Joshua H. Sylvester*, Micah A. Thornton, Jessie M. Henderson, Mitchell A. Thornton and Eric C. Larson

GTR: Granger-inspired test for randomness in bitstreams

https://doi.org/10.1515/itit-2025-0027
Received August 29, 2025; accepted November 23, 2025;
published online February 4, 2026

Abstract: Assessing the quality of random bitstreams used to support cryptographic and other applications is crucially important as any detectable deviations from true randomness can introduce exploitable vulnerabilities resulting in a loss of security. Existing randomness tests, such as those recommended by the U.S. National Institute of Standards and Technology (NIST), examine statistical characteristics of bitstreams such as bit distributions, periodicity, cumulative sums and other properties. However, these methods do not explicitly test for generalized causality within a bitstream – instances where earlier sequences influence the likelihood of later sequences that are undetectable through correlation-based analyses. To address this gap, we propose a new approach, the Granger-inspired Test for Randomness (GTR), that applies principles of Granger causality to detect causal relationships within a single bitstream. To validate our approach, we conduct experiments using a 10-million-bit sample acquired from the NIST Randomness Beacon as a baseline case. We compare GTR to other methods that assess random bitstream quality. Our findings suggest that GTR outperforms many randomness tests, identifying subtle structural dependencies in bitstreams that are not detected with many current tests.

Keywords: Granger causality; Random Bit Generator (RBG); test

1 Introduction

Many important applications in cryptography require the use of a random bitstream as an input. A bitstream is considered to be “random” by the cryptographic community when it is a sequence of bits such that each bit is equally likely to be zero- or one-valued and where each bit’s value is independent of the values of other bits in the sequence. “Random number generator attacks” are the class of methods where an adversary exploits or subverts weaknesses in cryptographic protocols that depend upon high-quality random bitstreams as input. Therefore, identifying whether a bitstream has some degree of determinism is an important and long-standing problem. Such determinism may manifest as non-uniformly distributed values or as dependencies between bit values in the sequence, as opposed to the bitstream being indistinguishable from an observation of a truly random process. Any form of predictability in these bitstreams can result in exploitable vulnerabilities that enable an adversary to defeat the intended confidentiality provided by cryptographic methods including key and nonce[1] generators, initialization vectors, temporary password generators and others [1]–[4].

Examples of adversarial attacks due to insecure sources of randomness in cryptographic applications include [5]–[9]. A detailed exposé regarding the controversy surrounding the discovered vulnerability in the previously NIST-standardized Dual Elliptic Curve Deterministic Random Bit Generator (Dual_EC_DRBG) as described in [10] was published in [11] and resulted in NIST making the decision to remove Dual_EC_DRBG from its standards for random bit generators [12]. Another example of a potential vulnerability is reported in [13] concerning CTR-DRBG, which is a pseudorandom number generator recommended by NIST. Potential vulnerabilities in both the specification as well as in common implementations of CTR-DRBG are discussed in [14]. Thus, cryptographic protocols relying upon the standardized CTR-DRBG could

***Corresponding author: Joshua H. Sylvester**, Southern Methodist University, Darwin Deason Institute for Cybersecurity, Dallas, TX, USA, E-mail: jsylvester@smu.edu
Micah A. Thornton, Department of Mathematics, Texas Woman’s University, Denton, TX, USA, E-mail: mthornton11@twu.edu
Jessie M. Henderson, Mitchell A. Thornton and Eric C. Larson, Southern Methodist University, Darwin Deason Institute for Cybersecurity, Dallas, TX, USA, E-mail: hendersonj@smu.edu (J. M. Henderson), mitch@smu.edu (M. A. Thornton), eclarson@smu.edu (E. C. Larson)

1 A “nonce” is a random or pseudorandom value that is only used once in cryptographic applications to prevent replay attacks. The word originates from a shortened form of the phrase “number once”.

be vulnerable to attack simply because the generated entropy values are insecure.

The recently adopted Post Quantum Cryptographic (PQC) algorithms by NIST, documented in the FIPS 203/204/205 standards [15], rely on high-quality entropy sources with higher output data rates in terms of bits per second (bps) than can be provided by currently implemented Random Bit Generators (RBGs). Migrating from currently implemented cryptographic algorithms to the new PQC standards requires the replacement of many currently deployed RBGs, as they are unable to provide the higher output data rates required. Many of these replacement RBGs will be newly designed and implemented to accommodate these more demanding specifications. Before deploying a new RBG, it should be thoroughly evaluated to ensure that the cryptographic applications it supports are not vulnerable to random number generator attacks.

When evaluating the randomness of a bitstream, a fundamental objective is to ensure that no information can be extracted that causes the probability of predicting the next bit in the sequence to deviate from a value of one-half. There is no known test that can directly prove that a given bitstream is random. Consequently, bitstream tests are employed that evaluate various secondary properties that should hold if the bitstream is indeed random. These tests check for properties such as the presence of appropriate ratios of bit values, lack of any periodic structure, proper sums of bit values, and other characteristics that should be present in truly random bitstreams. For example, the "Runs Test" within the NIST Statistical Test Suite (STS) [16] is based on characteristics of substrings that contain bits of the same value, either all zero or all one (*i.e.,* 'runs'). In a truly random bitstring, the sequence of values representing the lengths of the 'runs' should be of the form of a random variable that is binomially distributed. Thus, a statistical inference test is employed to check the hypothesis that run lengths are binomially distributed and the resulting p-value must fall below a threshold value for the test to be successful.

STS is a collection of 15 tests that compute certain statistical properties for a given bitstream that indicate whether it is distinguishable from an observation of a random process or not. However, none of the STS tests directly consider causality properties that may be present among subsequences, or "windows" of bits. That is, the presence of a subsequence of bits that influences the probability of observing a later subsequence of bits through an underlying "cause and effect" phenomenon that is not otherwise identifiable through correlation-based tests. This kind of influence can be complex to detect but still indicates a serious lack of randomness in a bitstream that could induce a vulnerability in a cryptographic method that depends upon the bitstream's randomness. Indeed, just as there is no direct test to ascertain if a bitstream is random, there is likewise no test to detect the presence of general "cause and effect" relationships within a bitstream. However, as we describe in this work, a restricted form of causality inspired by the so-called "Granger causality" of time series data [17], can be detected within a bitstream when it is present.

To this end, we develop a Granger causality-inspired methodology to detect this limited form of causal influence among bit subsequences within a stream when it is present. We refer to this test as the Granger-inspired test for randomness, or GTR. An initial version of GTR was reported as a presentation only in [18], but that prior version of GTR did not implement all of the capabilities that are described here. The enhanced version of GTR described here adapts the underlying principles of Granger causality and applies them in a context that is much more appropriate for binary-valued bitstreams in contrast to the real-valued time series ensembles considered in 1969 by Clive Granger, the original inventor. While our approach requires some significant modifications to the original implementation of Granger causality, it retains the core concepts of the method, aiming to determine whether predictability is present (or not) within a bitstream. To create the GTR method, we focus on two primary contributions:

1. We modify the traditional vector autoregression (VAR) framework by replacing ordinary least squares (OLS) with logistic regression, which is more appropriate for the binary values comprising an RBG-generated bitstream since logistic regression provides values bounded in the interval [0,1] that are valid probability values.
2. We develop a method for testing Granger causality within a single bitstream rather than among two or more time series sequences of real-valued data.

Granger causality [17] is traditionally applied to multiple instances of time series to assess the presence of predictability and time-lagged relationships. In the case of bitstreams, however, there is a single sequence of data rather than multiple time series sequences. Nevertheless, a bitstream can be segmented into subsequences, and in an ideal random sequence, knowledge of one subsequence should provide no advantage in predicting future subsequences. This observation motivates our development of the GTR to detect the presence or absence of predictability within a bitstream.

We begin with a fundamental question: Can we determine whether one sequence of bits "causes" another sequence to appear at a later point? Consider the example

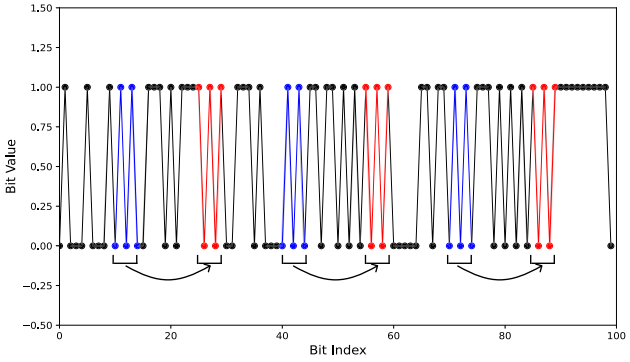


Figure 1: An example bitstream illustrating a repeated blue sequence consistently preceding a red sequence, suggesting a causal relationship.

in Figure 1, where a particular bit sequence (blue) consistently precedes another sequence (red). The goal of GTR is to detect such relationships by assessing whether knowledge of past bit sequences provides an advantage when predicting future bits.

It is recognized that the simple example of the blue causal bits and the red effect bits in Figure 1 could be readily detected by simple correlation-based tests since the red subsequences are all of the same value and, likewise, the blue subsequences are all of the same value. However, the GTR methodology does not necessarily depend on identical repetition for detection. In fact, because of its Granger-inspired design, the GTR should be able to detect limited forms of causality for the more general case where certain subsequences are *predictive* of other subsequences without each instance of this predictability being identical. In such scenarios, the GTR may generalize better than correlation-based tests. Nevertheless, while GTR is formulated to detect the presence of this more general form of predictability, we initially evaluate it using bitstreams with repeated subsequences as described in Figure 1 and leave additional investigations of predictability detection for future work.

2 Background concepts

A variety of different artifacts have been devised for the purpose of generating digit sequences that embody randomness properties. RBGs produce sequences of binary digits, or 'bits.' When higher-radix digit streams are produced, the artifact takes the form of the more general Random Number Generator (RNG). RBG and RNG may be implemented purely in the form of algorithms specified in a particular programming language, a dedicated hardware device, or as a combination of software and hardware. Although RBG and RNG have the word 'random' in common, it is extremely difficult for any human-designed artifact to generate truly random sequences; hence, the need to assess their quality is important and is the motivation for development of test suites. As previously discussed, it is impossible to prove that a given sequence is truly random. This fact is easy to understand since a sequence would have to be infinite in length to be deemed random since any finite subsequence could simply be repeated indicating a periodic component is present. Clearly all human-produced sequences are necessarily finite in length. However, it is possible to determine how closely properties of a finite sequence resemble those of a theoretical stream of infinite length. Therefore, various tests have been implemented in software that compute these properties and collections of these tests are bundled together to form test suites. The goal of a test suite is to test as many different properties as possible for the purpose of evaluating the quality of RBG and RNG.

2.1 Random number and bit generation

RNG are characterized by producing a sequence of non-binary digits whereas RBG produce a series of single bits and both RNG/RBG fall into two main categories: (a) "Pseudo-Random Number (or Bit) Generators" (PRNG/PRBG) and "True Random Number (or Bit) Generators" (TRNG/TRBG).

PRNG/PRBG are actually deterministic but they produce sequences of values that have sample statistics that closely match those from a theoretical purely random sequence. Several forms of PRNG/PRBG produce sequences that repeat and thus are more properly described as implementations of a periodic function from a mathematical point of view. However, when sequences are obtained within a single period, they have test statistics that closely match those of a purely random sequence. For this reason, a design goal for PRNG/PRBG is to ensure that the period is of maximal length. For cryptographic applications, PRNG/PRBG are usually undesirable choices since confidentiality is lost if the adversary can determine the architecture or method used as a basis for their implementation. However, there are cases where PRNG/PRBG are desirable. For example, many cell phone protocols use scrambling codes wherein it is desirable for the codes to appear random but where it also required that the cell phone and the tower must generate identical sequences to de-scramble the communications. Another example is the use of PRNG/PRBG in Monte Carlo software simulations. While Monte Carlo-based simulations are based on the use of random values, in this case pseudo-random values, it is desirable to have the ability to recreate simulation results in some cases. When PRNG/PRBG are employed in Monte Carlo simulations, results can be recreated by re-executing

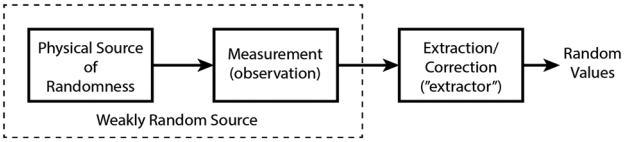


Figure 2: Block diagram of TRNG/TRBG.

the simulation using the same initialization seed for the PRNG/PRBG.

TRNG/TRBG are typically implemented as shown in the block diagram of Figure 2. The portion in the dashed line is the "weakly random source" and consists of a source of natural randomness with a measurement device. The reason why it is referred to as "weakly" random is due to the fact that the measurement device will always impart some amount of bias and often the probability mass function of the physical source of randomness is not perfectly uniform and may obey some other probability mass (or distribution) function. For these reasons, the "extractor" block is included in a TRNG/TRBG for the purpose of removing any bias or other deterministic behavior from the weakly random source and to also serve as a random variable transformation if the random nature of the physical source does not inherently adhere to a uniform probability distribution. Another block that is commonly included in TRNG/TRBG that is not shown in Figure 2 is a "health monitor" that checks the extractor output to ensure random values are continuing to be produced. Health monitors are important since the physical sources of entropy can vary over time or with changing environmental conditions. In fact, some of the statistical tests that are the subject of this paper can be employed as health monitors.

The physical source of randomness can include any of a variety of natural phenomena. Examples include atmospheric noise, electrical circuit thermal noise, network packet characteristics such as inter-packet arrival times, mechanical disk drive seek times, keyboard user typing patterns and various phenomena related to quantum mechanics (QM). In fact, many cryptographic experts promote the use of QM phenomena as physical sources since inherent randomness is a basic axiom of the theory of QM. Examples of QM-based physical sources include radioactive and nuclear decay times, photonic photodetector dark counts, tunneling behavior, and measurements or observables of coherent particles. Examples of TRNG and TRBG based on QM noise sources that are implemented as dedicated hardware devices are described in [19], [20]. Because Discrete Variable (DV), or gate-based, quantum computers utilize a measurement operator applied to a quantum state or wavefunction, it is possible to implement a TRNG in the form of software executing on a quantum computer. Previously, software-based RNG/RBG implied that a such RNG/RBG were necessarily PRNG/PRBG since they were implemented on deterministic classical electronic computers. An example TRNG implemented on a DV quantum computer is described in [21] that also has the advantage of supporting any arbitrary probability mass function rather than the typical uniform distribution. More recently, a programmable TRNG was devised and implemented on a Continuous Variable (CV) quantum computer, the Xanadu X8 photonic machine, based on the use of the boson sampling algorithm [22].

2.2 Random bit generation test suites

In addition to the previously mentioned STS test suite from NIST, a variety of other different tests have been formulated for the purpose of assessing sequences of digits and bits for properties consistent with truly random streams. Some of the more popular test suites include Diehard [23], Die*Harder* [24], AIS-31 [25], [26], ENT [27], TufTests [28] and NIST STS [29]. We summarize a few popular suites before delving into the NIST and STEER frameworks, which are employed in this paper.

2.2.1 Popular test suites

Some of the more popular test suites are summarized here to motivate the need for the addition of new tests such as the Granger-inspired causality test described in this paper.

2.2.1.1 Diehard test suite

Another set of RBG tests is named Diehard [23] that are available for download as an executable under the Microsoft DOS operating system for PCs with Intel 386/486 processors at [30]. These tests were originally written in the FORTRAN programming language by G. Marsaglia and were later automatically translated to the *C* programming language. At the time of this writing, the original source code for the Diehard test suite was not available for download.

2.2.1.2 Die*Harder* test suite

The Diehard test suite was completely re-implemented and extended to contain additional tests as an open source project named Die*Harder* by R. Brown and is available at [31] under a Gnu Public License (GPL). Additionally, a github download for Dieharder is available at [32], maintained by R. Urban. The Die*Harder* archives contain a manual that describes the content of the test suite [24]. An interface to the *R* statistical language for Dieharder is described in [33].

2.2.1.3 AIS-31 test suite

The AIS-31 test suite [25], [26] was developed in Germany and is available for download at [34]. AIS-31 is the counterpart to the U.S. NIST STS and RNG/RBG intended to support cryptographic applications in the European Union are required to satisfy this test suite.

2.2.1.4 ENT test suite

ENT is a test suite that contains several tests implemented in the *C* programming language produced by J. Walker and is open source under a Creative Commons Attribution-ShareAlike license and is available for download at [35]. More information regarding ENT, including the implemented tests and its usage, is available at [27].

2.2.1.5 Tuftests test suite

The author of the Diehard test suite, G. Marsaglia and his colleague, W.W. Tsang produced a test suite comprised of three tests that they describe as tests that are more likely to be failed than others in Diehard and other test suites [28]. These are named the "gcd," "gorilla," and "birthday spacings" test. Many RNG/RBG designers focus on passing this reduced set of tests first before evaluating their products with other test sets. The Tuftest suite is available for download at [36].

2.2.2 NIST statistical test suite

The NIST STS test suite is widely used and contains a set of fifteen tests that are summarized in Table 1. The tests are written in the *C* programming language and are available for download at [29]. The individual tests comprising the STS were written by different developers and care must be taken to ensure that they are executed with appropriate input parameters to ensure that test results are valid. It is recommended that users of STS have some background knowledge in statistics – particularly in the use of statistical inference tests since many of the individual tests are centered around concepts in hypothesis testing principles.

2.2.3 STEER test suite framework

In an attempt to offer the community an easily useable and extensible framework for RBG statistical test suites, Anametric, Inc. [37] and the Darwin Deason Institute for Cyber Security [38] at Southern Methodist University developed, maintains, and hosts the "STandard Entropy Evaluation Report" (STEER) framework [39] for RBG test suite integration. The STEER framework is an open source project under a Gnu Affero General Public License (AGPL) that currently contains the NIST STS version 2.1.2 set of tests. It is emphasized that the NIST STS test suite source files contained within STEER are not modified or changed in any way. However, some of the third-party mathematical functions used by some of the NIST STS tests were found to no longer be supported or maintained and, for these tests, functions from the GNU Scientific Library (GSL) were used as replacements [40].

The main goal of the "STandard Entropy Evaluation Report" (STEER) framework is to improve the accessibility, usability and scalability of the test suites for RBG while maintaining and extending its primacy in evaluating entropy sources. The STEER source code has been built and executed on several of the most common linux-based operating systems and is available for download at [39].

Table 1: Summary of NIST STS tests [16].

NIST STS tests	Test summary
2.1 Frequency (Monobit) test	Proportion of 0s and 1s
2.2 Frequency test within a block	Proportion of 1's within M-length blocks
2.3 Runs test	Total number of runs
2.4 Test for the longest run of ones in a block	Longest run of 1's within M-bit blocks
2.5 Binary matrix rank test	Check linear dependence among substrings
2.6 Discrete Fourier transform (spectral) test	Test periodic features
2.7 Non-overlapping template matching test	Check for occurrences of given aperiodic patterns
2.8 Overlapping template matching test	Use m-bit window to search for m-bit patterns
2.9 Maurer's "Universal Statistical" test	Test number of bits between matched patterns
2.10 Linear complexity test	Find length of a characterizing LFSR (complexity/randomness)
2.11 Serial test	Test number of occurrences of 2^m m-bit patterns
2.12 Approximate entropy test	Frequency of all possible overlapping m-bit patterns across entire sequence
2.13 Cumulative sums (Cusum) test	Test cumulative sums of partial sequences for expected value of random walks
2.14 Random excursions test	Test number of cycles having exactly K visits in cumulative sum random walk
2.15 Random excursions variant test	Determine total number of times a particular state is visited in cumulative sum random walk

Issues addressed in the STEER framework include scaling issues related to serialization of test execution, a requirement for interactive console input by the user, a lack of automatic build validation, a lack of standards for specifying test parameters and other input information, non-standardized output reports for individual tests, insufficient feedback resulting individual test executions, and difficulties in adding new tests to a particular test suite. These issues can cause users of test suites to easily misinterpret test results and to falsely declare a particular set of input test data as having passed (or failed) a test. For example, many of the tests in the NIST STS suite require input bitstreams comprising millions of bits to in order to produce statistically valid test results. Some tests provide no input parameter checking and can declare bitstreams to be pass a particular test when, in reality, the bitstring should be orders of magnitude larger before the results of the test become valid.

STEER addresses these issues by establishing common test data formats and programming interfaces to simplify test suite usage and the addition of new tests by developers. This is accomplished by establishing a JSON schema for test information, parameters and reports. Automation and scalability are addressed through the allowance of scripts to reduce tedious console input by the user for each test and the incorporation of multi-threading support to permit the tests to all run concurrently on different CPU cores rather than in a serial fashion. Additionally, STEER incorporates automated source code quality checks for new tests and validates user input data to ensure that test results are statistically sound and provide meaningful results. Output reports for test runs are consolidated into a common format and a test report viewer is included that allows users to easily browse test reports.

2.3 Principles of Granger causality

A notable omission in commonly-used test suites are methods that attempt to discern if a sequence contains generalized causal properties. This is likely due to the fact that no known test can determine the presence of a subsequence that causally influences the generation of a subsequent subsequence. However, restricted forms of causality have been formulated. One of these restricted forms is known as Granger causality [17]. This is the framework on which our GTR is built. In general terms, the original formulation of Granger causality uses statistical hypothesis testing to determine whether one time series is useful in forecasting another. Furthermore, Granger causality assumes that each time series under consideration has a linear relationship with time-lagged values and assumes that additive noise present.

Specifically, assuming two time series x and y, one may wish to know whether past values of x could be useful in predicting future values of y. A common way to test for this Granger causality is to fit two Vector Autoregressive Models (VARs). The first model is fit **only** to past values of y and is denoted the restricted model. The second model, denoted the unrestricted model, is fit to past values of **both** x and y. The restricted and unrestricted models are shown respectively in eqs. (1) and (2), where p represents the lag order. The error terms of the two models are represented by ϵ_t and η_t respectively.

$$y_t = b_0 + \sum_{i=1}^{p} b_i y_{t-i} + \epsilon_t, \tag{1}$$

$$y_t = b_0 + \sum_{i=1}^{p} b_i y_{t-i} + \sum_{j=1}^{p} a_j x_{t-j} + \eta_t, \tag{2}$$

These error terms can the be compared using a statistical test where the null hypothesis is that there is no difference between the error of the two models. The alternative hypothesis is that the variance of the error of the unrestricted model is significantly less than that of restricted. The null and alternative hypotheses are formally stated in eqs. (3) and (4). If the null hypothesis is rejected, this provides evidence in support of the alternative hypothesis and it is said that x Granger-causes y. One common way to perform this test is by comparing the fit of the two models via a log-likelihood ratio test, which we employ throughout this work.

$$H_0: \mathrm{Var}(\eta) = \mathrm{Var}(\epsilon) \tag{3}$$

$$H_1: \mathrm{Var}(\eta) < \mathrm{Var}(\epsilon) \tag{4}$$

3 Methodology

3.1 Logistic regression based Granger causality test

Ordinary least squares (OLS) provides a simple and widely used approach for fitting models in Granger causality testing. However, in contexts where the response variable is binary, logistic regression is a more appropriate modeling framework. Unlike OLS, which can produce predictions outside the [0,1] range, logistic regression models the probability of an event occurring, ensuring outputs remain valid probabilities. These probabilities can then be thresholded to make binary decisions, aligning naturally with the structure of the response variable. Substituting logistic regression for

OLS modifies the simplified regression formulation in eq. (5) to a logistic regression form in eq. (6).

$$Y = \alpha + \beta X + \epsilon \quad \text{(Ordinary Least Squares)} \tag{5}$$

$$P(Y = 1 \mid X) = \sigma(\alpha + \beta X) \quad \text{(Logistic Regression)} \tag{6}$$

where the sigmoid function $\sigma(z)$ is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{7}$$

To assess Granger causality within this logistic regression framework, we employ a likelihood ratio test to compare the restricted and unrestricted models. The log-likelihood function for a logistic regression model parameterized by $\boldsymbol{\beta}$ is given by eq. (8).

$$\ell(\boldsymbol{\beta}|\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} y_i \log p_i + (1 - y_i) \log(1 - p_i), \quad \text{where} \quad p_i = \sigma\left(\mathbf{x}_i^\top \boldsymbol{\beta}\right) \tag{8}$$

By framing Granger causality within logistic regression, we ensure that predicted probabilities remain valid and interpretable for binary bitstream testing, while maintaining consistency with the core hypothesis-testing framework of traditional Granger causality tests.

3.2 Reformulating Granger causality testing for bitstreams

The structure of our Granger-inspired test follows the same fundamental approach as a traditional Granger causality test: we compare a restricted model and an unrestricted model to determine whether the unrestricted model demonstrates improved predictability. The key distinction in our methodology lies in how we construct the datasets for fitting these models. In classical Granger causality testing, the restricted model is fit to lag matrix containing p lags of a single time series y, while the unrestricted model incorporates p lags of both x and y. To adapt this methodology to a bitstream context, we construct the unrestricted and restricted lag matrices based on data from separate regions of the bitstream.

Following the standard Granger causality framework, we define two models:

- **Restricted model (R):** Predicts the next bit in the sequences using a window of bits immediately preceding the target bit.
- **Unrestricted model (U):** Predicts the next bit in the sequence using two windows: a window of the most recent bits **as well as** a window of historical bits which occurred earlier in the sequence.

$$H_0 : \ell_{\mathrm{R}} = \ell_{\mathrm{U}} \tag{9}$$

$$H_A : \ell_{\mathrm{R}} < \ell_{\mathrm{U}} \tag{10}$$

$$\lambda_{\mathrm{GTR}} = -2 \times (\ell_{\mathrm{R}} - \ell_{\mathrm{U}}) \tag{11}$$

$$\lambda_{\mathrm{GTR}} \sim \chi^2_w \tag{12}$$

If incorporating historical bits improves the predictive performance of the unrestricted model relative to the restricted model, we conclude that prior bit sequences influence future bit sequences. We employ a log-likelihood ratio test as defined in eqs. (9)–(12) where ℓ represents the log-likelihood of the respective model. The null hypothesis states that historical bits add no predictive value beyond recent bits, while the alternative hypothesis posits that including historical bits improves model fit. The test statistic λ_{GTR} follows a chi-squared distribution with w degrees of freedom, where w is the window size (defined below). Practically, the degrees of freedom, representing the difference in the number of parameters between R and U, will always equal the window size (w). We evaluate the GTR using a 0.01 significance threshold to align with the convention used by the NIST-STS.

However, as will be discussed in Section 3.3, we apply a Null-LLR test to both the restricted and unrestricted models as a preliminary step to applying the GTR. Since this introduces multiple hypothesis tests on the same dataset, the probability of Type I errors increases. To control for this, we apply a Bonferroni correction to the significance threshold of the GTR, ensuring that our conclusions remain statistically robust. Given a significance level of 0.01 and the three tests (Null-LLR on the restricted, Null-LLR on the unrestricted, and the GTR), this corresponds to a Bonferroni-corrected significance level of $0.00\bar{3}$.

In conventional Granger causality applications, constructing the dataset involves generating a simple lagged matrix. However, in our case, the process is more nuanced. To structure the dataset, we define two key parameters: the **window size** (w) and the **offset** (o).

- **Window** (w): Specifies how many bits are used in each window for prediction. This terminology aligns with a sliding window methodology, where a fixed-length segment moves sequentially through the data. The restricted model relies solely on the most recent w bits, while the unrestricted model incorporates an additional historical window of w bits from earlier in the sequence.
- **Offset** (o): Determines how far back in the bitstream we select an additional window for the unrestricted model. This historical window is separated from the target bit

by $o \times w$ bits, where $o \geq 1$ ensures strict separation between input windows.

By structuring our test in this way, we retain the core principles of Granger causality while adapting them to evaluate the internal predictability of a single bitstream. Thus, we define the restricted and unrestricted logistic regression models for the GTR in eqs. (13) and (14), where $\mathbf{X}_{\text{Recent}}$ consists of the recent window of bits, i.e., the bits immediately preceding the next (target) bit, and $\mathbf{X}_{\text{Historical}}$ consists of bits from an earlier window in the sequence.

$$P(X_{\text{Next}} = 1 \mid \mathbf{X}_{\text{Rec.}}) = \underbrace{\sigma\left(\alpha + \sum_{j=1}^{w} \beta_j X_{\text{Rec.},j}\right)}_{\text{Restricted model}} \tag{13}$$

$$P(X_{\text{Next}} = 1 \mid \mathbf{X}_{\text{Rec.}}, \mathbf{X}_{\text{His.}}) = \underbrace{\sigma\left(\alpha + \sum_{j=1}^{w} \beta_j X_{\text{Rec.},j} + \sum_{j=1}^{w} \gamma_j X_{\text{His.},j}\right)}_{\text{Unrestricted model}} \tag{14}$$

Note that we use a sliding window to curate the $\mathbf{X}_{\text{Recent}}$ and $\mathbf{X}_{\text{Historical}}$ datasets, thus, convolutional accelerations with graphics processors can be utilized to improve computational efficiency.

3.2.1 Data curation example

Consider a bitstream of length 10,000 bits with $w = 100$ and $o = 6$. The restricted model uses the bits at positions 601–700 as input to predict the bit at position 701. In the unrestricted model, the offset parameter o explicitly defines how far back to retrieve additional input features for the unrestricted model relative to the target bit. That is, the historical window is separated from the target bit by a 600 bit gap, such that the unrestricted model uses both the bits at positions 601–700 (recent window) and the bits at positions 1–100 (historical window) to predict bit 701. The dataset is generated by shifting both windows forward by one bit at each step. This process continues until the target reaches the 10,000th bit.

Consider a smaller example (based on Figure 1) to visually demonstrate the intended application of the GTR. In this example, we use $w = 5$ and $o = 2$. Figure 3 illustrates how the unrestricted model's dataset is created:

- **The first window (blue)** indicates the historical bits used *only* by the unrestricted model.
- **The second window (red)** contains the recent bits used by *both* the restricted and unrestricted models.
- **The last window (orange)** represents the target (i.e., next) bit.

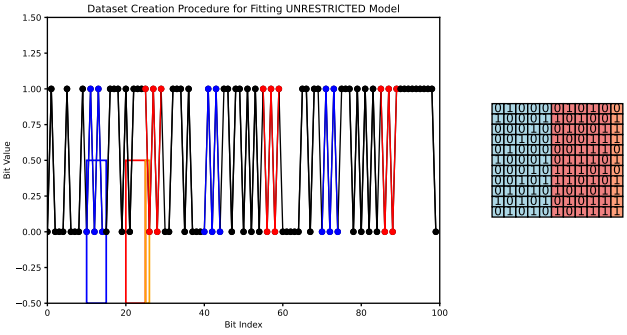


Figure 3: Sliding-window illustration for the unrestricted model dataset with repeated bit sequences. ($w = 5$, $o = 2$).

As shown on the right side of the figure, the dataset is populated by shifting these windows over the bitstream. This setup enables the unrestricted model to improve prediction by repeatedly encountering the blue bits when targeting the red sequences. Identical rows in the dataset, arising from different positions in the bitstream, allow the unrestricted model to capture patterns that the restricted model cannot. This detectability advantage motivates the use of GTR for bitstream testing.

3.3 Assessing model fit via a Null-LLR to ensure GTR validity

In traditional Granger causality testing, it is generally assumed that each time series exhibits some degree of autoregressive structure. This assumption ensures that both the restricted and unrestricted models fit their respective data reasonably well, allowing meaningful comparisons between them.

However, in the context of random bitstream testing, this assumption does not necessarily hold. If a bitstream is truly random, we would expect that no meaningful pattern exists within the data, making it difficult – if not impossible – for any model trained on subsequences of the bitstream to achieve a good fit. This presents a key challenge: for the GTR to yield meaningful conclusions, we must first establish that the models (restricted and unrestricted) achieve a sufficiently strong fit. If neither model fits the data well, this alone may serve as evidence that the bitstream is random, and applying the GTR in such cases could lead to spurious findings.

To address this, we use a **Null-LLR test** (shown in eqs. (15)–(18)), which we evaluate at a significance threshold of 0.01, as a preliminary step in the GTR procedure as is shown in Figure 4. This test assesses whether a given model (restricted or unrestricted) provides a statistically

significant improvement in fit compared to a null model that contains only a constant term. Specifically, we test whether the model trained on the observed bitstream offers any predictive advantage beyond the null model. The Null-LLR test serves as a safeguard to ensure that the fitted models are meaningful before proceeding with the GTR.

$$H_0: \ell_{\text{null}} = \ell_M \tag{15}$$

$$H_A: \ell_{\text{null}} < \ell_M \tag{16}$$

$$\lambda_{\text{Null},M} = -2 \times (\ell_{\text{null}} - \ell_M) \tag{17}$$

$$\lambda_{\text{Null},M} \sim \chi^2(k-1) \tag{18}$$

where $M \in \{R, U\}$ represents the restricted (R) and unrestricted (U) models, respectively, and k is the number of parameters in model M.

Failing the Null-LLR test does not necessarily preclude the use of the corresponding model in the GTR; however, it significantly limits the strength of any conclusions drawn from the test. If both the restricted and unrestricted models fail the Null-LLR test, this provides strong evidence that the bitstream lacks structure, reinforcing the hypothesis that it is truly random. However, we additionally hypothesize that cases where the unrestricted model rejects the Null-LLR test hypothesis while the restricted model fails to reject it could provide independent evidence of Granger causality. In such scenarios, the inclusion of past data enables the unrestricted model to achieve a significantly better fit, suggesting that past bit sequences influence future ones.

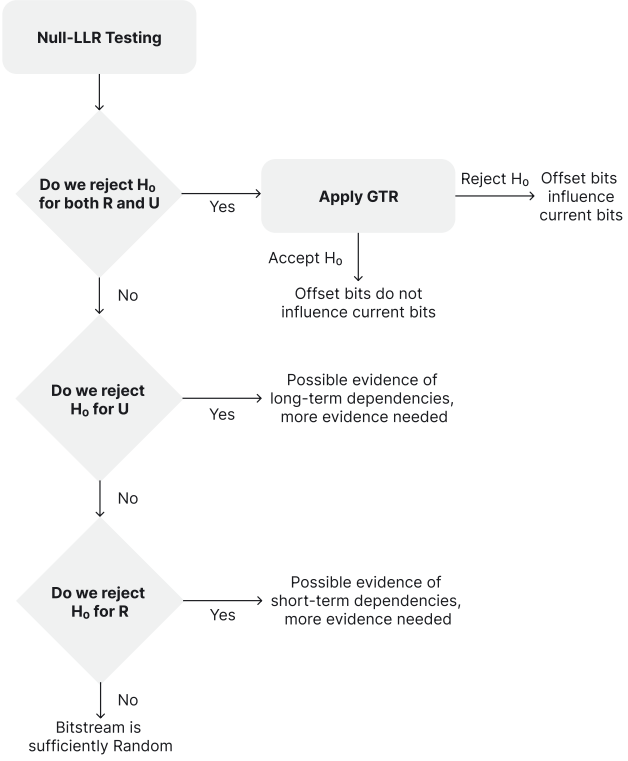


Figure 4: Decision diagram for applying the GTR contingent on the Null-LLR results.

4 Experimental results

To conduct a proof-of-concept experiment for our methodology, we initialize a bitstream with 10 million bits from the NIST Beacon [41], serving as a random baseline for evaluating our method. We perform the GTR on this random bitstream using $w = 31$ and $o = 6$, with results presented in the first entry of Table 2, designated as "NIST Baseline." Table 2 includes the GTR results alongside p-values from the preliminary Null-LLR test for both the restricted (R) and unrestricted (U) models. The Null-LLR test results indicate that neither model for the NIST baseline dataset is adequately fit, suggesting that the bitstream is sufficiently random, preventing the GTR test from being applied with total confidence. Furthermore, the GTR results themselves show no detectable difference in predictability between the restricted and unrestricted models for the NIST baseline.

In the remainder of Table 2, we evaluate the same 10 million bits from the NIST Beacon; however, we now insert repeated sequence pairs into the bitstream overwriting subsequences of the NIST Beacon data, such that the presence of these repeated sequence pairs should be detectable by the GTR. Specifically, we insert sequence A of 31 bits, advance by $w \times o$ bits, and then insert sequence B of 31 bits, such that when the GTR's unrestricted model's historical window contains only A bits, the first B bit will be the target bit.[2,3] We experiment with inserting 333, 1,111, 3,333, and 33,333 pairs of A and B. According to the reported Null-LLR tests in Table 2, for 1,111 pairs and higher, the restricted and unrestricted models are sufficiently fit. Additionally, the GTR detects significant differences in fit between the restricted and unrestricted models for 1,111 pairs and above. Notably, 1,111 pairs account for only 0.7 % of the bits in the bitstream, suggesting that the GTR may be reasonably sensitive to detecting repeated bit sequences.

For the same bitstream configurations reported in Table 2, we report the results on the NIST-STS test suite

2 Both A and B are generated using numpy's randint function.

3 We separate the insertion of each A by 300 bits to ensure that we exclusively target the influence of A on B and avoid confounding effects such as A sequences being influenced by past B sequences.

Table 2: Preliminary results for the GTR on a NIST Beacon bitstream with inserted repeat sequences.

Dataset		Null-LLR test		GTR (*R. vs U.*)		
# Repeat seq. pairs	*% of bits*	*Null-LLR (R) p-val*	*Null-LLR (U) p-val*	λ_{GTR}	*p-value*	*GC found*
Nist baseline: 0	0.000 %	0.290	0.107	41.247	0.103	No
333	0.213 %	0.186	0.058	42.599	0.080	No
1,111	0.711 %	8.9e-16	3.8e-21	90.2	1.1e-07	Yes
3,333	2.133 %	2.3e-231	0.0000	578.8	3.2e-102	Yes
33,333	21.333 %	0.000	0.0000	48,897.86	0.0000	Yes

in Table 3.[4] The GTR demonstrates strong performance in comparison to NIST-STS, even successfully detecting causality even in cases where some tests fail to identify any non-randomness. However, three tests from NIST-STS detected non-randomness in the dataset containing only 333 repeated sequence pairs, while the GTR failed to identify the presence of this non-randomness. These three NIST-STS tests are the approximate entropy test, the non-overlapping template matching test, and the serial test.

While the GTR is designed to detect the causation of one subsequence based on another, the serial and approximate entropy tests focus on identifying repeated occurrences of patterns. Because of this, the approximate entropy and serial tests are particularly well-suited for detecting the repetitiveness of the inserted sequence pairs in our test sets. Additional analysis is required to clearly distinguish the GTR from these methods, as the GTR evaluates whether past subsequences influence future ones, rather than merely detecting repetition. The current evaluation does not yet investigate cases where causal relationships exist with minimal repetition, which may further differentiate the GTR from these statistical tests.

Regarding the non-overlapping template matching test, we employed more than 100 default templates as a part of STEER. To consolidate the results of these tests into a single interpretable result, we applied Fisher's combined probability [42]. Of the templates evaluated, five of these templates occurred in either the inserted A or B sequences. Consequently, the repetitive insertion of these subsequences was readily identified by the non-overlapping template matching test. In contrast, for templates that did not match any bit sequences in A or B, the individual tests failed to reject the null hypothesis. This outcome underscores the importance of selecting the appropriate template in the non-overlapping template matching test, a constraint absent in the GTR method.

We perform granular testing by scaling the number of repeated sequence pairs and present the results in Figure 5. The top subplot shows GTR test statistics as the number of repeated sequence pairs increases from 333 to 1,323 in increments of 33. Also plotted are the GTR test statistics of 56, 52, and 45, corresponding to GTR p-values of $0.00\bar{3}$, 0.01, and 0.05, respectively. The bottom subplot displays the corresponding Null-LLR p-values for the unrestricted and restricted models, showing that the unrestricted model achieves a better fit with fewer repeated sequence pairs.

Table 3: Comparison of NIST-STS test suite (STEER) and GTR across increasing levels of repeated sequence pairs in the NIST Beacon bitstream.

Test	0 0.00 %	333 0.21 %	1,111 0.71 %	3,333 2.13 %	33,333 21.3 %
Approx. entropy ($m = 10$)	0.6030	0.0000	0.0000	0.0000	0.0000
Block frequency ($m = 128$)	0.6459	0.7785	0.9149	0.9997	1.0000
Cusum-Forward	0.0663	0.0706	0.0713	0.0842	0.0876
Cusum-Backward	0.0340	0.0340	0.0340	0.0340	0.0228
Discrete Fourier transform	0.9352	0.9306	0.0005	0.0000	0.0000
Linear complexity ($M = 500$)	0.7083	0.7613	0.6202	0.5180	0.5802
Longest run of ones	0.7729	0.7418	0.7199	0.5936	0.0000
Non-overlapping template ($m = 9$)	0.9324	0.0000	0.0000	0.0000	0.0000
Overlapping template ($m = 9$)	0.5416	0.5717	0.5018	0.0766	0.0000
Random excursions	0.2119	0.0669	0.2540	0.0707	0.0000
Random excursions variant	0.0866	0.2410	0.0701	0.0213	0.1103
Rank	0.1000	0.1201	0.0969	0.0318	0.8232
Runs	0.3270	0.1705	0.0184	0.0000	0.0000
Serial ($m = 16$, $\nabla\Psi^2_m$)	0.9602	0.0000	0.0000	0.0000	0.0000
Universal statistical	0.5191	0.1348	0.0000	0.0000	0.0000
GTR (ours)	0.1030	0.0800	1.1e-07	3.2e-102	0.0000

4 Note that for the random excursions and random excursions variant tests we report the minimum p-values. Except for the 33,333 test set, all random excursion tests agree to accept the null hypothesis. For the 33,333 test set, 4/8 tests agreed to reject the null hypothesis. Additionally, for the non-overlapping template matching test we use 100+ default templates provided in STEER and consolidate the results into a single p-value using Fisher's combined probability [42].

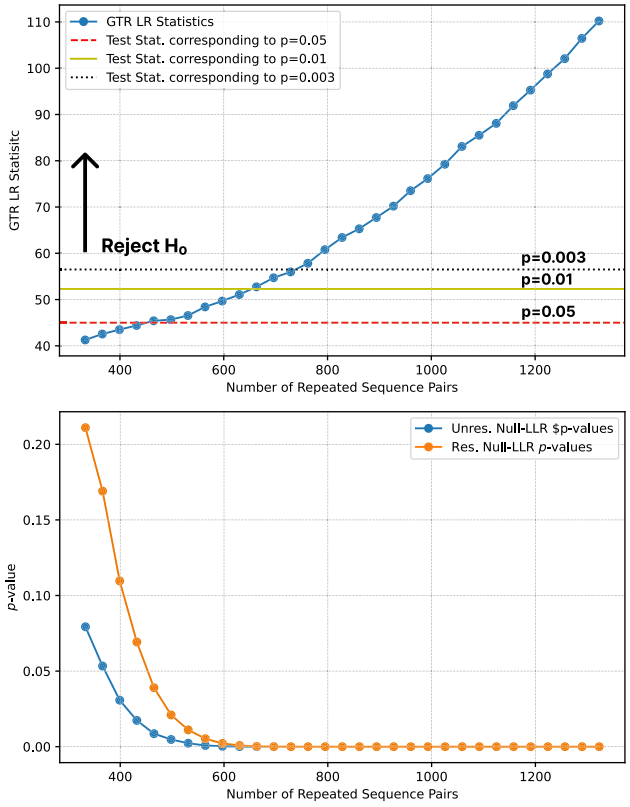


Figure 5: Top: Plotting GTR test statistics as the number of repeated bit sequences inserted into the NIST Beacon bitstream is scaled. Bottom: The corresponding Null-LLR test results for the unrestricted and restricted models.

The unrestricted model reaches a p-value below $0.00\bar{3}$ at 531 repeated sequence pairs, while the restricted model does not reach significance until 597. This discrepancy indicates that there may be instances where the restricted model does not sufficiently fit the data but the unrestricted model does. Drawing conclusions from the GTR in these cases is not advisable given that the restricted model cannot be adequately compared against. However, a well-fit unrestricted model and a poorly-fit restricted model could provide sufficient evidence that the historical bits add predictability.

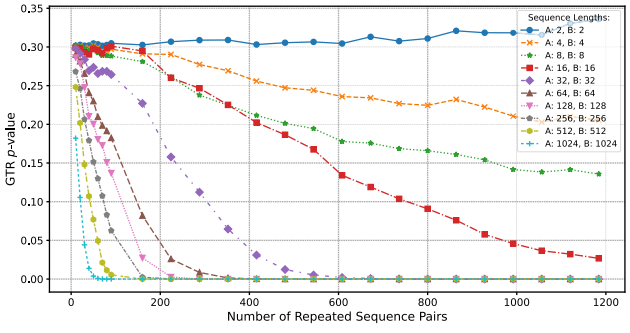


Figure 6: Scaling the lengths of both the A and B subsequences simultaneously from 2^1 to 2^{10} bits, with results shown as a function of the number of repeated sequence pairs.

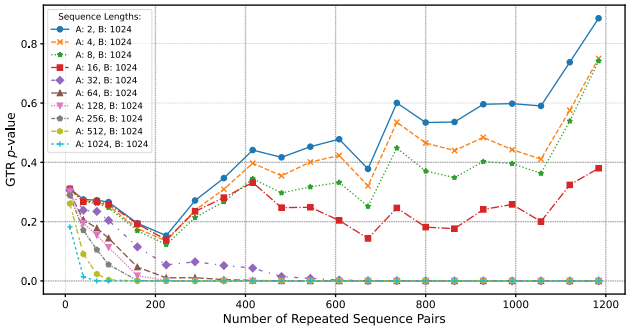


Figure 7: Scaling the length of the A subsequences from 2^1 to 2^{10} bits, with the length of the B fixed at 2^{10} bits.

We extend our analysis by maintaining the scaling of repeated sequence pairs while also varying sequence lengths. Two experiments are conducted: (1) both A and B are scaled simultaneously from 2^1 to 2^{10} bits, and (2) B is fixed at 2^{10} bits while only A's length varies. For the GTR, both use $w = 32$ and $o = 95$. To ensure alignment between the inserted sequences and the GTR we insert A and B every 8,192 bits with A starting at index 0 and B starting at index 3,072. This ensures that when the GTR's historical window is at the start of the A subsequence (e.g., bit positions 0–32), the target bit will reside at the start of the B subsequence (e.g., bit position 3,072).

Results for the first experiment (Figure 6) show that as both sequences grow, GTR increasingly detects added predictability, reflected in decreasing p-values. In the second experiment (Figure 7), a similar trend emerges for A sequences of at least 32 bits. However, for shorter A sequences, increasing their occurrences raises GTR's p-value, likely because the restricted model's ability to predict B from preceding instances of B overshadows the weaker predictive signal from shorter A sequences.

5 Summary and future extensions

The detection of non-random structures in bitstreams is essential for ensuring the security of cryptographic applications. While existing randomness tests evaluate statistical properties such as bit frequency distributions and periodic patterns, they do not directly assess causal dependencies within a sequence. In this work, we introduced the Granger-inspired Test for Randomness (GTR), a methodology designed to identify whether past segments of a bitstream influence future segments, thereby indicating

deviations from ideal randomness. By adapting Granger causality for binary data through logistic regression and applying it within a single sequence, we developed a test capable of uncovering hidden patterns that traditional methods may overlook.

Our evaluation demonstrated that the GTR does not falsely detect causality in a truly random dataset, using the NIST Beacon as a benchmark. However, when artificial patterns were embedded into the bitstream, the GTR successfully identified these structured dependencies, even when they accounted for a small fraction of the total data. These results highlight the sensitivity of our method and its potential as a complementary tool for randomness assessment. Future research could optimize GTR for increased sensitivity and explore applicability to cryptographic systems directly. By expanding the toolkit for randomness testing, the GTR contributes to strengthening the foundations of secure cryptographic practices.

Research ethics: Not applicable.
Informed consent: Not applicable.
Author contributions: All authors have accepted responsibility for the entire content of this manuscript and approved its submission.
Use of Large Language Models, AI and Machine Learning Tools: None declared.
Conflict of interest: The authors state no conflict of interest.
Research funding: None declared.
Data availability: Not applicable.

References

[1] E. B. Barker and J. M. Kelsey, "Recommendation for random number generation using deterministic random bit generators," Tech. Rep. NIST SP 800-90Ar1, National Institute of Standards and Technology, 2015.

[2] R. Gennaro, "Randomness in cryptography," *IEEE Secur. Priv.*, vol. 4, no. 2, pp. 64—67, 2006.

[3] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Cryptanalytic attacks on pseudorandom number generators," in *International Workshop on Fast Software Encryption*, Springer, 1998, pp. 168—188.

[4] E. Zenner, "Nonce generators and the nonce reset problem," in *Information Security*, *Lecture Notes in Computer Science*, vol. 5735, P. Samarati, M. Yung, F. Martinelli, and C. A. Ardagna, Eds., Berlin, Heidelberg, Springer, 2009, pp. 411—426.

[5] L. Dorrendorf, Z. Gutterman, and B. Pinkas, "Cryptanalysis of the random number generator of the windows operating system," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, pp. 1—32, 2009.

[6] B. Koerner, "Russians engineer a brilliant slot machine cheat — and casinos have no fix," *Wired*, 2017. Available at: https://www.wired.com/2017/02/russians-engineer-brilliant-slot-machine-cheat-casinos-no-fix/.

[7] F. Weimer, "New openssl packages fix predictable random number generator," 2008, https://lists.debian.org/debian-security-announce/2008/msg00152.html [Accessed: Mar. 08, 2025].

[8] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter, "Ron was wrong, Whit is right," *Cryptology ePrint Archive*, 2012. Available at: https://eprint.iacr.org/2012/064.pdf.

[9] F. Weimer, "Android security vulnerability," 2013, https://bitcoin.org/en/alert/2013-08-11-android [Accessed: Mar. 08, 2025].

[10] D. Shumow, N. Ferguson, and Microsoft, "On the possibility of a back door in the NIST SP800-90 dual EC," https://rump2007.cr.yp.to/15-shumow.pdf [Accessed: Mar. 08, 2025].

[11] N. Kostyuk and S. Landau, "Dueling over Dual_EC_DRBG: The consequences of corrupting a cryptographic standardization process," *Harv. Natl. Secur. J.*, vol. 13, no. 2, p. 224, 2022.

[12] E. Barker (NIST) and J. Kelsey (NIST), *NIST removes cryptography algorithm from random number generator recommendations*, vol. 08, p. 2025, Mar. Available at: https://www.nist.gov/news-events/news/2014/04/nist-removes-cryptography-algorithm-random-number-generator-recommendations.

[13] V. T. Hoang and Y. Shen, "Security analysis of NIST CTR-DRBG," in *Annual International Cryptology Conference*, Springer, 2020, pp. 218—247.

[14] J. Woodage and D. Shumow, "An analysis of NIST SP 800-90A," in *Advances in Cryptology — EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19—23, 2019, Proceedings, Part II 38*, Springer, 2019, pp. 151—180.

[15] NIST, "NIST releases first 3 finalized post-quantum encryption standards," 2024, https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards [Accessed: Mar. 08, 2025].

[16] A. Rukhin *et al*., "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Tech. Rep. NIST SP 800-22, Revision 1a, National Institute of Standards and Technology, 2010.

[17] C. W. J. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica*, vol. 37, no. 3, pp. 424—438, 1969.

[18] M. A. Thornton, "Assessing random bit generator quality with Granger causality extensions," in *International Cryptographic Modules Conference*, 2023. Presentation only, no written paper was produced.

[19] K. N. Smith, D. L. MacFarlane, and M. A. Thornton, "A quantum photonic TRNG based on quaternary logic," in *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL)*, IEEE, 2020, pp. 164—169.

[20] M. A. Thornton and D. L. MacFarlane, "Quantum photonic TRNG with dual extractor," in *Quantum Technology and Optimization Problems: First International Workshop, QTOP 2019, Munich, Germany, March 18, 2019, Proceedings 1*, Springer, 2019, pp. 171—182.

[21] A. Sinha, E. R. Henderson, J. M. Henderson, E. C. Larson, and M. A. Thornton, “A programmable true random number generator using commercial quantum computers,” in *Quantum Information Science, Sensing, and Computation XV*, vol. 12517, SPIE, 2023, pp. 35—49.
[22] J. W. Ange and M. A. Thornton, “Optimization and realization of boson sampling for true random number generation using the Xanadu X8,” in *Quantum Information Science, Sensing, and Computation XVII*, SPIE, 2025.
[23] G. Marsaglia, “Random number generators,” *J. Mod. Appl. Stat. Methods*, vol. 2, no. 1, pp. 2—13, 2003.
[24] R. G. Brown, “Robert G. Brown’s general tools page,” Github archive, 2006, https://rurban.github.io/dieharder/manual/dieharder.pdf [Accessed: Mar. 09, 2025].
[25] W. Killmann and W. Schindler, “A proposal for: Functionality classes for random number generators,” *ser. BDI*, Bonn, 2011.
[26] W. Schindler, *Evaluation Criteria for Physical Random Number Generators*, New York, NY, Springer, 2009, pp. 25—54.
[27] J. Walker, “A pseudorandom number sequence test program,” 2008, https://www.fourmilab.ch/random/ [Accessed: Mar. 09, 2025].
[28] G. Marsaglia and W. W. Tsang, “Some difficult-to-pass tests of randomness,” *J. Stat. Software*, vol. 7, no. 3, pp. 1—9, 2002.
[29] NIST Computer Security Resource Center, “Random bit generation RBG,” 2024, STS version 2.1.2, https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software [Accessed: Mar. 09, 2025].
[30] G. Marsaglia, “The Marsaglia random number CDROM including the diehard battery of tests of randomness,” Produced at Florida State University under a grant form the *National Science Foundation*, 1996, https://ani.stat.fsu.edu/diehard/ [Accessed: Mar. 09, 2025].
[31] R. G. Brown, “Die*Harder*: a Gnu public license random number tester,” 2005, https://webhome.phy.duke.edu/rgb/General/dieharder.php [Accessed: Mar. 09, 2025].
[32] R. Urban, “rurban/dieharder,” 2005, https://github.com/rurban/dieharder [Accessed: Mar. 09, 2025].
[33] D. Eddelbuettel and R. G. Brown, “RDieharder: An R interface to the dieharder suite of random number generator tests,” *Initial version as of May*, 2007, https://cran.r-project.org/web/packages/RDieHarder/vignettes/RDieHarder.pdf [Accessed: Mar. 09, 2025].
[34] German Federal Information Security Agency, “mjosaarinen/ais31-testsuite-v1.0,” 2020, https://github.com/mjosaarinen/ais31-testsuite-v1.0 [Accessed: Mar. 09, 2025].
[35] J. Walker, “Fourmilab/ent_random_sequence_tester,” 2022, https://github.com/Fourmilab/ent_random_sequence_tester [Accessed: Mar. 09, 2025].
[36] G. Marsaglia and W. W. Tsang, “Some difficult-to-pass tests of randomness,” 2002, https://www.jstatsoft.org/article/view/v007i03 [Accessed: Mar. 09, 2025].
[37] Anametric, Inc. “Lighting up the future,” 2025, https://anametric.com [Accessed: Mar. 09, 2025].
[38] Darwin Deason Institute for *Cyber Security, “Southern Methodist University (SMU),”* 2025, https://smu.edu/DDI [Accessed: Mar. 09, 2025].
[39] Darwin Deason Institute for Cyber Security, “SMU-DDI/steer-framework,” 2024, https://github.com/SMU-DDI/steer-framework [Accessed: Mar. 09, 2025].
[40] GNU Operating System, “GSL — GNU scientific library,” 2024, https://www.gnu.org/software/gsl/ [Accessed: Mar. 09, 2025].
[41] NIST Computer Security Division — Information Technology Laboratory, “NIST randomness beacon (version 2.0 beta) — interoperable randomness beacons | CSRC | CSRC,” 2019 [Online]. Available at: https://csrc.nist.gov/Projects/interoperable-randomness-beacons/beacon-20.
[42] R. A. Fisher, *Statistical Methods for Research Workers*, vol. 1, Edinburgh, Oliver & Boyd, 1925 [Online]. Available at: https://archive.org/details/statisticalmethoe7fish.

Bionotes

Joshua H. Sylvester
Southern Methodist University, Darwin Deason Institute for Cybersecurity, Dallas, TX, USA
jsylvester@smu.edu

Joshua H. Sylvester is a Ph.D. student in Computer Science at Southern Methodist University. He earned bachelor’s degrees in Computer Science and Geology in 2020 and a master’s degree in Computer Science in 2021, where his thesis focused on lightweight IoT-enabled marine observatories with cloud integration. His research interests span time series analysis, statistical inference, and machine learning, with a particular focus on Granger-inspired methods. His recent work includes the development of statistical tests for detecting predictive structure in cryptographic bitstreams, as well as related advances in lag selection, clustering, and representation learning for time series data.

Micah A. Thornton
Department of Mathematics, Texas Woman’s University, Denton, TX, USA
mthornton11@twu.edu

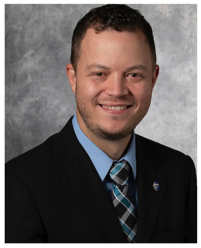
Micah A. Thornton is an Assistant Professor in the Department of Mathematics at Texas Woman’s University. He is a Ph.D. biostatistician trained through the joint UT Southwestern—SMU Biostatistics program, with an interdisciplinary background in Statistics and Computer Engineering. He earned a BS in Statistical Science and both a BS and MS in Computer Engineering in 2017, followed by a Ph.D. in Biostatistics in 2021, all from Southern Methodist University.

Jessie M. Henderson
Southern Methodist University, Darwin Deason Institute for Cybersecurity, Dallas, TX, USA
hendersonj@smu.edu

Jessie M. Henderson was a graduate research assistant at the Darwin Deason Institute for Cybersecurity while she completed her master's in Applied and Computational Mathematics. She is pursuing a Juris Doctorate at the University of Chicago, where she is the recipient of the Phil C. Neal Memorial Award, a Donald E. Egan Scholar, the President of the Law and Technology Society, the Vice President of the Law Students Association, and the Treasurer of the Intellectual Property Law Society.

Mitchell A. Thornton
Southern Methodist University, Darwin Deason Institute for Cybersecurity, Dallas, TX, USA
mitch@smu.edu

Mitchell A. Thornton is the Cecil H. Green Chair of Engineering and a Professor of Electrical and Computer Engineering at Southern Methodist University. He also serves as the Executive Director of the Darwin Deason Institute for Cyber Security and as Program Director for the M.S. program in Quantum Engineering within the Lyle School of Engineering. His research and professional interests include digital and embedded systems, cybersecurity, quantum informatics, and data processing algorithms and systems. He has authored or co-authored five books, more than 350 technical publications, and has led numerous sponsored research projects. He is also a named inventor on 33 U.S. and international patents. He received the Ph.D. degree in Computer Engineering from Southern Methodist University.



Eric C. Larson
Southern Methodist University, Darwin Deason Institute for Cybersecurity, Dallas, TX, USA
eclarson@smu.edu

Eric C. Larson is an Associate Professor of Computer Science at Southern Methodist University, with courtesy appointments in Electrical and Computer Engineering and in Operations Research and Engineering Management. His research lies at the intersection of machine learning and signal and image processing, with applications in security, health, education, human—machine teaming, and ubiquitous computing. He has secured over $9 million in federal and corporate research funding and has published extensively in peer-reviewed venues. He received the Ph.D. degree in Electrical and Computer Engineering from the University of Washington and an MS in Image and Signal Processing from Oklahoma State University.