# 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, United Kingdom*
[2] *Department of Computer Science, University College London, London, United Kingdom*
[3] *Quantinuum, 17 Beaumont Street, Oxford OX1 2NA, United Kingdom*
[4] *Department of Computer Science, University of Oxford, Oxford, United Kingdom*
[5] *Quantinuum, Terrington House, 13–15 Hills Road, Cambridge CB2 1NL, United Kingdom*

Random number generators (RNGs) are notoriously hard to build and test, especially in a cryptographic setting. Although one cannot conclusively determine the quality of an RNG by testing the statistical properties of its output alone, running numerical tests is both a powerful verification tool and the only universally applicable method. In this work, we present and make available a comprehensive statistical testing environment (STE) that is based on existing statistical test suites. The STE can be parameterised to run lightweight (i.e. fast) all the way to intensive testing, which goes far beyond what is required by certification bodies. With it, we benchmark the statistical properties of several RNGs, comparing them against each other. We then present and implement a variety of post-processing methods, in the form of randomness extractors, which improve the RNG's output quality under different sets of assumptions and analyse their impact through numerical testing with the STE.



**Contents**



*Electronic address: cameron.foreman@quantinuum.com
†Electronic address: richie.yeung@quantinuum.com

## I. 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 strongly depending on the context in which it is used. In cryptography, for example, random numbers should be *unpredictable* after they have been generated, 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 are 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 that sense, unpredictability from the user's perspective is a necessary (but not sufficient) condition for the unpredictability of an adversary. This idea motivates the use of numerical testing of a RNG's outputs, which serves as a means of randomness validation, i.e. to detect failure 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 getting 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). The purpose of this certification process is for a third party, for example NIST, to provide assurance that the RNG has been built and tested according to the best practices. In fact, in almost all applications related to cryptography, most companies see the certification by NIST (or another equivalent body) of the RNG as a requirement for using it. NIST and BSI's standards require both a detailed modelling of the underlying physical process and numerical testing of the RNG's output statistics in order for a hardware RNG to be compliant. Partly because of this, numerous statistical test suites have been developed. The best known are NIST's [1] and the Dieharder [2, 3] suite, but other useful ones exist e.g. [4–6]. Together, the tests contained within these suites enable the comprehensive analysis of a wide array of statistical characteristics. Despite their usefulness, these suites are often complicated to use and their output subtle to analyse. Moreover, it is desirable to combine tests from different suites in order to push the statistical testing further.

Because of this, in our work we select and parameterise specific statistical tests from existing suites to create a statistical testing environment (STE) with a tunable intensity from lightweight to intense, with a recommended setting offering a good trade-off between its computational cost and its effectiveness in detecting statistical bias. Due to its ability to be tuned, the suite enables testing beyond that of the individual suites it selects from, and allows for more rigorous testing of an RNG compared to the requirements of standards bodies. We make our testing environment easy to use and openly available at https://github.com/CQCL/random_test. Using the STE's most intense version, 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 for the analysis of the results of the overall numerical test results, which is not obvious otherwise.

We then use the STE to test the impact of different techniques to improve the RNG's output by post-processing it. For this, we present and implement a variety of randomness extraction methods, with the mathematical algorithms taken from `Cryptomite`, the software library of randomness extractors that we developed in [7]. Extraction methods are applied to the output of the RNG to improve its quality, e.g. by removing any bias or dependencies between bits. The set-up is illustrated in Figure 1. Each of the randomness extraction methods we present relies on its own set of assumptions, which can be compared against each other. Such assumptions are, for example, additional structure of the RNG's outputting process, i.e. that each output bit is produced in an identical and independently distributed (I.I.D) manner, or the need for a short pre-existing (near-)perfect random bit string as a resource. These assumptions need to be justified in practice, and statistical testing is used to assess whether each post-processing method was successful from a statistical perspective.

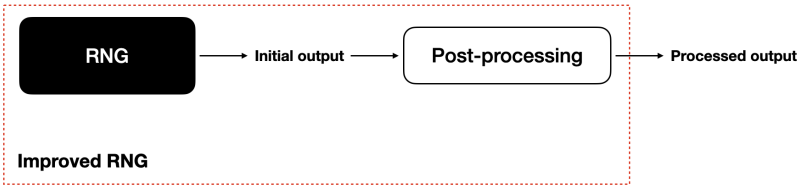


FIG. 1: This figure illustrates our implementation set-up. The black box represents one of the initial RNGs we test and the dashed box denotes the new, in principle improved, RNG with additional post-processing applied.

### A. Related work

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, empirical testing of specific RNGs was performed, such as [8, 9], which analysed and compared the results of numerous statistical tests on a variety of pseudo-RNGs (PRNGs). Other works have considered 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 [11] and [12–16] which develop, implement and statistically test different TRNGs. Recently, this has been extended to quantum RNGs (QRNGs), for example, [17–20] and [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 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 randomness extractors[1]. These conditioners are therefore 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 effect (from a statistical perspective) of different post-processing methods.

### B. Summary of results

In summary, our results and observations are:

- To make available our STE, a powerful, flexible and easy to use statistical testing environment together with the framework to analyse its results.
- To intensely test the output statistics 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].
- To present and implement a variety of post-processing techniques, in the form of randomness extractors, to improve the quality of the outputs for each of the three RNGs. This set of post-processing methods is made of four levels, requiring increasingly more sophisticated implementations: deterministic (level 1), seeded (level 2), two-source (level 3) and physical (level 4) extractors. It goes significantly further than the study and comparison of different types of extractors in [26] and [27], which respectively only focus on deterministic or seeded extractors. Together with the software library `Cryptomite`, presented separately in [7] and which we use for our extractor implementations, this allows RNG developers to carefully choose and implement suitable post-processing.
- To use the STE to intensely analyse the effect that each level of the post-processing has on the output of the different RNGs. Our main observations are that:
  - The RNGs that failed numerical testing without post-processing still fail when simpler post-processing methods (level 1) are applied, although an improvement is indeed observed.
  - All our implementations at levels 2, 3 and 4 successfully post-process, from a statistical perspective, the output of the three RNGs that we used.

[1] Randomness extractors can be understood as conditioners that have information-theoretic security, i.e. do not rely on computational assumptions on the adversary.

– Low entropy sources, for example the post-processed 32-bit LFSR, can successfully pass our intense statistical testing when the right post-processing is applied. This is indeed unsurprising based on the existence of PRNGs, but the poorness of the PRNG used illustrates the limitation of statistical testing when performed alone, i.e. without a precise model and justification for the unpredictability of the underlying physical process.

We note that there are sometimes important differences between the ideal version of a certain extraction method and its actual implementation. Because of this, we ensure to explicitly state both fundamental and added implementation assumptions at each level.

## II. TOOLS AND DEFINITIONS

As we stated in the introduction, randomness is different when considering the perspective of a user or that of an adversary. The difference between *statistical* and *cryptographic* randomness can be understood by considering a hypothetical game in which an adversary tries to distinguish the real output of an RNG from that of an ideal RNG, i.e. one whose output distribution is uniformly distributed. The difference then lies, mainly, in the information available to the adversary in the game[2]. In the case of cryptographic randomness, the adversary has access to additional (aka side) information that is not in the hands of the user, i.e. its predictive power cannot be quantified directly by studying the output's statistical properties alone. Examples of such side information include a better model of the underlying physical process (the entropy source) or of the device's environment, or even information leaked through side-channels. For statistical randomness, one assumes that there is no additional side information, since only the statistical properties of the output is analysed.

As an example, pseudo-RNGs (PRNGs) are mathematical algorithms that *expand* an initial, short, string of random numbers into a larger output string. The initial random string is called the seed of the PRNG and needs to be obtained independently. The (larger) output of the PRNG is then indistinguishable from the uniform distribution to an adversary who is unable to solve a given computational problem efficiently, e.g. the learning with errors (LWE) problem. In this case, there is a big difference between an adversary with no additional information and one who knows the seed. In one case where the seed is unknown, the output is statistically indistinguishable from the uniform distribution (and therefore should pass statistical tests). In the other case, however, the output can be recomputed directly from the seed and therefore be distinguished from the ideal output. This is why it is crucial to keep the seed private when using a PRNG.

In our work, we study the output distribution of different RNGs directly, without considering additional information i.e. we test for statistical randomness. We then study, in the same way, the effect of different post-processing methods applied to the output of the RNG. We consider the specific case of bits, i.e. the RNG outputs an $n$ length bit string $X \in \{0,1\}^n$, for simplicity - although one can also study RNGs whose output alphabet is larger. In some cases, we talk about sizes in bytes, where one byte is eight bits. We denote the random variable $X$ and its specific realisation $x$ for $X = x$. The set-up is illustrated in Figure 2.

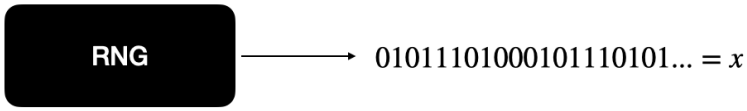


FIG. 2: An illustration of the set-up we consider. An RNG generates a bit string $X = x$ of length $n$. In this work, we first study the statistical properties of the realisation $x$ of the (random variable) $X$. Then, we analyse the effect of different post-processing methods applied to it.

The amount of randomness 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

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

[2] In practice, another difference is the computational power available, since a user can only run limited statistical testing when there is no such limitation for a potentially computationally unbounded adversary.

This can be interpreted as the minimum amount of randomness, in bits, a variable $X$ has, when there is no side information available. The quantity $P_g = \max_{x \in \{0,1\}^n} \Pr(X = x)$ is the guessing probability of $X$. 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 $X_i \in \{0,1\}^n$ for $i \in \mathbb{N}$ is said to have block min-entropy[3] $k$, if

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

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

**Definition 3** (Min-entropy rate)**.** The min-entropy rate of a random variable $X \in \{0,1\}^n$ is

$$\alpha = \frac{\mathrm{H}_\infty(X)}{n}. \tag{3}$$

This can be interpreted as the minimum amount of randomness $X$ has per bit, on average.

**Definition 4** (Statistical distance)**.** The statistical distance, $\Delta$, between two random variables, $X, Z \in \{0,1\}^n$ is defined as

$$\Delta(X, Z) = \frac{1}{2} \sum_{v \in \{0,1\}^n} |\Pr(X = v) - \Pr(Z = v)|. \tag{4}$$

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

**Definition 5** ($\epsilon$-perfect randomness)**.** A random variable $X$ on $\{0,1\}^n$ is said to be $\epsilon$-perfectly random, if,

$$\Delta(X, \mathrm{U}_n) \leq \epsilon, \tag{5}$$

where $\mathrm{U}_n$ 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 distinguishing advantage at most $\epsilon$, i.e. the distinguisher in the game described above is successful with probability at most $\frac{1}{2} + \epsilon$. When $\epsilon = 0$, the random variable is said to be perfectly random. This definition is *universally-composable* [28], i.e. $X$ can be used safely in other applications.

## III. 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*, which directly implies that the output can be distinguished from the ideal uniform distribution at some confidence level, as described in the previous section. The hypothesis that a random variable is uniformly distributed is known as the null hypothesis $\mathsf{H_0}$. 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 can be defined as follows:

---

[3] This definition can easily be generalised to the case where each block, or random variable, $X_i$ has different size $n_i$, i.e. $X_i \in \{0,1\}^{n_i}$, and min-entropy $k_i$.

**Definition 6** (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

$$\mathsf{p} = \Pr(T \leq t | \Delta(X, \mathrm{U}_n) = 0) \tag{6}$$

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

A range of p-values is defined that provide a threshold at which the null hypothesis is rejected i.e. when the test is deemed to fail. If a test ensures there is, at most, a 1% chance it incorrectly rejects that the RNG is producing uniform random numbers[4], 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 high probability. For example, if all statistical test 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 $\mathsf{p}_{\mathsf{type2}} = \mathsf{p}_{\mathsf{type2}}^{\mathsf{test1}} \cdot \mathsf{p}_{\mathsf{type2}}^{\mathsf{test2}} \cdot \ldots \cdot \mathsf{p}_{\mathsf{type2}}^{\mathsf{testn}}$, where $n$ is the number of tests performed.

### A. Existing test suites

#### 1. *NIST statistical test suite*

The NIST statistical test suite (SP 800-22) [1] is the best known and widely used. This suite contains 15 tests, some of which have multiple sub-tests, and passing them is a requirement for RNG certification by numerous governing bodies such as NIST and BSI. During testing, a file of randomness is split into sub-strings, and each sub-string is tested individually. The user can define the number of sub-strings and the total bit string size to analyse.

The user guide suggests using 100 sub-strings of $10^6$ bits, which requires a minimum of $10^8$ bits or equivalently, 12.5 megabytes (MB), for testing. For each statistical test, a set of p-values is calculated, where each p-value corresponds to one of the sub-strings. The pass/fail analysis is then performed using this set of p-values, giving two test results. The first result is a statistical test on the observed p-values, assessing the null hypothesis that the set of p-values is uniformly distributed, at the 1% significance level. The second result is to check that sufficiently many sub-strings pass the test at the 1% confidence level (i.e. have p-values in the range $[0.005, 0.995]$). In order for an RNG to be deemed as producing satisfactory random numbers, it must pass both results for each test. See [29] for further details.

#### 2. *Diehard(er) statistical test suite*

The Dieharder statistical test suite consists of the 18 Diehard tests and additional tests, including some from the NIST test suite. Similarly to the NIST suite, this is one of the core test suites used by RNG certification bodies. Failure is concluded when $\mathsf{p} \notin [0.0005, 1 - 0.0005]$ and a test is deemed 'weak' if $\mathsf{p} \in [0.0005, 0.005] \cup [1 - 0.0005, 1 - 0.005]$. This high tolerance for poor test statistics means that a bad RNG may sometimes pass the Diehard(er) test suite, but failing Diehard(er) is a strong indicator of non-uniformity.

The Diehard(er) tests require a significant amount of random numbers to avoid re-use of the input random numbers, which gives erroneous results. For this reason, we suggest test sizes of at least 1 gigabyte (GB) of random numbers. If testing smaller file sizes, one can modify the default parameters to avoid these issues. Our testing environment uses the default parameters for each test.

#### 3. *TestU01 statistical test suite*

TestU01 is a software library written in C that conducts RNG statistical testing with pre-compiled statistical test batteries. These batteries vary widely in the amount of tests and the amount of randomness they require. For full details of which tests these batteries include, see [4].

---

[4] Known as the type 1 error – when a statistical test incorrectly rejects a true null hypothesis.

Test p-values are shown if $\mathsf{p} \notin [0.001, 0.999]$, so we will use this as our failure criteria. During our testing, we use the Alphabit, Rabbit and SmallCrush batteries, which are all contained in TestU01. In order to run these tests, files should contain at least $2^{25}$ random bytes ($\approx$ 35MB). We omit Crush and BigCrush from our work due to their excessive runtime and their large random number requirement, however they can be executed with our statistical testing environment.

#### 4. *ENT statistical test suite*

The ENT test suite is a small but efficient set of 6 statistical tests. This test suite has been used to show bias in a commercial quantum RNG by showing consistent failure in the so-called $\chi^2$ test [30] (we replicate these results of an RNG which we acquired independently, see Table III).

The ENT tests output test statistics directly, without giving a pass/failure threshold, so we assess failure based on the table found in Table 3 of [31]. Although there is no specific guidance on required input lengths, we found that the tests give suspicious results when input sizes are below 0.5GB.

#### 5. *PractRand statistical test suite*

PractRand is a C++ library of statistical tests for RNGs. It was designed with practicality in mind – to be efficient, user-friendly and detect significant bias in RNGs. According to its documentation, it boasts quicker runtime than most test suites (which we confirm, see Table II), unique interfacing, no (in principle) maximum input length limit and some original tests. It performs tests based on size of the input file, testing on subsets of size $2^{24+x}$ bytes for $x \in \mathbb{N}$, and performing more tests as $x$ increases. In our testing, we limit the maximum test size to $2^{32}$ bytes ($\approx$ 4.3GB). For more information, see [6], where they have full details and additional analysis, including comparisons between PractRand and other test suites.

There are numerous result ranges that p-values may enter when testing using the PractRand suite, these are "unusual', "mildly suspicious", "suspicious", "suspect" and "fail". Failure is concluded when $\mathsf{p} \notin [10^{-11}, 1 - 10^{-11}]$.

### B. Our statistical testing environment

The interfacing code for our STE can be downloaded at https://github.com/CQCL/random_test. We provide a *Light*, *Recommended* and *All* setting for statistical testing which is detailed below and can be executed using `run_light`, `run_recommended` and `run_all` commands respectively. The NIST statistical test suite is not performed with these commands, since it requires user prompts, but it can be executed individually in the environment. We believe that the *recommended* setting offers a nice trade-off between computational (and thus, environmental) cost and rigorousness, yet goes beyond standard numerical testing required by certification bodies. Using the STE (or by downloading, parameterising and executing the relevant statistical test suites independently), all results in this work can be replicated.

### C. Suggested settings

We now suggest settings for statistical testing, based on knowledge acquired during this research. The runtimes shown are averaged over 10 executions and result from testing a 10Gbit file, except for NIST, where a 100Mbit file is tested (to align with the user guidance). Statistical testing was run on a Dell Precision 7540 personal laptop with 16GB of RAM and a 2.3GHz Intel I9-9880H processor, using the Ubuntu 20.04 operating system.

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

TABLE I: 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 that column is included in the setting.

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

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

#### 1. *Light*

Our suggested light test mode (executed with terminal command `run_light`) of ENT, SmallCrush and PractRand takes under 5 minutes to run and consists of approximately 941 tests. In our numerical testing, we show that this is a sufficient set of tests to detect failure to generate uniform randomness in the RNGs that fail (see Section IV and Section V).

#### 2. *Recommended*

Our recommended setting (executed with terminal command `run_recommended`) includes most light mode suites, with the exception of exchanging TestU01 SmallCrush for Rabbit, but also adding the NIST and Diehard tests. The inclusion of these additional tests cause the runtime to become around 2 hours and increase the total number of tests to 999. The only reduction from the full statistical test mode (executed with terminal command `run_all`) is the exclusion of TestU01 SmallCrush and Alphabit, which are omitted since most of the individual tests are highly correlated to those in Rabbit, for example, the same test with slightly different parameters. The recommended test suite includes all statistical tests required by RNG governing bodies (e.g. NIST and Dieharder), whilst being significantly more powerful than running those alone. In later sections, we show the importance of going beyond these individual test suites, in the sense that we show that an RNG that passes the NIST and Dieharder tests shows significant statistical bias once analysed with our combined STE (see Section IV).

### D. Shortcomings of statistical testing

Fundamentally, statistical tests have limited ability to validate if good random numbers are being produced by an RNG. They should rather be understood as a useful tool to detect 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"[5] (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 have been shown to have issues. For the NIST test suite, the list of implementation issues is extensive, so we summarize 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 results are analysed were discovered, for example, [36] found that although the NIST documentation provides guidance that the analysed RNG is random if all tests are passed, even though truly uniform data has a high probability (80%) of failing at least one NIST statistical (sub-)test. Some work 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 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] they find that over 50% of the Dieharder tests generated biased null hypothesis distributions (which are expected to be uniform).

[5] In this case, the adversary also requires control of an "unprivileged" process, which is a form of side-information that may be hard to obtain in practice.

## IV. STATISTICAL TESTING OF DIFFERENT RNGS

In this section, we use our STE to analyse the statistical properties of random numbers produced by some commonly used RNGs. At this stage, we do not apply any post-processing to the RNGs 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 those cases our statistical analysis applies to the joint system comprising both the source of randomness and the existing post-processing in the device. We then also use and discuss a NIST min-entropy estimation tool, which provides a min-entropy estimate for use in the second part of our work in which we add and analyse further post-processing.

The RNGs that we analyse are:

- 32-bit LFSR: a software pseudo-RNG.
- Intel RDSEED [11]: a hardware RNG based on thermal noise from a ring oscillator, 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 description of the RNGs can be found in Appendix A. The LFSR is still widely used in numerical simulations, although it is known to have flaws [39]. In our work, we study it mainly as a way to benchmark against, serving as the obvious bad choice in the sense that its output exhibits patterns – but we will see that the effect of post-processing its output is statistically non-trivial. Both Intel's RDSEED and IDQ's Quantis are sold as RNGs for cryptographic use. 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 $10 \times 10$Gbit files from each RNG and, similarly, using the NIST test suite performed on $10 \times 100$Mbit files split into 100 sub-strings each of 1Mbit. The NIST min-entropy estimators [29] are used in the so-called non-IID setting on $10 \times 1$Mbit files. This analysis far exceeds that required by certification bodies, so may be a result of independent interest. All testing is done using the default parameters, unless otherwise stated.

### A. Results

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[6], on average. The results we obtained for the three RNGs, are summarised in the following Table III and displayed visually in Figure 4 (level 0).

| 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) |

TABLE III: This table gives the average sum of tests failed for $5 \times 10$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 $5 \times 10$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.

In the statistical tests, the RDSEED RNG performs best, failing the least tests and, without surprise, the 32-bit LFSR performs worst. The poor performance of the LFSR is likely due to its periodicity, since bits repeat every $2^{32} - 1$ (4.3Gbits) and this is less than the size of the files tested. The IDQ Quantis device performs well in the NIST and Diehard tests however fails an ENT test and several of the TestU01 suites tests. 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 [29] are a collection of algorithms that give a standardised way of estimating the min-entropy (as defined in Definition 1) of an RNG's output. These estimators are both useful to evaluate the entropy

[6] 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 each statistical test is independent.

generation of the studied RNG and to calculate a min-entropy bound that we use later to choose the randomness extractors parameters.

| RNG | NIST Min-Entropy Estimator (/byte) | $\overline{\mathsf{est}}$: NIST Min-Entropy Estimator (/bit) | $\sigma$: Sample Standard Deviation (/bit) | $\alpha$: 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 |

TABLE IV: 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 different output test samples. Full results tables can be found in Appendix B.

In Table IV, the NIST min-entropy estimator per byte is the average observed per-byte min-entropy calculated by the NIST min-entropy estimator tool and $\overline{\mathsf{est}}$ is the per-bit average. The sample standard deviation, $\sigma$, describes how much the different test results fluctuated, which we calculate using the expression Equation (B1) and $\alpha$ is a lower bound (with probability 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.

The estimated min-entropy per bit for the IDQ Quantis device was the largest (0.895). In terms of sample standard deviation, the 32-bit LFSR was by far the highest, which indicates the observed min-entropy estimators for different samples fluctuated the most. NIST recommends that a good RNG should have a of min-entropy per bit of at least $1 - 2^{-32}$ [40], which is much larger than the values we observe from using their min-entropy estimator tool. That being said, results suggest that some estimator tests provide significant underestimates [41], which could explain the large disparity between the estimated results and the recommendation of NIST – but underestimates are not a problem in our case.

## V. A VARIETY OF POST-PROCESSING METHODS

### A. Overview

Randomness extractors are mathematical algorithms that distil *weakly* random bit strings[7], 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 want to answer is whether these methods have an observed impact on the statistical properties of the RNGs output. The recipe that we follow is the following:

1. We collect the output of each RNG that we 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.

### B. Randomness extraction methods

We now describe the different post-processing levels 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:

[7] 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.

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 [42], 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 often difficult to justify and hard 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[8] or if it has poor statistical properties. At a fundamental level, seeded extractors are unsatisfying as there is a 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 from 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 [43] or independence only in the sense of a Markov chain [44].

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[9]. 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 a few additional implementation assumptions (to the minimal ones). We come back in detail to physical extractors in Section V H.

When a second bit string of randomness is required (levels 2 and 3), we use the NIST Randomness Beacon [45]. For physical randomness extraction (level 4), we use a semi-device-independent randomness amplification protocol that is an adaptation of [46], which we describe in Section V H. 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.

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

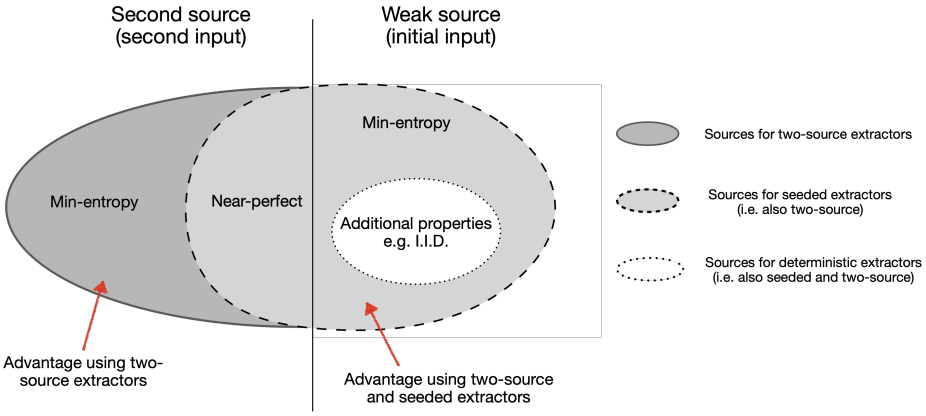


FIG. 3: Illustration of the set of sources, or input distributions, that can be successfully extracted from by different randomness extraction methods. (Right) weak input distributions and (Left) second input, or weak seed, distributions. Deterministic extractors (level 1) require additional properties on the weak input, but do not need a second input source. Seeded extractor (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 on the figure) requires special quantum hardware, which effectively provides the second input with a device-independent lower bound on the min-entropy, requiring minimal added assumptions.

[8] This could happen, for example, if the seed is generated whilst sharing the same environment as the RNG or by an adversary.
[9] For example, that information cannot travel faster than the speed of light.

### C. Results overview

We now present the main results of statistical testing 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 it is subject to, on average, when testing $5 \times 10$Gbit files[10]. This is the criterion we use to call randomness generation *successful* from a statistical perspective (green highlighted area).

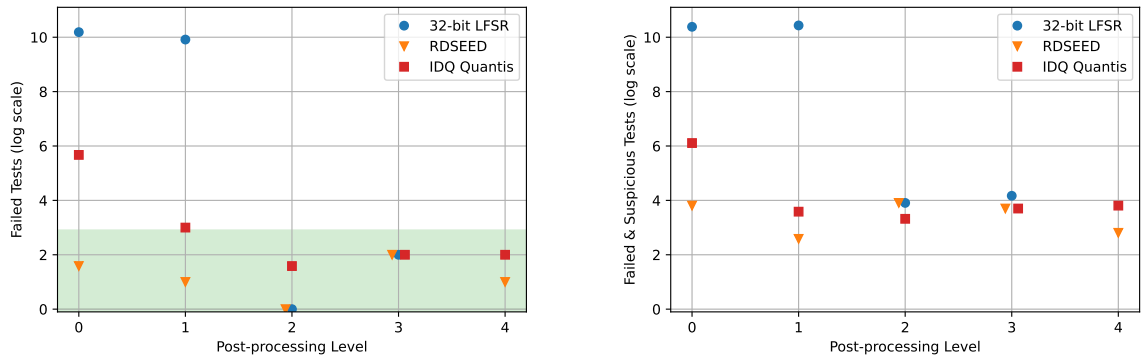


FIG. 4: The above plots show (left) the number of statistical tests failed and (right) failed and suspicious for each initial RNG at each post-processing level. The $x$ axis indicates the level, with step 0 being the initial RNG with no additional post-processing and steps 1-4 are deterministic, seeded, 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 on 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, $\alpha_{\mathsf{RNG}}$, as detailed and evaluated in Section IV.

Ideally, the results would reflect the different levels of the post-processing and the validity of the assumptions 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 which 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 level 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 nontrivial 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 give 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.

### D. 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 we made. For the post-processing algorithms, we use the randomness extractors publicly available from the software library `Cryptomite` [7]. In order to test the randomness quality at each step we generate $5 \times 10$Gbit test files of processed output and perform statistical testing with the 'all' setting (the most

[10] 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.

intense) in the STE.

All randomness post-processing and statistical tests were run on a Dell Precision 7540 personal laptop with 16GB of RAM and a 2.3GHz 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 choose the parameters of the different extractors such that, in theory, the processed output is $\epsilon_{\text{total}}$-perfectly random (see Definition 5), with $\epsilon_{\text{total}} \leq 2^{-32} \approx 10^{-10}$.

### E. 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 7** (Deterministic randomness extractor)**.** A deterministic randomness extractor is a function

$$\mathsf{Ext}_{\mathsf{d}} : \{0,1\}^n \to \{0,1\}^m \tag{7}$$

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

$$\Delta(\mathsf{Ext}_{\mathsf{d}}(X), \mathrm{U}_m) \leq \epsilon \tag{8}$$

where $\mathrm{U}_m$ is the uniform variable on $\{0,1\}^m$.

In words, a deterministic extractor is a function that maps random variables $X$ with specific characteristics, to a new variable $\mathsf{Ext}_{\mathsf{d}}(X)$ that is near-perfectly random. Note that the properties of $X$ required depend on the specific extractor – for example that all the bits in $X$ are I.I.D.. In practice, those properties are hard (or even impossible) to verify and it is preferable to make a claim about the min-entropy only.

The implementation of the deterministic extraction set-up is shown in Figure 5.

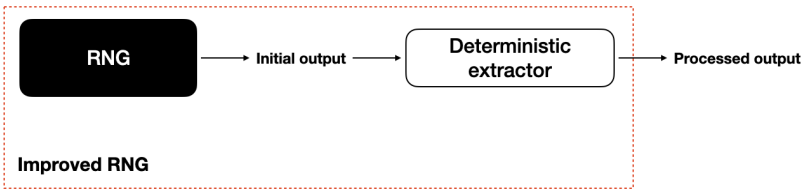


FIG. 5: The level 1 of our post-processing methods is performed by using a deterministic extractor, namely the Von Neumann extractor, on the initial output of the RNG.

We use the Von Neumann extractor [42] 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

$$\Pr(X_{2i} = 0) = \Pr(X_{2i+1} = 0) = p_i \tag{9}$$

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.

### 1. Results

| 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) |

TABLE V: This table gives the sum of tests failed for $5 \times 10$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 re-scale 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.

The statistical test results show that:

- There is an observed improvement, for both the LFSR and IDQ Quantis, compared with the results of the initial RNG testing (Table III), although they are still not successful.
- The number of NIST statistical tests failed by both the LFSR and IDQ Quantis increases when moving from no post-processing to deterministic post-processing. This could happen for many reasons, including that the RNGs have a specific bias structure that is incompatible (and indeed amplified) by the Von Neumann extractor or that there are some fundamental issues with the NIST tests (as suggested in [33–36]).

## F. Level 2: Seeded extraction

The properties required for deterministic extraction (level 1) from an RNG to be successful are hard to justify in practice. Seeded extraction requires only that the initial RNG output has a min-entropy guarantee, i.e. that it is only *somewhat* random – a much weaker requirement on the initial RNGs output, making it more realistic in practice. The cost for this weaker requirement is that a second, independent and (near-)perfectly random string (a seed) now needs to be provided.

**Definition 8** (Seeded randomness extractor)**.** A seeded randomness extractor is a function $\mathsf{Ext}_\mathsf{s} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ 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) then,

$$\Delta(\mathsf{Ext}_\mathsf{s}(X,S), \mathrm{U}_m) \leq \epsilon \tag{10}$$

where $\mathrm{U}_m$ is the uniform distribution on $\{0,1\}^m$.

A seeded extractor can be understood as a randomized function that maps a weakly random variable $X$ to a new variable $\mathsf{Ext}_\mathsf{s}(X,S)$ that is (near-)perfectly random. Note that the seed may be $\epsilon_s$-perfect only, with additive error in the statistical distance above, i.e. $\epsilon \to \epsilon + \epsilon_s$ (see, for example, Appendix A from [47] for a proof). Seeded extractors are a special case of a two-source extractor, which we define later in Definition 10.

**Definition 9** (Strong seeded extractor)**.** A *strong* seeded randomness extractor is a function $\mathsf{Ext}_\mathsf{s} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ such that

$$\Delta([\mathsf{Ext}_\mathsf{s}(X,S), S], [\mathrm{U}_m, S]) \leq \epsilon \tag{11}$$

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

A strong seeded extractor is a randomized 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 re-used with different weak input random variables, allowing a single seed to be used in many extraction rounds. The step of 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).

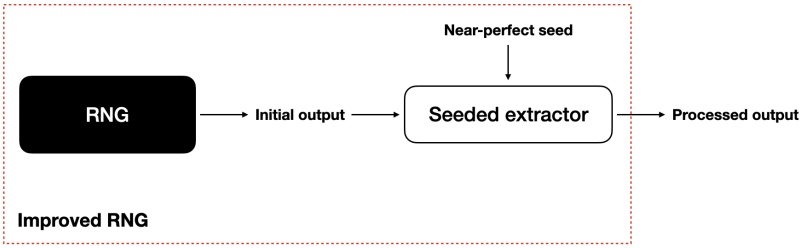


FIG. 6: The set-up for seeded extraction. In this case, the initial output of the RNG only needs to have min-entropy, but extraction requires an additional near-perfectly random bit string (the seed), which needs to be generated independently.

This step can be implemented with the Circulant [7], Dodis et al. [48], Toeplitz [49], and Trevisan [50] 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 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 $|S| = |X_i| + 1 = 10007$. Note that, using Circulant allows 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 [45]. The min-entropy $k_i^{\mathsf{RNG}}$ for each block $X_i$ is $k_i^{\mathsf{RNG}} = \alpha_{\mathsf{RNG}}|X_i|$, where $\alpha_{\mathsf{RNG}}$ is a lower bound on the min-entropy per bit for each initial RNG block of outputs $X_i$, with probability $\epsilon_{\mathsf{est}} < 2^{-32}$ (as found in Equation (B2) of Section IV). The output length after extraction, $m$, is then roughly $m \approx k_i^{\mathsf{RNG}}$.

In order to generate the required $5 \times 10$Gbit's 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:

$$\mathsf{Output} = \left[\mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_1, S), \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_2, S), \ldots, \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_n, S)\right], \tag{12}$$

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

### 1. *Results*

| 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) |

TABLE VI: This table gives the sum of tests failed for $5 \times 10$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.

The observations we draw from the results in Table VI are the following:

- The statistical test results show a significant improvement on the results using deterministic extraction, see Section V E. In particular, now 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, since many 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 $10007 + 32$

(the seed length of the extractor plus the seed length of the 32-bit LFSR), in the 50Gbit of processed output generated, i.e. a true min-entropy rate of, at most, $\alpha = (10007 + 32)/(5 \times 10^{10}) < 10^{-5}$.

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 get detected by our statistical testing.

### G. Level 3: Two-source extraction

Seeded extraction (level 2) requires an independent string of (near-)perfect randomness as an initial resource, which is hard to justify and leads to a 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[11] – 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 10** (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's $\mathrm{H}_\infty(X) \geq k_1$ and $\mathrm{H}_\infty(Y) \geq k_2$ respectively,

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

where $\mathrm{U}_m$ 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 11** (Strong two-source extractor)**.** A two-source randomness extractor is said to be *strong* in the input $Y$ if the function $\mathsf{Ext}_2$ is such that

$$\Delta([\mathsf{Ext}_2(X,Y), Y], [\mathrm{U}_m Y]) \leq \epsilon \tag{14}$$

where $[\cdot,\cdot]$ denotes the concatenation of random variables and $\mathrm{U}_m$ 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.

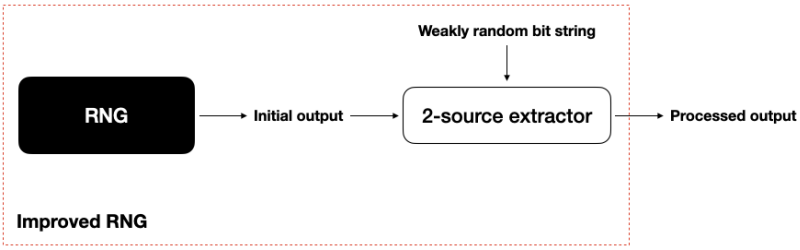


FIG. 7: The set-up for two-source extraction. In this case, the initial output of the RNG only needs to have some min-entropy and extraction requires an additional bit string which is weakly random only, in the sense that it also has min-entropy.

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 Circulant extractor, this requirement is that the sum of the

[11] For example, the case of using a two-source extractor secure in the Markov model [44], 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 [43].

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 $|Y| = |X_i| + 1 = 10007$.

To generate $Y$, we again use the NIST Randomness Beacon, but, in this case we minimise the amount of entropy we assume it contains instead of assuming 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 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 $\alpha_{\mathsf{NIST}}$, as

$$\alpha_{\mathsf{NIST}} = 1.02 - \alpha_{\mathsf{RNG}}, \tag{15}$$

where $\alpha_{\mathsf{RNG}}$ is the min-entropy rate of the initial RNG (found in Section IV). We use 1.02 instead of 1 to account 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 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) use this seed in multiple Circulant seeded extractions on $X_{i\geq 2}$. The multiple outputs of the seeded extractor are concatenated together to obtain a final output of $5 \times 10^{10}$ bits. In other words, the concatenation of the two-source and seeded extractors together form a two-source extractor with advantageous parameters. Therefore, the final output for statistical testing is given by:

$$\mathsf{Output} = \left[\mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_2, S), \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_3, S), \ldots, \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_n, S)\right], \qquad S = \mathsf{Ext}_{2}^{\mathsf{Circulant}}(X_1, Y), \tag{16}$$

where $[\cdot,\cdot]$ 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 $\epsilon_{\mathsf{total}} = \epsilon_{\mathsf{est}} + \epsilon_{\mathsf{ext_1}} + \sum_{j=2}^{n} \epsilon_{\mathsf{ext_j}} \leq 2^{-32}$. A 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 (16)) can be found in [51] Section 6.3. This, combined with the fact that composed output error is the sum of each of the individual extractor errors (in [7]) allows us to calculate $\epsilon_{\mathsf{total}}$.

### 1. *Results*

| 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) |

TABLE VII: This table gives the sum of tests failed for $5 \times 10$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 parenthesis. Full results can be found in Appendix E 2.

Our results show that all RNGs extracted at level 3 are successful from a statistical perspective, like 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 (16) 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, each block produced by the initial RNG must be independent from one another (as well as have block min-entropy). Even in this case, the results were successful statistically. Full explanation and results can be found in Appendix E 1.

## H. 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 required 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 done 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 [52]. The idea is to use the initial RNG to generate random challenges for the quantum device, and then studying 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. For a review on the subject, together with its minimal assumptions (called loopholes), we refer the reader to [53]. This approach crucially relies on quantum resources, which have this self-testing property, and has no classical analogue. See Figure 8 for an illustration.

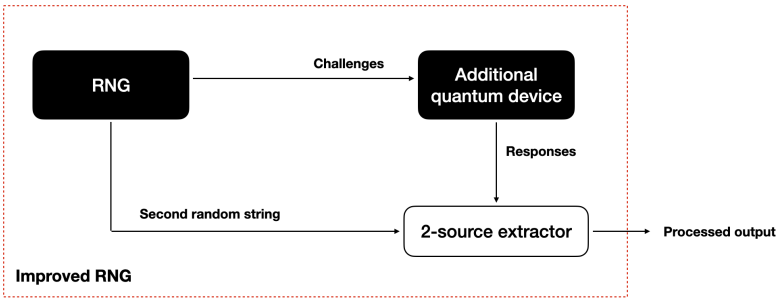


FIG. 8: The set up for level 4: physical randomness extraction. The initial RNG is used twice: first to generate challenges to the quantum device, and second, to provide an extra bit string as input to a two-source extractor. The role of the quantum device is to provide an additional source of randomness. 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.

Today, quantum devices that are capable of running device-independent protocols are extremely hard to build (they require the ability to perform a loophole free Bell Test [52]) 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 protocol thus still has comparatively fewer assumptions than adding standard hardware, but is not minimal either. Such a *semi*-device-independent protocol is the one we implement for our physical extraction method at level 4, based on the randomness amplification protocol described in [46] and implemented on quantum computers. For clarity, the assumptions are:

- 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 device, such as the quantum computers based on ion-traps that we use – see the discussion in [46] (Section 6.2, *Validity of quantum computers for Bell experiments and added assumptions*).

We used the H1-1 Quantinuum ion-trap quantum computer [54] as our device to obtain, from its output, a weakly random bit string size of $3.6 \times 10^6$ bits[12] 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 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[13]. The advantage of using a quantum device, and therefore level 4, is two-fold: a) one gets a rigorous, semi-device-independent,

[12] This means, because of using the Circulant extractor, the input length of the initial RNG block to the two-source extraction is also $3.6 \times 10^6$ bits.

[13] 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 logarithmic min-entropy rate only. However, up to our knowledge no such extractor has been implemented, let alone efficiently, since $0.482 + 0.482 < 1$, making the results of this section more interesting

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 the 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 is below 0.482, see Table IV).

The processed output is then generated in two steps. (1) 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) Use this seed in multiple Circulant seeded extractions on $X_{i\geq 2}$ which are concatenated together to obtain a final output of $5 \times 10^{10}$ bits. In other words, the concatenation of the two-source and seeded extractors together again form a two-source extractor with advantageous parameters. Therefore, the final output for statistical testing is given by:

$$\mathsf{Output} = \left[\mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_2, S), \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_3, S), \ldots, \mathsf{Ext}_{\mathsf{s}}^{\mathsf{Circulant}}(X_n, S)\right], \qquad S = \mathsf{Ext}_{2}^{\mathsf{Circulant}}(X_1, Y), \tag{17}$$

where $[\cdot,\cdot]$ 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 the following table.

### 1. *Results*

| 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) |

TABLE VIII: This table gives the sum of tests failed for $5 \times 10$Gbit samples from level 4. Note: The 32-bit LFSR does not generate any output in this setting, since it's 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 parenthesis. Full results can be found in Appendix F.

The statistical test results show, as for level 2 and 3, that the post-processed RNGs perform well at level 4.

## I. Environmental impact

A consumer-grade laptop consumes roughly 200 W/hour under heavy usage. The total runtime of our test suite with parameter 'all' on is 128 minutes, so we estimate the energy used in conducting a single run of statistical testing to be around 0.425kWh (1'530'000J), which equates to roughly 0.184kg of CO2 (using `https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator`). For the entirety of this work, we performed full testing (i.e. using the all setting) 105 times giving a total energy usage of around 44.625kWh (160'650'000J), and therefore, roughly generated 19.32kg of CO2.

## VI. CONCLUSION AND FUTURE WORK

In this work, we have presented both our STE, a statistical testing environment to analyse the output of RNGs, and a variety of extraction methods to post-process RNGs outputs. Our STE software, documentation and a build file can be found at https://github.com/CQCL/random_test, and the randomness extractor software library `Cryptomite` can be found at https://github.com/CQCL/cryptomite and [7]. Our objective is to make both statistical testing and randomness extraction easy to use, efficient and openly accessible.

The results from our statistical testing tell a mixed story. First, using our STE we intensely tested the output of three RNGs, showing failure for two of them (reproducing and strengthening previous results [10, 21]). For the RNGs that failed, we observed that all of our post-processing methods improved their statistical properties, and in particular, observed that for post-processing levels level 2 or above (seeded, two-source and physical extraction), the processed output was found to be statistically indistinguishable from uniform. Unfortunately, because of the limitations of statistical testing we were unable to find non-artificial examples of RNGs failing when post-processing of level 2 (seeded extraction) was applied, but that are successful when level 3 or higher levels are applied (two-source and physical) – although level 3 is strictly stronger than level 2. The full list of our observations can be found in Section V C and following sections.

We could have gone 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 [55] and Crypt-X [56]. Moreover, we were recently made aware of the numerical tests BitReps [57] and RaBiGeTe [58], 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.

Furthermore, it would be interesting to perform 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 we presented. One could use, for example, vetted conditioning components from NIST [59] and compare their results to the ones we obtained using information-theoretic randomness extractors.

## VII. ACKNOWLEDGEMENTS

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

[1] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, and San Vo. *A statistical test suite for random and pseudorandom number generators for cryptographic applications*, volume 22. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2001.

[2] George Marsaglia. The Marsaglia random number CDROM including the Diehard battery of tests of randomness. http://www.stat.fsu.edu/pub/diehard, 2008.

[3] Robert G Brown, Dirk Eddelbuettel, and David Bauer. Dieharder. *Duke University Physics Department Durham, NC*, pages 27708–0305, 2018.

[4] Pierre L'ecuyer and Richard Simard. TestU01: AC library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)*, 33(4):1–40, 2007.

[5] John Walker. *A Pseudorandom Number Sequence Test Program.*

[6] Chris Doty-Humphrey. PractRand official site. http://pracrand.sourceforge.net, 2018.

[7] Cameron Foreman, Richie Yeung, Alec Edgington, and Florian J Curchod. Cryptomite: A versatile and user-friendly library of randomness extractors. *arXiv preprint arXiv:2402.09481*, 2024.

[8] Juan Soto. Statistical testing of random number generators. In *Proceedings of the 22nd national information systems security conference*, volume 10, page 12. NIST Gaithersburg, MD, 1999.

[9] E. A. Tsvetkov. Empirical tests for statistical properties of some pseudorandom number generators. *Mathematical Models and Computer Simulations*, 3:697–705, 2011.

[10] Mike Hamburg, Paul Kocher, and Mark E Marson. Analysis of Intel's Ivy Bridge digital random number generator. http://www.cryptography.com/public/pdf/Intel_TRNG_Report_20120312.pdf, 2012.

[11] Benjamin Jun and Paul Kocher. The Intel random number generator. *Cryptography Research Inc. white paper*, 27:1–8, 1999.

[12] Kuen Hung Tsoi, Ka Hei Leung, and Philip Heng Wai Leong. High performance physical random number generator. *IET computers & digital techniques*, 1(4):349–352, 2007.

[13] Limeng Zhang, Biwei Pan, Guangcan Chen, Lu Guo, Dan Lu, Lingjuan Zhao, and Wei Wang. 640-Gbit/s fast physical random number generation using a broadband chaotic semiconductor laser. *Scientific Reports*, 7(1):45900, 2017.

[14] Caitlin RS Williams, Julia C Salevan, Xiaowen Li, Rajarshi Roy, and Thomas E Murphy. Fast physical random number generator using amplified spontaneous emission. *Optics express*, 18(23):23584–23597, 2010.

[15] Yingnan Sun and Benny Lo. Random number generation using inertial measurement unit signals for on-body IoT devices. 2018.

[16] Seong-Min Cho, Eungi Hong, and Seung-Hyun Seo. Random number generator using sensors for drone. *IEEE Access*, 8:30343–30354, 2020.

[17] Bingjie Xu, Ziyang Chen, Zhengyu Li, Jie Yang, Qi Su, Wei Huang, Yichen Zhang, and Hong Guo. High speed continuous variable source-independent quantum random number generation. *Quantum Science and Technology*, 4(2):025013, 2019.

[18] Seán Ó Dúill, Leidy Rodriguez, David Alvarez-Outerelo, Francisco J Diaz-Otero, Ankit Sharma, Frank Smyth, and Liam P Barry. Operation of an electrical-only-contact photonic integrated chip for quantum random number generation using laser gain-switching. *Optics*, 4(4):551–562, 2023.

[19] Marcin M Jacak, Piotr Jóźwiak, Jakub Niemczuk, and Janusz E Jacak. Quantum generators of random numbers. *Scientific Reports*, 11(1):16108, 2021.

[20] Pouyan Keshavarzian, Karthick Ramu, Duy Tang, Carlos Weill, Francesco Gramuglia, Shyue Seng Tan, Michelle Tng, Louis Lim, Elgin Quek, Denis Mandich, et al. A 3.3-gb/s spad-based quantum random number generator. *IEEE Journal of Solid-State Circuits*, 2023.

[21] Darren Hurley-Smith and Julio Hernandez-Castro. Quantum leap and crash: Searching and finding bias in quantum random number generators. *ACM Transactions on Privacy and Security (TOPS)*, 23(3):1–25, 2020.

[22] ID Quantique. Quantis: Quantum random number generator, 2004.

[23] Luca Crocetti, Pietro Nannipieri, Stefano Di Matteo, Luca Fanucci, and Sergio Saponara. Review of methodologies and metrics for assessing the quality of random number generators. *Electronics*, 12(3):723, 2023.

[24] Kübra Seyhan and Sedat Akleylek. Classification of random number generator applications in iot: A comprehensive taxonomy. *Journal of Information Security and Applications*, 71:103365, 2022.

[25] Ronen Shaltiel. An introduction to randomness extractors. In *International colloquium on automata, languages, and programming*, pages 21–41. Springer, 2011.

[26] Siew-Hwee Kwok, Yen-Ling Ee, Guanhan Chew, Kanghong Zheng, Khoongming Khoo, and Chik-How Tan. A comparison of post-processing techniques for biased random number generators. In *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication: 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings 5*, pages 175–190. Springer, 2011.

[27] Xiongfeng Ma, Feihu Xu, He Xu, Xiaoqing Tan, Bing Qi, and Hoi-Kwong Lo. Postprocessing for quantum random-number generators: Entropy evaluation and randomness extraction. *Physical Review A*, 87(6):062327, 2013.

[28] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[29] Kerry McKay et al. Users guide to running the draft NIST SP 800-90B entropy estimation suite. *NIST, Gaithersburg, MD, USA, Tech. Rep. SP*, 2016.

[30] Darren Hurley-Smith and Julio Hernandez-Castro. Certifiably biased: An in-depth analysis of a common criteria EAL4+ certified TRNG. *IEEE Transactions on Information Forensics and Security*, 13(4):1031–1041, 2017.

[31] Lara Ortiz-Martin, Pablo Picazo-Sanchez, Pedro Peris-Lopez, and Juan Tapiador. Heartbeats do not make good pseudo-random number generators: An analysis of the randomness of inter-pulse intervals. *Entropy*, 20(2):94, 2018.

[32] Thomas Shrimpton and R Seth Terashima. A provable-security analysis of Intel's secure key RNG. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 77–100. Springer, 2015.

[33] Paul Burciu and Emil Simion. A systematic approach of NIST statistical tests dependencies. *Journal of Electrical Engineering, Electronics, Control and Computer Science*, 5(1):1–6, 2019.

[34] Kenji Hamano and Toshinobu Kaneko. Correction of overlapping template matching test included in NIST randomness test suite. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 90(9):1788–1792, 2007.

[35] Katarzyna Anna Kowalska, Davide Fogliano, and Jose Garcia Coello. On the revision of NIST 800-22 test suites. *Cryptology ePrint Archive*, 2022.

[36] Kinga Marton and Alin Suciu. On the interpretation of results from the NIST statistical test suite. *Science and Technology*, 18(1):18–32, 2015.

[37] Markku-Juhani O Saarinen. NIST SP 800-22 and GM/T 0005-2012 tests: Clearly obsolete, possibly harmful.

[38] Marek Sỳs, Lubomír Obrátil, Vashek Matyáš, and Dušan Klinec. A bad day to die hard: Correcting the Dieharder battery. *Journal of Cryptology*, 35(1):1–20, 2022.

[39] Rafał Stępień and Janusz Walczak. Statistical analysis of the LFSR generators in the NIST STS test suite. *Computer applications in electrical engineering*, 11, 2013.

[40] Darryl Buller, Aaron Kaufer, Allen Roginsky, and Meltem Sönmez Turan. Discussion on the full entropy assumption of the SP 800-90 series. Technical report, National Institute of Standards and Technology, 2022.

[41] Shuangyi Zhu, Yuan Ma, Tianyu Chen, Jingqiang Lin, and Jiwu Jing. Analysis and improvement of entropy estimators in NIST SP 800-90B for non-IID entropy sources. *IACR Transactions on Symmetric Cryptology*, pages 151–168, 2017.

[42] John Von Neumann. Various techniques used in connection with random digits. *John von Neumann, Collected Works*, 5:768–770, 1963.

[43] Marshall Ball, Oded Goldreich, and Tal Malkin. Randomness extraction from somewhat dependent sources. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[44] Rotem Arnon-Friedman, Christopher Portmann, and Volkher B Scholz. Quantum-proof multi-source randomness extractors in the Markov model. *arXiv preprint arXiv:1510.06743*, 2015.

[45] John Kelsey, Luís TAN Brandão, Rene Peralta, and Harold Booth. A reference for randomness beacons: Format and protocol version 2. Technical report, National Institute of Standards and Technology, 2019.

[46] Cameron Foreman, Sherilyn Wright, Alec Edgington, Mario Berta, and Florian J Curchod. Practical randomness amplification and privatisation with implementations on quantum computers. *Quantum*, 7:969, 2023.

[47] D Frauchiger, R Renner, and M Troyer. True randomness from realistic quantum devices (2013). *URL http://arxiv. org/abs/1311.4547*.

[48] Yevgeniy Dodis, Ariel Elbaz, Roberto Oliveira, and Ran Raz. Improved randomness extraction from two independent sources. In *Approximation, randomization, and combinatorial optimization. Algorithms and techniques*, pages 334–344. Springer, 2004.

[49] Hugo Krawczyk. LFSR-based hashing and authentication. In *Annual International Cryptology Conference*, pages 129–139. Springer, 1994.

[50] Luca Trevisan. Construction of extractors using pseudo-random generators. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 141–148, 1999.

[51] Salil P Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.

[52] Nicolas Brunner, Daniel Cavalcanti, Stefano Pironio, Valerio Scarani, and Stephanie Wehner. Bell nonlocality. *Reviews of modern physics*, 86(2):419, 2014.

[53] Antonio Acín and Lluis Masanes. Certified randomness in quantum physics. *Nature*, 540(7632):213–219, 2016.

[54] Quantinuum. H1-1. https://www.quantinuum.com/, 1-4 Nov, 2021.

[55] Michael Mascagni and Ashok Srinivasan. Algorithm 806: SPRNG: A scalable library for pseudorandom number generation. *ACM Transactions on Mathematical Software (TOMS)*, 26(3):436–461, 2000.

[56] Helen Gustafson, Ed Dawson, Lauren Nielsen, and William Caelli. A computer package for measuring the strength of encryption algorithms. *Computers & Security*, 13(8):687–697, 1994.

[57] Julio Hernandez-Castro Jamie Pont, Calvin Brierley. BitReps. https://github.com/jjp31/bitreps-1/tree/master.

[58] Cristiano Piras. RaBiGeTe—Random Bit Generators Tester. http://cristianopi.altervista.org/RaBiGeTe_MT/.

[59] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A McKay, Mary L Baish, Mike Boyle, et al. Recommendation for the entropy sources used for random bit generation. *NIST Special Publication*, 800(90B):102, 2018.

[60] Debarshi Datta, Bipa Datta, and Himadri Sekhar Dutta. Design and implementation of multibit LFSR on FPGA to generate pseudorandom sequence number. In *2017 Devices for Integrated Circuit (DevIC)*, pages 346–349. IEEE, 2017.

[61] M. Sahithi, B. MuraliKrishna, M. Jyothi, K. Purnima, A. Jhansi Rani, and N. Sudha. Implementation of random number generator using LFSR for high secured multi purpose applications. *International Journal of Computer Science and Information Technologies*, 3(1):3287–3290, 2012.

[62] Patrik Ekdahl. *On LFSR based Stream Ciphers-analysis and design*. Lund University, 2003.

[63] Amit Kumar Panda, Praveena Rajput, and Bhawna Shukla. FPGA implementation of 8, 16 and 32 bit LFSR with maximum length feedback polynomial using VHDL. In *2012 International Conference on Communication Systems and Network Technologies*, pages 769–773. IEEE, 2012.

[64] N David Mermin. Extreme quantum entanglement in a superposition of macroscopically distinct states. *Physical Review Letters*, 65(15):1838, 1990.

[65] Erik Woodhead, Boris Bourdoncle, and Antonio Acín. Randomness versus nonlocality in the Mermin-Bell experiment with three parties. *Quantum*, 2:82, 2018.

## Appendix A: RNG Descriptions

### 1. Linear Feedback Shift Register (LFSR)

The LFSR is a class of pseudo RNG that is commonly used in applications, due to its speed and ease of implementation in both software and hardware, e.g. [60] [61]. Notably, LFSRs are used in cryptography, including in hashing and authentication [49] and stream ciphers [62]. For this work, we implement the maximal period LFSR found in [63].

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. Initialize 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 step (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).

### 2. Intel RDSEED

Intel manufacture 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].

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, which are very simple statistical tests, with the goal of detecting critical failure in the entropy source.
3. Processing of randomness through a cryptographic hash function.

A user then calls randomness from the true-RNG by using RDSEED. An in-depth description of all the above steps can be found in [11] and [10].

Using the Intel true-RNG, we are unable to output raw randomness from the entropy source. The best we can do is use RDSEED. In this case, some post-processing has already been performed (as described above). Some independent analysis of the quality of the Intel true-RNG has been done, including [10] and [32], where the former find that the min-entropy rate of RDSEED is around 0.65, which is similar to our result (see Section IV).

### 3. IDQ Quantis QRNG

The IDQ Quantis (USB) is 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 perfect random numbers due to the laws of quantum mechanics. We revert the reader to the IDQ Quantis QRNG brochure for further details of its construction [22].

## Appendix B: Initial RNG Analysis

### 1. Full results: Statistical testing

| 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 IX: The number of failed tests for the raw output from the 32-bit LFSR. Note, 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 parenthesis.

| 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 X: 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 parenthesis.

| 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) |
| 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)** |

TABLE XI: 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 parenthesis.

### 2. Full results: Min entropy estimators

| 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 XII: Observed NIST min-entropy estimators for 32-bit LFSR 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 XIII: Observed NIST min-entropy estimators for RDSEED 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 |
| 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** |

TABLE XIV: Observed NIST min-entropy estimators for IDQ Quantis raw output.

#### a. *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 IV. We use subscripts to index a single min-entropy estimator test and superscript to index the RNG which the variable refers to. Let $\mathsf{est}_i$ denote the $i$th 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 $\sigma$, (using Bessel's correction), given by

$$\sigma^{\mathsf{RNG}} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\overline{\mathsf{est}}^{\mathsf{RNG}} - \mathsf{est}_{\mathsf{i}}^{\mathsf{RNG}})^2}. \tag{B1}$$

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

$$\Pr(\mathsf{est}_{\mathsf{i}}{}^{\mathsf{RNG}} < \alpha^{\mathsf{RNG}}) = \epsilon_{\mathsf{est}} < 2^{-32}, \tag{B2}$$

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

$$\alpha^{\mathsf{RNG}} = \overline{\mathsf{est}}^{\mathsf{RNG}} - 7\sigma^{\mathsf{RNG}} \tag{B3}$$

where $\overline{\mathsf{est}}^{\mathsf{RNG}}$ is the average NIST min-entropy estimator for the RNG and $\sigma^{\mathsf{RNG}}$ is the observed sample standard deviation satisfies Equation (B2), 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

### 1. Full results

| 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) |
| 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 XV: The number of failed tests for the output of the 32-bit LFSR after post-processing with the Von Neumann extractor. Note, 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 parenthesis.

| 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 XVI: 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 parenthesis.

| 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)** |

TABLE XVII: 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 parenthesis.

## Appendix D: Seeded Extraction in Detail

### 1. Full results

| 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 XVIII: 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 parenthesis.

| 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 XIX: 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 parenthesis.

| 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)** |

TABLE XX: 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 parenthesis.

## Appendix E: Two-source Extraction in Detail

### 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 2-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 IV is too low.

| 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 XXI: This table gives the sum of tests failed for $5 \times 10$Gbit 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 it's 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 parenthesis.

| 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 XXII: 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 parenthesis.

| 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)** |

TABLE XXIII: 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 parenthesis.

### 2. Two-source extraction using the NIST randomness beacon

| 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 XXIV: 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 parenthesis.

| 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) |
| 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 XXV: 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 parenthesis.

| 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)** |

TABLE XXVI: 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 parenthesis.

## Appendix F: Physical Randomness Extraction in Detail

### 1. Protocol for physical randomness extraction

For this physical randomness extraction, we roughly follow the protocol developed in [46], with some adaptions to improve the randomness generation speed. 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 [52].

The adapted protocol 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, that select 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 [46].

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 that round.

3. After $n$ rounds, calculate the observed probability distribution $\Pr(a, b, c|x, y)$.

4. Evaluate the Mermin inequality [64] value $\mathsf{M}_{\mathsf{obs}}$, where

$$\mathsf{M}_{\mathsf{obs}} = E_{0,0,0} - E_{0,1,1} - E_{1,0,1} - E_{1,1,0} \tag{F1}$$

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

$$E_{x,y,x\oplus y} = \sum_{a\oplus b\oplus c=0} \Pr(a, b, c \mid x, y, x \oplus y) - \sum_{a\oplus b\oplus c=1} \Pr(a, b, c \mid x, y, x \oplus y) \tag{F2}$$

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 [46]. Let $\epsilon_{\text{est}}$ be the estimation error

and $\mathsf{M}$ 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

$$\mathsf{M}_{\mathsf{adj}} = \mathsf{M}_{\mathsf{obs}} - 16t. \tag{F3}$$

and

$$\epsilon_{\text{est}} = \exp(-2t/n) \tag{F4}$$

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 [65], which applies to 2 output bits per round. For details, see Section 4.3 *Quantum devices, Bell tests, and guessing probabilities* [46] and note that, the relationship between guessing probability and min-entropy can be found in Appendix A.3, Equation (28) of [46].

7. Take 2 of the 3 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 min-entropy's 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: two-source extraction.

For the extractor implementation, we again use the Circulant extractor, and steps (8) and (9) can be viewed as analagous 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 $1.8 \times 10^6$ circuits to obtain a weakly random seed size of $3.6 \times 10^6$ bits. This experiment took approximately 33.5 hours 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 $\alpha_{\mathsf{Q}}$ is the min-entropy rate of the quantum computer outputs $(\mathbf{a}, \mathbf{b})$. Note: The min-entropy of the LSFR was too low to perform physical extraction, as we need $\alpha_{\mathsf{RNG}} > 1 - \alpha_{\mathsf{Q}}$.

### 2. Full results

| 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) |
| 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 XXVII: 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. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parenthesis.

| 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)** |

TABLE XXVIII: 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. In cells with multiple entries, failed tests are on the left and suspicious tests (when applicable) are on the right in parenthesis.