entropy

Article

Statistical Testing of Random Number Generators and Their Improvement Using Randomness Extraction

Cameron Foreman [1,2,*], Richie Yeung [3,4] and Florian J. Curchod [5]

[1] Quantinuum, Partnership House, Carlisle Place, London SW1P 1BX, UK
[2] Department of Computer Science, University College London, London WC1E 6BT, UK
[3] Quantinuum, 17 Beaumont Street, Oxford OX1 2NA, UK; richie.yeung@quantinuum.com
[4] Department of Computer Science, University of Oxford, Wolfson Building, Parks Rd, Oxford OX1 3QG, UK
[5] Quantinuum, Terrington House, 13–15 Hills Road, Cambridge CB2 1NL, UK; florian.curchod@quantinuum.com
* Correspondence: cameron.foreman@quantinuum.com

Abstract: Random number generators (RNGs) are notoriously challenging to build and test, especially for cryptographic applications. While statistical tests cannot definitively guarantee an RNG's output quality, they are a powerful verification tool and the only universally applicable testing method. In this work, we design, implement, and present various post-processing methods, using randomness extractors, to improve the RNG output quality and compare them through statistical testing. We begin by performing intensive tests on three RNGs—the 32-bit linear feedback shift register (LFSR), Intel's 'RDSEED,' and IDQuantique's 'Quantis'—and compare their performance. Next, we apply the different post-processing methods to each RNG and conduct further intensive testing on the processed output. To facilitate this, we introduce a comprehensive statistical testing environment, based on existing test suites, that can be parametrised for lightweight (fast) to intensive testing.

Keywords: statistical testing; random number generation; randomness extractors; information-theoretic security

check for updates

Citation: Foreman, C.; Yeung, R.; Curchod, F.J. Statistical Testing of Random Number Generators and Their Improvement Using Randomness Extraction. *Entropy* **2024**, *26*, 1053. https://doi.org/10.3390/e26121053

Academic Editors: Andrei Khrennikov and Karl Svozil

Received: 30 October 2024
Revised: 27 November 2024
Accepted: 29 November 2024
Published: 4 December 2024

1. Introduction

The notion of randomness plays an important role in numerous fields, ranging from philosophy to science. In science, it is used in optimisation and numerical integration (e.g., using the Monte Carlo method), algorithm randomisation, or cryptography. Although there is something universal about the concept of randomness, its definition varies substantially depending on the context in which it is used. In cryptography, for example, random numbers should be *unpredictable*, in the sense that they should be indistinguishable from uniformly distributed and secret ones, even by an adversary potentially possessing information about the random number generator (RNG) that the user does not have. Therefore, randomness—or unpredictability—from the perspective of the RNG user and from the perspective of a hypothetical adversary is fundamentally different. However, if the output of the RNG exhibits patterns that are detectable by the user, then these patterns also imply predictive power from the perspective of the adversary, since the adversary needs to be considered to have at least as much information as the user. In this sense, unpredictability from the user's perspective is a necessary (but not sufficient) condition for the unpredictability of an adversary. This idea motivates the numerical testing of RNGs' outputs, which serves as a means of randomness validation, i.e., to detect failures to generate randomness.

Because numerical testing is a useful implementation check and the only universally applicable method to test different RNGs, it is an essential part of obtaining a cryptographic RNG certified by standards bodies—for example, the National Institute of Standards and Technology (NIST) or the Bundesamt für Sicherheit in der Informationstechnik (BSI). This certification process ensures that the RNG has been constructed and tested following best practices. For nearly all cryptographic applications, companies typically regard certification from a standards body as a prerequisite for RNG usage. The NIST and BSI standards require both the detailed modelling of the underlying physical process and the numerical testing of the RNG's output statistics in order for a hardware RNG to be compliant. This has led to the development of statistical test suites, the best known being NIST's [1] and the Dieharder [2,3] suite, but there are also others, e.g., [4–6]. Together, these test suites enable the comprehensive analysis of a wide array of statistical characteristics. Despite their usefulness, they are often complicated to use and their outputs are challenging to analyse.

To address this, we design a tunable statistical testing environment (STE) by combining existing suites, allowing for adjustable intensity levels that balance the computational cost with effectiveness in detecting statistical bias, enabling more rigorous RNG testing than standard requirements. We make our testing environment openly available at https://github.com/CQCL/random_test (accessed on 28 November 2024). Using the STE's most intense setting, we test the output of three different RNGs representative of those used across commercial applications, allowing us to benchmark them against each other. We also provide a framework to analyse the overall numerical test results generated by the STE, which is not obvious otherwise.

We then present a range of post-processing methods, utilising various randomness extractors, to enhance the quality of each RNG's output by, for instance, removing bias or dependencies between bits. Our set-up is illustrated in Figure 1. Each extraction method is based on distinct assumptions, which can be compared against each other. Examples of these assumptions include specific structures in the RNG's output process, such as assuming that each output bit is generated in an identically and independently distributed (i.i.d.) manner or the availability of a short, pre-existing (near-)perfect random bit string as a resource. To evaluate the effectiveness of each method, we implement them using the Cryptomite software library [7] and apply statistical testing through our STE to analyse the impact of these techniques. These assumptions must be practically justified, and statistical testing is employed to determine whether each post-processing method succeeds in producing statistically sound results.

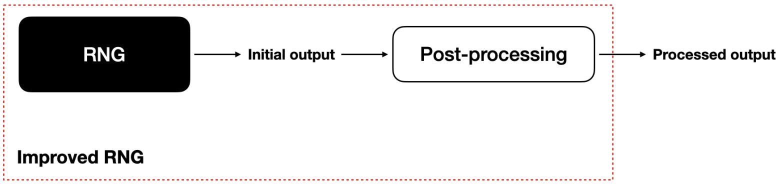

Figure 1. This figure illustrates our implementation set-up. The black box represents one of the initial RNGs that we test, and the dashed box denotes the new—in principle, improved—RNG with additional post-processing applied.

1.1. Related Work

The statistical testing of RNGs has a long history, dating back to the implementation of the Diehard CD-ROM tests in the 1990s [2]. Since then, two main research directions have emerged, which are both relevant to our work. First, researchers developed other test suites, such as the NIST Statistical Test Suite [1], the TestU01 suite [4], ENT [5], and PractRand [6], all of which we utilise for this work. Second, the empirical testing of specific RNGs was performed, such as [8,9], which analysed and compared the results of statistical tests on a variety of pseudo-RNGs (PRNGs). Other works have considered the empirical testing of so-called true-RNGs (TRNGs)—for example, [10], which tests the statistical properties of the entropy source in Intel's Ivy Bridge TRNG, and [11–16], which develop, implement, and statistically test different TRNGs. Recently, this has been extended to quantum RNGs (QRNGs)— for example, [17–21], in which the authors extensively statistical test an ID Quantique QRNG [22]. Other works give a universally applicable RNG statistical testing framework, such as [23,24].

Randomness extraction also has a rich body of literature; see [25] for an introduction. In cryptographic randomness standards, e.g., in NIST's SP 800 [1], so-called *conditioners* are standardised, whose role is similar to that of randomness extractors (randomness extractors can be understood as conditioners that have information-theoretic security, i.e., they do not rely on computational assumptions on the adversary). These conditioners are the only post-processing that has been vetted for use by governing bodies and are the most commonly used as a consequence. To the best of our knowledge, our work represents the first attempt at comparing the effects (from a statistical perspective) of different post-processing methods.

1.2. Summary of Results

In summary, our main results and observations are as follows.

- We perform intense statistical testing of three different RNGs: the 32-bit Linear Feedback Shift Register (LFSR) PRNG, Intel's RDSEED TRNG, and IDQuantique's 'Quantis' QRNG. We show the failure of two of them and provide evidence that one behaves well from a statistical perspective, extending and confirming the results of [10,21].
- We present and implement a variety of post-processing methods, in the form of randomness extractors, to improve the output quality from each of the three RNGs. The post-processing methods form a set of four levels, where each level requires increasingly more sophisticated implementation: deterministic (level 1), seeded (level 2), two-source (level 3), and physical (level 4) extractors. Our contribution goes significantly further than the study and comparison of different types of extractors in [26,27], which focus only on deterministic or seeded extractors, respectively.
- We present and experimentally demonstrate the implementation of a physical extractor (level 4) using a high-fidelity quantum computer. This allows us to execute a complex quantum protocol that has no classical equivalent.
- We intensively test the statistical effect that each level of post-processing has on the output of the different RNGs. Our main observations are as follows.
  - RNGs that fail statistical testing without post-processing continue to fail when simple post-processing methods (level 1) are applied, although some improvement is observed. Notably, one of these failing post-processed RNGs implements the *self-shrinking generator* [28], which is studied for cryptographic applications.
  - All our level 2, 3, and 4 implementations successfully post-process the output of the three RNGs used from a statistical perspective, passing the statistical testing.
  - Low-entropy sources, such as the post-processed 32-bit LFSR, can pass rigorous statistical testing when suitable post-processing is applied. While this result aligns with the existence of cryptographically secure PRNGs, the low quality of the PRNG used highlights the limitations of statistical testing alone, i.e., without a precise model and justification for the unpredictability of the underlying physical process.
- We make publicly available our STE, a powerful, flexible, and easy to use statistical testing environment, together with suggested settings, which provide a valuable trade-off between the intensity and computation time, and a framework to analyse the cumulative test results.

2. Tools and Definitions

As discussed in the Introduction, randomness is different when considering the perspective of a user or that of an adversary. In this work, we study the statistical properties of different RNGs' outputs directly, without considering any additional information that a potential adversary might obtain, such as a detailed model of the entropy source or information from invasive or non-invasive attacks. We then examine, in the same manner, the effects of various post-processing methods applied to the RNG's output. We consider the specific case of bits, i.e., the RNG outputs a bit string of length n, denoted $X \in \{0,1\}^n$, although one can also study RNGs whose output alphabet is larger. In some cases, we refer to sizes in bytes, where one byte equals eight bits. We denote the random variable produced by an RNG as X and its specific realisation as x (i.e., $X = x$). The set-up is illustrated in Figure 2.

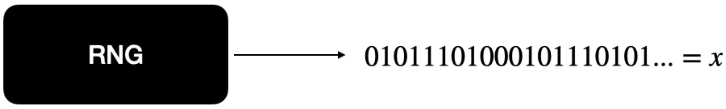

Figure 2. An illustration of the set-up that we consider. An RNG generates a bit string $X = x$ of length n. In this work, we first study the statistical properties of the realisation x of the (random variable) X. Then, we analyse the effects of different post-processing methods applied to it.

The amount of randomness that a random variable X has is captured by its min-entropy $\mathrm{H}_\infty(X)$.

Definition 1 (Min-entropy)**.** *The min-entropy, k, of a random variable, $X \in \{0,1\}^n$, is defined as*

$$k = \mathrm{H}_\infty(X) = -\log_2 \max_{x \in \{0,1\}^n} \Pr(X = x) \,. \tag{1}$$

This can be interpreted as the minimum amount of randomness, in bits, that a variable X has when there is no side information available. The min-entropy rate of a random variable $X \in \{0,1\}^n$ is $\alpha = \mathrm{H}_\infty(X)/n$. This can be interpreted as the minimum amount of randomness that X has per bit, on average. Since RNGs output sequentially, we generalise this definition to consider the min-entropy of the current random variable conditioned on all previously produced random variables. This is known as *block min-entropy*.

Definition 2 (Block min-entropy)**.** *A set of random variables $X_i \in \{0,1\}^{n_i}$ for $i \in \mathbb{N}$ is said to have block min-entropy k_i, if*

$$\mathrm{H}_\infty(X_i | X_0, X_1, ..., X_{i-1}) = -\log_2 \max_{x \in \{0,1\}^{n_i}} \Pr(X_i = x | X_0, X_1, ..., X_{i-1}) \geq k_i, \qquad \forall i \,. \tag{2}$$

This can be interpreted as the minimum number of random bits that a variable X_i has when conditioned on all previous random variables, indexed by $0, \ldots, i-1$.

Definition 3 (Statistical distance)**.** *The statistical distance, Δ, between two random variables, $X, Z \in \{0,1\}^n$, is defined as*

$$\Delta(X, Z) = \frac{1}{2} \sum_{v \in \{0,1\}^n} |\Pr(X = v) - \Pr(Z = v)| \,. \tag{3}$$

This is a measure of how close to one another, or indistinguishable from one another, two random variables are.

Definition 4 (ϵ-perfect randomness)**.** *A random variable X on $\{0,1\}^n$ is said to be ϵ-perfectly random if*

$$\Delta(X, \mathrm{U}_n) \leq \epsilon \,, \tag{4}$$

where U_n is the uniform variable on $\{0,1\}^n$, i.e., $\Pr(\mathrm{U}_n = u) = \frac{1}{2^n}$ for all $u \in \{0,1\}^n$.

This definition is equivalent to saying that the variable X is distinguishable from a uniform distribution with a distinguishing advantage of at most ϵ, i.e., a distinguisher can guess that X is not uniform with a success probability of at most $\frac{1}{2} + \epsilon$. When $\epsilon = 0$, the random variable is said to be perfectly random. This definition is *universally composable* [29], i.e., X can be used safely in other applications.

3. Statistical Testing

Statistical test suites are collections of algorithms that analyse the numerical properties of a set of random numbers to determine whether there is evidence to reject the possibility that they are uniformly distributed. If there is sufficient evidence to reject this possibility, a statistical test is said to be *failed* and the RNG output can be distinguished from the uniform distribution at some confidence level. The hypothesis that a random variable is uniformly distributed is known as the null hypothesis H_0. For an RNG producing a random variable $X \in \{0,1\}^n$, the null hypothesis is $\mathsf{H}_0 : \Delta(X, \mathrm{U}_n) = 0$. If the null hypothesis is rejected, then the alternative hypothesis $\mathsf{H}_1 : \Delta(X, \mathrm{U}_n) > 0$ is accepted.

However, a random variable cannot be tested directly; only its realisation can—i.e., the bit string $x \in \{0,1\}^n$ produced by the random variable X. To assess whether to accept or reject the null hypothesis, a statistical test calculates a specific measure of x (e.g., its mean), known as the *test statistic t*, and analyses how likely this test statistic is to be observed, assuming that the underlying random variable is uniform. Test statistics calculated from realisations of a uniform distribution are normally distributed, so one can calculate how likely observing certain ranges of the test statistic is by using concentration inequalities. More precisely, this likelihood is captured by a probability known as the p-value, which is defined as follows.

Definition 5 (p-value)**.** *Given an observed test statistic t obtained by calculating a measure from the realisation of a random variable $X = x \in \{0,1\}^n$ and T, the (normally distributed) variable associated with all the possible measure values, the* p*-value* $\mathsf{p} \in [0,1]$ *is defined as*

$$\mathsf{p} = \Pr(T \leq t | \Delta(X, \mathrm{U}_n) = 0)\,, \tag{5}$$

where $\mathrm{U}_n \in \{0,1\}^n$ *is uniformly distributed.*

A range of p-values is defined that provides a threshold at which the null hypothesis is rejected, i.e., when the test is deemed to fail. If a test ensures that there is, at most, a 1% chance that it incorrectly rejects that the RNG is producing uniform random numbers (known as the type 1 error—when a statistical test incorrectly rejects a true null hypothesis), then it would, for example, conclude failure if $\mathsf{p} \notin [0.01, 1]$. This threshold for failure is on one tail only, so it only fails test statistics that are sufficiently biased away from the expected value in one direction. More generally, tests are two-tailed and conclude failure if the observed p-values are outside of a sufficiently large interval—for example, if $\mathsf{p} \notin [0.005, 0.995]$.

The failure of numerous statistical tests is a strong indicator that an RNG is not producing (near-)perfect random numbers, as its output can be distinguished from the uniform distribution with a high probability. For example, if all statistical tests performed on the RNG are independent, the probability that the null hypothesis is accepted given that the alternative hypothesis is true (known as the type 2 error) is $\mathsf{p}_{\mathsf{type2}} = \mathsf{p}^{\mathsf{test1}}_{\mathsf{type2}} \cdot \mathsf{p}^{\mathsf{test2}}_{\mathsf{type2}} \cdot \ldots \cdot \mathsf{p}^{\mathsf{testn}}_{\mathsf{type2}}$, where n is the number of tests performed.

We now describe several existing statistical test suites used in this work. It is important to note that, while each suite contains multiple tests, many test outcomes are correlated. For instance, a source with an unusually high number of ones may fail both the monobit and poker tests. Similarly, different test suites often include the same tests with slight parameter variations, meaning that failure in one suite is likely to result in failure in another. For example, both NIST and Dieharder include a runs test.

3.1. Existing Test Suites

3.1.1. NIST Statistical Test Suite

The NIST statistical test suite (SP 800-22) [1] is the best known. It includes 15 tests, some with multiple sub-tests, and passing certain tests is required for RNG certification by organisations such as NIST and BSI. During testing, a randomness file is split into sub-strings, with each sub-string tested individually. Users can specify the number of sub-strings and the total bit string size to analyse, although the guide recommends using 100 sub-strings of 10^6 bits, requiring at least 10^8 bits, or 12.5 MB, for testing. For each test, an analysis is performed on each sub-string, and the suite provides two results: (1) the p-value for a statistical test on the uniformity of the distribution of results across sub-strings and (2) the number of sub-strings that pass each test. Both results are assessed at the 1% significance level. We note that the individual p-values for each sub-string in each test are not accessible to the user.

3.1.2. Diehard(er) Statistical Test Suite

The Dieharder statistical test suite includes the 18 original Diehard tests along with additional tests, including some from the NIST suite. Like the NIST suite, it is widely used by RNG certification bodies. A failure is determined when $\mathsf{p} \notin [0.0005, 1-0.0005]$, and a test is considered 'weak' if $\mathsf{p} \in [0.0005, 0.005] \cup [1-0.0005, 1-0.005]$. This higher tolerance for poor test statistics means that a flawed RNG may occasionally pass Dieharder, but failure is a strong indicator of non-uniformity. The Dieharder tests require a large quantity of random numbers to prevent the re-use of input data, which can lead to inaccurate results. We recommend using at least 1 gigabyte (GB) of random numbers for testing. For smaller file sizes, the test parameters can be adjusted to avoid these issues. In our testing environment, we use the default parameters for each test.

3.1.3. TestU01 Statistical Test Suite

TestU01 is a C-based software library for conducting RNG statistical testing with pre-compiled test batteries. These batteries vary significantly in the number of tests and the amount of randomness required. For details on the specific tests included in each battery, see [4]. Test p-values are reported if $\mathsf{p} \notin [0.001, 0.999]$, which we use as our failure criterion. In our testing, we use the Alphabit, Rabbit, and SmallCrush batteries from TestU01. To run these tests, the input files must contain at least 2^{25} random bytes (approximately 35 MB). We omit the Crush and BigCrush batteries due to their long runtimes and large randomness requirements, although they can be run within our statistical testing environment.

3.1.4. ENT Statistical Test Suite

The ENT test suite is a small but efficient set of six statistical tests. It has been used to demonstrate bias in a commercial quantum RNG by consistently failing the χ^2 test [30] (we replicate these results with an independently acquired RNG; see Table 3. ENT outputs test statistics without providing a pass/fail threshold, so we assess failure based on the criteria in Table 3 of [31]. Although there is no specific guidance on the required input sizes, we found that the tests produced unreliable results with inputs smaller than 0.5 GB.

3.1.5. PractRand Statistical Test Suite

PractRand is a C++ library of statistical tests designed for practicality—they are efficient, user-friendly, and capable of detecting significant biases in RNGs. According to its documentation, it runs faster than most test suites (which we confirm; see Table 2), offers unique interfacing, has no theoretical maximum input length, and includes some original tests. PractRand tests input files based on size, examining subsets of 2^{24+x} bytes for $x \in \mathbb{N}$, with more tests performed as x increases. In our testing, we limit the maximum test size to 2^{32} bytes (approximately 4.3 GB). For full details and comparisons with other test suites, see [6]. PractRand uses various p-value ranges, including "unusual", "mildly suspicious", "suspicious", "suspect", and "fail". A failure occurs when $\mathsf{p} \notin [10^{-11}, 1-10^{-11}]$.

3.2. Our Statistical Testing Environment

The interfacing code for our STE can be downloaded at https://github.com/CQCL/random_test (accessed on 28 November 2024). We offer three testing modes, *Light*, *Recommended*, and *All*, which can be executed using the commands `run_light`, `run_recommended`, and `run_all`, respectively. The NIST statistical test suite is not included in these commands due to its need for user prompts, but it can be run separately within the environment. We believe that the *Recommended* setting strikes a good balance between the computational (and environmental) cost and rigor, exceeding the standard testing required by certification bodies. All results in this work can be replicated using the STE or by downloading, configuring, and running the relevant statistical test suites independently.

3.2.1. Suggested Settings

We now propose the recommended settings for statistical testing using our STE, based on insights gained during this research. The runtimes are averaged over 10 executions using a 10 Gbit file, except for the NIST suite, where a 100 Mbit file is tested, in accordance with the user guidelines. All testing was conducted on a Dell Precision 7540 laptop with 16 GB of RAM and a 2.3 GHz Intel I9-9880H processor, running the Ubuntu 20.04 operating system. All the test runtimes are given in Tables 1 and 2.

Table 1. This table details our settings for light, recommended, and all statistical testing using the code provided. A 'Y' in a specific column indicates that the associated test suite of this column is included in the setting.

Test Mode	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)	Total Runtime	Total Tests
Light			Y	Y			Y	4 m 44 s	941
Recommended	Y	Y	Y			Y	Y	114 m 31 s	999
All	Y	Y	Y	Y	Y	Y	Y	127 m 41 s	1015

Table 2. This table gives the average runtimes of all statistical test suites contained in our statistical testing environment. The average is taken when running each test suite 10 times on independent inputs. For the NIST test suite, the runtime relates to testing a 100 Mbit file. For all other test suites, the runtime is for a 10 Gbit file.

	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
Average Runtime	37 m 3 s	18 m 12 s	1 m 24 s	0 m 32 s	12 m 38 s	55 m 4 s	2 m 48 s

Light

Our suggested light test mode, executed with the command `run_light`, includes ENT, SmallCrush, and PractRand, running in under 5 min and covering approximately 941 tests. Our numerical analysis shows that this set of tests is sufficient to detect failures in generating uniform randomness for RNGs that fail (see Sections 4 and 5).

Recommended

Our recommended setting, executed with the command `run_recommended`, includes most of the light mode test suites, replacing TestU01 SmallCrush with Rabbit and adding the NIST and Diehard tests. These additional tests increase the runtime to approximately 2 h and bring the total number of tests to 999. The full statistical test mode (`run_all`) includes all suites, except for SmallCrush and Alphabit, which are omitted due to their significant correlations with the Rabbit tests, where only the parameters differ slightly. The recommended suite covers all tests required by RNG certification bodies (e.g., NIST and

Dieharder) while offering a more comprehensive analysis than when running these tests alone. The later sections demonstrate the value of extending beyond individual test suites, as an RNG that passes the NIST and Dieharder tests can still show significant statistical bias when analysed with our combined STE (see Section 4).

3.3. Shortcomings of Statistical Testing

Fundamentally, statistical tests have a limited ability to validate that good random numbers are being produced by an RNG. They should rather be understood as a useful tool to detect a failure to generate uniform random numbers, since passing statistical tests gives no guarantee of (near-)perfect randomness. This is especially important in the case of cryptographic RNGs. For example, in [32], a thorough analysis of Intel's RDSEED hardware RNG is performed, and one of their conclusions is that "RDSEED delivers truly random bits but with a security margin that becomes worrisome if an adversary can see a large number of outputs from either interface. If he controls an unprivileged process on the same physical machine, this could happen very quickly" (in this case, the adversary also requires control of an "unprivileged" process, which is a form of side information that may be difficult to obtain in practice) (on page 4). As we shall see next, our statistical testing results do not detect that RDSEED's output can be distinguished from uniform.

At the implementation level, the available software for numerous statistical test suites has been shown to have issues. For the NIST test suite, the list of implementation issues is extensive, so we summarise a few problems that the reader may find interesting. Research has found significant dependencies between the tests [33] and implementation issues with certain tests; for example, [34] found that the settings of both the Discrete Fourier Transform test and Lempel-Ziv test were wrong, and [35] found an error within the probability calculations for the Overlapping Template Matching test. Moreover, problems with how the results are analysed have been discovered; for example, [36] found that although the NIST documentation provides guidance that the analysed RNG is random if all tests are passed, truly uniform data have a high probability (80%) of failing at least one NIST statistical (sub-)test. Some work has even suggested that the tests are "harmful" [37], namely that "the weakest pseudo-random number generators will easily pass these tests, promoting false confidence in insecure systems". During this work, we found an additional issue with the NIST test suite that we could not find reported elsewhere: the results showed that all tests failed whenever the CPU was being used for other computations simultaneously. The NIST Random Bit Generation Team have been made aware of this. Other test suites have also had their own reported problems, including Dieharder. In [38], it was found that over 50% of the Dieharder tests generated biased null hypothesis distributions (which were expected to be uniform).

4. Statistical Testing of Different RNGs

In this section, we use our STE to analyse the statistical properties of the random numbers produced by some commonly used RNGs. At this stage of our analysis, we do not apply any post-processing to the RNG's output; however, some of the RNGs that we consider already have post-processing included, in the form of so-called conditioners or deterministic randomness extractors. Therefore, in cases where the post-processing already exists, our statistical analysis applies to the joint system, which comprises both the randomness (or entropy) source and the existing post-processing in the device. Additionally, we use and discuss a NIST min-entropy estimation tool, which provides a min-entropy estimate for our subsequent analysis incorporating various levels of post-processing.

The RNGs that we analyse are as follows:

- 32-bit LFSR: a software pseudo-RNG;
- Intel RDSEED [11]: a hardware RNG based on thermal noise, i.e., a chaotic process;
- IDQuantique (IDQ) Quantis [22]: a hardware RNG based on the quantum effect of detecting photons at the output of a semi-transparent mirror.

Further details and descriptions of the RNGs can be found in Appendix A.3. The LFSR remains widely used in numerical simulations, despite its well-known flaws [39]. In this work, we primarily use it as a benchmark, serving as an example of a poor choice due to the patterns in its output and its short period. However, we will show that post-processing its output has a statistically significant impact. We note that IDQ's Quantis is marketed as an RNG for cryptographic use, with certifications for compliance with various security and cryptographic standards. (IDQ Quantis has reportedly passed certifications or government validations, including "NIST SP800-22 Test Suite Compliance, METAS Certification, CTL Certification, multiple iTech Labs certificates, and compliance with BSI's AIS31 standard (dedicated version of Quantis)" [22]). These RNGs are used in numerous applications and are a sample of the different types of RNGs available today. The statistical analysis is performed using the `run_all` function in our statistical test environment on 10 × 10 Gbit files from each RNG and, similarly, using the NIST test suite performed on 10 × 100 Mbit files split into 100 sub-strings, each of 1 Mbit. The NIST min-entropy estimators [40] are used in the non-IID setting on 10 × 1 Mbit files. This analysis far exceeds that required by certification bodies, so it may be a result of independent interest. All testing is performed using the default parameters, unless otherwise stated.

An RNG producing near-perfect randomness should pass almost all statistical tests. More concretely, we mean that the ideal RNG would fail less than 7.5 of the 4600 individual statistical tests on average. (This number is the expected amount of type 2 error, i.e., the expected maximum number of failed tests, given that the underlying distribution is indistinguishable from uniform. Note that we are implicitly assuming that each statistical test is independent). The results obtained for the three RNGs are summarised in the following Table 3 and displayed visually in Figure 4 (level 0).

Table 3. This table gives the average sum of statistical tests failed for 5 × 10 Gbit samples from each RNG (after testing 10 samples). The results are presented in this way to allow for direct comparison to later results, where only 5 × 10 Gbit samples are tested. Due to the 32-bit LFSR failing PractRand quickly, only 635 tests were conducted (instead of the full 4600), so we rescale these results. In cells with multiple entries, failed tests are on the left and *suspicious* tests (when applicable) are on the right in parentheses. The full results can be found in Appendix B.

RNG	NIST (75)	Diehard (90)	ENT (30)	SmallCrush (75)	Alphabit (85)	Rabbit (200)	PractRand (4600)
32-bit LFSR	10	40 (3)	5	51	73	131	855 (167)
RDSEED	0	0 (4)	0	1	0	1	0 (7)
IDQ Quantis	0	0 (3)	5	0	17	25	3 (15)

In the statistical tests, the RDSEED RNG performs the best, failing the fewest tests, and the 32-bit LFSR performs the worst, failing the most. The poor performance of the LFSR is likely due to its periodicity, since bits repeat every $2^{32} - 1$ (4.3 Gbits), and this is less than the size of the files tested. The IDQ Quantis device performs well in the NIST and Diehard tests but fails an ENT test and several tests in TestU01's Rabbit and Alphabit suites. These observations reproduce (and add confidence to) the results of previous work [21]. These results, especially for IDQ's device, exhibit the need to go beyond the requirements of certification bodies for statistical testing, with additional tests providing a noticeable advantage in detecting failures.

The NIST min-entropy estimators [40] are a collection of algorithms that give a standardised means of estimating the min-entropy (as defined in Definition 1) of an RNG's output. These estimators are useful both in evaluating the entropy generation of the RNG and in calculating a min-entropy bound, which we later use to determine the parameters for randomness extractors. Although the estimators are designed to test entropy sources without post-processing, this is not feasible in the case of RDSEED or the IDQ Quantis device.

In Table 4, the NIST min-entropy estimator per byte is the average observed per-byte min-entropy calculated by the NIST min-entropy estimator tool, while $\overline{\text{est}}$ is the per-bit average. The sample standard deviation, σ, reflects the variability in the different test results calculated using Equation (A1), and α is a lower bound (with probability of at least $1 - 2^{-32}$) on the per-bit min entropy for any test sample. Details of this derivation can be found in Appendix B.2.1.

Table 4. This table shows the average NIST min-entropy estimator, the sample standard deviation, and a lower bound for min-entropy/bit for each RNG. These results are the average of 10 tests on different 1′000′000 bit samples, each generated with significant time gaps between the generation of each test sample. Full results can be found in Appendix B.2.

RNG	NIST Min-Entropy Estimator (/byte)	$\overline{\text{est}}$: NIST Min-Entropy Estimator (/bit)	σ: Sample Standard Deviation (/bit)	α: Lower Bound Min-Entropy (/bit)
32-bit LFSR	6.870	0.859	0.058	0.453
RDSEED	6.189	0.852	0.022	0.698
IDQ Quantis	7.157	0.895	0.006	0.853

The IDQ Quantis device had the highest estimated min-entropy per bit, with a value of 0.895, although all three RNGs had similar values. The 32-bit LFSR had the largest sample standard deviation, indicating the greatest fluctuation in the min-entropy estimates across different test samples. We note that the NIST SP800-90 series recommends min-entropy per bit of at least $1 - 2^{-32}$ for an RNG to be considered to have full entropy [41]. This value is significantly higher than any values that we observed. However, some of the NIST min-entropy estimator tests are known to produce significant underestimates [42], which potentially explains the large disparity between our estimates and the NIST requirement. Moreover, in our case, underestimates are not problematic, since we desire a lower bound on the min-entropy of the RNG's output.

5. A Variety of Post-Processing Methods

Randomness extractors are mathematical algorithms that distil *weakly* random bit strings (more precisely, a necessary (but not sufficient) condition for randomness extraction to be successful is that the source has some min-entropy; see Definition 1), in the sense that they are not uniformly distributed, into a near-perfect random bit string. In this section, we present, implement, and test a variety of randomness extraction processes. The main question that we seek to answer is whether these methods have an observed impact on the statistical properties of the RNG's output. The procedure that we follow is the following.

1. We collect the output of each RNG that was tested in the previous section. We call this the *initial* output.
2. We apply different post-processing methods, or randomness extractors, to this initial output to produce a new, *processed* output. Each time, we precisely define and explain the underlying assumptions of the used extractors required for the extraction method to be successful. These different sets of assumptions, for each extraction method, can be compared with each other and form the different post-processing levels.
3. We analyse the new, processed output with our STE to determine whether each extraction method had an impact from a statistical perspective. We also compare the results obtained using the different post-processing methods for each RNG.

A schematic of the set-up can be found in Figure 1.

5.1. Randomness Extraction Methods

We now describe the different post-processing levels that we consider in this work, i.e., the types of randomness extractors that we will use to improve the different RNGs. We consider four classes of randomness extractor, which form the different levels, each with increasingly elaborate implementations.

- **Level 1: Deterministic extractors—**This class of extractors requires certain properties of the initial output's distribution to hold, beyond just a min-entropy assumption. An example is the seminal `Von Neumann` extractor [43], which works if every bit of the initial output is identically and independently generated (although a sufficient condition is that the input forms an exchangeable sequence). In practice, assumptions of this type are difficult to justify and to control.
- **Level 2: Seeded extractors—**These extractors require a second string, called a *seed*, of independent and (near-)perfectly random bits as the resource. This seed needs to be carefully generated and can lead to problems if, for example, it is not generated independently of the initial output of the RNG (this could happen, for example, if the seed is generated whilst sharing the same environment as the RNG or by an adversary) or if it has poor statistical properties. At a fundamental level, seeded extractors are unsatisfying as there is circularity in having to generate near-perfect randomness as a resource to build an RNG.
- **Level 3: Two-source extractors—**These extractors are a generalisation of seeded extractors in which the assumptions on the seed are relaxed. Namely, the second, additional source of randomness (previously the seed) now only needs to have some known min-entropy and be independent of the initial output. Moreover, the independence condition can also be relaxed—for example, allowing coordination, cross-influence, or bounded mutual information with respect to the input [44] or independence only in the sense of a Markov chain [45].
- **Level 4: Physical device-independent extractors—**The last class that we consider are extractors requiring special additional hardware, providing the second randomness source needed in level 3 whilst making only minimal assumptions (for example, that information cannot travel faster than the speed of light). This is made possible by a particular type of interactive proof system in which quantum hardware can be verified to perform as promised, as opposed to having to rely on modelling the physical process, as would be done normally. This 'black box' verification gives a guaranteed lower bound on the min-entropy of the output, which can then be used together with the RNG's initial output in a two-source extractor as in level 3. These *physical* extractors are referred to, in the quantum information science community, as device-independent randomness amplification protocols and have no classical analogue. With today's technology, such extractors require making additional implementation assumptions (to the minimal ones). We return in detail to physical extractors in Section 5.2.4.

When a second bit string of randomness is required (levels 2 and 3), we use the NIST Randomness Beacon [46]. For physical randomness extraction (level 4), we use a semi-device-independent randomness amplification protocol that is an adaptation of [47], which we describe in Section 5.2.4. All the algorithms for extraction used in this work are from the software library `Cryptomite` [7], which can be found at https://github.com/CQCL/cryptomite (accessed on 28 November 2024).

The assumptions that the different post-processing methods require are illustrated in Figure 3.

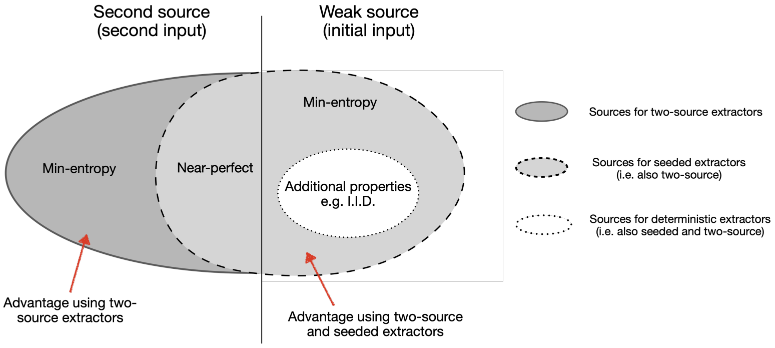
Second source
(second input)
Weak source
(initial input)
Min-entropy
Min-entropy
Near-perfect
Additional properties
e.g. I.I.D.
Sources for two-source extractors
Sources for seeded extractors
(i.e. also two-source)
Sources for deterministic extractors
(i.e. also seeded and two-source)
Advantage using two-source extractors
Advantage using two-source and seeded extractors

Figure 3. Illustration of the set of sources, or input distributions, that can be successfully extracted from by different randomness extraction methods. (Right) weak input distributions and (Left) second input, or weak seed, distributions. Deterministic extractors (level 1) require additional properties on the weak input but do not need a second input source. Seeded extractors (level 2) relax the need for additional properties of the weak input and extract from sources with min-entropy only, at the cost of requiring a second string of (near-)perfect randomness. Two-source extractors (level 3) relax the assumptions of seeded ones to a second source that also has min-entropy only. Physical extractors (level 4, not in the figure) require special quantum hardware, which effectively provides the second input with a device-independent lower bound on the min-entropy, requiring minimal added assumptions.

5.1.1. Results Overview

We now present the main results of the statistical testing of the different post-processing methods in Figure 4, with more details and tables in the following sections. As stated before, we expect that an RNG producing near-perfect random numbers fails less than 7.5 of the 4600 tests that it is subject to, on average, when testing 5 × 10 Gbit files. (This number is the expected amount of type 2 error, i.e., the expected maximum number of failed tests, given that the underlying distribution is indistinguishable from uniform. Note that we implicitly assume that each statistical test is independent). This is the criterion that we use to call randomness generation *successful* from a statistical perspective (green highlighted area).

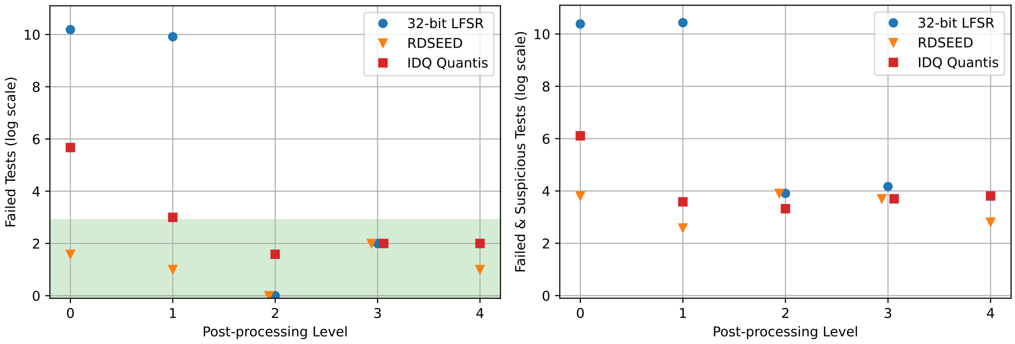

Figure 4. The above plots show (**left**) the number of statistical tests failed and (**right**) failed and suspicious for each initial RNG at each post-processing level. The x axis indicates the level, with step 0 being the initial RNG with no additional post-processing, and steps 1–4 are deterministic, seeded, two-source, and physical extraction, respectively. The y axis is the number of statistical tests failed (**left**) or failed and suspicious (**right**), out of 4600, using a logarithmic scale: for f failed or failed and suspicious tests, $y = \log_2(f+1)$. The shaded region in the left plot illustrates the *successful* region, whereby the RNG fails less than 7.5 tests, and the white region illustrates the 'unacceptable' region, in which, with high probability, near-perfect randomness is not produced. We note that we are unable to use the 32-bit LFSR at level 4 because of its low initial estimated min-entropy rate, α_{RNG}, as detailed and evaluated in Section 4.

Ideally, the results would reflect the different levels of post-processing and the validity of the assumptions that these imply. Our results in Figure 4 tell a mixed story.

- For the RNGs that fail the tests when unprocessed, we observe that additional post-processing indeed improves the quality of the initial output. Considering the LFSR, for example, any extraction method higher than level 1 applied to the initial output produced a processed output that passed the numerical tests well. IDQ's device, as a second example, is significantly improved already with level 1 of extraction, but only gives successful results when higher levels are applied.
- Although level 3 is strictly a relaxation of the assumptions made at level 2, we were unable to observe a difference in the numerical results. This is because level 2, from a statistical perspective, seems to be giving results that are already successful. Moreover, we are unable to distinguish between levels 2, 3, and 4. We interpret this as another illustration of the difference between statistical and cryptographic randomness, in which weaker assumptions are desirable even if no statistical advantage can be witnessed from the user's perspective. It is also likely that, in order to give non-trivial examples of step 2 failing, one would need to generate the seed in a manner that is either significantly biased or correlated to the RNG (both of which could happen in practice).
- All our implementations above level 1 gave successful numerical test results on the three RNGs that we tested. In particular, from a statistical perspective, this means that a poor PRNG (here, the 32-bit LFSR) can be concatenated with an extractor to form a good PRNG.

5.2. Implementations of the Post-Processing Methods

We now describe how we implemented the post-processing, i.e., different extractors in our levels, together with the parameter choices and compromises that we made. For the post-processing algorithms, we used the randomness extractors publicly available from the software library Cryptomite [7]. To assess the randomness quality at each step, we generated 5×10 Gbit test files of the processed output and performed statistical testing using the 'all' setting (the most intense) in the STE. All randomness post-processing and statistical tests were run on a Dell Precision 7540 personal laptop with 16 GB of RAM and a 2.3 GHz Intel i9 processor, using the Ubuntu 20.04 operating system. We state all input and output sizes and give detailed descriptions of each test setting and implementation of each level with the parameter choices, so that all results can be reproduced. For each level, we chose the parameters of the different extractors such that, in theory, the processed output was $\epsilon_{\mathsf{total}}$-perfectly random (see Definition 4), with $\epsilon_{\mathsf{total}} \leq 2^{-32} \approx 10^{-10}$.

5.2.1. Level 1: Deterministic Extraction

A deterministic extractor will generate a near-perfectly random output when processing the initial output of RNGs with some well-defined properties. These well-defined properties vary depending on the extractor that is used, with different choices possible.

Definition 6 (Deterministic randomness extractor)**.** *A deterministic randomness extractor is a function*

$$\mathsf{Ext_d} : \{0,1\}^n \to \{0,1\}^m \tag{6}$$

such that, for random variables $X \in \{0,1\}^n$ *with specific properties [25],*

$$\Delta(\mathsf{Ext_d}(X), \mathrm{U}_m) \leq \epsilon \,, \tag{7}$$

where U_m *is the uniform variable on* $\{0,1\}^m$*.*

In other words, a deterministic extractor is a function that maps random variables X with specific characteristics to a new variable $\mathsf{Ext_d}(X)$ that is near-perfectly random. Note that the properties of X required depend on the specific extractor—for example, that all bits in X are I.I.D.

The implementation of the deterministic extraction set-up is shown in Figure 5. We use the Von Neumann extractor [43] to extract from the initial output $X \in \{0,1\}^n$ of the RNG, with the implementation from [7]. This extractor requires that all two subsequent input bits have a fixed bias, i.e., for bits $X_{2i}, X_{2i+1} \in \{0,1\}$ with $i = 1, \ldots, \lfloor \frac{n}{2} \rfloor$ and $p_i \in (0,1)$, we require that

$$\Pr(X_{2i} = 0) = \Pr(X_{2i+1} = 0) = p_i \, . \tag{8}$$

The Von Neumann extractor works by grouping subsequent bits in pairs and outputting the first (or second) bit only when the bits in the pair are different, giving an output length of $m \approx p(1-p)$ (if the bias is fixed, $p_i = p$ for all i) and $\epsilon = 0$, i.e., perfect randomness at the output.

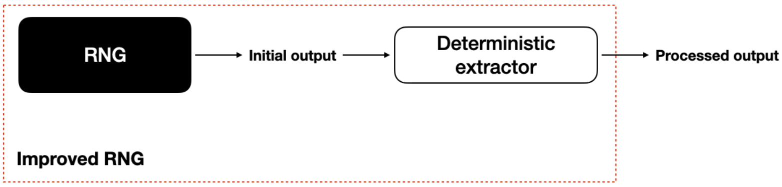

Figure 5. Here, level 1 of our post-processing methods is performed by using a deterministic extractor, namely the Von Neumann extractor, on the initial output of the RNG.

The statistical test results in Table 5 show the following.

- Both the LFSR and IDQ Quantis show improvements compared to the initial RNG testing results (Table 3), although they still do not pass overall.
- The number of NIST statistical test failures for both the LFSR and IDQ Quantis increases when moving from no post-processing to deterministic post-processing. This may be due to specific biases in the RNGs that are incompatible with, or even amplified by, the Von Neumann extractor (e.g., successive bits are not independent) or fundamental issues with the NIST tests, as suggested in [33–36].

Table 5. This table gives the sum of statistical tests failed for 5 × 10 Gbit samples from each RNG, after deterministic extraction using the Von Neumann extractor. Due to the 32-bit LFSR failing PractRand quickly, only 635 tests were conducted (instead of the full 4600), so we rescale these results. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses. Full results can be found in Appendix C.

RNG	NIST (75)	Diehard (90)	ENT (30)	SmallCrush (75)	Alphabit (85)	Rabbit (200)	PractRand (4600)
32-bit LFSR	25	10 (5)	5	18	76	106	724 (413)
RDSEED	0	0 (2)	0	0	0	1	0 (2)
IDQ Quantis	4	0 (1)	0	0	0	3	0 (3)

Interestingly, applying the Von Neumann extractor to an LFSR results in a stream cipher known as the self-shrinking generator, which has been studied for cryptographic use [28]. Although the self-shrinking generator fails fewer tests compared to the unprocessed LFSR, a substantial number of failures remain.

As noted at the start of this section, a deterministic extractor can produce a near-perfectly random output if the input source satisfies certain specific properties. However, these properties are often difficult or even impossible to verify in practice. As a result, it is more practical to base claims solely on the min-entropy of the source. In [48], it was demonstrated that deterministic extraction from a source characterised solely by min-entropy is impossible. Such sources require additional, independent randomness to enable extraction.

5.2.2. Level 2: Seeded Extraction

Seeded extraction requires only a min-entropy guarantee for the initial RNG output but comes at the cost of needing a second, independent and (near-)perfectly random input (the *seed*) to enable extraction.

Definition 7 (Seeded randomness extractor)**.** *A seeded randomness extractor is a function* $\mathsf{Ext_s} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *such that, for a random variable* $X \in \{0,1\}^n$ *with min-entropy* $\mathrm{H}_\infty(X) \geq k$*, and seed* $S \in \{0,1\}^d$ *with min-entropy* $\mathrm{H}_\infty(S) = d$ *(i.e., S is perfectly random),*

$$\Delta(\mathsf{Ext_s}(X,S), \mathrm{U}_m) \leq \epsilon\,, \tag{9}$$

where U_m *is the uniform distribution on* $\{0,1\}^m$*.*

A seeded extractor can be understood as a randomised function that maps a weakly random variable X to a new variable $\mathsf{Ext_s}(X,S)$ that is (near-)perfectly random. Note that the seed may be ϵ_s-perfect only, with an additive error in the statistical distance above, i.e., $\epsilon \to \epsilon + \epsilon_s$ (see, for example, Appendix A from [49] for proof). Seeded extractors are a special case of two-source extractors, which we define later in Definition 9.

Definition 8 (Strong seeded extractor)**.** *A strong seeded randomness extractor is a function* $\mathsf{Ext_s} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *such that*

$$\Delta([\mathsf{Ext_s}(X,S), S], [\mathrm{U}_m, S]) \leq \epsilon\,, \tag{10}$$

where $[\cdot,\cdot]$ *denotes the concatenation of random variables and* U_m *is the uniform variable on* $\{0,1\}^m$*.*

A strong seeded extractor is a randomised function that gives a (near-)uniform output, even when conditioned on the seed S (the output is therefore independent of the seed). This has some interesting consequences, which we exploit to generate the large amounts of processed output needed for statistical testing. Specifically, S can be reused with different weak input random variables, allowing a single seed to be used in many extraction rounds. The set-up for seeded extraction (implemented using a strong seeded extractor) is shown in Figure 6. The initial output from the RNG is split into blocks X_i for $i = 1, \ldots, n$ with a promise on each block's min-entropy (Definition 2).

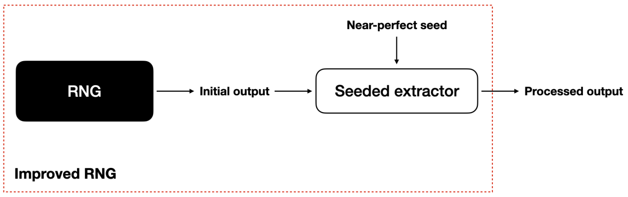

Figure 6. The set-up for seeded extraction. In this case, the initial output of the RNG only needs to have min-entropy, but extraction requires an additional near-perfectly random bit string (the seed), which needs to be generated independently.

This step can be implemented with the Circulant [7], Dodis et al. [50], Toeplitz [51], and Trevisan [52] extractors from Cryptomite, as they can all be used as strong seeded extractors. Among these extractors, Circulant offers the best trade-off between security parameters and computational complexity and is therefore the one that we chose. The Circulant extractor requires that the seed length is the input length plus one and that the seed length is a prime. We set the seed length $|S|$ and RNG input block lengths

$|X_i|$ to $|S| = |X_i| + 1$ = 10,007. Note that using Circulant allows us to generate cryptographic randomness even against an adversary able to store (and process) side information in quantum systems without changing the extraction algorithm, i.e., the extractor is *quantum-proof*; see [7] for details.

To generate the seed S, we use the NIST Randomness Beacon, which is a public source of randomness produced by the US Government agency (NIST), mixing different randomness sources together, including chaotic classical and quantum processes [46]. The min-entropy k_i^{RNG} for each block X_i is $k_i^{\mathsf{RNG}} = \alpha_{\mathsf{RNG}}|X_i|$, where α_{RNG} is a lower bound on the min-entropy per bit for each initial RNG block of outputs X_i, with probability $\epsilon_{\text{est}} < 2^{-32}$ (as found in Equation (A2) of Appendix B.2.1). The output length after extraction, m, is then roughly $m \approx k_i^{\mathsf{RNG}}$.

In order to generate the required 5 × 10 Gbits of processed output, the Circulant extractor is used multiple times on different initial output blocks X_i with the same seed. The extractor's outputs are then concatenated together until a final output, Output, of sufficient size is generated. The Output is given by

$$\mathsf{Output} = \left[\mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_1, S), \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_2, S), \ldots, \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_n, S)\right], \tag{11}$$

where $[\cdot, \cdot]$ denotes the concatenation of random variables. Each extraction round, which we index i, has an associated error $\epsilon_{\text{ext}_{\text{i}}}$, and we choose the total security parameter to be $\epsilon_{\text{total}} \leq 2^{-32}$—namely, everything is chosen so that $\epsilon_{\text{total}} = \epsilon_{\text{est}} + \sum_{j=1}^{n} \epsilon_{\text{ext}_{\text{j}}} \leq 2^{-32}$. This derivation for ϵ_{total}, specifically that the composed output error is the sum of each of the individual extractor errors, can be found in [7].

The observations that we draw from the results in Table 6 are the following.

- The statistical test results show a significant improvement in the results using deterministic extraction; see Section 5.2.1. In particular, all RNGs have been successfully post-processed from a statistical perspective.
- Even the 32-bit LFSR is successfully extracted from, which suggests that one can, from a statistical perspective, build good PRNGs by appending an extractor to poor PRNGs.
- Randomness that has a small amount of min-entropy only can pass statistical tests successfully. This is somewhat unsurprising as cryptographically secure PRNGs exist, but we find it interesting to comment on nonetheless. The total entropy of the final output of the processed LFSR output is upper-bounded by 10,007 +32 (the seed length of the extractor plus the seed length of the 32-bit LFSR), in the 50 Gbit of processed output generated, i.e., a true min-entropy rate of, at most, $\alpha = (10,007 + 32)/(5 \times 10^{10}) < 10^{-5}$.

Table 6. This table gives the sum of statistical tests failed for 5 × 10 Gbit samples from each RNG, after a strong seeded extractor has been applied to its initial output. The seed is generated using the NIST Randomness Beacon. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right. Full results can be found in Appendix D.

RNG	**NIST (75)**	**Diehard (90)**	**ENT (30)**	**SmallCrush (75)**	**Alphabit (85)**	**Rabbit (200)**	**PractRand (4600)**
32-bit LFSR	0	0 (3)	0	0	0	0	0 (6)
RDSEED	0	0 (7)	0	0	0	0	0 (7)
IDQ Quantis	0	0 (2)	0	0	0	2	0 (5)

Our results at this level are disappointing, in the sense that the successful test results mean that we will not be able to distinguish the next levels (3 and 4) from level 2 from a statistical perspective—for example, that level 3 is strictly better than level 2. It would be interesting to find non-trivial examples where the output of a seeded extractor fails statistical tests because of a seed generated in a way that is not independent or near-uniform.

Unfortunately, we could only find artificial examples (i.e., when all seed bits are the same) that were detected by our statistical testing.

5.2.3. Level 3: Two-Source Extraction

Seeded extraction (level 2) requires an independent string of (near-)perfect randomness as an initial resource, which is difficult to justify and leads to circularity: one needs near-perfect randomness to generate more of it. Two-source extraction relaxes this requirement, allowing the second string to be only weakly random, in the sense that it has some min-entropy and/or only a relaxed notion of independence (for example, the case of using a two-source extractor secure in the Markov model [45], where the two input sources can be correlated through a common cause, or if the sources may have bounded coordination, cross-influence, or mutual information [44])—although, in this work, we calculate our two-source extractor parameters based on standard independence between the two input sources. Two-source extractors can be used as seeded extractors, simply by assuming that one of the input strings is already near-perfect and independent; therefore, level 3 is strictly a relaxation of the assumptions of level 2.

Definition 9 (Two-source randomness extractor)**.** *A two-source randomness extractor is a function* $\mathsf{Ext}_2 : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^m$ *such that, for statistically independent random variables* $X \in \{0,1\}^{n_1}$ *and* $Y \in \{0,1\}^{n_2}$ *with min-entropy* $\mathrm{H}_\infty(X) \geq k_1$ *and* $\mathrm{H}_\infty(Y) \geq k_2$*, respectively,*

$$\Delta(\mathsf{Ext}_2(X,Y), \mathrm{U}_m) \leq \epsilon\,, \tag{12}$$

where U_m *is the uniform variable on* $\{0,1\}^m$*.*

In other words, a two-source extractor is a weakly randomised function that maps a random variable X to a new variable $\mathsf{Ext}_2(X,Y)$ that is near-perfect.

Definition 10 (Strong two-source extractor)**.** *A two-source randomness extractor is said to be strong in the input* Y *if the function* Ext_2 *is such that*

$$\Delta([\mathsf{Ext}_2(X,Y), Y], [\mathrm{U}_m Y]) \leq \epsilon\,, \tag{13}$$

where $[\cdot,\cdot]$ *denotes the concatenation of random variables and* U_m *is the uniform variable on* $\{0,1\}^m$*.*

Strong two-source extractors, like strong seeded extractors, allow for one input source to be used in multiple extraction rounds. The set-up for seeded extraction (implemented using a strong seeded extractor) is shown in Figure 7.

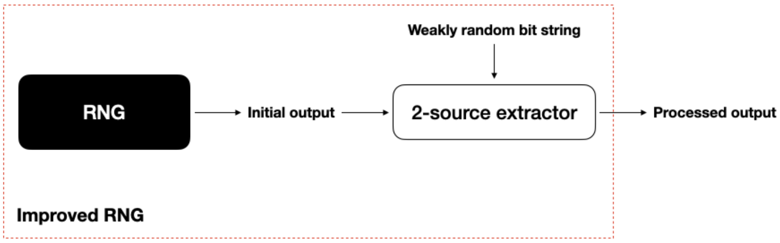

Figure 7. The set-up for two-source extraction. In this case, the initial output of the RNG only needs to have some min-entropy and extraction requires an additional bit string that is weakly random only in the sense that it also has min-entropy.

From the Cryptomite library, we again use the Circulant extractor [7], but this time as a strong two-source extractor. Other extractors in Cryptomite can be used too, but, since the Circulant extractor offers the best parameters and efficiency, we use it in our implementation. For full details, we refer the reader to [7]. Two-source extraction requires a second input source with min-entropy above some threshold based on the specific two-source extractor construction. For the $\mathsf{Circulant}$ extractor, this requirement is that the sum of the min-entropy rates of the two weak inputs is at least 1. X_i is the initial RNG output blocks and Y is the additional weakly random input (which we sometimes call the *weak seed*) and, as in level 2, we set $|Y| = |X_i| + 1 = 10{,}007$.

To generate Y, we again use the NIST Randomness Beacon, but, in this case, we minimise the amount of entropy that we assume that it contains, instead of assuming that it has full entropy as in level 2. This change in the assumption increases the likelihood that the assumption holds in practice. The output length of the $\mathsf{Circulant}$ extractor is roughly $(\alpha_{\mathsf{NIST}} + \alpha_{\mathsf{RNG}} - 1)|Y|$, which we impose by adjusting the min-entropy rate assumption of the NIST Randomness Beacon as α_{NIST}, as

$$\alpha_{\mathsf{NIST}} = 1.02 - \alpha_{\mathsf{RNG}} \tag{14}$$

where α_{RNG} is the min-entropy rate of the initial RNG (found in Section 4). We use 1.02 instead of 1 to account for spurious terms in the parameter calculation that reduce the output length; see [7] for the explicit calculation of these penalty terms. In other words, we use the computed min-entropy rate of the RNG under study to minimise the assumption about the second source's min-entropy rate, whilst imposing a non-trivial output length from the extractor.

The processed output is then generated in two steps: (1) using the $\mathsf{Circulant}$ extractor as a two-source extractor on the two input strings X_1 and Y, we generate a (near-)perfect output, which will be the seed in the next step; (2) we use this seed in multiple $\mathsf{Circulant}$ seeded extractions on $X_{i \geq 2}$. The multiple outputs of the seeded extractor are concatenated together to obtain a final output of 5×10^{10} bits. In other words, the concatenation of the two-source and seeded extractors together forms a two-source extractor with advantageous parameters. Therefore, the final output for statistical testing is given by

$$\mathsf{Output} = \left[\mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_2, S), \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_3, S), \ldots, \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_n, S)\right], \tag{15}$$

where $S = \mathsf{Ext}_{2}^{\mathsf{Circulant}}(X_1, Y)$, $[\cdot,\cdot]$ denotes the concatenation of random variables and the extractor round with input X_i has error $\epsilon_{\mathsf{ext_i}}$. The total error of the final output is $\epsilon_{\mathsf{total}} = \epsilon_{\mathsf{est}} + \epsilon_{\mathsf{ext_1}} + \sum_{j=2}^{n} \epsilon_{\mathsf{ext_j}} \leq 2^{-32}$. Proof that a strong two-source extractor and strong seeded extractor can be composed into $\mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_{i>1}, S)$, for S the output of a two-source extractor (right-hand side of Equation (15)) can be found in [53] Section 6.3. This, combined with the fact that the composed output error is the sum of each of the individual extractor errors (in [7]), allows us to calculate $\epsilon_{\mathsf{total}}$.

Our results in Table 7 show that all RNGs extracted at level 3 are successful from a statistical perspective, as in the seeded extraction case (level 2). In the Appendices, we implement a variant of level 3 (two-source extraction) where all input strings are drawn from the initial RNG and there is no randomness from an alternative RNG, i.e., rewriting the Output in Equation (15) using $Y = X_0$, where X_0 is another output block from the initial RNG. In this regime, for near-perfect randomness to be generated, all blocks produced by the initial RNG must be independent of one another (as well as having block min-entropy). Even in this case, the results were successful statistically. Full explanations and results can be found in Appendix E.1.

Table 7. This table gives the sum of statistical tests failed for 5 × 10 Gbit samples from each RNG, after strong two-source extraction, taking the RNG as one weak source and randomness from the NIST Randomness Beacon as the second. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses. Full results can be found in Appendix E.2.

RNG	NIST (75)	Diehard (90)	ENT (30)	SmallCrush (75)	Alphabit (85)	Rabbit (200)	PractRand (4600)
32-bit LFSR	0	0 (6)	0	0	2	1	0 (8)
RDSEED	0	0 (4)	0	0	0	3	0 (5)
IDQ Quantis	0	0 (3)	0	0	0	1	0 (5)

5.2.4. Level 4: Physical Randomness Extraction

Two-source extraction (level 3) allows for the generation of near-perfect randomness if two weakly random but independent strings of randomness are available. In the final level, we consider post-processing with a *physical* randomness extractor. This level is called physical because it requires a quantum device, in addition to the initial RNG, while the other levels only require mathematical algorithms to perform extraction. At a high level, the role of this additional hardware is to provide a second string of random numbers, whilst making minimal assumptions only.

Adding quantum hardware may initially seem to imply introducing numerous assumptions; however, following the *device-independent* approach, this hardware can, in principle, be treated as an untrusted *black box* (which could even have been built by an adversary, so long as it can be shielded once in use and meets some minimal requirements). We call the added assumptions *minimal* because they are either fundamental to physics—e.g., information cannot travel faster than light speed—or no cryptography can ever be performed without them—e.g., the devices are shielded (there are no backdoors). This is made possible by the development of device-independent protocols, which rely on Bell tests [54]. The idea is to use the initial RNG to generate random challenges for the quantum device and then study its response. With ideal (noiseless) devices, this approach can be used to *self-test* the inner functioning of the device, i.e., one can uniquely identify the implemented quantum states and measurements from the observed challenge–response statistics alone. For real (noisy) devices, this approach can be used to bound the adversary's guessing power, and thus guarantee min-entropy, over the device's outputs or responses. This approach crucially relies on quantum resources, which have this self-testing property, and has no classical analogue. For a review on the subject, together with its minimal assumptions (called loopholes), we refer the reader to [55]. See Figure 8 for an illustration.

Today, quantum devices that are capable of running device-independent protocols are extremely difficult to build (they require the ability to perform a loophole-free Bell Test [54]) and exist as experiments on lab benches only. Because of this, more practical implementations have been developed in which a few well-justified assumptions are added (to the minimal ones). The resulting protocols have comparatively fewer assumptions than standard hardware, but not fundamentally minimal. Such a *semi*-device-independent protocol is the one that we implement for our physical extraction method at level 4, based on an adaptation of the randomness amplification protocol described in [47] and implemented on quantum computers. For clarity, the assumptions that we make are the following.

- The initial RNG has a block min-entropy structure (as in seeded and two-source extraction).
- The quantum device is independent of the initial RNG's output; we do not consider correlations between the two (although this can be added). This assumption is well motivated since the quantum computer is distant from the initial RNG.
- The quantum device is assumed to perform a faithful Bell test. This assumption is well motivated when using particular types of devices, such as the quantum computers

based on ion traps that we use—see the discussion in [47] (Section 6.2 , *Validity of quantum computers for Bell experiments and added assumptions*).

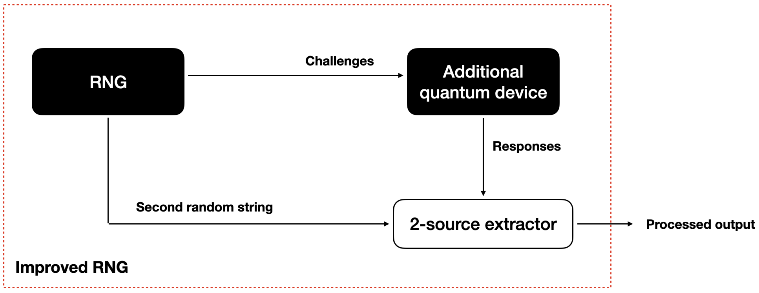

Figure 8. The set-up for level 4: physical randomness extraction. The initial RNG is used twice: first to generate challenges to the quantum device and second to provide an extra bit string as input to a two-source extractor. The role of the quantum device is to provide an additional source of randomness. The device-independent protocol is performed by using the challenge–response behaviour of the device to obtain a lower bound on the amount of randomness in the device's responses (without characterising the device itself). The second bit string of the initial RNG and the responses from the quantum device form the two input strings to a two-source extractor, implemented as in level 3.

We used the H1-1 Quantinuum ion-trap quantum computer [56] as our device to obtain, from its output, a weakly random bit string size of 3.6×10^6 bits (this means that, due to using the Circulant extractor, the input length of the initial RNG block to the two-source extraction is also 3.6×10^6 bits) with min-entropy rate $\alpha_{\mathsf{Q}} \geq 0.518$, certified in the semi-device-independent manner described above. The Circulant extractor requires $\alpha_{\mathsf{Q}} + \alpha_{\mathsf{RNG}} > 1$ to give a non-vanishing output, implying that the rate of an initial RNG must satisfy $\alpha_{\mathsf{RNG}} > 0.482$ to allow for physical extraction with our implementation. (This minimum requirement is particularly interesting, since, even if one has access to two identical (and independent) copies of an initial RNG with $\alpha_{\mathsf{RNG}} = 0.482 + \delta$ for $\delta \in (0, 0.18)$, one would be unable to extract from the two (step 3) with today's implemented extractors. Note that this is not a fundamental limitation, as other two-source extractors allow for one of the strings to have a logarithmic min-entropy rate only. However, to our knowledge no such extractor has been implemented, let alone efficiently). The advantage of using a quantum device, and therefore level 4, is two-fold: (a) one obtains a rigorous, semi-device-independent lower bound on a second bit string's min-entropy and (b) the min-entropy rate of the quantum device is above 0.5, allowing extraction from a weak initial output with rate 0.5 using the Circulant extractor. Note that the min-entropy of the LSFR was too low to perform physical extraction (its min-entropy rate was below 0.482; see Table 4).

The processed output is then generated in two steps. (1) We generate a (near-)perfect seed using the Circulant extractor as a two-source extractor on the two input strings X_1, from the initial RNG, and Y, from the H1-1 Quantinuum quantum computer. (2) We use this seed in multiple Circulant seeded extractions on $X_{i \geq 2}$, which are concatenated together to obtain a final output of 5×10^{10} bits. In other words, the concatenation of the two-source and seeded extractors together again forms a two-source extractor with advantageous parameters. Therefore, the final output for statistical testing is given by

$$\mathsf{Output} = \left[\mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_2, S), \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_3, S), \ldots, \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_n, S) \right], \tag{16}$$

where $S = \mathsf{Ext}_2^{\mathsf{Circulant}}(X_1, Y)$, $[\cdot, \cdot]$ denotes concatenation and the extractor round with input X_i has error $\epsilon_{\mathsf{ext_i}}$. The total error of the final output is $\epsilon_{\mathsf{total}} = \epsilon_{\mathsf{est}} + \epsilon_{\mathsf{ext_1}} + \sum_{j=2}^{n} \epsilon_{\mathsf{ext_j}} \leq 2^{-32}$. This last step is similar to that of level 3, where the NIST Randomness Beacon is replaced by the H1-1 Quantinuum quantum computer. The statistical test results are given in Table 8.

Table 8. This table gives the sum of statistical tests failed for 5 × 10 Gbit samples from level 4. Note: The 32-bit LFSR does not generate any output in this setting, since its min-entropy is too low for extraction. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses. Full results can be found in Appendix F.

RNG	NIST (75)	Diehard (90)	ENT (30)	SmallCrush (75)	Alphabit (85)	Rabbit (200)	PractRand (4600)
32-bit LFSR	-	-	-	-	-	-	-
RDSEED	0	0 (2)	0	0	0	1	0 (3)
IDQ Quantis	0	0 (3)	1	0	0	2	0 (7)

The results of the statistical tests in Table 8 show, as for levels 2 and 3, that the post-processed RNGs perform well at level 4.

6. Discussion

In this work, we have presented a variety of extraction methods to post-process the output of random number generators (RNGs) and evaluated their impacts on the statistical properties of three widely used RNGs. We started by extensively testing the output from three RNGs and identified statistical failures in two of them, corroborating and extending previous findings [10,21]. For the RNGs that failed, all post-processing methods improved the statistical properties. Specifically, we found that the processed outputs processed with level 2 or higher (seeded, two-source, or physical extraction) were statistically indistinguishable from uniform distributions. However, due to the inherent limitations of statistical testing, we could not identify any examples that failed under level 2 post-processing (seeded extraction) but succeeded with level 3 or higher (two-source and physical extraction), even though level 3 is provably stronger than level 2.

Our statistical testing environment, STE, software, documentation, and build file can be found at https://github.com/CQCL/random_test (accessed on 28 November 2024), and the randomness extractor software library Cryptomite can be found at https://github.com/CQCL/cryptomite (accessed on 28 November 2024) and in [7]. These tools may be independently interesting, making both statistical testing and randomness extraction easy to use and openly accessible.

A number of interesting future directions arise. It would be interesting to perform the statistical testing of other RNGs with our test environment to analyse how they perform when tested beyond what is required by standardisation bodies. Similarly, it would be interesting to include different post-processing methods than the ones that we have presented. One could use, for example, vetted conditioning components from NIST [57] and compare their results to the ones obtained using information-theoretic randomness extractors.

We could have moved even further in our numerical testing but, because numerical tests consume substantial computational resources, decided to omit certain test suites from our analysis, including SPRNG [58] and Crypt-X [59]. Moreover, we were recently made aware of the numerical tests BitReps [60] and RaBiGeTe [61], which are also not included in STE. It would be interesting to include these in the analysis to obtain an even more intense statistical testing environment.

Author Contributions: Conceptualisation, C.F. and F.J.C.; methodology, C.F. and F.J.C.; software, R.Y.; validation, C.F., R.Y. and F.J.C.; formal analysis, C.F.; investigation, C.F. and F.J.C.; resources, F.J.C.; data curation, C.F.; writing—original draft preparation, C.F. and F.J.C.; writing—review and editing, C.F., R.Y. and F.J.C.; visualisation, C.F., R.Y. and F.J.C.; supervision, F.J.C.; project administration, C.F. and F.J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data presented in this study are openly available in https://github.com/CQCL/random_test (accessed on 28 November 2024).

Acknowledgments: We thank Erik Woodhead and Ela Lee for the useful discussions and suggestions.

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

Appendix A. RNG Descriptions

Appendix A.1. Linear Feedback Shift Register (LFSR)

The LFSR is a class of pseudo-RNGs that is commonly used in applications due to its speed and ease of implementation in both software and hardware, e.g., [62,63]. Notably, LFSRs are used in cryptography, including in hashing and authentication [51] and stream ciphers [64]. For this work, we implement the maximal period LFSR found in [65].

This LFSR generates pseudo-randomness as follows. Let $s = b_1, \ldots, b_{32}$ denote the initial 32-bit state, where b_i denotes the $i = 1, \ldots, 32$th bit.

1. Initialise the LFSR with the 32-bit initial state $s = b_1, b_2, ..., b_{31}, b_{32}$.
2. Calculate the feedback f of s, where $f = b_{32} \oplus b_{22} \oplus b_2 \oplus b_1 \oplus 1$, where $\oplus$ denotes addition modulo 2.
3. Output bit b_1.
4. Replace bit b_i with bit b_{i-1} for all $i \in (2, 32)$.
5. Set $b_{32} = f$.
6. Repeat steps (2–5) until the desired amount of bits has been generated.

The maximum period for a 32-bit LFSR is $2^{32} - 1$. This means that bits repeat every $2^{32} - 1$ generated bits (approximately every 4.3 Gbits).

Appendix A.2. Intel RDSEED

Intel manufactures a hardware RNG based on thermal noise, which is present in their computer processing units. This true-RNG is constructed as follows, although a more in-depth description can be found in [10,11].

1. Initial weak randomness is generated from an entropy source. This source is a self-clocking circuit designed such that, when the clock is running, the circuit enters a meta-stable state, which then resolves to one of two possible states—determined randomly by thermal noise. The state in which the circuit resolves is the random bit output from the entropy source. This self-clocking occurs irregularly at around 3 GHz.
2. Health and swellness checks are performed, which are very simple statistical tests, with the goal of detecting critical failures in the entropy source.
3. Cryptographic processing of the randomness with AES is performed.

A user requests randomness from the true-RNG via the RDSEED instruction. Using Intel's true-RNG, we cannot directly output raw randomness from the entropy source; the best available option is RDSEED, which includes some level of post-processing (as described above). Independent analyses of the Intel true-RNG's quality, including [10,32], report a min-entropy rate for RDSEED of around 0.65, closely matching our result (see Section 4).

Appendix A.3. IDQ Quantis QRNG

The IDQ Quantis (USB) is a QRNG based on photons hitting a 50:50 beam splitter and being detected in position 0 (reflected) and 1 (transmitted). In principle, if all components are accurately modelled and the device is shielded from any outside influence, the output is perfectly random numbers due to the laws of quantum mechanics. We refer the reader to the IDQ Quantis QRNG brochure for further details of its construction [22].

Appendix B. Initial RNG Analysis

Appendix B.1. Full Results: Statistical Testing

Table A1. The number of failed tests for the raw output from the 32-bit LFSR. Note that only 127/920 PractRand tests were run due to the numerous failings in the 2^{25} byte case. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
LFSR 1	2	8 (0)	1	10	15	28	23 (5)
LFSR 2	2	8 (0)	1	10	14	27	25 (3)
LFSR 3	2	8 (2)	1	11	15	26	24 (4)
LFSR 4	2	8 (0)	1	10	15	25	23 (5)
LFSR 5	2	8 (2)	1	10	15	25	25 (4)
LFSR 6	2	8 (1)	1	10	15	27	23 (5)
LFSR 7	2	8 (0)	1	10	14	27	24 (6)
LFSR 8	2	8 (0)	1	10	15	25	25 (3)
LFSR 9	2	8 (0)	1	10	14	26	21 (6)
LFSR 10	2	8 (1)	1	10	14	26	23 (4)
Total	**20**	**80 (6)**	**10**	**101**	**146**	**262**	**236 (45)**

Table A2. The number of failed tests for the raw output from RDSEED. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
RDSEED 1	0	0 (0)	0	1	0	0	0 (1)
RDSEED 2	0	0 (1)	0	0	0	0	0 (8)
RDSEED 3	0	0 (0)	0	0	0	0	0 (0)
RDSEED 4	0	0 (1)	0	0	0	0	0 (1)
RDSEED 5	0	0 (1)	0	0	0	1	0 (1)
RDSEED 6	0	0 (0)	0	0	0	1	0 (0)
RDSEED 7	0	0 (1)	0	0	0	0	0 (0)
RDSEED 8	0	0 (1)	0	0	0	0	0 (0)
RDSEED 9	0	0 (0)	0	0	0	0	0 (1)
RDSEED 10	0	0 (2)	0	0	0	0	0 (0)
Total	**0**	**0 (7)**	**0**	**1**	**0**	**2**	**0 (12)**

Table A3. The number of failed tests for the raw output from IDQ Quantis. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
IDQ Quantis 1	0	0 (0)	1	0	3	5	0 (3)
IDQ Quantis 2	0	0 (1)	1	0	4	5	0 (2)
IDQ Quantis 3	0	0 (0)	1	0	2	4	0 (11)
IDQ Quantis 4	0	0 (1)	1	0	3	5	0 (0)

Table A3. *Cont.*

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
IDQ Quantis 5	0	0 (1)	1	0	3	5	0 (2)
IDQ Quantis 6	0	0 (0)	1	0	2	5	0 (1)
IDQ Quantis 7	0	0 (0)	1	0	5	5	0 (4)
IDQ Quantis 8	0	0 (1)	1	0	3	6	2 (1)
IDQ Quantis 9	0	0 (1)	1	0	3	4	0 (4)
IDQ Quantis 10	0	0 (1)	1	0	6	5	3 (2)
Total	**0**	**0 (6)**	**10**	**0**	**34**	**49**	**5 (28)**

Appendix B.2. Full Results: Min-Entropy Estimators

Table A4. Observed NIST min-entropy estimators for 32-bit LFSR raw output.

RNG	NIST Min-Entropy Estimator (/byte)	NIST Min-Entropy Estimator (/bit)
LFSR 1	6.956997	0.869624625
LFSR 2	5.792304	0.724038
LFSR 3	7.161811	0.895226375
LFSR 4	6.638405	0.829800625
LFSR 5	7.353758	0.91921975
LFSR 6	7.121091	0.890136375
LFSR 7	7.213483	0.901685375
LFSR 8	7.188889	0.898611125
LFSR 9	6.638383	0.829797875
LFSR 10	6.638383	0.829797875
Average	**6.8703504**	**0.8587938**

Table A5. Observed NIST min-entropy estimators for RDSEED raw output.

RNG	NIST Min-Entropy Estimator (/byte)	NIST Min-Entropy Estimator (/bit)
RDSEED 1	6.737815	0.842226875
RDSEED 2	6.530758	0.81634475
RDSEED 3	6.846048	0.855756
RDSEED 4	6.995008	0.874376
RDSEED 5	6.861225	0.857653125
RDSEED 6	7.086914	0.88586425
RDSEED 7	6.638399	0.829799875
RDSEED 8	7.024343	0.878042875
RDSEED 9	6.747707	0.843463375
RDSEED 10	6.724567	0.840570875
Average	**6.8192784**	**0.8524098**

Table A6. Observed NIST min-entropy estimators for IDQ Quantis raw output.

RNG	NIST Min-Entropy Estimator (/byte)	NIST Min-Entropy Estimator (/bit)
IDQ Quantis 1	7.149988	0.8937485
IDQ Quantis 2	7.142161	0.892770125
IDQ Quantis 3	7.152185	0.894023125
IDQ Quantis 4	7.088475	0.886059375

Table A6. *Cont.*

RNG	NIST Min-Entropy Estimator (/byte)	NIST Min-Entropy Estimator (/bit)
IDQ Quantis 5	7.161971	0.895246375
IDQ Quantis 6	7.169887	0.896235875
IDQ Quantis 7	7.260033	0.907504125
IDQ Quantis 8	7.188102	0.89851275
IDQ Quantis 9	7.115609	0.889451125
IDQ Quantis 10	7.142009	0.892751125
Average	**7.157042**	**0.89463025**

Appendix B.2.1. Deriving a Min-Entropy Lower Bound

In this subsection, we derive a min-entropy lower bound from the observed NIST min-entropy estimators in Table 4. We use subscripts to index a single min-entropy estimator test and superscript to index the RNG which the variable refers to. Let est_i denote the ith observed NIST min-entropy estimate per bit for a test, $i = 1, \ldots, 10$, and $\overline{\mathsf{est}}$ the average estimate per bit. The sample standard deviation σ (using Bessel's correction) is given by

$$\sigma^{\mathsf{RNG}} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\overline{\mathsf{est}}^{\mathsf{RNG}} - \mathsf{est}_{\mathsf{i}}^{\mathsf{RNG}})^2}\,. \tag{A1}$$

We compute the lower bound for the min-entropy rate, α^{RNG}, including a finite statistics correction term to lower-bound the true estimated min-entropy rate of each RNG with a high probability. Specifically, we desire

$$\Pr(\mathsf{est}_{\mathsf{i}}{}^{\mathsf{RNG}} < \alpha^{\mathsf{RNG}}) = \epsilon_{\mathsf{est}} < 2^{-32}, \tag{A2}$$

where $\mathsf{est}_{\mathsf{i}}$ is the ith NIST min-entropy estimator for a specific RNG. Selecting

$$\alpha^{\mathsf{RNG}} = \overline{\mathsf{est}}^{\mathsf{RNG}} - 7\sigma^{\mathsf{RNG}} \tag{A3}$$

where $\overline{\mathsf{est}}^{\mathsf{RNG}}$ is the average NIST min-entropy estimator for the RNG and σ^{RNG} is the observed sample standard deviation that satisfies Equation (A2), giving $\epsilon_{\mathsf{est}} \approx 2^{-39}$. Here, we have made the assumption that the NIST min-entropy estimator results are normally distributed (which we believe is reasonable due to each test sample being generated a significant time apart) and used the standard probability density function for normally distributed variables.

Appendix C. Deterministic Extraction in Detail

Appendix C.1. Full Results

Table A7. The number of failed tests for the output of the 32-bit LFSR after post-processing with the $\mathsf{Von\ Neumann}$ extractor. Note that only 127/920 PractRand tests were run due to the numerous failings in the 2^{25} byte case. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
LFSR VN 1	5	2 (2)	1	4	16	21	19 (12)
LFSR VN 2	5	2 (0)	1	3	15	21	21 (11)
LFSR VN 3	5	2 (1)	1	3	15	22	22 (10)

Table A7. *Cont.*

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
LFSR VN 4	5	2 (2)	1	3	15	22	19 (12)
LFSR VN 5	5	2 (0)	1	5	15	20	19 (12)
Total	**25**	**10 (5)**	**5**	**18**	**76**	**106**	**100 (57)**

Table A8. The number of failed tests for the output of RDSEED after post-processing with the Von Neumann extractor. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
RDSEED VN 1	0	0 (0)	0	0	0	0	0 (0)
RDSEED VN 2	0	0 (0)	0	0	0	0	0 (0)
RDSEED VN 3	0	0 (0)	0	0	0	0	0 (1)
RDSEED VN 4	0	0 (1)	0	0	0	0	0 (1)
RDSEED VN 5	0	0 (1)	0	0	0	1	0 (0)
Total	**0**	**0 (2)**	**0**	**0**	**0**	**1**	**0 (2)**

Table A9. The number of failed tests for the output of IDQ Quantis after post-processing with the Von Neumann extractor. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
IDQ Quantis VN 1	0	0 (0)	0	0	0	0	0 (1)
IDQ Quantis VN 2	0	0 (0)	0	0	0	2	0 (0)
IDQ Quantis VN 3	0	0 (0)	0	0	0	0	0 (1)
IDQ Quantis VN 4	2	0 (1)	0	0	0	1	0 (0)
IDQ Quantis VN 5	2	0 (0)	0	0	0	0	0 (1)
Total	**4**	**0 (1)**	**0**	**0**	**0**	**3**	**0 (3)**

Appendix D. Seeded Extraction in Detail

Appendix D.1. Full Results

Table A10. Statistical test results for the 32-bit LFSR as the weak input source to the strong seeded randomness Circulant extractor. The seed is randomness generated from the NIST Randomness Beacon. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
LFSR NIST SE 1	0	0 (2)	0	0	0	0	0 (2)
LFSR NIST SE 2	0	0 (0)	0	0	0	0	0 (1)
LFSR NIST SE 3	0	0 (0)	0	0	0	0	0 (1)
LFSR NIST SE 4	0	0 (1)	0	0	0	0	0 (0)
LFSR NIST SE 5	0	0 (0)	0	0	0	0	0 (2)
Total	**0**	**0 (3)**	**0**	**0**	**0**	**0**	**0 (6)**

Table A11. Statistical test results for RDSEED as the weak input source to the strong seeded randomness Circulant extractor. The seed is randomness generated from the NIST Randomness Beacon. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
RDSEED NIST SE 1	0	0 (3)	0	0	0	0	0 (4)
RDSEED NIST SE 2	0	0 (1)	0	0	0	0	0 (2)
RDSEED NIST SE 3	0	0 (1)	0	0	0	0	0 (0)
RDSEED NIST SE 4	0	0 (0)	0	0	0	0	0 (1)
RDSEED NIST SE 5	0	0 (2)	0	0	0	0	0 (0)
Total	**0**	**0 (7)**	**0**	**0**	**0**	**0**	**0 (7)**

Table A12. Statistical test results for IDQ Quantis as the weak input source to the strong seeded randomness Circulant extractor. The seed is randomness generated from the NIST Randomness Beacon. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
IDQ Quantis NIST SE 1	0	0 (0)	0	0	0	0	0 (3)
IDQ Quantis NIST SE 2	0	0 (0)	0	0	0	1	0 (0)
IDQ Quantis NIST SE 3	0	0 (2)	0	0	0	1	0 (0)
IDQ Quantis NIST SE 4	0	0 (0)	0	0	0	0	0 (2)
IDQ Quantis NIST SE 5	0	0 (0)	0	0	0	0	0 (0)
Total	**0**	**0 (2)**	**0**	**0**	**0**	**2**	**0 (5)**

Appendix E. Two-Source Extraction in Detail

Appendix E.1. Two-Source Extraction with a Single RNG

In this subsection, we test the use of two strings from each RNG as the inputs to a strong two-source extractor. For near-perfect randomness to be generated, the unique strings from the RNG must be independent; otherwise, this violates some of the assumptions of this level. Due to the limitations of the Circulant strong two-source extractor, we are unable to perform this step for the LFSR, since the min-entropy lower bound derived in Table 4 is too low.

Table A13. This table gives the sum of statistical tests failed for 5 × 10Gbit samples from each RNG, after strong 2-source extraction taking strings of randomness from the same RNG and assuming independence. Note: The 32-bit LFSR does not generate any output in this setting, since its min-entropy is too low. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (75)	Diehard (90)	ENT (30)	SmallCrush (75)	Alphabit (85)	Rabbit (200)	PractRand (4600)
32-bit LFSR	-	-	-	-	-	-	-
RDSEED	0	0 (1)	1	0	0	1	0 (7)
IDQ Quantis	0	0 (2)	0	0	2	1	0 (8)

Table A14. Statistical test results for RDSEED as the weak input source to the strong two-source randomness Circulant extractor. The seed is randomness generated from RDSEED. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
RDSEED Self 2E 1	0	0 (1)	0	0	0	0	0 (2)
RDSEED Self 2E 2	0	0 (0)	0	0	0	1	0 (0)
RDSEED Self 2E 3	0	0 (0)	0	0	0	0	0 (2)
RDSEED Self 2E 4	0	0 (0)	1	0	0	0	0 (3)
RDSEED Self 2E 5	0	0 (0)	0	0	0	0	0 (0)
Total	**0**	**0 (1)**	**1**	**0**	**0**	**1**	**0 (7)**

Table A15. Statistical test results for IDQ Quantis as the weak input source to the strong two-source randomness Circulant extractor. The seed is randomness generated from IDQ Quantis. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
IDQ Quantis Self 2E 1	0	0 (1)	0	0	0	0	0 (3)
IDQ Quantis Self 2E 2	0	0 (0)	0	0	0	0	0 (2)
IDQ Quantis Self 2E 3	0	0 (0)	0	0	1	1	0 (3)
IDQ Quantis Self 2E 4	0	0 (0)	0	0	1	0	0 (0)
IDQ Quantis Self 2E 5	0	0 (1)	0	0	0	0	0 (0)
Total	**0**	**0 (2)**	**0**	**0**	**2**	**1**	**0 (8)**

Appendix E.2. Two-Source Extraction Using the NIST Randomness Beacon

Table A16. Statistical test results for the 32-bit LFSR as the weak input source to the strong two-source randomness Circulant extractor. The seed is randomness generated from the NIST Randomness Beacon. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
LFSR NIST 2E 1	0	0 (2)	0	0	0	0	0 (0)
LFSR NIST 2E 2	0	0 (3)	0	0	1	0	0 (0)
LFSR NIST 2E 3	0	0 (1)	0	0	0	0	0 (1)
LFSR NIST 2E 4	0	0 (0)	0	0	1	1	0 (4)
LFSR NIST 2E 5	0	0 (0)	0	0	0	0	0 (3)
Total	**0**	**0 (6)**	**0**	**0**	**2**	**1**	**0 (8)**

Table A17. Statistical test results for RDSEED as the weak input source to the strong two-source randomness Circulant extractor. The seed is randomness generated from the NIST Randomness Beacon. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
RDSEED NIST 2E 1	0	0 (0)	0	0	0	1	0 (0)
RDSEED NIST 2E 2	0	0 (1)	0	0	0	1	0 (0)
RDSEED NIST 2E 3	0	0 (2)	0	0	0	1	0 (1)

Table A17. *Cont.*

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
RDSEED NIST 2E 4	0	0 (1)	0	0	0	0	0 (1)
RDSEED NIST 2E 5	0	0 (0)	0	0	0	0	0 (3)
Total	**0**	**0 (4)**	**0**	**0**	**0**	**3**	**0 (5)**

Table A18. Statistical test results for IDQ Quantis as the weak input source to the strong two-source randomness Circulant extractor. The seed is randomness generated from the NIST Randomness Beacon. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
IDQ Quantis NIST 2E 1	0	0 (0)	0	0	0	0	0 (0)
IDQ Quantis NIST 2E 2	0	0 (1)	0	0	0	0	0 (2)
IDQ Quantis NIST 2E 3	0	0 (1)	0	0	0	0	0 (0)
IDQ Quantis NIST 2E 4	0	0 (0)	0	0	0	1	0 (2)
IDQ Quantis NIST 2E 5	0	0 (1)	0	0	0	0	0 (1)
Total	**0**	**0 (3)**	**0**	**0**	**0**	**1**	**0 (5)**

Appendix F. Physical Randomness Extraction in Detail

Appendix F.1. Protocol For Physical Randomness Extraction

For this physical randomness extraction, we roughly follow the protocol developed in [47], with some adaptations to improve the randomness generation speed. (Other protocols could be used and adapted—for example [66]—but additional analysis would need to be performed. Some other protocols allow for fewer assumptions than ours in [47], e.g., [67,68], but these require significant classical computation so cannot be implemented on the current hardware). This produces a semi-device-independent protocol for randomness amplification using a remote quantum computer, based on Bell tests. Roughly speaking, a Bell test requires a device to be challenged with inputs and then, based on the observed input–output statistics, a certain amount of entropy can be certified in the outputs. For a good description of Bell tests, see "Non-local games" in [54].

The adapted protocol that we use is constructed as follows.

1. During each of the n rounds, prepare a circuit that generates the GHZ state $\frac{1}{\sqrt{2}}(|000\rangle + i\,|111\rangle)$ and measure each qubit with a local X or Y measurement decided by the inputs at that round, which are selected from the set of measurements $\{(X,X,X),(X,Y,Y),(Y,X,Y),(Y,Y,X)\}$. Labelling local X and Y measurements as 0 and 1, respectively, allows us to write each measurement setting in the set as $(x_i, y_i, x_i \oplus y_i)$, where subscript i denotes the i-th round and x_i, y_i are input bits selected using the initial RNG. See Section 6.3 *Implementations of Mermin inequality violations on quantum computers* in [47].
2. Run the circuit of round $i \in 1,2,...,n$, recording the measurement settings $x_i, y_i, x_i \oplus y_i$ and measurement outcomes a_i, b_i, c_i of this round.
3. After n rounds, calculate the observed probability distribution $\Pr(a,b,c|x,y)$.
4. Evaluate the Mermin inequality [69] value $\mathsf{M}_{\mathsf{obs}}$, where

$$\mathsf{M}_{\mathsf{obs}} = E_{0,0,0} - E_{0,1,1} - E_{1,0,1} - E_{1,1,0} \tag{A4}$$

from the observed probability distribution, where $E_{x,y,x\oplus y}$ denotes the correlator for measurements $(x, y, x \oplus y)$, defined by

$$E_{x,y,x\oplus y} = \sum_{a\oplus b\oplus c=0} \Pr(a,b,c \mid x,y,x\oplus y) - \sum_{a\oplus b\oplus c=1} \Pr(a,b,c \mid x,y,x\oplus y) \tag{A5}$$

5. Reduce $\mathsf{M}_{\mathsf{obs}}$ to account for finite statistics using the Höeffding inequality, using the relationship between $\mathsf{M}_{\mathsf{obs}}$ and the 'losing probability', found at the beginning of Appendix A.2 of [47]. Let ϵ_{est} be the estimation error and M be the true (asymptotic) value of the Mermin inequality for some I.I.D. quantum device; then, we find $\mathsf{M}_{\mathsf{adj}}$ such that $\Pr(\mathsf{M}_{\mathsf{adj}} > \mathsf{M}) \leq \epsilon_{\text{est}}$ by defining

$$\mathsf{M}_{\mathsf{adj}} = \mathsf{M}_{\mathsf{obs}} - 16t. \tag{A6}$$

and

$$\epsilon_{\text{est}} = \exp(-2t/n) \tag{A7}$$

for $t > 0$.
6. Based on the adjusted value $\mathsf{M}_{\mathsf{adj}}$, evaluate the amount of min-entropy in the measurement outcomes of the quantum device. This is performed using the analytic expression in [70], which applies to two output bits per round. For details, see Section 4.3 *Quantum devices, Bell tests, and guessing probabilities* [47]. Note that the relationship between the guessing probability and min-entropy can be found in Appendix A.3, Equation (28) of [47].
7. Take two of the three output bits (e.g., $\mathbf{a}, \mathbf{b}$, discard $\mathbf{c}$) for randomness extraction.
8. Perform strong two-source randomness extraction using the quantum computer outputs $(\mathbf{a}, \mathbf{b})$ with a fresh string of randomness from an RNG, if the sum of the min-entropies of each bit string is high enough for extraction. The output is a near-perfect bit string, which we call the seed.
9. Repeatedly perform strong seeded extraction using the generated seed and fresh strings from the initial RNG, concatenating the output following the same logic as level 3, i.e., two-source extraction.

For the extractor implementation, we again use the Circulant extractor, and steps (8) and (9) can be viewed as analogous to level 3, where the source Y is instead generated by the described quantum process. We used the H1-1 Quantinuum ion-trap quantum computer as our quantum device, executing 1.8 × 10^6 circuits to obtain a weakly random seed of 3.6 × 10^6 bits. This experiment took approximately 33.5 h of quantum computing time. We obtain $\mathsf{M}_{\mathsf{obs}} = 3.83 \to \mathsf{M}_{\mathsf{adj}} \approx 3.75 \to \alpha_{\mathsf{Q}} \geq 0.518$, where α_{Q} is the min-entropy rate of the quantum computer outputs $(\mathbf{a}, \mathbf{b})$. We note that the min-entropy of the LSFR was too low to perform physical extraction, as a necessary condition is $\alpha_{\mathsf{RNG}} > 1 - \alpha_{\mathsf{Q}}$.

Appendix F.2. Full Results

Table A19. Statistical test results for RDSEED as the weak input source to the physical randomness extractor hierarchy level, implemented with the Circulant extractor. The seed is randomness generated using the semi-device-independent randomness amplification protocol outlined in Appendix F.1. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
RDSEED PE 1	0	0 (1)	0	0	0	0	0 (0)
RDSEED PE 2	0	0 (0)	0	0	0	0	0 (2)
RDSEED PE 3	0	0 (0)	0	0	0	0	0 (0)

Table A19. *Cont.*

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
RDSEED PE 4	0	0 (1)	0	0	0	0	0 (1)
RDSEED PE 5	0	0 (0)	0	0	0	1	0 (0)
Total	**0**	**0 (2)**	**0**	**0**	**0**	**1**	**0 (3)**

Table A20. Statistical test results for IDQ Quantis as the weak input source to the physical randomness extractor hierarchy level, implemented with the Circulant extractor. The seed is randomness generated using the semi-device-independent randomness amplification protocol outlined in Appendix F.1. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parentheses.

RNG	NIST (15)	Diehard (18)	ENT (6)	SmallCrush (15)	Alphabit (17)	Rabbit (40)	PractRand (920)
IDQ Quantis PE 1	0	0 (1)	0	0	0	0	0 (1)
IDQ Quantis PE 2	0	0 (0)	0	0	0	1	0 (2)
IDQ Quantis PE 3	0	0 (0)	1	0	0	0	0 (3)
IDQ Quantis PE 4	0	0 (2)	0	0	0	0	0 (0)
IDQ Quantis PE 5	0	0 (0)	0	0	0	1	0 (1)
Total	**0**	**0 (3)**	**1**	**0**	**0**	**2**	**0 (7)**

References

1. Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Barker, E.; Leigh, S.; Levenson, M.; Vangel, M.; Banks, D.; Heckert, A.; et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; US Department of Commerce, Technology Administration, National Institute of Standards and Technology: Washington, DC, USA, 2001; Volume 22,
2. 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 28 November 2024).
3. Brown, R.G.; Eddelbuettel, D.; Bauer, D. *Dieharder*; Physics Department, Duke University: Durham, NC, USA, 2018.
4. L'ecuyer, P.; Simard, R. TestU01: AC library for empirical testing of random number generators. *ACM Trans. Math. Softw.* **2007**, *33*, 1–40. [CrossRef]
5. Walker, J. A Pseudorandom Number Sequence Test Program. Available online: https://www.fourmilab.ch/random/ (accessed on 28 November 2024).
6. Doty-Humphrey, C. PractRand Official Site. 2018. Available online: http://pracrand.sourceforge.net (accessed on 28 November 2024).
7. Foreman, C.; Yeung, R.; Edgington, A.; Curchod, F.J. Cryptomite: A versatile and user-friendly library of randomness extractors. *arXiv* **2024**, arXiv:2402.09481.
8. Soto, J. Statistical testing of random number generators. In Proceedings of the 22nd national information systems security conference, Arlington, VA, USA, 18–21 October 1999; NIST: Gaithersburg, MD, USA, 1999; Volume 10, p. 12.
9. Tsvetkov, E.A. Empirical tests for statistical properties of some pseudorandom number generators. *Math. Model. Comput. Simulations* **2011**, *3*, 697–705. [CrossRef]
10. Hamburg, M.; Kocher, P.; Marson, M.E. Analysis of Intel's Ivy Bridge Digital Random Number Generator. 2012. Available online: http://www.cryptography.com/public/pdf/Intel_TRNG_Report_20120312.pdf (accessed on 28 November 2024).
11. Jun, B.; Kocher, P. *The Intel Random Number Generator*; White Paper; Cryptography Research Inc.: San Francisco, CA, USA, 1999; Volume 27, pp. 1–8.
12. Tsoi, K.H.; Leung, K.H.; Leong, P.H.W. High performance physical random number generator. *IET Comput. Digit. Tech.* **2007**, *1*, 349–352. [CrossRef]
13. Zhang, L.; Pan, B.; Chen, G.; Guo, L.; Lu, D.; Zhao, L.; Wang, W. 640-Gbit/s fast physical random number generation using a broadband chaotic semiconductor laser. *Sci. Rep.* **2017**, *7*, 45900. [CrossRef]
14. Williams, C.R.; Salevan, J.C.; Li, X.; Roy, R.; Murphy, T.E. Fast physical random number generator using amplified spontaneous emission. *Opt. Express* **2010**, *18*, 23584–23597. [CrossRef]
15. Sun, Y.; Lo, B. Random Number Generation Using Inertial Measurement Unit Signals for On-body IoT Devices. In *Living in the Internet of Things: Cybersecurity of the IoT*; IET: London, UK, 2018; pp. 1–9. [CrossRef]
16. Cho, S.M.; Hong, E.; Seo, S.H. Random number generator using sensors for drone. *IEEE Access* **2020**, *8*, 30343–30354. [CrossRef]
17. Xu, B.; Chen, Z.; Li, Z.; Yang, J.; Su, Q.; Huang, W.; Zhang, Y.; Guo, H. High speed continuous variable source-independent quantum random number generation. *Quantum Sci. Technol.* **2019**, *4*, 025013. [CrossRef]

18. Ó Dúill, S.; Rodriguez, L.; Alvarez-Outerelo, D.; Diaz-Otero, F.J.; Sharma, A.; Smyth, F.; Barry, L.P. Operation of an Electrical-Only-Contact Photonic Integrated Chip for Quantum Random Number Generation Using Laser Gain-Switching. *Optics* **2023**, *4*, 551–562. [CrossRef]
19. Jacak, M.M.; Jóźwiak, P.; Niemczuk, J.; Jacak, J.E. Quantum generators of random numbers. *Sci. Rep.* **2021**, *11*, 16108. [CrossRef]
20. Keshavarzian, P.; Ramu, K.; Tang, D.; Weill, C.; Gramuglia, F.; Tan, S.S.; Tng, M.; Lim, L.; Quek, E.; Mandich, D.; et al. A 3.3-Gb/s SPAD-Based Quantum Random Number Generator. *IEEE J. Solid-State Circuits* **2023**, *58*, 2632–2647. [CrossRef]
21. Hurley-Smith, D.; Hernandez-Castro, J. Quantum leap and crash: Searching and finding bias in quantum random number generators. *ACM Trans. Priv. Secur.* **2020**, *23*, 1–25. [CrossRef]
22. ID Quantique. *Quantis: Quantum Random Number Generator*; ID Quantique: Geneve, Switzerland, 2004.
23. 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. [CrossRef]
24. Seyhan, K.; Akleylek, S. Classification of random number generator applications in IoT: A comprehensive taxonomy. *J. Inf. Secur. Appl.* **2022**, *71*, 103365. [CrossRef]
25. Shaltiel, R. An introduction to randomness extractors. In Proceedings of the International Colloquium on Automata, Languages, and Programming, Zurich, Switzerland, 4–8 July 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 21–41.
26. Kwok, S.H.; Ee, Y.L.; Chew, G.; Zheng, K.; Khoo, K.; Tan, C.H. A comparison of post-processing techniques for biased random number generators. In Proceedings of the Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication: 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Greece, 1–3 June 2011; Proceedings 5. Springer: Berlin/Heidelberg, Germany, 2011; pp. 175–190.
27. Ma, X.; Xu, F.; Xu, H.; Tan, X.; Qi, B.; Lo, H.K. Postprocessing for quantum random-number generators: Entropy evaluation and randomness extraction. *Phys. Rev. A* **2013**, *87*, 062327. [CrossRef]
28. Meier, W.; Staffelbach, O. The self-shrinking generator. In Proceedings of the Workshop on the Theory and Application of of Cryptographic Techniques, Perugia, Italy, 9–12 May 1994; Springer: Berlin/Heidelberg, Germany, 1994; pp. 205–214.
29. Canetti, R. Universally composable security: A new paradigm for cryptographic protocols. In Proceedings of the Proceedings 42nd IEEE Symposium on Foundations of Computer Science, Newport Beach, CA, USA, 8–11 October 2001; IEEE: Piscataway, NY, USA, 2001; pp. 136–145.
30. Hurley-Smith, D.; Hernandez-Castro, J. Certifiably biased: An in-depth analysis of a common criteria EAL4+ certified TRNG. *IEEE Trans. Inf. Forensics Secur.* **2017**, *13*, 1031–1041. [CrossRef]
31. Ortiz-Martin, L.; Picazo-Sanchez, P.; Peris-Lopez, P.; Tapiador, J. Heartbeats do not make good pseudo-random number generators: An analysis of the randomness of inter-pulse intervals. *Entropy* **2018**, *20*, 94. [CrossRef]
32. Shrimpton, T.; Terashima, R.S. A provable-security analysis of Intel's secure key RNG. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, 26–30 April 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 77–100.
33. Burciu, P.; Simion, E. A Systematic Approach of NIST Statistical Tests Dependencies. *J. Electr. Eng. Electron. Control Comput. Sci.* **2019**, *5*, 1–6.
34. Hamano, K.; Kaneko, T. Correction of overlapping template matching test included in NIST randomness test suite. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **2007**, *90*, 1788–1792. [CrossRef]
35. Kowalska, K.A.; Fogliano, D.; Coello, J.G. On the revision of NIST 800-22 Test Suites. *Cryptology ePrint Archive*. 2022. Available online: https://eprint.iacr.org/2022/540.pdf (accessed on 28 November 2024).
36. Marton, K.; Suciu, A. On the interpretation of results from the NIST statistical test suite. *Sci. Technol.* **2015**, *18*, 18–32.
37. Saarinen, M.J.O. NIST SP 800-22 and GM/T 0005-2012 Tests: Clearly Obsolete, Possibly Harmful. In Proceedings of the 2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Genoa, Italy, 6–10 June 2022.
38. Sýs, M.; Obrátil, L.; Matyáš, V.; Klinec, D. A Bad Day to Die Hard: Correcting the Dieharder Battery. *J. Cryptol.* **2022**, *35*, 1–20. [CrossRef]
39. Stępień, R.; Walczak, J. Statistical analysis of the LFSR generators in the NIST STS test suite. *Comput. Appl. Electr. Eng.* **2013**, *11*, 356–362.
40. McKay, K. *Users Guide to Running the Draft NIST SP 800-90B Entropy Estimation Suite*; Technical Report SP; NIST: Gaithersburg, MD, USA, 2016.
41. Buller, D.; Kaufer, A.; Roginsky, A.; Sönmez Turan, M. *Discussion on the Full Entropy Assumption of the SP 800-90 Series*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2022.
42. Zhu, S.; Ma, Y.; Chen, T.; Lin, J.; Jing, J. Analysis and improvement of entropy estimators in NIST SP 800-90B for non-IID entropy sources. *IACR Trans. Symmetric Cryptol.* **2017**, *2017*, 151–168. [CrossRef]
43. Von Neumann, J. Various techniques used in connection with random digits. *John Neumann Collect. Work.* **1963**, *5*, 768–770.
44. Ball, M.; Goldreich, O.; Malkin, T. Randomness extraction from somewhat dependent sources. In Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS 2022), Berkeley, CA, USA, 31 January–3 February 2022; Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Wadern, Germany, 2022.
45. Arnon-Friedman, R.; Portmann, C.; Scholz, V.B. Quantum-proof multi-source randomness extractors in the Markov model. *arXiv* **2015**, arXiv:1510.06743.

46. Kelsey, J.; Brandão, L.T.; Peralta, R.; Booth, H. *A Reference for Randomness Beacons: Format and Protocol Version 2*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2019.
47. Foreman, C.; Wright, S.; Edgington, A.; Berta, M.; Curchod, F.J. Practical randomness amplification and privatisation with implementations on quantum computers. *Quantum* **2023**, *7*, 969. [CrossRef]
48. Santha, M.; Vazirani, U.V. Generating quasi-random sequences from semi-random sources. *J. Comput. Syst. Sci.* **1986**, *33*, 75–87. [CrossRef]
49. Frauchiger, D.; Renner, R.; Troyer, M. True randomness from realistic quantum devices. *arXiv* **2013**, arXiv:1311.4547.
50. Dodis, Y.; Elbaz, A.; Oliveira, R.; Raz, R. Improved randomness extraction from two independent sources. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 334–344.
51. Krawczyk, H. LFSR-based hashing and authentication. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 21–25 August 1994; Springer: Berlin/Heidelberg, Germany, 1994; pp. 129–139.
52. Trevisan, L. Construction of extractors using pseudo-random generators. In Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, Atlanta, GA, USA, 1–4 May 1999; pp. 141–148.
53. Vadhan, S.P. Pseudorandomness. *Found. Trends® Theor. Comput. Sci.* **2012**, *7*, 1–336. [CrossRef]
54. Brunner, N.; Cavalcanti, D.; Pironio, S.; Scarani, V.; Wehner, S. Bell nonlocality. *Rev. Mod. Phys.* **2014**, *86*, 419. [CrossRef]
55. Acín, A.; Masanes, L. Certified randomness in quantum physics. *Nature* **2016**, *540*, 213–219. [CrossRef]
56. Quantinuum. H1-1. 1–4 November 2021. Available online: https://www.quantinuum.com/ (accessed on 28 November 2024).
57. Turan, M.S.; Barker, E.; Kelsey, J.; McKay, K.A.; Baish, M.L.; Boyle, M. Recommendation for the entropy sources used for random bit generation. *NIST Spec. Publ.* **2018**, *800*, 102.
58. Mascagni, M.; Srinivasan, A. Algorithm 806: SPRNG: A scalable library for pseudorandom number generation. *ACM Trans. Math. Softw.* **2000**, *26*, 436–461. [CrossRef]
59. Gustafson, H.; Dawson, E.; Nielsen, L.; Caelli, W. A computer package for measuring the strength of encryption algorithms. *Comput. Secur.* **1994**, *13*, 687–697. [CrossRef]
60. Pont, J.; Calvin Brierley, J.H.C. BitReps. Available online: https://github.com/jjp31/bitreps-1/tree/master (accessed on 28 November 2024).
61. Piras, C. RaBiGeTe—Random Bit Generators Tester. Available online: http://cristianopi.altervista.org/RaBiGeTe_MT/ (accessed on 28 November 2024).
62. Datta, D.; Datta, B.; Dutta, H.S. Design and implementation of multibit LFSR on FPGA to generate pseudorandom sequence number. In Proceedings of the 2017 Devices for Integrated Circuit (DevIC), Kalyani, India, 23–24 March 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 346–349.
63. Sahithi, M.; MuraliKrishna, B.; Jyothi, M.; Purnima, K.; Rani, A.J.; Sudha, N. Implementation of random number generator using LFSR for high secured multi purpose applications. *Int. J. Comput. Sci. Inf. Technol.* **2012**, *3*, 3287–3290.
64. Ekdahl, P. *On LFSR Based Stream Ciphers-Analysis and Design*; Lund University: Lund, Sweden, 2003.
65. Panda, A.K.; Rajput, P.; Shukla, B. FPGA implementation of 8, 16 and 32 bit LFSR with maximum length feedback polynomial using VHDL. In Proceedings of the 2012 International Conference on Communication Systems and Network Technologies, Rajkot, India, 11–13 May 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 769–773.
66. Kessler, M.; Arnon-Friedman, R. Device-independent randomness amplification and privatization. *IEEE J. Sel. Areas Inf. Theory* **2020**, *1*, 568–584. [CrossRef]
67. Chung, K.M.; Shi, Y.; Wu, X. Physical randomness extractors: Generating random numbers with minimal assumptions. *arXiv* **2014**, arXiv:1402.4797.
68. Ramanathan, R. Finite Device-Independent Extraction of a Block Min-Entropy Source against Quantum Adversaries. *arXiv* **2023**, arXiv:2304.09643.
69. Mermin, N.D. Extreme quantum entanglement in a superposition of macroscopically distinct states. *Phys. Rev. Lett.* **1990**, *65*, 1838. [CrossRef]
70. Woodhead, E.; Bourdoncle, B.; Acín, A. Randomness versus nonlocality in the Mermin-Bell experiment with three parties. *Quantum* **2018**, *2*, 82. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.