimation

In order to perform this test, coordinates are generated in the rectangle $[0, 2^{24} - 1]$. In order to do so, 3 bytes form the value of x and another 3 characterize y. Since 3 bytes are 24 bits, these represent unsigned integer values from 0 to $2^{24} - 1$. Once a pair (x, y) is generated, we check whether it is inside the closed Euclidean ball of centre $(0, 0)$ and radius $r = 2^{24} - 1$. This is true if and only if $x^2 + y^2 \leq r^2$.

Then the number of generated points within the circle is counted. The rate between the points within the circle area

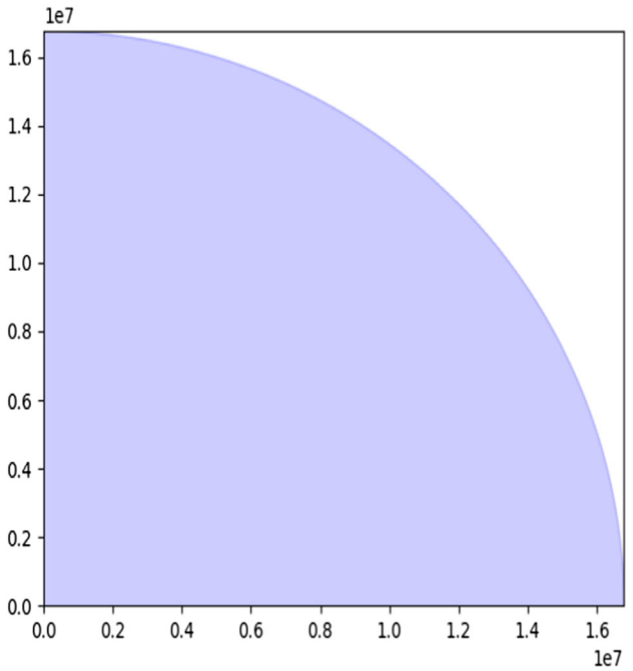

Fig. 1 Area of quarter of circle compared to rectangle $[0, r]^2$

and the total number of points approximates the ratio between the area of a quarter of a circle and the rectangle's area. This is given by:

$$\frac{\text{points within circle}}{\text{total points}} \simeq \frac{\text{circle part's area}}{\text{rectangle's area}} = \frac{\pi \cdot r^2/4}{r^2} = \pi/4. \tag{4}$$

So the π estimation is given by $\tilde{\pi} = 4 \cdot \frac{\text{points within circle}}{\text{total points}} = 4 \cdot \frac{pts_{in}}{pts}$. The error percentage is also shown (Fig. 1).

2.3.5 Serial correlation

This test measures how one byte depends on the previous byte. Alternatively, one can see it as a way of measuring the predicting power of the last output byte over the following one. It counts the appearances of each pair of bytes and returns a value between -1 and 1, with 0 meaning no linear relation between bytes, and an absolute value of 1 being complete linear dependence. This coefficient is computed as the Pearson's r correlation coefficient between the sequence $x = x_0 \ldots x_{n-1}$ and the sequence shifted a byte "to the left" $x' = x_1 \ldots x_{n-1}x_0$.

The Pearson's correlation coefficient r between two samples x and y is given by the following expression:

$$r = \frac{\sum_{i=0}^{n-1}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=0}^{n-1}(x_i - \bar{x})^2}\sqrt{\sum_{i=0}^{n-1}(y_i - \bar{y})^2}}. \tag{5}$$

Entropy = 7.787095 bits per byte.
Optimum compression would reduce the size of this 11619 byte file by 2 percent.
Chi square distribution for 11619 samples is 6567.22, and randomly would exceed this value less than 0.01 percent of the times.
Arithmetic mean value of data bytes is 110.0003 (127.5 = random).
Monte Carlo value for Pi is 3.392561983 (error 7.99 percent).
Serial correlation coefficient is 0.098324 (totally uncorrelated = 0.0).

Fig. 2 Results of applying tests to file *entitle.gif* of the battery

which in our case means

$$\begin{aligned} r &= \frac{\sum_{i=0}^{n-1}(x_i - \bar{x})(x'_i - \bar{x})}{\sum_{i=0}^{n-1}(x_i - \bar{x})^2} \\ &= \frac{\sum_{i=0}^{n-1} x_i x'_i - n\bar{x}^2 - n\bar{x}^2 + n\bar{x}^2}{\sum_{i=0}^{n-1}(x_i^2 - 2x_i\bar{x} + \bar{x}^2)} \\ &= \frac{\sum_{i=0}^{n-1} x_i x_{(i+1) \bmod n} - n\bar{x}^2}{\sum_{i=0}^{n-1} x_i^2 - n\bar{x}^2}. \end{aligned}$$

This is the only test in the original ENT suite that tries, to a certain extend, to measure independence as opposed to the others which are concerned exclusively with uniformity.

Unfortunately only one of these tests produces an approximate p value. This means the interpretation of the rest of statistics is not trivial. We will address this issue in Sect. 5. In Fig. 2, we show the result we get from applying the classical ENT test to a *gif* file included with the source code.

3 Independence of tests

We studied the tests in ENT with good-quality pseudo-random data, by analysing 10,000 files of size 15 MB, drawn from Unix's PRNG */dev/urandom*. Firstly, we run the suite to compute the resulting 50,000 statistics. We show a histogram for the data in Fig. 3. One of the first things we can notice is that the entropy takes very few different values for these random-like data. More precision for this output might be desirable.

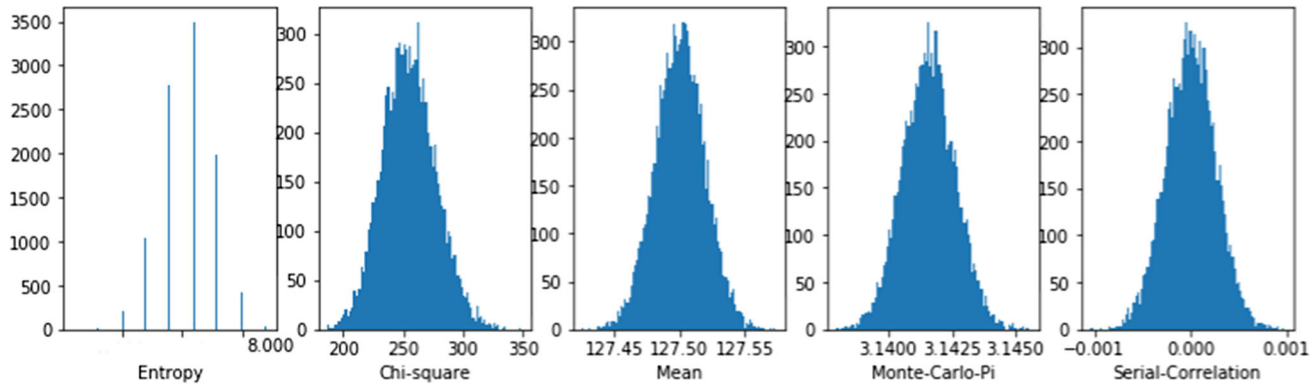

Fig. 3 Histograms for the */dev/urandom* data

Table 1 *P* values of normality tests performed on two sets totalling 10, 000 15 MB files from */dev/urandom*

	Mean	MC π	SC
S1: Shapiro–Wilk p value	0.4267	0.0480	0.6155
S2: Shapiro–Wilk p value	0.5221	0.6927	0.7666
S1: Anderson–Darling p value	0.2896	0.1589	0.4614
S2: Anderson–Darling p value	0.5157	0.7258	0.8955

Another interesting fact we could notice in Fig. 3 is that the Mean, Monte Carlo π and Serial Correlation statistics seem to be normally distributed. In order to check that this is indeed the case, we run the Shapiro–Wilk test and the Anderson–Darling test. Either test returns a p value. We divide the 10, 000 files in two groups of 5, 000 and perform a normality test in both sets (S1 and S2 respectively).

Table 1 confirms that indeed the Mean, Monte Carlo π and Serial Correlation statistics are normally distributed. This will be very helpful in developing p values for these statistics later in Sect. 5.

A quick look at the definition of the tests can hint that the Chi-square test and the Entropy test might be closely related since they both aim to measure the uniformity of bytes in the sequence. First, we graph a scatter plot of the statistics obtained for our data. See Fig. 4.

Figure 4 shows that the data for the entropy and the Chi-square tests lie almost exactly on a line. In order to have a closer look at this, we compute the Pearson's r correlation coefficient, as defined in Eq. 5, between these two statistics. This is shown in Table 2.

Table 2 shows there is an almost perfect negative correlation between the entropy and the Chi-square statistic. This is rather intuitive since an even allocation of numbers 0–255 in the sequence that leads to high entropy also leads to a lower X^2 value for the goodness-of-fit statistic. Nonetheless, we have to beware of the fact that the entropy took very few different values. A linear regression model showed that residuals were not normal, and so a linear expression cannot be extrapolated that will work for both random and non-random data. Nonetheless, it is clear that the Chi-square test and the Entropy test are highly correlated. Note that the π estimation and Mean test also have a remarkable correlation.

In Table 2, we have coloured each correlation coefficient for which their corresponding p value under the independence hypothesis was smaller than 0.01. This means the compared tests are very likely not independent.

4 Modified counter with good results

Since most of ENT's tests only focus on the uniformity of the sequence or source under analysis, there might be many easily predictable sequences capable of obtaining good results in the battery. For instance, a counter with values from 0 to $2^{16} - 1$ where two bytes are output for each counter value will seemingly have perfect entropy, very low error for Monte Carlo π estimation, perfect arithmetic mean, and no serial correlation. We show the results in Fig. 5 for 16 full loops of this counter.

However, we can notice in Fig. 5 that the X^2 statistic is 0.00 which is **too good** a fit. By modifying this counter a bit, we can get a more "random looking" X^2 statistic but still very good randomness results, even if the sequence still remains essentially a counter.

In order to do so, first we express the 0 to $2^{16} - 1$ counter over two bytes as pairs (i, j) with $i, j \in \{0, 1, ..., 255\}$. That is, i is the most significant or left byte of the counter and j the least significant or right one. Now, we are going to change some appearances of the value 250 by appearances of 251, as shown in Algorithm 1.

Note that in this pseudo-counter, all byte values from 0 to 255 will have precisely the number of expected appearances $\frac{n}{256}$ except for 250 and 251.

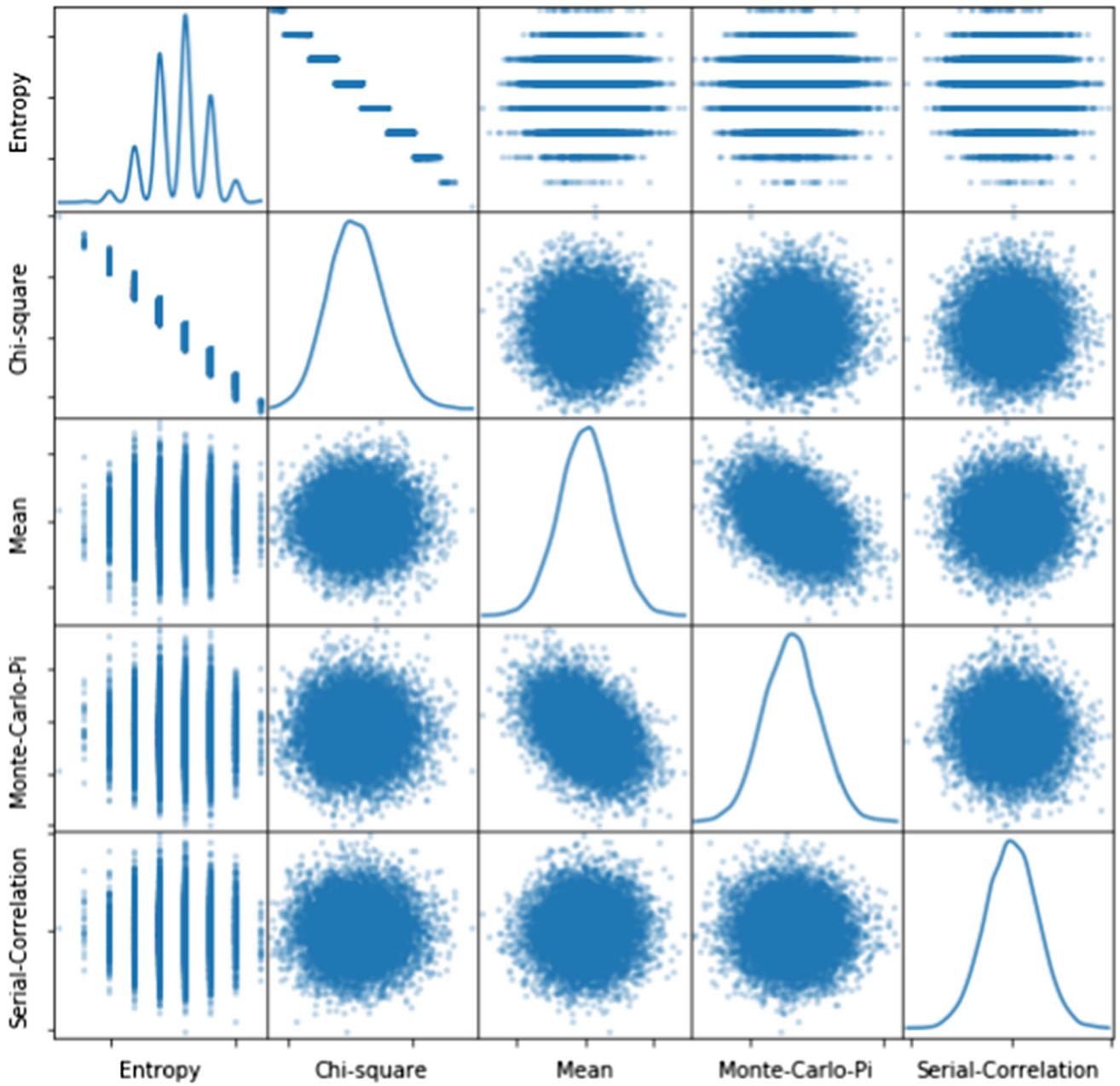

Fig. 4 Scatter matrix for statistics obtained for */dev/urandom*

Table 2 Correlation matrix for ENT statistics of */dev/urandom*

	Entropy	χ^2	Mean	MC π	SC
Entropy	1.000	**−0.966**	−0.003	0.002	−0.006
χ^2	**−0.966**	1.000	0.007	−0.006	0.008
Mean	−0.003	0.007	1.000	**−0.424**	0.022
MC π	0.002	−0.006	**−0.424**	1.000	**−0.033**
SC	−0.006	0.008	0.022	**−0.033**	1.000

The change will be performed every time $i \mod k = 0$, and so this will result in $L \cdot 256/k$ more appearances than expected for 251. The value 250 will be that same amount short from the expected amount. Therefore, the X^2 statistic of the sequence will be

$$X^2 = \frac{(n/256 + L \cdot 256/k - n/256)^2}{n/256} + \frac{(n/256 - L \cdot 256/k - n/256)^2}{n/256} = 512\frac{L^2 256^2}{k^2 n}. \tag{6}$$

Algorithm 1 Pseudo-code of modified k, L-counter.

```
for 1 ≤ l ≤ L do
    for 0 ≤ i ≤ 255 do
        for 0 ≤ j ≤ 255 do
            if j = 250 and i mod k = 0 then
                output ← i
                output ← 251
            else
                output ← i
                output ← j
            end if
        end for
    end for
end for
```

```
Entropy = 8.000000 bits per byte.
Optimum compression would reduce the size
of this 2097152 byte file by 0 percent.
Chi square distribution for 2097152 samples is
0.00, and randomly would exceed this value
more than than 99.99 percent of the times.
Arithmetic mean value of data bytes is
127.5000 (127.5 = random).
Monte Carlo value for Pi is 3.140982762 (error
0.02 percent).
Serial correlation coefficient is 0.000000
(totally uncorrelated = 0.0).
```

Fig. 5 Results for a 16-loop 2-byte counter in the range 0 to $2^{16}-1$

Now, considering that the length of the file processed by Algorithm 1 is $n = L \cdot 2 \cdot 256^2$ (which is obvious by multiplying the range of the loops and considering that every inner iteration outputs two bytes) it is easy to see from Eq. 6 that it suffices to take L and k such that $k^2 = L$ in order to get a statistic of exactly 256, which is less suspicious that the 0.00 one we obtained before and much closer to the expected value (of ~ 255.3 for 255 degrees of freedom).

That is, if $k^2 = L$ we get

$$512\frac{L^2 256^2}{k^2 n} = 512\frac{L^2 256^2}{k^2 L \cdot 2 \cdot 256^2} = 512/2 = 256.$$

For instance, we can take $L = 16$, to loop 16 times as we did for the initial counter, and $k = 4$. This yields the results shown in Fig. 6.

Therefore, we have obtained impeccable results for the battery using our modified counter. Figures 7 and 8 show two representations that should make abundantly clear how non-random this source is, despite its excellent results with ENT.

```
Entropy = 7.999912 bits per byte.
Optimum compression would reduce the size
of this 2097152 byte file by 0 percent.
Chi square distribution for 2097152 samples is
256.00, and randomly would exceed this value
47.06 percent of the times.
Arithmetic mean value of data bytes is
127.5005 (127.5 = random).
Monte Carlo value for Pi is 3.140925542 (error
0.02 percent).
Serial correlation coefficient is -0.000000
(totally uncorrelated = 0.0).
```

Fig. 6 Results for modified counter with $k^2 = L = 16$

```
0000 0001 0002 0003 0004 0005 0006 0007
0008 0009 000a 000b 000c 000d 000e 000f
0010 0011 0012 0013 0014 0015 0016 0017
0018 0019 001a 001b 001c 001d 001e 001f
0020 0021 0022 0023 0024 0025 0026 0027
0028 0029 002a 002b 002c 002d 002e 002f
0030 0031 0032 0033 0034 0035 0036 0037
0038 0039 003a 003b 003c 003d 003e 003f
0040 0041 0042 0043 0044 0045 0046 0047
0048 0049 004a 004b 004c 004d 004e 004f
0050 0051 0052 0053 0054 0055 0056 0057
0058 0059 005a 005b 005c 005d 005e 005f
0060 0061 0062 0063 0064 0065 0066 0067
0068 0069 006a 006b 006c 006d 006e 006f
0070 0071 0072 0073 0074 0075 0076 0077
0078 0079 007a 007b 007c 007d 007e 007f
0080 0081 0082 0083 0084 0085 0086 0087
0088 0089 008a 008b 008c 008d 008e 008f
0090 0091 0092 0093 0094 0095 0096 0097
0098 0099 009a 009b 009c 009d 009e 009f
00a0 00a1 00a2 00a3 00a4 00a5 00a6 00a7
00a8 00a9 00aa 00ab 00ac 00ad 00ae 00af
00b0 00b1 00b2 00b3 00b4 00b5 00b6 00b7
00b8 00b9 00ba 00bb 00bc 00bd 00be 00bf
00c0 00c1 00c2 00c3 00c4 00c5 00c6 00c7
00c8 00c9 00ca 00cb 00cc 00cd 00ce 00cf
00d0 00d1 00d2 00d3 00d4 00d5 00d6 00d7
00d8 00d9 00da 00db 00dc 00dd 00de 00df
00e0 00e1 00e2 00e3 00e4 00e5 00e6 00e7
00e8 00e9 00ea 00eb 00ec 00ed 00ee 00ef
00f0 00f1 00f2 00f3 00f4 00f5 00f6 00f7
00f8 00f9 00fb 00fb 00fc 00fd 00fe 00ff
```

Fig. 7 First 512 bytes of modified counter

5 Calculating *p* values from existing statistics

One of the most obvious improvements the battery could use is offering p values from the statistics that do not provide one. This would allow us to perform proper hypothesis testing, in order to find statistically sound results from which we can conclude whether the hypothesis test can be rejected. For instance, say we have a sequence with arithmetic mean

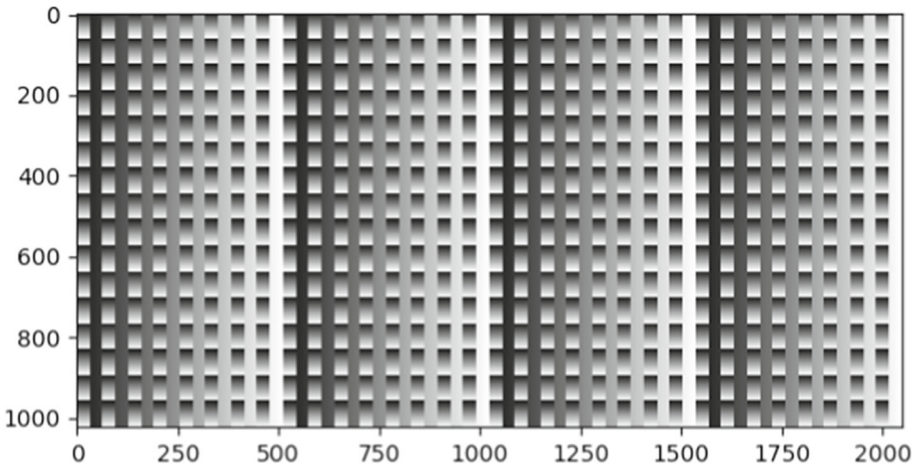

Fig. 8 Greyscale representation of modified counter

127.9523. How do we know if this is too far from 127.5? That will depend on the sequence length. That approximation might be excellent for a 50-byte sequence, but extremely poor for a 20 MB file.

However, there are some derivations by which we can provide a p value, which will tell us how likely it would be to obtain results "more extreme" than the ones observed if the sequence were truly random.

5.1 Arithmetic mean

The arithmetic mean is arguably the most simple statistic in the battery. For a sequence $x = x_0 ... x_{n-1}$, we get the statistic $\bar{x} = \frac{1}{n}\sum_{i=0}^{n-1} x_i$. Under the randomness hypothesis, x_i are i.i.d. samples from a discrete uniform distribution $Y \simeq \mathcal{U}\{0, 255\}$. Therefore, we can apply the Central Limit and compare our results to a standard normal distribution $\mathcal{N}(0, 1)$.

If each x_i for $i \in \{0, 1, \ldots, n-1\}$ is an independent sample from Y, the Central Limit Theorem states that the following holds:

$$\sqrt{n}\frac{\bar{x} - \mu}{\sigma} \longrightarrow \mathcal{N}(0, 1) \quad \text{as } n \to \infty. \tag{7}$$

It is straightforward to find μ and σ. Since Y is a discrete uniform distribution between 0 and 255, we get:

$$\begin{aligned}
\mathbb{E}[Y] &= \sum_{i=0}^{255} i \cdot P_Y(i) = \frac{1}{256}\sum_{i=0}^{255} i = 127.5 \\
\mathbb{E}[Y^2] &= \sum_{i=0}^{255} i^2 \cdot P_Y(i) = \frac{1}{256}\sum_{i=0}^{255} i^2 = \frac{255 \cdot (2 \cdot 255 + 1)}{6} \\
\sigma^2 &= \mathbb{E}[Y^2] - \mathbb{E}[Y]^2 = \frac{256^2 - 1}{12} \Rightarrow \sigma = \sqrt{\frac{256^2 - 1}{12}}
\end{aligned} \tag{8}$$

Therefore we can obtain a two-tailed p value. First we compute the following z statistic:

$$z = \sqrt{n}\frac{\bar{x} - \mu}{\sigma}. \tag{9}$$

Under the randomness hypothesis, z will approximately follow a standard normal distribution. z will be near 0 if the mean was near 127.5. Therefore, "more extreme" results than z are those that have a bigger absolute value than $|z|$. Now if we denote by Φ the cumulative standard normal distribution, we get the p value from the following equation, which resembles the two-tailed p values from [20]:

$$p = P(\mid Z \mid \geq \mid z \mid) = 2 \cdot (1 - \Phi(\mid z \mid)). \tag{10}$$

Since the original ENT code has a numerical approximation of Φ from which it approximates the value of χ^2_{255}, we can perform this computation without adding too much code, and thus turn the arithmetic mean into a more useful and insightful test.

5.2 Serial correlation

For the serial correlation, the same problem holds. It's hard to tell what constitutes a critical boundary for serial correlation, and how to factor in the sequence length.

If two sequences were independent, their serial correlation should approximate zero. In the case of a random sequence, we would expect the correlation between the sequence and the sequence shifted one byte to be near zero. If this value was too big, it would indicate the generator is predictable.

In order to obtain a p value from the Pearson's correlation coefficient between two sets of samples $x = x_0 \ldots x_{n-1}$ and $y = y_0 \ldots y_{n-1}$, we use the statistic

$$t = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}}, \tag{11}$$

which follows a Student t-distribution with $n-2$ degrees of freedom when the samples come from a bivariate normal distribution. However, this requirement is usually dismissed when the number of samples is large enough. Nonetheless, there is always a caveat with this omission, and this can lead to issues with small sample sizes [24].

Since the sequences analysed by StringENT will typically consist of at least thousands of bytes, this approximation will be good enough. We will compute statistic t and find a two-tailed p value from Student's t-distribution with $n-2$ degrees of freedom, n being the number of bytes of the sequence. We will denote such distribution T. Since T is symmetric, this p value's computation is analogous to the one in Eq. (10):

$$p = P(\mid T \mid \geq \mid t \mid). \tag{12}$$

Nonetheless, the code in the ENT battery does not come with any tables of Student's t-distribution. Therefore, in order to avoid adding too much new code to the battery, we have made a compromise in accuracy vs. code length and used an approximation of the t-distribution by the normal distribution, which yields three decimal digits' precision. The formula is given by [25,26]

$$P(T \leq t) \simeq \Phi\left(t\left(1-\frac{1}{4n}\right)\left(1+\frac{1}{2n}t^2\right)^{\frac{-1}{2}}\right) \tag{13}$$

Moreover, since the t-distribution converges to the normal distribution as the degrees of freedom grow, and we will typically use it with very large values for $df = n-2$, the accuracy will be reasonably good. By joining Eqs. (12) and (13), our two-tailed p value will be given by:

$$p = 2 \cdot \left(1 - \Phi\left(|t|\left(1-\frac{1}{4n}\right)\left(1+\frac{1}{2n}t^2\right)^{\frac{-1}{2}}\right)\right). \tag{14}$$

5.3 Monte Carlo Pi estimation

We look at Eq. (4). Given a coordinate $(x, y) \in [0, 2^{24}-1]^2$ at random, if x and y are independently and uniformly distributed according to $\mathcal{U}(0, 2^{24}-1)$, the probability that (x, y) lies within the circle is given by $\pi/4$, as is easy to deduce from Fig. 1. Therefore, in our case, in which x and y will be, under randomness hypothesis, i.i.d. samples from $\mathcal{U}\{0, 2^{24}-1\}$, we get that the probability that (x, y) lies within the circle should be around $\pi/4$.

Therefore, we can model the experiment "draw a coordinate and see whether it lies within the circle" as a $Bernoulli(\pi/4)$ distribution. That is, an experiment with probability $\pi/4$ of success, and $1-\pi/4$ of failure.

Hence, since under randomness hypothesis each coordinate is independent from all others, the number of coordinates that lie within the circle follows a Binomial distribution with $p = \pi/4$ and $n =$ number of points, $Binomial(\pi/4, n)$.

A particular case of the Central Limit Theorem, known as the De Moivre–Laplace Theorem [27], states that if B_n follows a $Binomial(p, n)$ distribution, and $a, b \in \mathbb{R} \cup \{\pm\infty\}$, $a < b$, we get

$$\lim_{n\to\infty} P\left(a \leq \frac{B_n - np}{\sqrt{np(1-p)}} \leq b\right) = \frac{1}{\sqrt{2\pi}}\int_a^b e^{-\frac{x^2}{2}}dx. \tag{15}$$

Therefore, the number of points that will fall within the circle can lead us to an approximately standard normal distribution.

This allows us to find a p value. Denoting by $\tilde{\pi}$ the approximation of π obtained through the observed sequence, and using Eq. (4), we get that, if we refer to the number of coordinates in the sequence as pts, where $pts = \left\lfloor \frac{n_{bytes}}{6} \right\rfloor$, the number of points within the circle is given by

$$pts_{in} = \frac{\tilde{\pi} \cdot pts}{4}.$$

Therefore, since under the randomness hypothesis pts_{in} is approximately distributed as a $Binomial(\pi/4, pts)$, due to Eq. (15) we get the following statistic with approximate standard normal distribution:

$$z = \frac{pts_{in} - pts \cdot \frac{\pi}{4}}{\sqrt{pts \cdot \frac{\pi}{4} \cdot \left(1-\frac{\pi}{4}\right)}}.$$

We now follow the same path we did before. Under the null hypothesis, the sequence is random, and the number of points that is expected to be within the circle are around $\frac{\pi}{4} \cdot pts$, and so the p value, just like in Eq. (10), is given by

$$P(\mid Z \mid \geq \mid z \mid) = 2 \cdot (1 - \Phi(\mid z \mid)). \tag{16}$$

In order to double check that these formulae are indeed well derived, we confirmed p value uniformity for the 10,000 files of */dev/urandom* which we used in Sect. 3.

If the p values are well formulated, they should be uniformly distributed between 0 and 1. In order to check that such is the case, we perform a Chi-square uniformity test, dividing the p values in 15 bins that partition [0, 1] in even intervals, and measuring the goodness of fit for the number of p values in each bin compared to the expected 10,000/15. The resulting p values for this uniformity test are shown in Table 3, and they confirm our expectations of uniformity. In order to see this graphically, we also show the corresponding histograms in Fig. 9.

Table 3 P values after applying a 15-bin Chi-square uniformity test to the p values obtained from */dev/urandom*

Test	Mean	Serial correlation	Monte Carlo π estimate
p value	0.5605	0.7504	0.8766

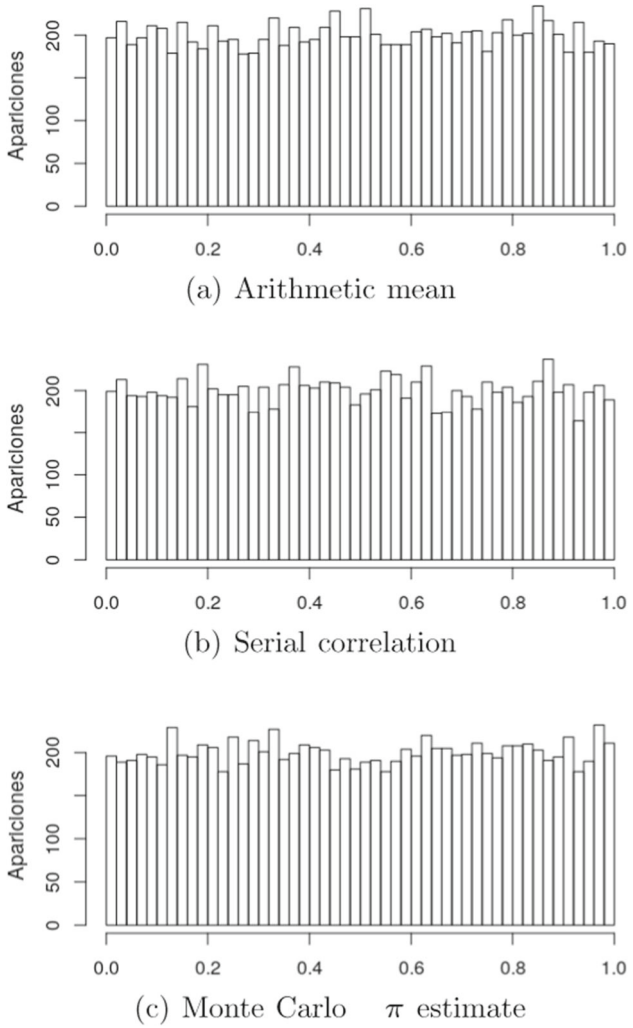

Fig. 9 Histograms of our new p values for 10,000 15 MB files from */dev/urandom*

6 New tests

We decided to add a couple of very fast tests in order to extend the capabilities of the original ENT battery and address the issue detailed in Sect. 4.

6.1 Runs test

Firstly we add a simple runs test, as explained in [28], which in turn takes it from [29]. This test is one of the most well-known and used for randomness testing. In fact, all the most popular test suites include this test as an important element, for example, the NIST SP 800-22 battery, Dieharder, CryptX, and SPRNG, among others, as well as older batteries such as the Knuth or Diehard batteries.

Table 4 Signs for runs test

Number	23	201	3	165	242
Sign	−	+	−	−	+

Given a sequence $x = x_0 \ldots x_{n-1}$, we create a sequence of n + and − signs, by comparing each value to the median 127.5. If a value is below the median, we assign the sign −, and if it is above it we assign a +. We show a small example in Table 4.

A run is then defined as a subsequence of the sign sequence consisting entirely of +'s or −'s, such that the signs at either end of the run are different from the sign of the run.

For instance, the example of Table 4 consists of 4 runs: 2 of + signs and 2 of − signs. We will define n_1 as the number of − signs and n_2 as the number of + signs. Then, the expected number of runs is given by

$$\bar{R} = \frac{2n_1n_2}{n_1+n_2} + 1 \qquad s_R^2 = \frac{2n_1n_2(2n_1n_2 - n_1 - n_2)}{(n_1+n_2)^2(n_1+n_2-1)}.$$

We denote by R the number of observed runs and s_R^2 its variance [29]. Since R follows a binomial distribution, we get that

$$Z = \frac{R - \bar{R}}{s_R} \longrightarrow \mathcal{N}(0,1) \text{ as } n \to \infty.$$

Therefore, a two-tailed p value is obtained just like in Sect. 5.1.

6.2 Local means test

We also created a specific test to address the problem of the pseudo-counter passing the battery successfully. This test is based on two principles: the arithmetic mean's p value of Sect. 5.1 and a Chi-square goodness-of-fit test.

We divide the sequence in N parts of M bytes each (and discard $n \mod M$ bytes). We denote each block $X_i = x_{i\cdot M} x_{i\cdot M+1} \ldots x_{(i+1)\cdot M-1}$ for $i = 0, \ldots, N-1$. We get a z_i statistic by replicating Eq. (9)

$$z_i = \sqrt{n}\frac{\bar{X}_i - \mu}{\sigma}$$

where σ y μ are the same as in Equation (8), and $X_i = x_{i\cdot M} x_{i\cdot M+1} \ldots x_{(i+1)\cdot M-1}$. Since the z_i are approximately distributed as standard normals, we can now take a goodness-of-fit statistic, via

```
StringENT | Results report
-------------------------------------------------------------
Entropy = 7.99991171413 bits per byte.
Optimum compression would reduce the size
of this 2097152 byte file by 0 percent.
-------------------------------------------------------------
Chi square distribution for 2097152 samples is
256.00.
p-value    0.470609
-------------------------------------------------------------
Arithmetic mean value of data bytes is
127.5005 (127.5 = random).
p-value    0.992366
-------------------------------------------------------------
Monte Carlo value for Pi is 3.140925542 (error
0.02 percent).
p-value    0.810200
-------------------------------------------------------------
Serial correlation coefficient is -0.000000
(totally uncorrelated = 0.0).
p-value    0.999690
-------------------------------------------------------------
The number of runs is 1048576.
p-value    0.998898
-------------------------------------------------------------
The local means test's X^2 statistic is
524263.812 for 2048 blocks.
p-value     0.000000 ***
-------------------------------------------------------------
*:     Significant for α=0.05
**:    Significant for α=0.01
***:   Significant for α=0.001
```

Fig. 10 Results of applying StringENT to modified 4,16-counter

$$X^2 = \sum_{i=0}^{N-1} z_i^2. \tag{17}$$

Under randomness hypothesis, each block X_i is independent from all blocks X_j with $j \neq i$, and so the statistic of Eq. 17 has a χ^2_N distribution, that is, a Chi-square distribution with N degrees of freedom. Therefore, we can obtain a one-tailed p value by reusing the Chi-square test already in the battery, with no extra code.

Note that this test in the bit mode becomes the block frequency test used in [20], and is inspired by it. We will take $M = 1024$ as default.

After introducing these changes, and setting a bigger precision for the entropy output, we run the extended battery on the modified counter of Sect. 4. The output is shown in Fig. 10.

We see that the local means test is able to easily detect the non-randomness of the modified counter. Since each block in the output of the counter has very skewed means from the expected 127.5, the goodness-of-fit statistic is extremely high, and so the corresponding p value is less than 10^{-6}. Therefore, we get a significant result at $\alpha = 0.01$ that allows us to confidently reject the randomness hypothesis for the modified counter.

7 Execution times

In this section, we look at the new changes we introduced to the battery code, and what computational cost they bring.

1. *Computation of the p value for the arithmetic mean test* When the battery calculates the arithmetic mean, the cost associated with this computation is a case of computing Eqs. (10) and (9), which is done in a constant number of steps. Thus these computations are on the order of $\mathcal{O}(1)$.
2. *Computation of the p value for the serial correlation test* Similarly to the previous case, this is performed in a constant number of operations that does not depend on the length of the sequence, so its complexity is of the order of $\mathcal{O}(1)$.
3. *Computation of the p value for the estimation of the Monte Carlo estimation of* π It is also a constant number of operations, since the battery has already calculated the number of points inside the circle and the total number of points. Therefore we have a complexity on the order of $\mathcal{O}(1)$.
4. *Runs test* The runs can be calculated in a single pass to the sequence, because while the i-th byte is being studied, its sign can be compared with that of the $i-1$ byte, and it can be determined at the same time whether there is a new run or not. Finally, the number of operations must be added to find the p value, but this is constant whatever the number of runs and the number of positive and negative signs. Therefore, this has a complexity order of $\mathcal{O}(n)$, where n is the number of bytes in the sequence.
5. *Local means test* Like the runs, in a single pass through the sequence, all the means are found, normalized, squared, and the X^2 statistic of Eq. 17 is updated. Thus, the complexity is of the order of $\mathcal{O}(n)$.

Therefore, while doing more calculations that will take slightly longer, the order of complexity of StringENT is still at $\mathcal{O}(n)$, which is the same that all ENT computations have. To empirically verify this result, we experimented with different files of varying sizes and compared the results. We will also include the results when applying the NIST SP 800-22 battery as a reference, it is one of the most commonly used in practice. We note that, although the execution time has gone up, it is still extremely fast, particularly when compared to the NIST SP 800-22 suite. The results are shown in Table 5. These were computed using a standard Windows 10 (64-bit)

Table 5 Execution time comparisons

File size (MB)	ENT (s)	StringENT (s)	NIST SP 800-22
1	0.028	0.049	6.065 s
5	0.122	0.202	30.367 s
10	0.197	0.39	1 m 1.045 s
13	0.255	0.473	1 m 19.608 s

machine with an AMD Ryzen 3700XT (3.6 Ghz, 8 cores, 16 threads) CPU and 64GB of RAM.

8 Redundancies

After the derivation of the arithmetic mean p value, we noticed that p values for the arithmetic mean and the uniformity Chi-square test are the same in the bit mode of the suite. This makes sense since in the ENT bit mode, the Chi-square test just measures how many ones and zeros there are, and compares that to the expected rate of 0.5. That is, in bit mode, the Chi-square test and the arithmetic mean test perform the very same computation.

This is also true for the newly introduced runs test and the serial correlation test, and consequently they output the same p values (save for digit precision). Therefore, we acknowledge that by using the bit mode of the battery we can incur in many undesirable redundancies and hence we discourage its usage.

On the other hand, we saw in Sect. 3 that the Entropy test and the Chi-square test are very highly correlated, which means they also provide similar information. Both tests focus

```
StringENT | Results report
---------------------------------------------------------------
Entropy = 7.96850018978 bits per byte.
Optimum compression would reduce the size
of this 780470 byte file by 0 percent.
---------------------------------------------------------------
Chi square distribution for 780470 samples is
34797.80.
p-value    0.000000 ***
---------------------------------------------------------------
Arithmetic mean value of data bytes is
126.6121 (127.5 = random).
p-value    0.000000 ***
---------------------------------------------------------------
Monte Carlo value for Pi is 3.184796814 (error
1.38 percent).
p-value    0.000000 ***
---------------------------------------------------------------
Serial correlation coefficient is 0.010767
(totally uncorrelated = 0.0).
p-value    0.000000 ***
---------------------------------------------------------------
The number of runs is 384137.
p-value    0.000000 ***
---------------------------------------------------------------
The local means test's X^2 statistic is
1417.428 for 762 blocks.
p-value    0.000000 ***
---------------------------------------------------------------
*:     Significant for α=0.05
**:    Significant for α=0.01
***:   Significant for α=0.001
```

Fig. 12 StringENT results for Zumaia seaside picture

Fig. 11 Seaside picture of Zumaia (copyright-free by Ángel Pérez)

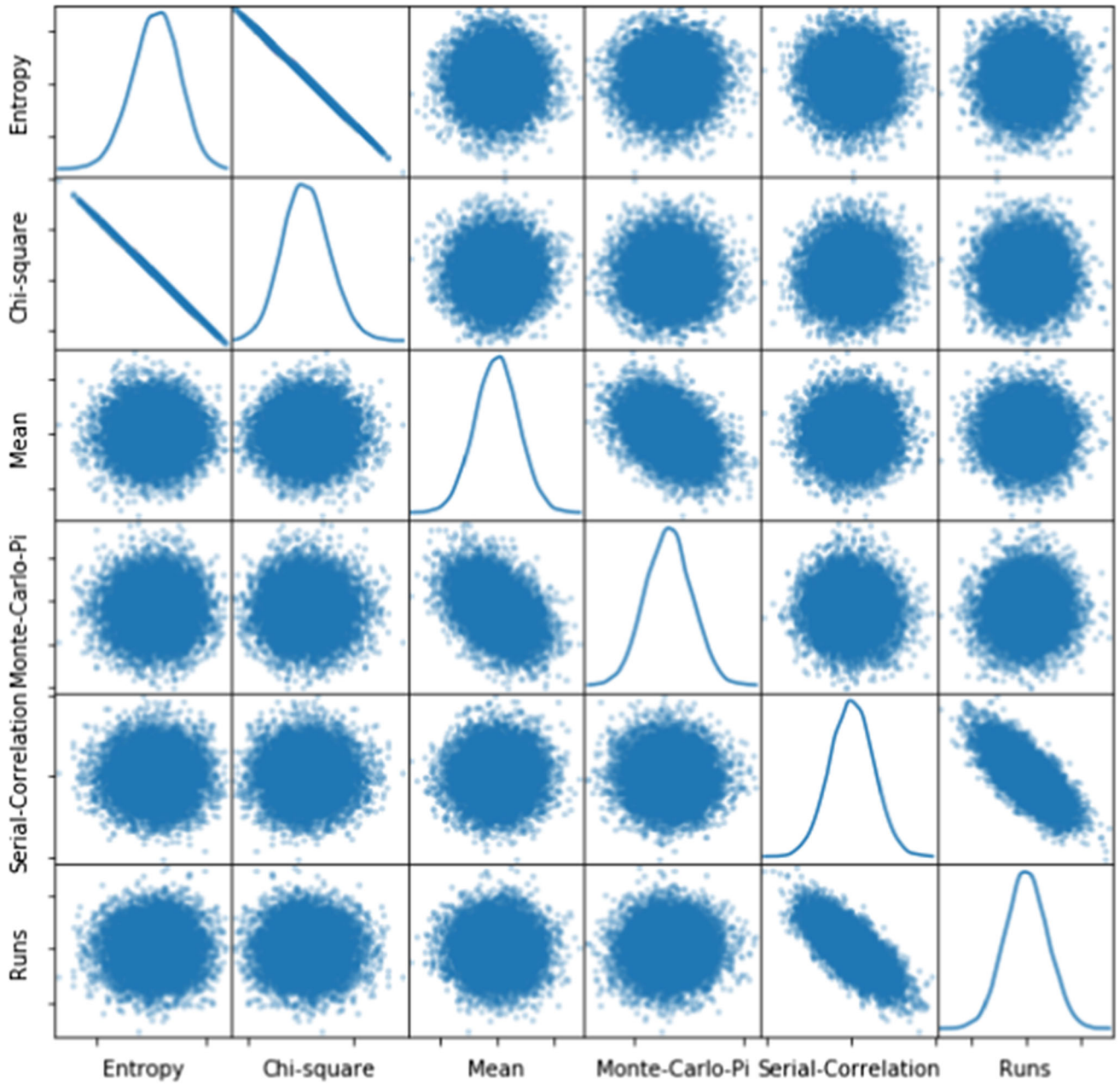

Fig. 13 Scatter plot matrix including better Entropy precision and Runs test

Table 6 Correlations of StringENT statistics for */dev/urandom* files

	Entropy	Chi-square	Mean	Monte-Carlo-Pi	Serial-correlation	Runs	LM-p-value
Entropy	1.0000	−1.0000	−0.0071	0.0058	−0.0083	0.0084	−0.0094
Chi-square	−1.0000	1.0000	0.0071	−0.0058	0.0083	−0.0084	0.0094
Mean	−0.0071	0.0071	1.0000	−0.4239	0.0219	−0.0194	0.0057
Monte-Carlo-Pi	0.0058	−0.0058	−0.4239	1.0000	−0.0334	0.0364	−0.0050
Serial-Correlation	−0.0083	0.0083	0.0219	−0.0334	1.0000	−0.7458	−0.0279
Runs	0.0084	−0.0084	−0.0194	0.0364	−0.7458	1.0000	0.0264
LM-p-value	−0.0094	0.0094	0.0057	−0.0050	−0.0279	0.0264	1.0000

Table 7 *P* values of observed correlations in StringENT statistics for */dev/urandom* files

	Entropy	Chi-square	Mean	Monte-Carlo-Pi	Serial-correlation	Runs	LM-*p*-value
Entropy	0.0000	**0.0000**	0.4753	0.5638	0.4076	0.4009	0.3478
Chi-square	**0.0000**	0.0000	0.4769	0.5636	0.4080	0.4004	0.3485
Mean	0.4753	0.4769	0.0000	**0.0000**	0.0285	0.0521	0.5711
Monte-Carlo-Pi	0.5638	0.5636	**0.0000**	0.0000	**0.0008**	**0.0003**	0.6154
Serial-Correlation	0.4076	0.4080	0.0285	**0.0008**	0.0000	**0.0000**	**0.0053**
Runs	0.4009	0.4004	0.0521	**0.0003**	**0.0000**	0.0000	**0.0082**
LM-*p*-value	0.3478	0.3485	0.5711	0.6154	**0.0053**	**0.0082**	0.0000

on the uniformity of the values $\{0, 1, \ldots, 255\}$. Since the compression rate is linearly related with the Entropy, it adds no additional information. However, what tests are run can be ultimately decided by the user.

In order to illustrate this, we run the battery on a *jpg* picture of a seaside landscape. It is displayed in Fig. 11, and the corresponding results in Fig. 12. By looking at the output we can confirm:

1. Displaying a 0 per cent compression rate is not necessarily a good indicator of randomness at all, as it takes "relatively little" entropy (it is just 0.03 below the maximum of 8 bits/byte for this file). Therefore, even a picture with multiple non-random patterns can achieve a 0 per cent compression rate, although the picture is clearly non-random. This is particularly true of compressed formats such as jpg/mpeg/webP, etc.
2. The rest of tests all fail with a very high statistical significance, as one should expect.

Also, having now more digits of precision for the Entropy and the new tests, we checked correlations again. Figure 13 shows a scatter plot matrix of some of the StringENT statistics for the 10,000 */dev/urandom* files. Table 6 shows the correlations obtained for the data in these tests, and Table 7 has the associated *p* values. We can see, as highlighted in bold, that many tests are not independent. Therefore, if we really want to maximize StringENT efficiency, we can remove some of these.

9 Conclusions and future work

We have studied the popular ENT test suite for evaluating random number generators, shown that some trivially flawed generators (i.e. pseudo-counter) can fool it and then proposed an extension we call StringENT.

With StringENT, we keep most of the simplicity and speed that made ENT so widely used, but add some much needed functionalities (in particular the computation of *p* values for all existing test) and two new tests. Now the user can call the execution of various tests through a parameter in the executable, to find a convenient trade-off between speed and coverage.

With all tests now offering a *p* value, it becomes easier to analyse their correlation, and we do this next in our work.

This also means the interpretation of the output statistics is easier now.

Future work could focus on designing, implementing and selecting new sets of tests which are similarly fast but do not incur on any kind of correlation.

To sum up, we believe that a promising way forward will consist in adding more tests to StringENT that maintain the characteristic speed of the battery while offering independent, uncorrelated insights.

Acknowledgements We would like to thank the reviewers for their comments, which have helped us to improve our work. This work has received funding from THEIA (Techniques for Integrity and Authentication of Multimedia Files of Mobile Devices) UCM project (FEI-EU-19-04)and from THEIA I (Techniques for integrity, authentication and scene recognition in multimedia files of mobile devices - Part I) UCM project (FEI-EU-21-01).Julio Hernandez-Castro was supported during this work by EPSRC's Quantum Communications Hub (grant number EP/T001011/1) and Innovate UK Industrial Strategy Challenge Fund (ISCF) Project No. 106374-49229 AQuRand (Assurance of Quantum Random Number Generators).

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Kawai, R., Masuda, H.: On simulation of tempered stable random variates. J. Comput. Appl. Math. **235**(8), 2873–2887 (2011). https://doi.org/10.1016/j.cam.2010.12.014
2. Wang, Y., Nicol, T.: On statistical distance based testing of pseudo random sequences and experiments with PHP and Debian Open SSL. Comput. Secur. **53**, 44–64 (2015). https://doi.org/10.1016/j.cose.2015.05.005
3. Wang, P., You, F., He, S.: Design of broadband compressed sampling receiver based on concurrent alternate random sequences. IEEE Access **7**, 135525–135538 (2019). https://doi.org/10.1109/ACCESS.2019.2942687
4. Yao, Y., Chen, X., Kang, W., Zhang, Y., Zhao, W.: Thermal Brownian motion of Skyrmion for true random number generation. IEEE Trans. Electron Devices **67**(6), 2553–2558 (2020). https://doi.org/10.1109/TED.2020.2989420
5. Srikanth, C.: Certain sequence of arithmetic progressions and a new key sharing method. Cryptogr. Commun. **12**, 597–612 (2020). https://doi.org/10.1007/s12095-019-00416-z
6. Gómez, A.I., Gómez-Pérez, D., Pillichshammer, F.: Secure pseudorandom bit generators and point sets with low star-discrepancy. J. Comput. Appl. Math. **396**, 1–8 (2020). https://doi.org/10.1016/j.cam.2021.113601
7. De Matteis, A., Pagnutti, S.: Pseudorandom permutation. J. Comput. Appl. Math. **142**(2), 367–375 (2020). https://doi.org/10.1016/S0377-0427(01)00425-3
8. Sarkar, P.: Modes of operations for encryption and authentication using stream ciphers supporting an initialisation vector. Cryptogr. Commun. **6**, 189–231 (2014). https://doi.org/10.1007/s12095-013-0097-7
9. Ak, M., Hanoymak, T., Selçuk, A.A.: IND-CCA secure encryption based on a Zheng–Seberry scheme. J. Comput. Appl. Math. **259**, 529–535 (2014). https://doi.org/10.1016/j.cam.2013.06.042
10. Mohanty, A., Sutaria, K.B., Awano, H., Sato, T., Cao, Y.: RTN in scaled transistors for on-chip random seed generation. IEEE Trans. Very Large Scale Integr. VLSI Syst. **25**(8), 2248–2257 (2017). https://doi.org/10.1109/TVLSI.2017.2687762
11. Hamann, M., Krause, M.: On stream ciphers with provable beyond-the-birthday-bound security against time-memory-data tradeoff attacks. Cryptogr. Commun. **10**, 959–1012 (2018). https://doi.org/10.1007/s12095-018-0294-5
12. Bharadwaj, B., Sairabanu, J.: Image encryption using a modified pseudo-random generator. In: 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), pp. 1–6 (2020). https://doi.org/10.1109/ic-ETITE47903.2020.094
13. Tuncer, T., Avaroğlu, E.: Random number generation with LFSR based stream cipher algorithms. In: 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 171–175 (2017). https://doi.org/10.23919/MIPRO.2017.7973412
14. Sarkar, S., Dey, P., Adhikari, A., Maitra, S.: Probabilistic signature based generalized framework for differential fault analysis of stream ciphers. Cryptogr. Commun. **9**, 523–543 (2017). https://doi.org/10.1007/s12095-016-0197-2
15. Saha, R., Geetha, G., Kumar, G., Kim, T.-H., Buchanan, W.J.: MRC4: a modified RC4 algorithm using symmetric random function generator for improved cryptographic features. IEEE Access **7**, 172045–172054 (2019). https://doi.org/10.1109/ACCESS.2019.2956160
16. Bontchev, B.: Modern trends in the automatic generation of content for video games. Serdica J. Comput. **2**, 133–166 (2016)
17. Kristian Lundedal Nielsen, R., Grabarczyk, P.: Are loot boxes gambling? Random reward mechanisms in video games. Trans. Digit. Games Res. Assoc. **4**(3), 171–207 (2019)
18. Almaraz Luengo, E., García Villalba, L.J.: Recommendations on statistical randomness test batteries for cryptographic purposes. ACM Comput. Surv. **54**(4), 1–34 (2022). https://doi.org/10.1145/3447773
19. Walker, J.: ENT: a pseudorandom number sequence test program. https://www.fourmilab.ch/random/ (2008)
20. 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., Heckert, N.A., Dray, J.F., Vo, S.: 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)
21. L'ecuyer, P., Simard, R.: Testu01: a C library for empirical testing of random number generators. ACM Trans. Math. Softw. **33**(4), 1–40 (2007)
22. Brown, R.G., Eddelbuettel, D., Bauer, D.: Dieharder: a random number test suite (version 3.31.1). https://webhome.phy.duke.edu/~rgb/General/dieharder.php (2014)
23. Gray, R.M.: Entropy and Information Theory, 1st edn. Springer, New York (2013)
24. Kowalski, C.J.: On the effects of non-normality on the distribution of the sample product-moment correlation coefficient. Appl. Stat. **21**(1), 1–12 (1972). https://doi.org/10.2307/2346598
25. Yerukala, R., Boiroju, N.K., Krishna, M.: Approximations to the t-distribution. Int. J. Stat. Math. **8**(1), 19–21 (2013)
26. Zogheib, B., Elsaheli, A.: Approximations to the t-distribution. Appl. Math. Sci. **9**, 2445–2449 (2015)
27. Dunbar, S.R.: Topics in probability and stochastic processes: the de Moivre–Laplace central limit theorem. https://www.math.unl.edu/~sdunbar1/ProbabilityTheory/Lessons/BernoulliTrials/DeMoivreLaplaceCLT/demoivrelaplaceclt.pdf
28. NIST: Runs test for detecting non-randomness. https://www.itl.nist.gov/div898/handbook/eda/section3/eda35d.htm (2013)
29. Bradley, J.V.: Distribution-Free Statistical Tests, 1st edn. Prentice-Hall, Englewood Cliffs (1968)