electronics

MDPI

Article

Review of Methodologies and Metrics for Assessing the Quality of Random Number Generators

Luca Crocetti *, Pietro Nannipieri *, Stefano Di Matteo, Luca Fanucci and Sergio Saponara

Department of Information Engineering, University of Pisa, Via G. Caruso, 16, 56122 Pisa, Italy
* Correspondence: luca.crocetti@unipi.it (L.C.); pietro.nannipieri@unipi.it (P.N.)

Abstract: Random number generators are a key element for various applications, such as computer simulation, statistical sampling, and cryptography. They are used to generate/derive cryptographic keys and non-repeating values, e.g., for symmetric or public key cyphers. The strength of a data protection system against cyber attacks corresponds to the strength of the weakest point in the security chain. Therefore, from a mathematical point of view, the security chain can be compromised even if the strongest algorithm is implemented. In fact, if the system requires keys or other random values and the generation process shows a certain vulnerability, the security of the system itself can be compromised. In this article, we present the most reliable tools and methodologies and the main standardisation efforts in the field of computer security to assess the quality of random number generators and ensure that they can be applied to computer security applications by offering adequate security strength. We offer a comprehensive guide that can be used as a quick and practical reference by developers of random number generators of any type to evaluate the random bit streams generated by implemented modules and determine whether or not they can be used in cybersecurity applications. Finally, we also present some use cases to which we applied the presented approach.

Keywords: random number generator; RNG; entropy; CSPRNG; TRNG; PRNG; cryptographic key; cryptography

Citation: Crocetti, L.; Nannipieri, P.; Di Matteo, S.; Fanucci, L.; Saponara, S. Review of Methodologies and Metrics for Assessing the Quality of Random Number Generators. *Electronics* **2023**, *12*, 723. https://doi.org/10.3390/electronics12030723

Academic Editor: Paris Kitsos

Received: 3 January 2023
Revised: 30 January 2023
Accepted: 31 January 2023
Published: 1 February 2023

1. Introduction

Random number generation is the process of generating a sequence of numbers that are truly random and unpredictable. These numbers are essential for many modern applications, including cryptography, which relies on random numbers to generate secure keys and codes that can protect sensitive information [1]. In the field of cryptography, random numbers are used to generate keys that can be used to encrypt and decrypt messages and data. The security of these keys depends on their unpredictability, which makes it difficult for attackers to guess or break them. To ensure the security of these keys, it is important to use high-quality random number generators that can produce truly random and unpredictable numbers. Random number generation is also important in many other fields, including gaming, simulation, and scientific research. In these fields, random numbers are used to simulate real-world events, test hypotheses, and analyse data [2]. Overall, random number generation is a vital tool for many modern applications, and the quality and security of these numbers are crucial for the reliability and integrity of these systems.

The quality of a Random Number Generator (RNG) is important because weak or predictable random numbers can compromise the security of a cryptographic system [3]. There are several ways to evaluate the quality of an RNG, including statistical tests and hardware evaluations. Statistical tests involve analysing the output of the RNG to determine whether it exhibits certain statistical properties that are characteristic of truly random numbers. These tests can include measures of uniformity, independence, and unpredictability. Hardware evaluations involve physically examining the hardware used to

generate the random numbers to ensure that it is functioning properly and that there are no vulnerabilities or weaknesses that could compromise the security of the RNG [4]. The National Institute of Standards and Technologies (NIST) is a non-regulatory agency of the United States Department of Commerce that is responsible for developing and promoting standards, guidelines, and best practices in a wide range of fields, including cryptography and random number generation.

In the field of cryptography, the NIST plays a key role in the development and promotion of cryptographic standards, guidelines, and best practices. This includes the development of standard algorithms and protocols, as well as the testing and evaluation of cryptographic systems and products. The NIST also publishes guidelines and recommendations on how to use cryptography effectively and securely. The Bundesamt für Sicherheit in der Informationstechnik (BSI) is the German national cybersecurity agency. Like the NIST, the BSI is responsible for developing and promoting standards, guidelines, and best practices in the field of cybersecurity, including cryptography and random number generation. The BSI also works with other government agencies, industry partners, and international organisations to improve cybersecurity and protect critical infrastructure. Both the NIST and the BSI play important roles in the field of cryptography and random number generation by developing and promoting standards, guidelines, and best practices that help ensure the security and reliability of cryptographic systems. Several other organisations and research groups provide tools and documents that can be used to evaluate RNGs, in addition to the NIST and BSI test suites. Some of these tools include:

- Dieharder [5]: This is a suite of statistical tests designed to evaluate the quality of RNGs. It includes a wide range of tests that measure different aspects of the output, including uniformity, independence, and unpredictability.
- PractRand [6]: This is another suite of statistical tests that are designed to evaluate the quality of RNGs. It includes a range of tests that measure different aspects of the output, including uniformity, independence, and unpredictability.
- ENT [7]: ENT (short for "Entropy") is a command-line tool that can be used to test the quality of Pseudo-Random Number Generators (PRNGs). ENT can be used to evaluate the quality of a PRNG by comparing its output to the output of a True Random Number Generator (TRNG).
- RaBiGeTe [8]: RaBiGeTe (short for "Random Bit Generators Tester") is a tool for evaluating the quality of PRNGs. It runs a series of statistical tests on their output. The tests are designed to detect biases or patterns in the output of the PRNG that may indicate poor quality.

However, the NIST and BSI test suites are usually preferred over these tools because they have been widely adopted and are widely recognised as reliable and effective methods for evaluating the quality of RNGs. They are also regularly updated to ensure that they are relevant and effective in detecting vulnerabilities and weaknesses in RNGs.

The NIST Entropy Assessment (EA) test suite [9] and procedure B (tests *T6*–*T8*) of the BSI suite [10] are sets of statistical tests that are used to evaluate the quality of RNGs. Both test suites are used to assess the entropy of RNGs, which is a measure of the randomness or unpredictability of the numbers generated by the RNG. The NIST EA test suite is a set of statistical tests designed to evaluate the statistical properties of the output of an RNG. The test suite includes a range of tests that measure different aspects of production, including uniformity, independence, and unpredictability. The test suite is used to evaluate the quality of RNGs used in cryptographic systems and other applications requiring high-quality random numbers. The BSI *T6*–*T8* test suite is a set of statistical tests used to evaluate the quality of RNGs. The test suite includes a range of tests that measure different aspects of the output, including uniformity, independence, and unpredictability. The test suite is used to evaluate the quality of RNGs used in cryptographic systems and other applications requiring high-quality random numbers. Both the NIST EA test suite and the BSI *T6*–*T8* test suite are important tools for evaluating the quality of RNGs. These test suites help ensure that the RNGs used in cryptographic systems and other applications are reliable

and secure, and they help protect against potential vulnerabilities or weaknesses that could compromise the security of the systems.

The NIST Statistical Test Suite (STS) [11] and procedure A (tests *T0* and *T1–T5*) of the BSI suite [10] are sets of statistical tests that are used to evaluate the quality of RNGs. Both test suites are used to assess the randomness of RNGs, which is a measure of the unpredictability or lack of bias in the numbers generated by the RNG. The NIST STS is a set of statistical tests that are designed to evaluate the statistical properties of the output of an RNG. The test suite includes a range of tests that measure different aspects of production, including uniformity, independence, and unpredictability. The test suite is used to evaluate the quality of RNGs that are used in cryptographic systems and other applications that require high-quality random numbers. Procedure A of BSI suite is a set of statistical tests used to evaluate the quality of RNGs. The test suite includes a range of tests that measure different aspects of the output, including uniformity, independence, and unpredictability. The test suite is used to evaluate the quality of RNGs used in cryptographic systems and other applications requiring high-quality random numbers. Both the NIST STS and the BSI *T0–T1* test suites are important tools for evaluating the quality of RNGs. These test suites help ensure that the RNGs used in cryptographic systems and other applications are reliable and secure, and they help protect against potential vulnerabilities or weaknesses that could compromise the security of the systems.

The objective of this contribution is to review the methodologies and the related metrics for reviewing the quality of an RNG in order to provide system developers and evaluators with a quick and practical reference for assessing the quality of their implementations according to the most recognised standardisation agencies' metrics. The rest of this paper is organised as follows:

- In Section 2, we present and describe the taxonomy and the classification methods for RNGs;
- In Section 3, we analyse and schematise the entropy validation procedure according to the most important standardisation agency and provide a simplified workflow to ease the setup of such procedures;
- In Section 4, the same is carried out for randomness evaluation procedures;
- In Section 5, we conclude our work.

2. RBG: Classification, Construction, and Functionality Classes

This section provides a detailed description and classification of Random Bit Generators (RBGs) while referring to the main standardisation organisations (i.e., NIST and BSI) and covering two main points: *Nomenclature, functionalities, and properties of RBGs and their components*, as well as *methods for the construction of RBGs to accomplish specific goals, target applications, and functionality classes*.

2.1. NIST Classification and Construction

For what concerns the NIST, the Special Publication (SP) 800-90 series specifies the guidelines for the construction of high-quality RBGs for both cryptographic and non-cryptographic purposes. These guidelines state that an RBG must be composed of two main components: an entropy source that generates true random values (by exploiting physical and non-physical noise sources) and a Deterministic Random Bit Generator (DRBG) that ensures the indistinguishability of the generated output from an ideal random number generator. The SP 800-90 series includes three parts.

2.1.1. SP800-90A [12]

The first is the Recommendation for Random Number Generation Using Deterministic Random Bit Generators, which specifies the construction of a DRBG. A DRBG uses an approved cryptographic algorithm to produce a sequence of bits (from an initial value) for cryptographic applications. Since the generation of random numbers by a DRBG is a deterministic process, a DRBG is said to produce pseudorandom bits. To obtain an RBG

from a DRBG, the initial value (seed) must be supplied by a source of entropy. A DRBG must include the following functionalities:

- An instantiate function, which involves the acquisition of randomness to initialise the DRBG and its internal state.
- A generate function, which produces the output bitstream and updates the internal state of the DRBG.
- Health testing to determine whether the DRBG operates properly.

A DRBG may also include a *reseeding* function to introduce fresh entropy into the internal state and an *uninstantiate* function to destroy all of the internal information that it has previously stored. All of the approved DRBGs in [12] respect the backtracking resistance property, which states that a compromise of the DRBG's internal state does not affect the security of prior outputs; it can be guaranteed by ensuring that the DRBG's generation algorithm is a one-way function. The prediction resistance property, which states that a compromise of the DRBG's internal state does not affect the security of future DRBG outputs, can be obtained only if a DRBG is effectively reseeded with fresh entropy between producing outputs for consecutive DRBG requests.

2.1.2. SP 800-90B [9]

The second is the Recommendation for the Entropy Sources Used for Random Bit Generation, which specifies the construction and validation of physical or non-physical noise sources used as input for a DRBG or an RBG. The main component of the entropy source model defined by the NIST is the noise source; it contains the non-deterministic and entropy-providing process responsible for the uncertainty associated with the output bitstream of the whole entropy source. The noise source can be a physical noise source, which uses dedicated hardware for the generation of randomness, or a non-physical source, which uses system data, such as human inputs or other system functions. The entropy source model of the NIST includes health tests that are intended to ensure that the noise source operates as expected and an optional conditioning component, which is a deterministic function for reducing bias and/or operating on the output bits to increase the entropy rate.

2.1.3. SP 800-90C [13]

The third is the Recommendation for RBGs Construction, which specifies the construction of three classes of RBGs by using DRBGs that are compliant with [12] and an entropy source that complies with [9].

Table 1 summarises the nomenclature and constructions of RBGs defined by the NIST. The construction RBG.1 does not include an internal noise source (it belongs to the category of DRBGs); the initial seed must be provided by an external RGB over a secure channel. In addition, the reseeding procedure is not supported; thus, it cannot provide prediction resistance. The RBG.1 implementation can be used in devices in which an entropy source cannot be included or in any device that has no access to an entropy source after instantiation.

The RBG.2 construction includes entropy sources that are used to instantiate and reseed the DRBG included in the construction. The randomness source can be physical or non-physical. The prediction resistance property can be guaranteed, since the reseeding procedure can be supported. The RBG.3 construction concerning the RBG.2 requires a physical noise source as an entropy source.

Table 1. Nomenclature and taxonomy for RBGs put out by the NIST standardisation organisation.

General Term	**RBGs**		
Deterministic or non-deterministic	DRBGs	Entropy source	
Noise sources	–	Physical	Non-physical
Constructions	RBG.1		
	RBG.2		
	RBG.3		

2.2. BSI Classification and Construction

Concerning the BSI, the standard [14] defines the nomenclature and taxonomy for RBGs, as reported in Table 2. The general term used by the BSI to indicate a general random number generator is RNG, unlike the NIST, which makes use of RBG to define a general random number generator. The BSI defines a Deterministic Random Number Generator (DRNG) as an RNG that produces random numbers by applying a deterministic algorithm from a secret initial value called a seed, along with other possible additional inputs. It is equivalent to the DRBG defined by the NIST in [12]. A TRNG is a device or mechanism for which the output values depend on a noise source; both Physical True Random Number Generators (PTRNGs) and Non-Physical True Random Number Generators (NPTRNGs) belong to this category; the PTRNG class uses a noise source that exploits physical phenomena (thermal noise, shot noise, jitter, metastability, etc.) from dedicated hardware designs, while the NPTRNG class uses noise sources that typically exploit system data and user interactions to produce digitised random data. The functionality classes defined by the BSI can be briefly summarised and compared as follows.

Table 2. Nomenclature and taxonomy for RNGs put out by the BSI standardisation organisation.

General Term	**RNGs**		
Deterministic or non-deterministic	DRNGs	TRNGs	
Noise sources	–	PTRNGs	NPTRNGs
Functionality classes	DRG.2 DRG.3 DRG.4	PTG.2 PTG.3	NTG.1

2.2.1. DRG.2, DRG.3, and DRG.4

These constructions define the requirements for deterministic RNGs that are suitable for cryptographic applications. The DRG.2 construction is suitable for applications for which the disclosure of previous random numbers due to a compromise of the internal state is not an issue (e.g., for challenges in challenge–response protocols). This means that this class of RNGs does not support the prediction resistance property. The seed material must be generated by RNGs belonging to the PTG.2, PTG.3, or NTG.1 classes. The DRG.3 construction is suitable for all cryptographic applications, except for those that require guaranteed fresh entropy. Therefore, the underlying cryptographic function of this class of RNG is a one-way function (backtracking resistance). The seed can be supplied by both TRNGs and DRNGs. The DRG.4 course includes a reseeding procedure and a high-entropy additional input; in this way, it can ensure the forwarding resistance property. In addition, DRG.4 requires the output of a PTRNG for seeding and reseeding procedures and for the additional input.

2.2.2. PTG.2 and PTG.3

These define requirements for non-deterministic RNGs that use physical noise sources. The PTG.2 class generates high-entropy random numbers that may be distinguishable from ideal randomness when testing large amounts of data. This means that the post-processing may not be cryptographic. The PTG.3 class is the most robust functionality, and it is appropriate for any cryptographic application. The security of the PTG.3 class is ensured by both the physical noise source and the computational security ensured by the cryptographic post-processing. Concerning DRG.4, the cryptographic post-processing of the PTG.3 class does not extend its input data.

2.2.3. NTG.1

The NTG.1 class generates true random bits by employing noise sources such as system data or human interactions instead of dedicated hardware. In addition, cryptographic post-processing is needed. The NTG.1 class is usually employed for devices that do not have access to a physical RNG.

3. Procedures for Entropy Assessment

The procedure illustrated herein is aimed at the assessment of entropy sources or, from a more general point of view, the evaluation of the outputs of modules whose purpose is to generate entropy. Hence, the samples to be collected and used as inputs for the assessment procedure must be extracted from raw data or the eventually conditioned and/or post-processed output of such modules according to the taxonomy, features, and specifications described in Section 2.

The output of this procedure is a numerical value indicating the level of entropy expressed in bits. For example, a module designed to generate entropy with an output data width of 1 bit will result in having an entropy in the range of 0 to 1 bits (theoretical upper limit), while a similar module but with an output data width of 8 bits (1 byte) will result in having an entropy in the range of 0 to 8 bits.

According to what was expressed in Section 1, the reference tools to be used for this purpose are:

- procedure B of the BSI suite, i.e., the battery of tests *T6* to *T8*;
- the NIST EA suite.

3.1. Entropy Assessment with Procedure B of the BSI Suite

Procedure B of the BSI suite is included in the unique BSI tool described in Section 1, which can be downloaded at [10]. This a tool is a Java application with a GUI that allows the user to choose between test *T0* (procedure A), the battery of tests *T1* to *T5* (procedure A), and the battery of tests *T6* to *T8* (procedure B). Once the tool is run, the latter option has to be selected, and several other configuration settings have to be specified, as shown in Figure 1.

Concerning Figure 1, in addition to the selection of procedure B (i.e., the check on the box *Proc. B: T6–T8*), it is necessary to specify the path of the input file (i.e., the file containing the samples of the entropy source module), the output format (if normal or detailed), the input data format (*Data Format* tab), the type of test (i.e., *Normal Test* or *Repeat Test*), and the bit width of the samples (*Internal Random Numbers* tab). Particular attention has to be paid to these last three points.

First of all, the choices for the *Data Format* setting indicate if 1 byte of the input sample file corresponds to 1 random bit (i.e., option *1 Byte = 1 RNDbit*) or if 1 byte of the input samples file corresponds to 8 random bits or 1 random byte (i.e., option *1 Byte = 8 RNDbit*). The former option has to be selected if the input file is a textual file; therefore, every random bit is represented by the corresponding ASCII character. Since an ASCII character occupies 1 byte of disk storage, each byte of the input file (i.e., an ASCII character) represents the value of 1 random bit. The latter option instead refers to the binary format (based on a personal disk's storage capabilities, one option or the other one can be chosen without

restrictions after having accordingly generated the input sample file; however, the usage of the binary format allows for the generation of more compact input files and for saving space on the disk; hence, if possible, this choice is suggested) of the input file (*.bin*) because, in this case, each byte of the input file is a concatenation of eight single binary values, each representing a different sample (1 random bit); hence, 1 byte of the input file corresponds to 8 random bits to be analysed.

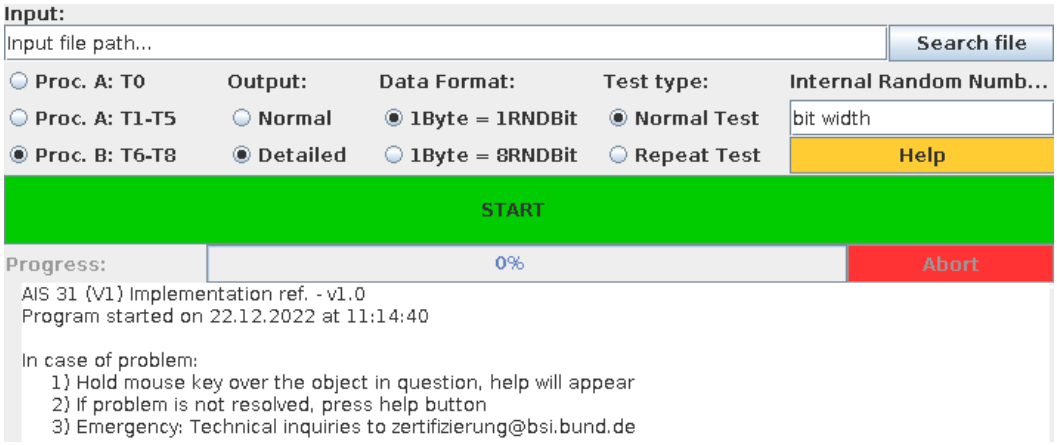

Figure 1. BSI tool for entropy assessment.

Concerning the *Test type* configuration, it allows an indication of if the test to be run is the first attempt (literally, the first time the test is executed on a specific input file) or not. Indeed, the reference document of the BSI suite specifies that in the case of specific unsuccessful outcomes, procedure B can be repeated for a second and last time. In particular, because procedure B consists of several internal tests, if, at the first attempt, only one internal test fails, the procedure can be applied again, but to a new set of samples. For this reason, when applying procedure B for the first time (i.e., *Normal Test*), the BSI tool only loads the required N samples into the RAM of the host PC, while it separates and stores the residual samples in a dedicated file that it creates automatically (by default, this file is automatically generated by the BSI tool in the same folder of the input file, and the word *_rest* is appended to the name of the input file). If the first attempt fails with at most one failing internal test, procedure B can be applied for a second (and final) time to the residual samples by setting the test type to the value *Repeat Test*. If the second attempt also fails (i.e., at least one internal test fails), the procedure has to be considered as failed; in other words, this means that the measured value of entropy is not reliable. In any other case (i.e., the first or second attempt succeeds without any failures), procedure B is passed, and the measured entropy level is reliable.

The third and last main configuration option (*Internal Random Numbers*) has to be used to indicate the bit width of the samples (by typing in the corresponding entry). For instance, the entropy source module that we developed for the European Processor Initiative (EPI) project and that we illustrated in [15,16] generates eight independent random bits in parallel as a unique output byte per time; hence, the bit width of input samples to be specified is 8.

Once all of the configuration parameters are set, the test can be run by pressing the *START* button; during its execution, the logs are displayed in the text box under the progress bar, which is filled accordingly. At the end of the test, the logs that contain both the intermediate results and the final ones are also stored in a dedicated file that constitutes the main and most significant output of procedure B of the BSI tool. As an example, the content of the output file is reported in Listing 1 to show the main results that we obtained when we tested our entropy source module [15,16].

Listing 1. Output of procedure B of BSI suite.

```
TEST STARTED.
TEST-SUITE:       Procedure B/T6--T8
FILE NAME:        entropy_src_samples.bin
OUTPUT DETAILS:   Enabled.
DATA FORMAT:      1 file byte = 8 random bit (1 random byte).
TEST TYPE:        Normal Test.
RND-Bit Width:    8 bit.
   Reading file.
       entropy_src_samples.bin
       Copying bit stream file to RAM...
       Converting data file to byte stream...
       Writing residual file: entropy_src_samples.bin_rest
       7200000 elements copied into RAM.
   File reading completed.
       Test procedure T6a for the verification of Procedure B.i) (vii.a) started
           .
       |P(1) - 0.5| = 0.0027800000000000047
       Last Element: 100000
   Test procedure T6a passed.
   Test procedure T6b for the verification of Procedure B.i) (vii.b) started.
       p(01) = 0.49988
       p(11) = 0.49901
       |p_(01) - p_(11)| = 8.699999999999819E-4
       Last Element: 500136
   Test procedure T6b passed.
   Test procedure T7a for the verification of Procedure B.i)(vii.c) started.
       Test Statistic[0] = 0.015680013271563233
       Test Statistic[1] = 0.13448132593208115
       Last Element: 1705488
   Test procedure T7a passed.
   Test procedure T7b for the verification of Procedure B.i)(vii.d) started.
       Test Statistic[0] = 0.7920246001580802
       Test Statistic[1] = 1.6245080785162238
       Test Statistic[2] = 0.706880000282752
       Test Statistic[3] = 0.5056204187042688
       Last Element: 4927816
   Test procedure T7b passed.
   Test T8 for the verification of Procedure B.i)(vii.e) started.
       Test Statistic: Entropy = 7.999997762809403
       Last Element: 6996296
   Test T8 passed.
Step successfully completed.
```

Several elements in Listing 1 can be individuated as further confirmation of how much is expressed in this section. For instance, the values set for the configuration parameters in the first rows of the output file, such as the data format for binary files (row 5) associated with the usage of an input binary file (row 3), the type of test (*Normal*, row 6), the bit width of random samples (8 bits at row 7, according to our implementation in [15,16]), the number of samples used for testing (7,200,000, row 13), and the generation of the file containing the residual samples, can be noted. In addition, the main and most significant elements are the outcomes of the intermediate tests (which are nine in total, i.e., *T6a*, *T6b*, two statistics for *T7a*, and four statistics for *T7b* and *T8*), respectively, at rows 18, 23, 26, 27, 31, 32, 33, 34, and 40, and the measure of entropy, which is a direct result of test *T8*. In the former case, all of the tests passed; in the latter one, the measured value of entropy is reported in row 38 (entry *Test Statistic: Entropy*), and it corresponds to 7.999.

According to the BSI specifications in [17], the final entropy value calculated by procedure B and the correlated results can be considered valid with a confidence of the 99.98%.

As a final summary of this section, Figure 2 reports a flow diagram for the entropy assessment when using the BSI suite.

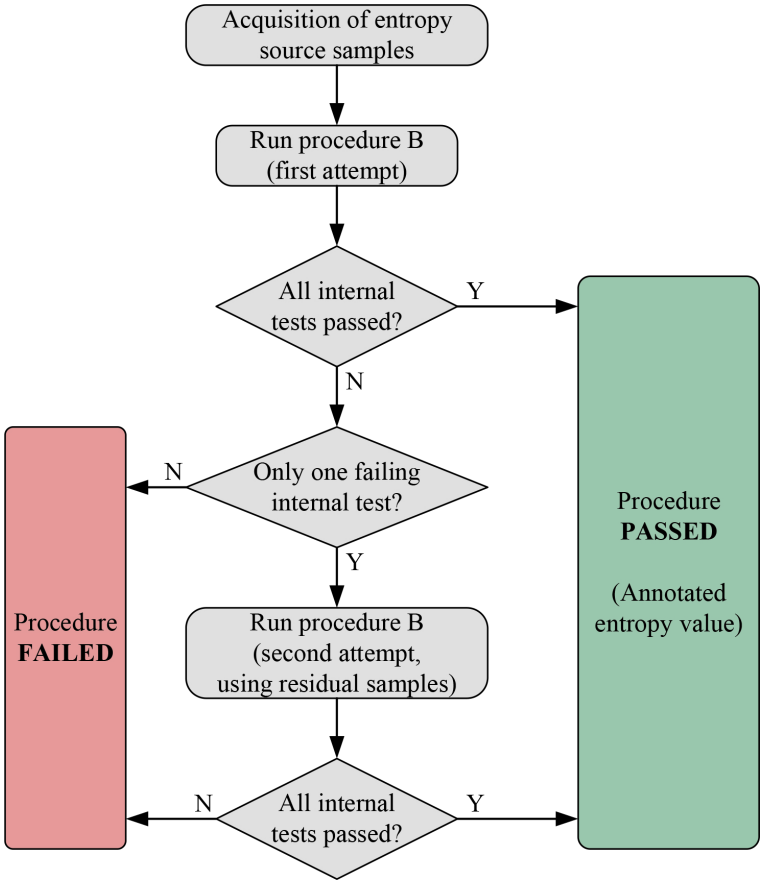

Figure 2. Flow diagram for entropy assessment when using the BSI suite (procedure B).

Concerning the number N of random samples to be acquired, a specific value cannot be defined because it depends on the value of acquired bits that are partitioned based on specific patterns. In any case, the minimum amount of bits to be collected can be extracted from the documentation of the BSI suite [17], and it is 6,968,480 bits (≈7 Mb). Considering that the testing procedure can also be applied a second time if the first attempt fails, when collecting the output samples for the entropy module, we suggest collecting at least double the amount of data, i.e., at least almost 14 Mb.

3.2. Entropy Assessment with the NIST EA Suite

Similarly to procedure B of the BSI tool, the NIST EA suite takes as input the collection of samples acquired from the entropy source, performs tests that need to be configured according to the characteristics of the implementation, and provides as its final and main output a measure of the entropy. Released as C source files in [18], it can be easily compiled on a host platform to generate the corresponding executable to be used for the assessment of entropy. This suite relies on an overall implementation model that is illustrated in Figure 3 and consists of:

- an analogue source of noise (i.e., the analogue entropy source);
- a block for the digitisation of the analogue source of noise;
- an optional conditioning block (for post-processing of the digitised source of noise);
- an optional (parallel) unit for online health tests.

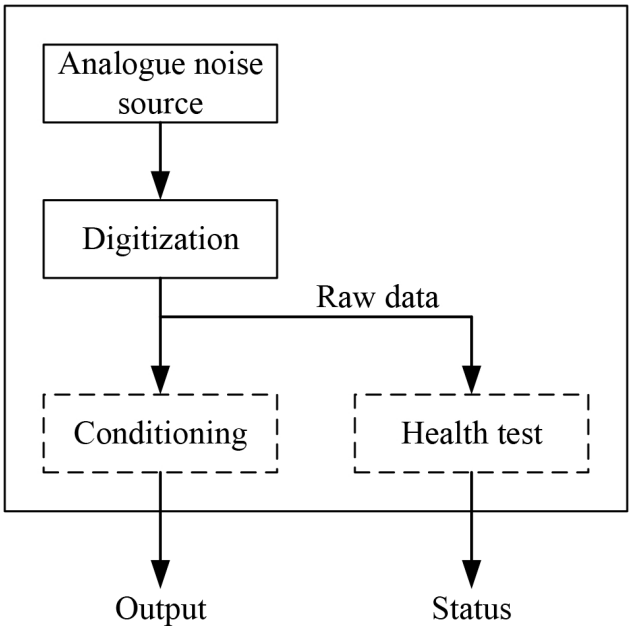

Figure 3. Overall implementation model for the application of the NIST EA.

According to the model in Figure 3, the NIST EA suite provides the following four distinct tools (i.e., executables) for testing the different sections and data of an entropy source module:

1. EA Independent and Identically Distributed (IID) (*ea_iid* executable);
2. EA non-IID (*ea_non_iid* executable);
3. EA *Restart* (*ea_restart* executable);
4. EA *Conditioning* (*ea_conditioning* executable),

The first two executables are mutually exclusive and must be applied to the raw data generated by the digitisation block as the first step of the assessment procedure to determine which of them best fits the acquired samples; then, the other executables have to be applied in that order while also using the output of the previous step that was updated as input. In other words, it is expected that after the first step, the other steps of the entropy measurement process progressively update the entropy value determined by the previous step.

To determine if the IID or the non-IID tool must be used, as a preliminary step, the IID assumption is made by applying the IID track, i.e., the sequence of executables *ea_iid*, *ea_restart*, and *ea_conditioning*. Each of these programs provides not only an entropy estimation, but also a result based on a pass/fail criterium; if all of the programs return the result *passed*, the IID assumption is verified, and the last entropy estimation corresponds to the entropy measure. The *ea_conditioning* executable has to be used only if the optional conditioning block is included within the implementation of the entropy source module. In the opposite case, i.e., if any of the programs of the IID track produce the result *failed*, the non-IID track has to be used; therefore, the sequence of executables to be applied is *ea_non_iid*, *ea_restart*, and (eventually) *ea_conditioning*. Figure 4 illustrates a flow diagram for the usage of the NIST EA suite that was described.

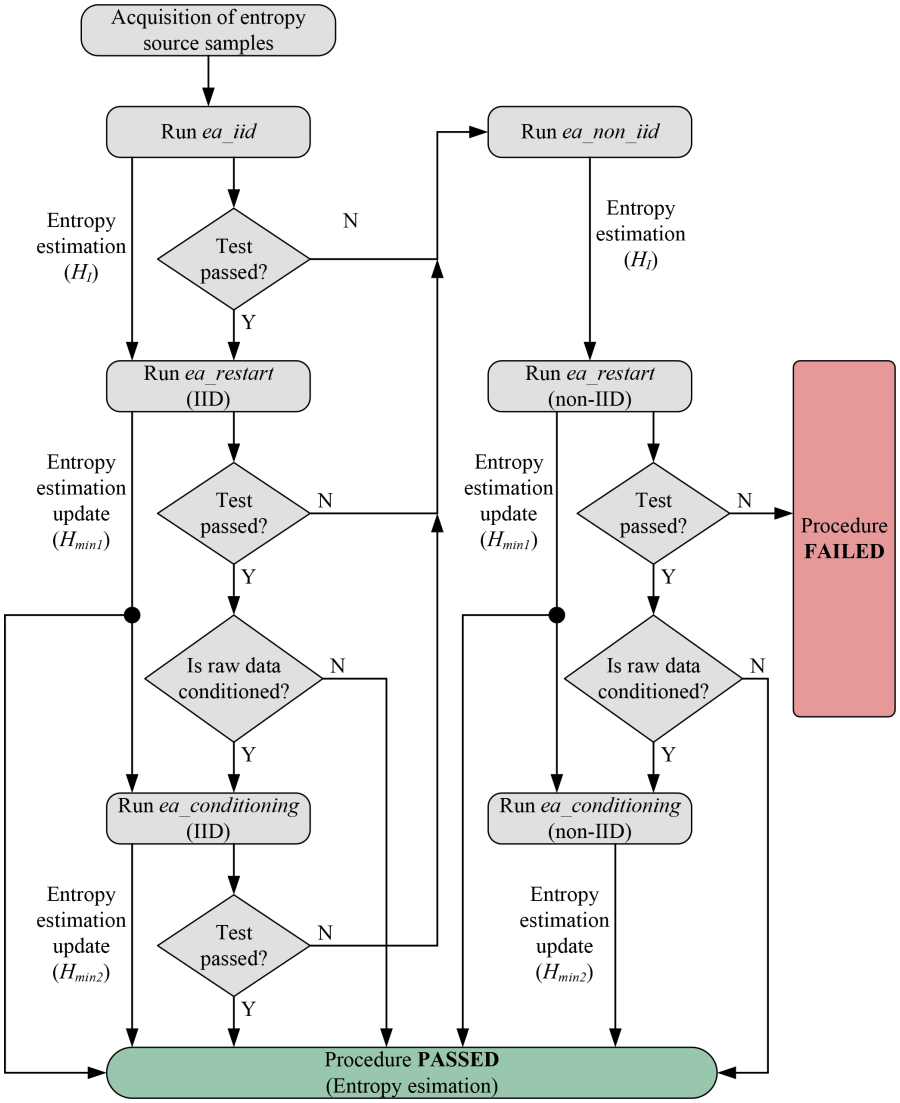

Figure 4. Flow diagram for entropy assessment when using the NIST EA suite.

Regardless of which track is used—IID or non-IID—the same constraints and configurations have to be used for the measurement of the entropy, and similar processes are performed according to the following steps:

1. first entropy (H_I) estimation by using the *ea_iid* executable (or *ea_non_iid*);
2. second entropy (H_{min_1}) estimation by using the *ea_restart* executable to update H_I;
3. third entropy (H_{min_2}) estimation by using the *ea_conditioning* executable to update H_{min_1}.

For the first entropy estimation, a sequence of 1,000,000 random samples has to be collected and provided as input to the *ea_iid* executable (or *ea_non_iid*) by also specifying the bit width b of the random samples. If $b = 1$, only the entropy value ($H_{original}$) is calculated, returning $H_I = H_{original}$; if $b > 1$, the entropy value $H_{bitstring}$ is also computed by using the first 1,000,000 bits and the output value if $H_I = min(H_{bitstring}, b \cdot H_{bitstring})$, i.e., the minimum value between $H_{bitstring}$ and $b \cdot H_{bitstring}$. If the implemented entropy module is not able to generate 1,000,000 consecutive samples, smaller sets from the same entropy source can be concatenated to form an overall dataset of at least 1,000,000 samples, for which each smaller set contains at least 1000 samples.

For the second entropy estimation, the input data to be provided to the *ea_restart* executable have to be formatted as a 1000×1000 matrix. Each row of this matrix corresponds to a different realisation of 1000 samples after the reset (or system power-on). In addition,

in this case, the bit width b of the random samples must be specified, and the entropy estimation from the previous step (H_{min_1}) has to be provided as input. The *ea_restart* executable calculates an entropy estimation per row (H_r) and an entropy estimation per column (H_c), and finally, it returns $H_{min_1} = min(H_I, H_r, H_c)$.

For the third (optional) entropy estimation, no file must be provided as input to the *ea_conditioning* executable, but its usage (necessary only if the conditioning block is integrated within the implementation of the entropy module) requires as input only the entropy estimation from the previous step (i.e., H_{min_1}) and the configuration of some parameters indicating the bit width of the input data of the conditioning block and the bit width of the output data of the conditioning block. Based on these values, the entropy estimation H_{min_2} is automatically calculated.

Hence, the final entropy estimation is the value called *min-entropy*, which is $H_{min} = H_{min_1}$ in the case of no conditioning or $H_{min} = H_{min_2}$ in the case of conditioning.

As an example, in Listings 2 and 3, the output of the *ea_iid* executable and the output of the *ea_restart* executable are, respectively, reported, with both being applied to our entropy source module [15,16] according to the illustrated procedure for the IID track. Both outputs show the results of the tests dedicated to the verification of the IID assumption, which is confirmed. In addition, in Listing 2, one can note, for instance, the calculation of the values of $H_{original}$ and $H_{bitstring}$, which determine H_I (in Listing 3), according to the analysis of random samples with a bit width of 8 bits, while Listing 3 also shows the values of H_r and H_c, for which a final entropy value of 7.888 (bits per byte) was obtained. This value, i.e., the output of the *ea_restart* test, is the final entropy value for our implementation [15,16] because we did not include a conditioning block at the output of the digitisation one and, thus, directly used the raw data on the output of this last block as the output of the whole entropy source module.

Listing 2. Output of the *ea_iid* test of the NIST EA suite. The elements H_original and H_bitstring correspond, respectively, to the values of $H_{original}$ and $H_{bitstring}$ in the text.

```
Calculating baseline statistics...
H_original: 7.892085
H_bitstring: 0.997924
min(H_original, 8 X H_bitstring): 7.892085

** Passed chi square~tests

** Passed length of longest repeated substring~test

Beginning initial tests...
Beginning permutation tests... these may take some time
** Passed IID permutation tests
```

Listing 3. Output *ea_restart* test of the NIST EA suite. The elements H_I, H_r, and H_c correspond, respectively, to the values of H_I, H_r, and H_c in the text.

```
H_I: 7.892085
ALPHA: 5.0251553006530614e-06, X_cutoff: 20
X_max: 16

Running IID~tests...

Running Most Common Value~Estimate...

H_r: 7.888291
H_c: 7.888291
H_I: 7.892085

Validation Test~Passed...

min(H_r, H_c, H_I): 7.888291
```

3.3. Shannon Entropy

The entropy values produced by both procedure B of the BSI suite and the NIST EA suite are estimations of the *min-entropy* H_{min}, which is the most conservative measure of the unpredictability of a set of outcomes, as the negative logarithm of the probability of the most likely outcome. However, based on the application of the implemented entropy source module, it can be necessary to extract another metric that is related to this one. The second metric that we are referring to is the Shannon entropy H_S, which can be calculated from the *min-entropy* by using Equation (1):

$$H_S = -2^{-H_{min}} \cdot \log_2\left(2^{-H_{min}}\right) - \left(1 - 2^{-H_{min}}\right) \cdot \log_2\left(1 - 2^{-H_{min}}\right) \quad (1)$$

For instance, the BSI also specifies for the RNG classes defined in its standard [17] a minimum requirement in terms of the Shannon entropy to be met to claim the validity of the realised module.

The formula for deriving the Shannon entropy must be applied to the *min-entropy* normalised to the bit unit, i.e., if the H_{min} value from the BSI suite or the NIST EA suite gives a measure of entropy samples with a bit width higher than 1 bit, this H_{min} value must be normalised by dividing it by the bit width of the entropy samples. For instance, in our case study, we obtained the values 7.999 and 7.888, which correspond, respectively, to a normalised *min-entropy* of $7.999/8 = 0.999$ and $7.888/8 = 0.986$. Applying the formula for the Shannon entropy, the corresponding H_S value (the effective values of H_S are 0.99993141 and 0.99999965, respectively, for the H_{min} values of 0.999 (procedure B of the BSI suite) and 0.986 (NIST EA), and both can be rounded to the value of 1.000 when using four significant digits for representing those numbers) is 1 in both cases. Ref. [17] specified the minimum value of 0.997 for the class *PTG.2* in terms of the Shannon entropy (which corresponds to the value of 0.910 in terms of the *min-entropy*). Therefore, our implementation in [15,16] can be claimed to be a suitable candidate for the *PTG.2* class of the BSI.

4. Procedure for Randomness Assessment

The procedure illustrated herein is aimed at the assessment of randomness or, in other words, giving a measure of how much the output bitstream generated by the implemented module is distinguishable concerning an ideal RNG. Therefore, the samples to be collected and used as input for the assessment procedure must be extracted from the final output of the modules whose purpose is the generation of random numbers, not the provision of bits of entropy.

According to what was expressed in Section 1, to evaluate the output bitstreams of RNGs, the reference tools to be used are procedure A of the BSI suite and the NIST STS put out by the corresponding standardisation organisations. Similarly to the process for the assessment of entropy, these tools require the collection of the output samples from the implemented RNG, which must be provided as an input file; consequently, they generate an output report that gives measures about the quality of the random sequences. Without entering into the mathematical background of the statistical tests that these suites rely on, it is sufficient to focus on the fact that the procedure essentially compares the results obtained by applying a battery of tests on the random sequences concerning the results that an ideal RNG would show. If the results of the tested RNG are equal to or better than those of an ideal RNG, the implemented RNG can be claimed to be able to generate random numbers that can be employed in cryptographic applications with a certain confidence level. In particular, both the BSI suite and the NIST STS rely on the following two parameters:

- the *p-value* , which is defined as *the probability that a perfect random number generator would have produced a sequence that is less random than the tested sequence*;
- the significance (or confidence) level, which is denoted by α.

A claim on the randomness of a sequence can be determined based on the relation between the *p-value* and α; if *p-value* $\geq \alpha$, then the sequence can be considered random with

a confidence of $(1 - \alpha) \cdot 100\%$; otherwise, if *p-value* $< \alpha$, the sequence can be considered as non-random with a confidence of $(1 - \alpha) \cdot 100\%$. For cryptographic applications, α is typically chosen in the range of $[0.001 - 0.01]$; hence, for example, assuming $\alpha = 0.01$, if *p-value* $\geq \alpha$, then the sequence can be considered random with a confidence of 99%; otherwise (*p-value* $< \alpha$), the tested sequence can be considered non-random with a confidence of 99%. In the case of $\alpha = 0.001$, the confidence that the tested sequence is random rises to 99.9%; if *p-value* $\geq \alpha$ and, accordingly, if *p-value* $< \alpha$, 99.9% is the confidence that the tested sequence can be considered non-random.

4.1. Randomness Assessment with Procedure A of the BSI Suite

Procedure A of the BSI suite for the evaluation of randomness operates similarly to procedure B that is included in the same suite, but it is applied to the output sequences of RNGs whose purpose is to generate random numbers (not to provide bits of entropy). The same tool as that shown in Section 3.1 and Figure 1 has to be used—hence, with the same configuration options—but in this case, the tests to be selected are not *T6* to *T8*, but test *T0* and the group of tests *T1* to *T5*. Then, the tool works as in the case of entropy assessment by loading the required samples into RAM, saving the residual samples in a dedicated file, and providing an output report both in the text box under the progress bar and as a textual file.

The first difference concerning the entropy evaluation case is that this procedure (denoted as procedure A) consists of two different steps, with each one requiring a different number of random input bits that need to be run manually:

1. execution of test *T0*;
2. execution of the group of tests *T1*–*T5*.

The first step (test *T0*) takes 3,145,728 bits as input and saves the remaining bits in a dedicated file for successive usage with tests *T1* to *T5*. The second step takes an input of 5,160,000 bits. Similarly to procedure B, if the first attempt (*Normal Test* type) does not report any failing tests, the procedure can be considered completed and successful. In addition, the BSI also specifies that the confidence level of the results is 99.87%, that is, the tested RNG cannot be distinguished from an ideal RNG with a confidence of 99.87%. Otherwise, if only one intermediate test of the entirety of procedure A fails, the same procedure can be applied for a second and last time, but on different samples with respect to those used in the first attempt, and with the application of the same considerations for the results that were applied to those of the first run. Instead, if more than one intermediate tests fail, the procedure has to be considered failed. Based on this, a flow diagram for the randomness assessment by using procedure A of the BSI suite can be derived, and it is illustrated in Figure 5.

As an example, we applied the procedure described herein to the DRBG mechanism that we implemented for the realisation of the Cryptographically Secure Pseudo-Random Generator (CSPRNG), which we illustrated in [19,20] and exploited in [21]. The corresponding outcomes of the two steps of procedure A are reported in Listings 4 and 5. Concerning the first one, which is related to test *T0*, the whole output is reported, while in the second one, which is related to the group of tests *T1* to *T5*, we transcribed only the initial part of the output report, and the final row reports the overall outcome of the test. This is because each of the tests from *T1* to *T5* was repeated 257 times, and the whole report would require a huge number of pages (indeed, it occupies 8000 rows, more or less). However, both listings show that the results of these case studies are all positive.

With reference to Listings 4 and 5, similar remarks can be made to those made for procedure B of the BSI. For instance, a way to count the number of failing (intermediate) tests is to count all of the passed ones (by searching for the matches of the keyword *passed* within the log) and to check the difference between this number and 1543. This is because, in the case of a successful outcome, the total number of occurrences of a *passed* keyword is $1 + (5 \cdot 257) + 257 = 1543$, respectively, for test *T0* (single test, performed only one time),

the 257 repetitions of the five tests (*T1* to *T5*), and an additional 257 due to the overall result reported for each repetition (e.g., row 29 and row 35 in Listing 5).

A final consideration can be made concerning the number of random samples to be collected. In the examples reported in Listings 4 and 5, we used different sample data for *T0* and *T1–T5*, but the same dataset could be used for both steps because they performed independent tests. Therefore, the minimum amount of data required to run procedure A of the BSI suite can be calculated as the maximum between the number of bits required by the first step, i.e., 3,145,728, and those required by the second step, i.e., 5,160,000, which is, indeed, 5,160,000. By also including the possibility of an eventual repetition of procedure A (in case only one intermediate test fails in the first attempt), this number should be doubled; hence, we suggest collecting at least 10 million samples, which, in our case, was about 10 Mb, with each sample being 1 bit.

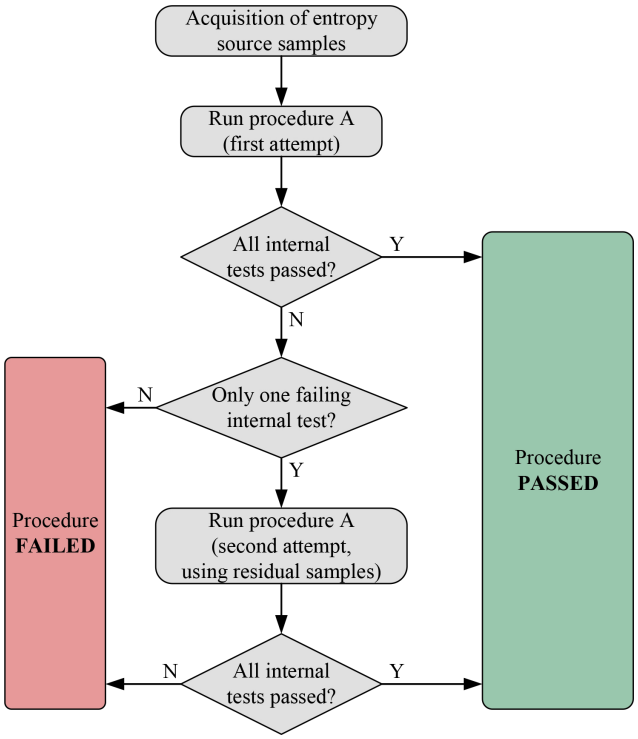

Figure 5. Flow diagram for randomness assessment when using the BSI suite (procedure A).

Listing 4. Output of procedure A (test *T0*) of the BSI suite.

```
TEST STARTED.
TEST-SUITE:       Procedure A/T0
FILE NAME:        drbg_samples.bin
OUTPUT DETAILS:   Enabled.
DATA FORMAT:      1 file byte = 8 random bits (1 random byte).
TEST TYPE:        Normal Test.
RND-Bit Width:    1 bit.
    Reading file.
        drbg_samples.bin
        Copying bit stream file to RAM...
        Converting data file to byte stream...
        Writing residual file: drbg_samples.bin_rest
        3,145,728 elements copied into RAM.
    File reading completed.
    Run Test T0 (Disjuncture Test); Criterion Procedure A.i(i)
    Test T0 passed.
Step successfully completed; remaining file selected for test
```

Listing 5. Output of procedure A (tests *T1–T5)* of the BSI suite.

```
TEST STARTED.
TEST-SUITE:       Procedure A/T1-T5
FILE NAME:        drbg_samples.bin_rest
OUTPUT DETAILS:   Enabled.
DATA FORMAT:      1 file byte = 8 random bits (1 random byte).
TEST TYPE:        Normal Test.
RND-Bit Width:    1 bit.
    Reading file.
        drbg_samples.bin_rest
        Copying bit stream file to RAM...
        Converting data file to byte stream...
        Writing residual file: drbg_samples.bin_rest_rest
        5,160,000 elements copied into RAM.
    File reading completed.
    Step 1 from 257 begins.
    Test within block.
        Run Test T1 (Monobit Test);  Criterion Procedure A.i(ii)
            1s number: 10030
        Permitted range: [9655; 10345]
        Test T1 passed.
        Run Test T2 (Poker Test);  Criterion Procedure A.i(ii)
            Test Statistic = 10.400000000000546
        Test T2 passed.
        Run Test T3 (Run Test);  Criterion Procedure A.i(ii)
        ...
        ...
        ...
        Test T5 passed.
    Step 1 passed.
    Step 2 from 257 begins.
        ...
        ...
        ...
        Test T5 passed.
    Step 257 passed.
Step successfully completed; remaining file selected for test
```

4.2. Randomness Assessment with the (Fast) NIST STS Suite

For the evaluation of randomness with the NIST STS suite, an additional parameter must be defined: the number of sequences to be tested. Indicating this with k, this parameter must satisfy the constraint $k \geq 1/\alpha$; in other words, if $\alpha = 0.01$ is set, at least 100 sequences have to be tested, while for $\alpha = 0.001$, at least 1000 have to be tested, and so on. Hence, based on the *p-value*, α, and k, the NIST STS computes the metrics PRoportion (PR) and *p-value* of *p-values* (PoP) according to the following steps:

- The *p-value* of each sequence is calculated, and the sequences for which *p-value* $< \alpha$ are discarded;
- The PR value is calculated as the ratio between the number of sequences that passed the test (*p-value* $\geq \alpha$) and the total number of tested sequences (k);
- The *p-values* of sequences that passed the test are distributed in the range $[0, 1)$ by splitting it into 10 equal sub-intervals named C1, C2, C3, and so on, up to C10, respectively, for ranges $[0, 0.1)$, $[0.1, 0.2)$, $[0.3, 0.4)$, ..., up to $[0.9, 1.0)$, and the uniformity of the *p-values*' distribution is calculated by exploiting the chi-square (other notations of this function are chi-squared or χ^2) function. This measure of uniformity corresponds to the PoP value.

This procedure is performed for each statistical test included within the NIST suite, and all results are collected in the report file *finalAnalysisReport.txt*. A test is considered passed (with a confidence of $(1 - \alpha) \cdot 100\%$) if both of the following conditions are satisfied:

- The PR value lies in the confidence interval defined as $(1 - \alpha) \pm 3\sqrt{\frac{\alpha(1-\alpha)}{k}}$;
- PoP ≥ 0.0001.

Any failing test is highlighted in the report file with a "*" symbol near the test results. The NIST indicates that tested sequences should be considered random if all tests are passed and that additional tests should be performed on different sets of data, similarly to the repetition of procedure A or B of the BSI suite. Anyway, this specification is too stringent, as analysed by the authors of [22]. They investigated the NIST STS in depth by analysing more than 100 GB of data produced by a physical quantum random number generator, and they concluded that an ideal random number generator also has a high probability (about 80%) of failing at least one test of that statistical suite. To overcome this issue, they defined several tolerated failing tests, in addition to a more accurate confidence interval (using the new constant value, the formula for the calculation of the confidence interval for the PR metric becomes $(1-\alpha) \pm 2.6\sqrt{\frac{\alpha(1-\alpha)}{k}}$) for the PR metric, by substituting the constant 3 with the value 2.6. For instance, when testing $k = 1000$ sequences with a significance level of 1% ($\alpha = 0.01$), three (six) failing tests are admitted for the PR metric when using the NIST's (proposed) confidence interval that includes the constant 3 (2.6) in its formula. The authors of [22] also proposed an alternative optimised implementation of the NIST STS called *Fast* NIST STS, which promises an overall speedup of 30 times with respect to the original NIST version, and they released it in a git repository at [23]. After conducting some comparison tests by using the reference vectors included in both of the STS implementations, we found that the speedup claimed by the authors for *Fast* NIST STS is effective and significant, in addition to being reliable because it generates the same numerical results as those calculated by the original version. For this reason, we strongly suggest using this optimised version of the NIST STS.

Both the original and the optimised version of the NIST STS were released as C source files, similarly to the NIST EA presented in Section 3.2, and they can be downloaded at [24] and [23], respectively. Once compiled, the program can be run as any other executable by using the command line of the host platform, and before starting the execution of tests, it asks for some parameters, which are:

- the input file containing the samples;
- the number of sequences (k);
- the bit length of input sequences (n);
- the block length (M) for the *Block Frequency Test*;
- the block length (m) for the *NonOverlapping Template Test*;
- the block length (m) for the *Overlapping Template Test*;
- the block length (m) for the *Approximate Entropy Test*;
- the block length (m) for the *Serial Test*;
- the block length (M) for the *Linear Complexity Test*.

For the definition of the values to apply to such parameters, it is useful to refer to the indications provided by the BSI in [25], which recommended a set of values for reliable results. These values are:

- $k = 1073$ (for $\alpha = 0.01$);
- n = 1,000,000;
- M (Block Frequency Test) = 20,000;
- m (NonOverlapping Template Test) = 10;
- m (Overlapping Template Test) = 10;
- m (Approximate Entropy Test) = 8;
- m (Serial Test) = 16;
- M (Linear Complexity Test) = 1000.

Therefore, the flow diagram for the usage of the (Fast) NIST STS can be drawn, as illustrated in Figure 6.

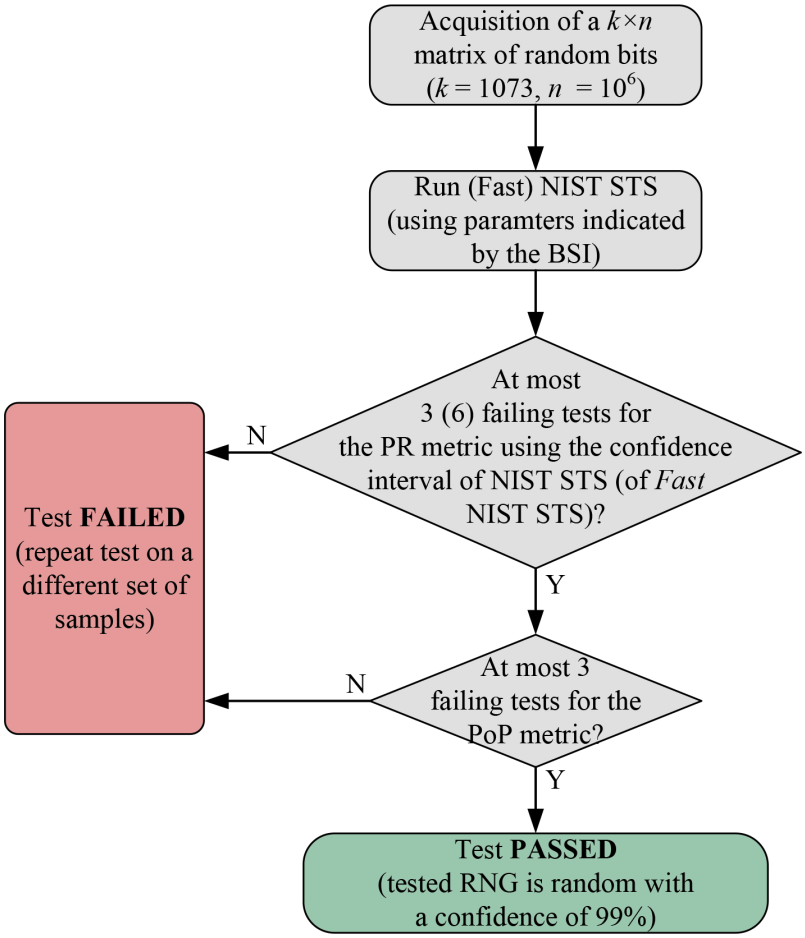

Figure 6. Flow diagram for randomness assessment when using the *Fast* NIST STS suite for a significance level of $\alpha = 0.01$.

In Figure 6, a flow diagram for the randomness assessment of RNGs when using the (Fast) NIST STS is shown for the case in which $\alpha = 0.01$. The use cases for different significance levels α can be derived by referring to [22,25] and using the collection of several sequences $k = 10.73 \cdot (1/\alpha)$, each with 10^6 bits, as a rule of thumb.

We applied the presented procedure to the DRBG implemented in [19,20] and obtained, respectively, zero and five failing tests for the PR metric when using the original NIST STS confidence interval and the more stringent one proposed by the authors of [22], as shown in Figure 7. For that graph, the former confidence interval corresponds to the widest one (dark blue dashed lines), while the latter corresponds to the narrowest one (light-blue dashed lines). In addition, for some tests, a different confidence interval is depicted. This depends on the fact that those tests (automatically) used a different number of sequences (k') that was derived from the specified one (k). However, the test passed in both cases, and 6 was the number of tolerated failing tests for the PRs metric when using the more stringent confidence interval. In addition, for the PoP metric, the test passed, and zero failing tests were reported, as shown in Figure 8.

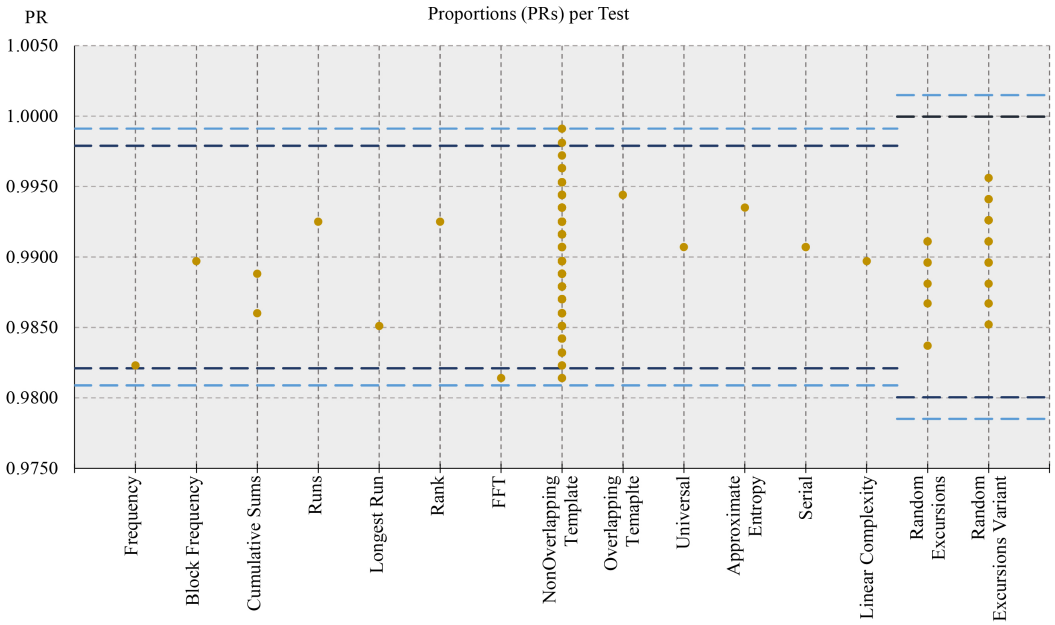

Figure 7. Graph of the results for the PR metric for the tested DRBG [19,20]. The dark golden points represent the PR values for each test or sub-test, and the blue dashed lines represent the boundaries of the confidence interval(s). The widest confidence interval is that of the original NIST STS suite, while the narrowest one is that proposed by the authors of the *Fast* NIST STS.

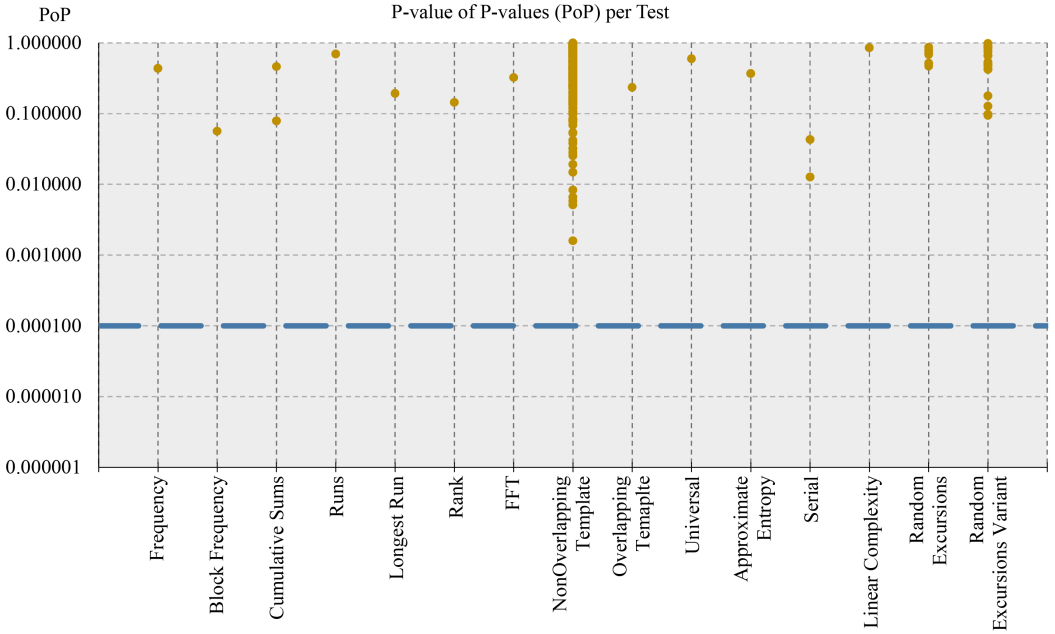

Figure 8. Graph of the results for the PoP metric for the tested DRBG [19,20]. The dark golden points represent the PoP values for each test, and the light-blue dashed line traces the threshold for the pass/fail criterion. The vertical axis is on a logarithmic scale.

As a further element for testifying the outcome of the PoP metric's outcome, in Figure 9, we illustrate the distribution of the tested sequences for the single-experiment tests only, i.e., those that were not composed of multiple sub-tests. In addition, with a rapid and visual inspection, it can be noted that these distributions are uniform.

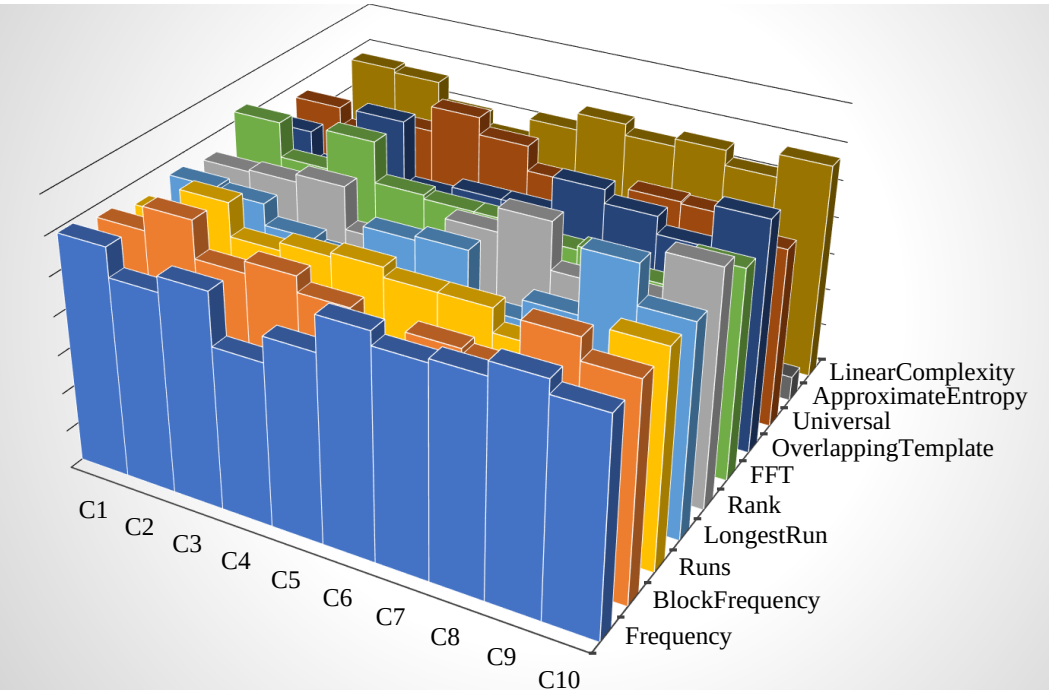

Figure 9. Histograms of the *p-values'* distributions. The (three-dimensional) histograms of the distributions are reported only for the single-experiment tests according to the results for the PoP metric (in Figure 8).

5. Conclusions

This work presents a brief but comprehensive review of the main tools and methodologies for constructing and evaluating RNGs according to the most stringent metrics for cybersecurity purposes that have been released by the most reliable reference standardisation organisations. We illustrated all of the required terms and methods for identifying, organising, and classifying the architecture of a RNG that is intended to be implemented and, correspondingly, the assessment procedures to be used according to the scope of the implemented RNG, i.e., for the generation of bits of entropy for the generation of random numbers. We also included the reference links that can be used to download the assessment suites from the corresponding reference organisations (NIST and BSI) and all of the required indications for a quick but exhaustive evaluation of the results obtained when evaluating the output of the implemented RNG. The presented contribution is expected to help designers in assessing the quality of their RNGs in detail so that they will be facilitated in ensuring quality, avoiding biases, improving the design, meeting industry standards, and enhancing their reputations. Any RNGs developer can use this work as a guideline for properly characterising a module that is implemented for random number applications and to determine the quality of their work. Following the presented procedure, a developer can claim compliance with the most qualified high-security requirements specified by the corresponding reference cybersecurity standards without the necessity of investigating and analysing the mathematical background and the statistical formulas used for the evaluation of the output of the implemented module.

Author Contributions: Conceptualisation and methodology, L.C., S.D.M. and P.N.; validation, L.C.; project administration and funding acquisition, S.S. and L.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the European Union's Horizon 2020 research and innovation programme *European Processor Initiative* EPI SGA2 under grant agreement no. 101036168 (EPI SGA2), it was partially supported by the Italian Ministry of University and Research (MUR) by means of the Recovery and Resilience Plan (PNRR) project CN4—CN00000023 under the grant

agreement no. I53C22000720001., and partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the FoReLab project (Departments of Excellence).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This work was partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the FoReLab project (Departments of Excellence).

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

References

1. Simion, E. Entropy and Randomness: From Analogic to Quantum World. *IEEE Access* **2020**, *8*, 74553–74561. [CrossRef]
2. Lin, Y.; Wang, F.; Liu, B. Random number generators for large-scale parallel Monte Carlo simulations on FPGA. *J. Comput. Phys.* **2018**, *360*, 93–103. [CrossRef]
3. Ergun, S. Security analysis of a chaos-based random number generator for applications in cryptography. In Proceedings of the 2015 15th International Symposium on Communications and Information Technologies (ISCIT), Nara, Japan, 7–9 October 2015; pp. 319–322. [CrossRef]
4. Ryabko, B. Time-adaptive statistical test for random number generators. *Entropy* **2020**, *22*, 630. [CrossRef] [PubMed]
5. Marsaglia, G. The Marsaglia Random Number CDROM Including the Diehard Battery of Tests of Randomness. 2008. Available online: http://www.stat.fsu.edu/pub/diehard/ (accessed on 2 January 2023).
6. Sleem, L.; Couturier, R. TestU01 and Practrand: Tools for a randomness evaluation for famous multimedia ciphers. *Multimed. Tools Appl.* **2020**, *79*, 24075–24088. [CrossRef]
7. Walker, J. ENT—A Pseudorandom Number Sequence Test Program. Available online: https://www.fourmilab.ch/random/ (accessed on 30 November 2022).
8. Random Bit Generator Tester (RaBiGeTe)—Random Bit Generators Tester. Available online: http://cristianopi.altervista.org/RaBiGeTe_MT/ (accessed on 30 November 2022).
9. NIST. *Recommendation for the Entropy Sources Used for Random Bit Generation*; SP 800-90B; NIST: Gaithersburg, MD, USA, 2018.
10. BSI. Implementation of Test Procedure A and Test Procedure B for Application Notes and Interpretation of the Scheme (AIS) 20/31 Standard. Available online: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_testsuit_zip.zip (accessed on 30 November 2022).
11. NIST. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; SP 800-22; NIST: Gaithersburg, MD, USA, 2010.
12. NIST. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*; SP 800-90A Rev. 1; NIST: Gaithersburg, MD, USA, 2015.
13. NIST. *Recommendation for Random Bit Generator (RBG) Constructions*; SP 800-90C (Draft); NIST: Gaithersburg, MD, USA, 2016.
14. BSI. A Proposal for Functionality Classes for Random Number Generators. Available online: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Certification/Interpretations/AIS_31_Functionality_classes_for_random_number_generators_e.pdf?__blob=publicationFile&v=4 (accessed on 30 November 2022).
15. Nannipieri, P.; Di Matteo, S.; Baldanzi, L.; Crocetti, L.; Belli, J.; Fanucci, L.; Saponara, S. True Random Number Generator Based on Fibonacci-Galois Ring Oscillators for FPGA. *Appl. Sci.* **2021**, *11*, 3330. [CrossRef]
16. Crocetti, L.; Di Matteo, S.; Nannipieri, P.; Fanucci, L.; Saponara, S. Design and Test of an Integrated Random Number Generator with All-Digital Entropy Source. *Entropy* **2022**, *24*, 139. [CrossRef] [PubMed]
17. Killmann, W.; Schindler, W. A proposal for: Functionality Classes for Random Number Generators, Version 2.0. In *Mathematical-Technical Reference of AIS 20/31*; T-Systems GEI GmbH: Bonn, Germany, 2011.
18. NIST. SP 800-90B Entropy Assessment Software, Version 1.0. 2019. Available online: https://github.com/usnistgov/SP800-90B_EntropyAssessment (accessed on 30 November 2022).
19. Baldanzi, L.; Crocetti, L.; Falaschi, F.; Belli, J.; Fanucci, L.; Saponara, S. Digital Random Number Generator Hardware Accelerator Intellectual Property (IP)-Core for Security Applications. In *Applications in Electronics Pervading Industry, Environment and Society. ApplePies 2019*; Lecture Notes in Electrical Engineering (LNEE); Springer: Berlin/Heidelberg, Germany, 2020; Volume 627, pp. 117–123. [CrossRef]
20. Baldanzi, L.; Crocetti, L.; Falaschi, F.; Bertolucci, M.; Belli, J.; Fanucci, L.; Saponara, S. Cryptographically Secure Pseudo-Random Number Generator IP-Core Based on Secure Hash Algorithm (SHA)2 Algorithm. *Sensors* **2020**, *20*, 1869. [CrossRef] [PubMed]
21. Nannipieri, P.; Bertolucci, M.; Baldanzi, L.; Crocetti, L.; Di Matteo, S.; Falaschi, F.; Fanucci, L.; Saponara, S. SHA2 and SHA-3 accelerator design in a 7 nm technology within the European Processor Initiative. *Microprocess. Microsyst.* **2021**, *87*, 103444. [CrossRef]
22. Sýs, M.; Riha, Z.; Matyas, V.; Marton, K.; Suciu, A. On the Interpretation of Results from the NIST Statistical Test Suite. *Rom. J. Inf. Sci. Technol. (ROMJIST)* **2015**, *18*, 18–32.

23. Sỳs, M.; Říha, Z. NIST STS Optimised v6.0.1. 2020. Available online: https://github.com/sysox/NIST-STS-optimised/files/4052762/Fast_NIST_STS_v6.0.1.zip (accessed on 30 November 2022).
24. NIST. SP 800-22 STS Software, Version 2.1.1. 2010. Available online: https://csrc.nist.gov/CSRC/media/Projects/Random-Bit-Generation/documents/sts-2_1_2.zip (accessed on 30 November 2022).
25. BSI. *Functionality Classes and Evaluation Methodology for Physical Random Number Generators, Version 1*; AIS 31; BSI: Herndon, VA, USA, 2001.