# Public Comments on the Decision Proposal for SP 800-22 Rev. 1a

Comment period: January 12 – February 14, 2022

On August 6, 2021, NIST's Crypto Publication Review Board initiated a review of SP 800-22 Rev. 1a, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* (April 2010). In response to the public comments received, NIST proposed to revise SP 800-22 Rev. 1a to make a number of changes. This document includes the public comments on the decision proposal.

More details about this review are available at NIST's Crypto Publication Review Project site.

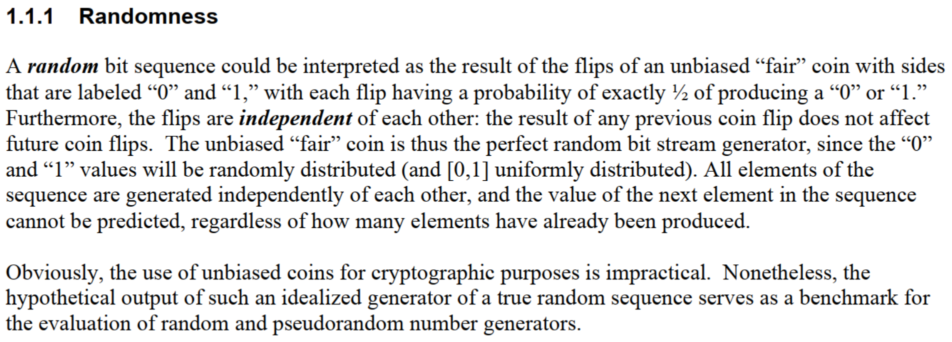
## LIST OF COMMENTS

## 1. Comments from David Johnston, February 9, 2022

Comments on SP800-22 Rev. 1a.

David Johnston
Principle Engineer, Security Researcher, Intel Corporation.
dj.johnston@intel.com

### Section 1.1.1 – Randomness



**1.1.1 Randomness**

A ***random*** bit sequence could be interpreted as the result of the flips of an unbiased "fair" coin with sides that are labeled "0" and "1," with each flip having a probability of exactly ½ of producing a "0" or "1." Furthermore, the flips are ***independent*** of each other: the result of any previous coin flip does not affect future coin flips. The unbiased "fair" coin is thus the perfect random bit stream generator, since the "0" and "1" values will be randomly distributed (and [0,1] uniformly distributed). All elements of the sequence are generated independently of each other, and the value of the next element in the sequence cannot be predicted, regardless of how many elements have already been produced.

Obviously, the use of unbiased coins for cryptographic purposes is impractical. Nonetheless, the hypothetical output of such an idealized generator of a true random sequence serves as a benchmark for the evaluation of random and pseudorandom number generators.

**Issue:**

This poorly describes the same notions as an unbiased IID bitstream or a full-entropy bitstream if we think coin tosses have entropy. These things are defined more clearly in SP800-90(A, B and draft C). For a seeded PRNG we do not have full-entropy and IID output. We do have some claim of a computational bound for the ability to distinguish the data from a uniform random sequence. For SP800-90A DRBGs that's commonly O($2^{128}$), O($2^{192}$) or O($2^{256}$).

**Resolution:**

Don't use coin-tosses for a model of PRNGs.

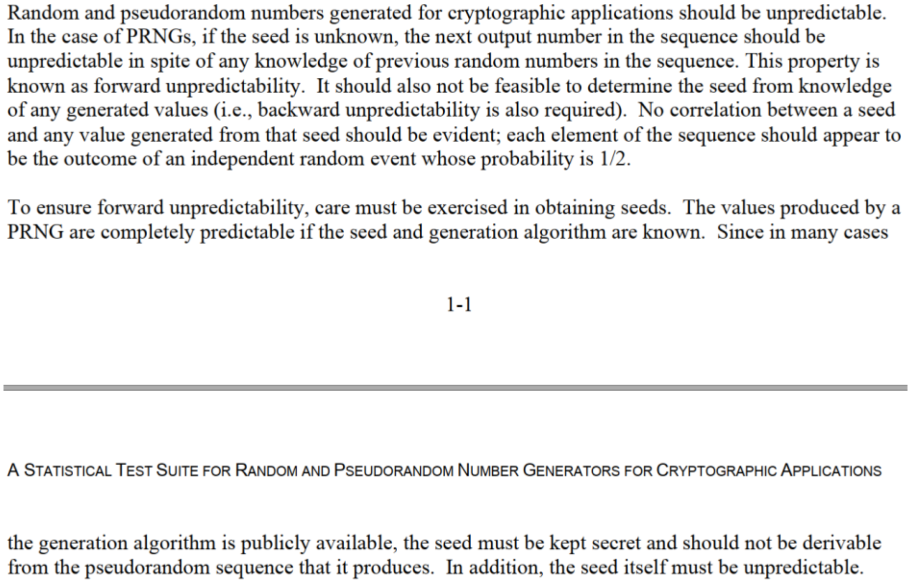
**Issue:**

The term PRNG for Pseudo Random Number Generator is used when other SP800 documents use the term DRBG, Deterministic Random Bit Generator to mean the same thing.

**Resolution:**

Bring this text into line with the text in the SP800-90 standards or refer directly to the definitions in SP800-90A. Use DRBG in place of PRNG.

**Section 1.1.2 – Unpredictability**



Random and pseudorandom numbers generated for cryptographic applications should be unpredictable. In the case of PRNGs, if the seed is unknown, the next output number in the sequence should be unpredictable in spite of any knowledge of previous random numbers in the sequence. This property is known as forward unpredictability. It should also not be feasible to determine the seed from knowledge of any generated values (i.e., backward unpredictability is also required). No correlation between a seed and any value generated from that seed should be evident; each element of the sequence should appear to be the outcome of an independent random event whose probability is 1/2.

To ensure forward unpredictability, care must be exercised in obtaining seeds. The values produced by a PRNG are completely predictable if the seed and generation algorithm are known. Since in many cases

1-1

A STATISTICAL TEST SUITE FOR RANDOM AND PSEUDORANDOM NUMBER GENERATORS FOR CRYPTOGRAPHIC APPLICATIONS

the generation algorithm is publicly available, the seed must be kept secret and should not be derivable from the pseudorandom sequence that it produces. In addition, the seed itself must be unpredictable.

**Issue:**
Terms like ‘not be feasible’ contain no quantification when we have precise quantification for DRBGs.

**Resolution:**
Express unpredictability in terms of the amount of computational effor required to predict future or past values without the key or the amount of computational effort to predict the key from the output data.

**Sections 1.1.3 and 1.1.4**

**Issue:**
‘RNG’ is used where SP800-90 would use ‘noise source’ or ‘entropy source’ depending on where in the processing chain you are looking.
‘PRNG’ is used

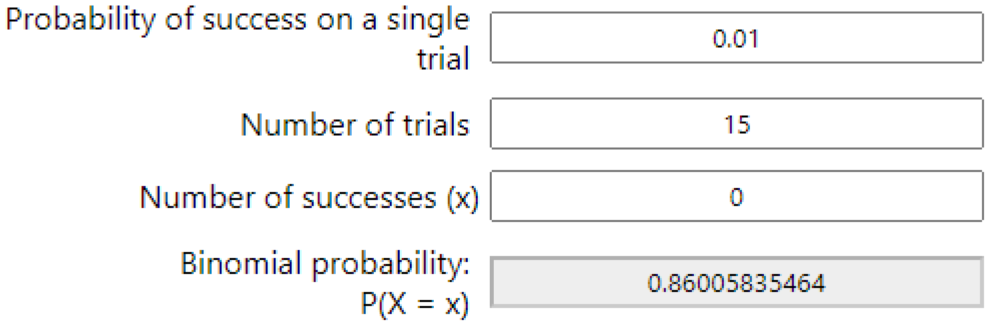
**Resolution:**
Bring the terminology into alignment with SP800-90 terms DRBG, entropy source and noise source.

**Section 1.1.5 Testing**

**Issue:**
The false positive rate of the test method is very high leading to actual false positive errors in testing uniform data.

There are 15 tests. Each has a 1% false positive rate with uniform data.

Plugging the numbers into a randomly selected online binomial calculator gives:



Probability of success on a single trial: 0.01

Number of trials: 15

Number of successes (x): 0

Binomial probability: P(X = x): 0.86005835464

So the chances of uniform data being failed by SP800-22 is approximately 1.0 – 0.86 or 13.139%
This explains why I frequently see SP800-22 indicate a failure over the output of an SP800-90A DRBG.

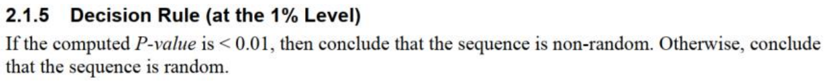
**Resolution:**
The data size of 1,000,000 bits limits the confidence you can have in the results.
Increase the data size and reduce the false positive rate to a tolerable limit (E.G. 1 in 10,000).

Alternatively, require the tests to be run multiple times over fresh data and apply a decision test over the sequence of results.

**Sections 2.[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15].5 Decision Rules**

The decision rule text in each test contains a dubious inference.

E.G.:



**2.1.5 Decision Rule (at the 1% Level)**

If the computed *P-value* is < 0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

Concluding the sequence is random is not a safe inference. We can only concluding that the test has failed to detect the failing criteria with 99% confidence.

Example:
djenrandom () is a program to create random data with well defined statistical properties. It is only nondeterministic with the -s option, but uses a built in seed otherwise. The 'pure' model produces uniform data with a CTR-DRBG. It so happens that the default sequence is failed by SP800-22.

```
dj@deadhat:/home/dj/src/sp800_22_tests$ djenrandom -b -k 128 -m pure >
random.bin
dj@deadhat:/home/dj/src/sp800_22_tests$ python3 sp800_22_tests.py random.bin >
result.txt
```

```
dj@deadhat:/home/dj/src/sp800_22_tests$ grep FAIL result.txt
  FAIL
random_excursion_test                    0.00040325014333107326 FAIL
```

If I run it again with -s to get a different random sequence, it usually passes:

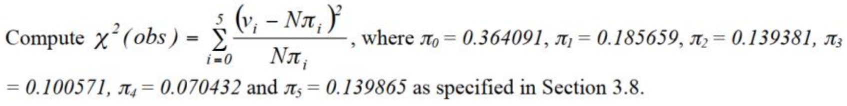
```
root@deadhat:/home/dj/src/sp800_22_tests# djenrandom -s -b -k 128 -m pure >
random2.bin
root@deadhat:/home/dj/src/sp800_22_tests# python3 sp800_22_tests.py random2.bin
> result2.txt
root@deadhat:/home/dj/src/sp800_22_tests# tail -20 result2.txt
P=0.8259553663237427
P=0.908642203453824

SUMMARY
-------
monobit_test                             0.6903077202130459 PASS
frequency_within_block_test              0.9878402262367767 PASS
runs_test                                0.8895869147970908 PASS
longest_run_ones_in_a_block_test         0.8843489342929933 PASS
binary_matrix_rank_test                  0.9805685792290544 PASS
dft_test                                 0.5945070174258117 PASS
non_overlapping_template_matching_test   0.9999997562576374 PASS
overlapping_template_matching_test       0.20729804098987048 PASS
maurers_universal_test                   0.1302644392380732 PASS
linear_complexity_test                   0.9897870378716215 PASS
serial_test                              0.948918339134405  PASS
approximate_entropy_test                 0.9488895296808474 PASS
cumulative_sums_test                     0.71559524543709   PASS
random_excursion_test                    0.049287497897004144 PASS
random_excursion_variant_test            0.03886253574485431 PASS
root@deadhat:/home/dj/src/sp800_22_tests#
```

Section 2.8 Overlapping Template Matching Test

**Issue:**
This issue was first published in “Random Number Generators, Principles and Practices”. Section 9.6

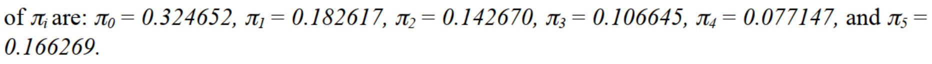
The $\pi_i$ coefficients are inconsistently described.

For the defined test the values are:



Compute $\chi^2(obs) = \sum_{i=0}^{5} \frac{(v_i - N\pi_i)^2}{N\pi_i}$, where $\pi_0 = 0.364091$, $\pi_1 = 0.185659$, $\pi_2 = 0.139381$, $\pi_3 = 0.100571$, $\pi_4 = 0.070432$ and $\pi_5 = 0.139865$ as specified in Section 3.8.

For the example the values are:



of $\pi_i$ are: $\pi_0 = 0.324652$, $\pi_1 = 0.182617$, $\pi_2 = 0.142670$, $\pi_3 = 0.106645$, $\pi_4 = 0.077147$, and $\pi_5 = 0.166269$.

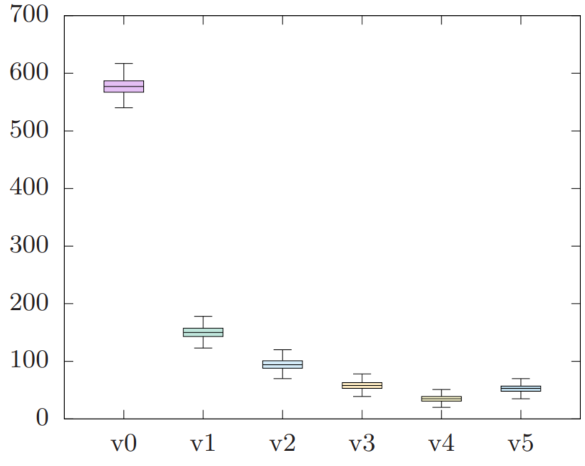
The NIST example STS-2.1.2 test code uses the example coefficients, not the proper coefficients.

Section 3.8 shows the example test coefficients as the proper ones and so is not in sync with the text in 2.8, which states the reverse.

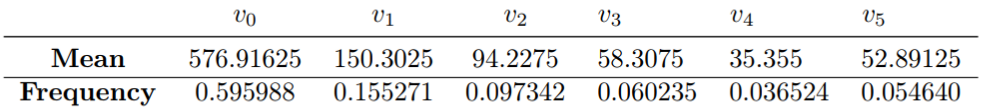
**Resolution:**
Clarify which are the correct values to use.

**Issue:**
The coefficients don’t match reality.

800 runs were performed over the recommended 1,028,016 bits of uniform random data and the distribution was measured.



Which leads to the empirically derived coefficients as follows:



| | $v_0$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ |
|---|---|---|---|---|---|---|
| **Mean** | 576.91625 | 150.3025 | 94.2275 | 58.3075 | 35.355 | 52.89125 |
| **Frequency** | 0.595988 | 0.155271 | 0.097342 | 0.060235 | 0.036524 | 0.054640 |

This is different to the example and proper coefficients in SP800-22.

**Resolution:**
Fix whatever is wrong with the mathematics in section 3.8 and use coefficients that match actual pattern frequencies in uniform random data.

**Issue:**
The data sizes don't add up

The test procedure states the minimum data size is 1,000,000 bits.
The test indicates to set M=1032, but this results in a total data size of less than 1,000,000 bits.

**Resolution:**
Set M=1062, not 1032

**Section: All the Document**

**Issue:**
This standard defines tests that unreliably distinguishes nonuniform data from uniform, but it not sufficient to identify a secure CSPRNG from an insecure PRNG for all but the worst case PRNG algorithms.

These tests are widely applied in papers and journal articles to justify the performance of RNGs. The NIST stamp of approval adds an air of suitability for the tests when in reality they do not achieve the stated goal.

I question the reason for this document existing. We have moved on to an approach of cryptanalysis of PRNGs and DRBGs with the SP800-90B procedures for assessing and conditioning noise sources to seed those DRBGs. This document does more harm than good.

**Resolution:**
Either add stronger wording at the start of the document, rejecting it’s use for assessing secure RNGs or withdraw the document entirely.

## 2. Comments from Markku-Juhani O. Saarinen, February 14, 2022

Dear NIST Crypto Publication Review Board,

Please find attached a PDF that contains commentary on the SP 800-22 Rev. 1a Decision Proposal. The document can also be downloaded via the URL: https://mjos.fi/doc/saarinen_sts1a.pdf

SP 800-22 and GM/T 0005-2012 Tests: Clearly Obsolete, Possibly Harmful

Comments on SP 800-22 Rev. 1a Decision Proposal. February 14, 2022.

Markku-Juhani O. Saarinen, PQShield Ltd.

Abstract. NIST SP 800-22 describes 15 statistical tests and suggests that they can be used for the evaluation of random and pseudorandom number generators in cryptographic applications. The Chinese standard GM/T 0005-2012 describes similar tests. The weakest of pseudorandom number generators will easily pass these tests, which promotes false confidence in insecure systems. Evaluation of pseudorandom generators and sequences should be based on cryptanalytic principles. Implementation validation should be focused on algorithmic correctness, not the randomness of output. For true random (entropy sources), the focus should be on the true entropy content and reliability of the construction and health tests. If the SP 800-22 is to be revised, we suggest the new SP focuses on evaluating stochastic models for entropy sources as the SP 800-90 series currently does not address this issue in depth. We further suggest that pseudorandom generators are analyzed for their suitability for post-quantum cryptography and lack of (asymmetric) backdoors or covert channels. We illustrate this by discussing the ``reference generators'' in SP 800-22 Appendix D, none of which are suitable for use in modern cryptography.

Best Regards,

- Markku

Dr. Markku-Juhani O. Saarinen <mjos@pqshield.com> PQShield, Oxford UK.

# NIST SP 800-22 and GM/T 0005-2012 Tests: Clearly Obsolete, Possibly Harmful

*Comments on SP 800-22 Rev. 1a Decision Proposal. February 14, 2022.*

Markku-Juhani O. Saarinen

PQShield Ltd., Oxford, UK, mjos@pqshield.com

**Abstract.** NIST SP 800-22 describes 15 statistical tests and suggests that they can be used for the evaluation of random and pseudorandom number generators in cryptographic applications. The Chinese standard GM/T 0005-2012 describes similar tests. The weakest of pseudorandom number generators will easily pass these tests, which promotes false confidence in insecure systems. Evaluation of pseudorandom generators and sequences should be based on cryptanalytic principles. Implementation validation should be focused on algorithmic correctness, not the randomness of output. For true random (entropy sources), the focus should be on the true entropy content and reliability of the construction and health tests. If the SP 800-22 is to be revised, we suggest the new SP focuses on evaluating stochastic models for entropy sources as the SP 800-90 series currently does not address this issue in depth. We further suggest that pseudorandom generators are analyzed for their suitability for post-quantum cryptography and lack of (asymmetric) backdoors or covert channels. We illustrate this by discussing the "reference generators" in SP 800-22 Appendix D, none of which are suitable for use in modern cryptography.

**Keywords:** SP 800-22 Rev 1a, GM/T 0005-2012, Statistical Randomness Tests

## 1 Introduction

In 2021 NIST's Cryptographic Publication Review Board initiated a review process for NIST Special Publication (SP) 800-22 Rev. 1a, *"A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications."* [RSN+10] In January 2022, it was announced that – against the expectations of many cryptographers – that the document is not withdrawn but will be revised instead[1].

**Obsoleted by SP 800-90 series.** SP 800-22 is not actually used – and should not be used – in NIST's Deterministic Random Bit Generator (DRBG) and Entropy Source (ENT) validation processes [NC21, Annex D.J]. FIPS 140-3 [NIS19] adopts the much more robust methodology of the NIST SP 800-90 series instead [BK15, TBK+18, BKR+21].

SP 800-90B also forms the basis for NIAP Entropy Assessment Reports (EAR), used in U.S. Government NSS and Common Criteria Protection Profiles [NIA13a, NIA13b].

The SP 800-22 document mainly finds use by amateur cryptographers and vendors of insecure systems, as it trivializes random bit generator validation to black-box statistical testing. Its use as evidence of security usually signals that competent specialists have not been involved in the design and analysis of a random bit generator.

[1] January 12, 2022: Announcement of Proposal to Revise Special Publication 800-22 Revision 1a https://csrc.nist.gov/News/2022/proposal-to-revise-sp-800-22-rev-1a

## 2 The Tests: Cryptanalysis-in-a-Box?

The SP 800-22 Rev. 1a document [RSN$^+$10] contains descriptions of 15 statistical tests, which have also been implemented in a software package available from the NIST website[2]. We note that the current Chinese standard GM/T 0005-2012 [SCA12] contains a very similar set of 15 tests. The Chinese standard is also coming into a review, with a revised standard GM/T 0005-2021 coming into effect in May 2022 (the changes in the revised Chinese standard are unknown to the author at the time of writing.)

The 15 tests take in a sequence of output bits, typically 1,000,000 bits, and produce P values based on that information, which leads to a PASS/FAIL metric (with a very high false-positive rate, as noted in other comments.)

1. Frequency (Monobit).
2. Block Frequency.
3. Runs Test.
4. Longest Run of Ones.
5. Binary Matrix Rank Test.
6. Discrete Fourier (Spectral).
7. Non-overlapping Template.
8. Overlapping Template Matching.
9. Maurer Universal Statistical.
10. The Linear Complexity Test.
11. The Serial Test.
12. Approximate Entropy.
13. Cumulative Sums (Cusums).
14. Random Excursions.
15. Random Excursions Variant.

## 3 A Systemic Problem: Security is Not Considered

The SP 800-22 tests are based on purely statistical interpretation of uniform and independent randomness [RSN$^+$10, Sect. 1.1.1]. The definitions and stated goals of the tests do not relate to computational indistinguishability [KL14, Sect. 7] or other relevant cryptographic security notions. Randomness is not viewed from a cryptanalytic security perspective: How hard is it to “break” the random and pseudorandom generators?

The word “cryptanalysis” can be found in the abstract of SP 800-22 three times, but not once in the body of the publication. Some of the included tests have a cryptanalytic flavor, however. As an illustrative example, the “linear complexity test” is motived by stating that *“An LFSR that is too short implies non-randomness.”* That may be true in relation to some non-cryptographic notion of randomness, but every cryptanalyst knows that a plain LFSR is never a secure pseudorandom generator.

Since LFSRs are linear, it is a simple exercise in cryptanalysis to solve the internal state of a fixed LFSR from the output. The linear algebra required runs in polynomial time; hence plain LFSRs are cryptanalytically broken regardless of their length.

**Suggestion: Cryptanalysis, Security Proofs, and Design Reviews.** The process of assessing the security of a pseudorandom number generator is similar to that of other cryptographic modes and constructions. In practice, one wishes to demonstrate (via mathematical proofs) that breaking a DRBG (“pseudorandom generator” in the language of SP 800-22) implies a break of an underlying vetted cryptographic algorithm such as AES or SHA-2/3.

The security of physical entropy sources is mainly assessed via review and testing of entropy production processes (stochastic models) and the robustness of their implementation. These are much more important aspects than the statistical uniformity of final output, which can be guaranteed with cryptographic conditioning in any case. In fact, entropy sources should avoid hiding statistical properties in their unconditioned “raw noise”; access to the raw noise is necessary for validating the actual entropy content and also the correctness of their stochastic models (See Section 5).

[2] Visited February 10, 2022: “NIST SP 800-22: Download Documentation and Software.” https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software

## 4 Validating Implementations, not Uniform Distributions

"Testing Strategy and Result Interpretation" [RSN+10, Sect. 4] suggests that the 15 statistical tests are applied to random bit generators based on cryptographic hash functions and block ciphers. It should be obvious that no new or useful statistical features can be found in the output of standard cryptographic algorithms with these tests. This is true even if the algorithms are run without any secret seeding material.

The only sensible motivation for using the tests to test a pseudorandom bit generator would be to discover flaws in their implementation. However, a buggy RNG should not be used, even if the bugs are so minor that the output still passes these trivial tests. It is much more useful to validate that a cryptographic hash function or block cipher is implemented according to the standard than it is to verify their statistical qualities.

As noted, even good physical entropy sources are generally not expected to yield strictly uniform output. In security evaluation NIST can forbid the direct use of physical entropy sources without cryptographic post-processing (conditioning) [BKR+21].

**Suggestion: Validate the Implementations.** Clearly, a statistical test is a poor way of verifying that a deterministic algorithm has been correctly implemented. Typically one would at least run DRBGs with known or chosen inputs and utilize Known Answer Tests (KATs) with reference tests vectors. This is one of the things that is done in NIST's CAVP[3]. For additional assurance, one can use formal methods to validate the correctness of the implementation, which is standard practice with hardware modules.

## 5 Suggestion for a replacement SP: Stochastic Models

There is a statistical aspect of random bit generation that a replacement to SP 800-22 could address: Entropy Source Stochastic Models. While stochastic models are mentioned and suggested (they're a "may" not a "should") in SP 800-90B [TBK+18, Sect 3.2.2], the creation of stochastic models is not really addressed in the current SP 800-90 series.

Stochastic models are a common requirement in the AIS-20/31 [KS11] evaluation methodology from German BSI. It would benefit vendors and testing labs if the NIST and BSI requirements were further harmonized. The definition of a stochastic model for an entropy source (or a "TRNG" in AIS-20/31) is already very similar in the two documents.

> *A stochastic model is a mathematical description (of the relevant properties) of an entropy source using random variables. A stochastic model used for an entropy source analysis is used to support the estimation of the entropy of the digitized data and finally of the raw data. In particular, the model is intended to provide a family of distributions, which contains the true (but unknown) distribution of the noise source outputs. Moreover, the stochastic model should allow an understanding of the factors that may affect the entropy. The distribution of the entropy source needs to remain in the family of distributions, even if the quality of the digitized data goes down. [TBK+18, Appendix B, Page 65]*

In practice, one studies the noise source and identifies the stochastic processes that generate entropy. Understanding the entropy process allows a stochastic model, output distributions, and min-entropy estimates to be developed. It also helps in deriving failure threshold parameters for the health tests required by SP 800-90B. A stochastic model can also have environmental components (e.g., temperature, voltage, interference metrics).

The statistical hypothesis testing (in entropy source validation) would mainly involve testing whether a physical entropy source behaves as predicted by its stochastic model.

[3] NIST Cryptographic Algorithm Validation Program https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program

# A Appendix: All the bad "Reference Generators"

Appendix D of SP 800-22 contains descriptions of *"Reference Pseudorandom Number Generators."* It is unclear what the purpose of these generators is, apart from testing the proposed statistical suite itself.

We use this to illustrate certain specific features of secure pseudorandom number generators (DRBGs) that can be assessed in a design review. This is in addition to the standard high-level considerations of statistical unconspiciousness, and (enhanced) forward and backward security [KS11, Sect 2.2.2].

- **Protection of private state.** The determination of the internal state from DRBG output should be demonstratively hard. *Analysis: All of the generators in Appendix D fail in this task, either by design, due to the use of weak cryptography, or due to potential backdoor access.*
- **Private state size.** Pseudorandom number generator (DRBG) should have a sufficiently large private internal state to resist cryptanalysis. Due to time-memory tradeoff attacks, no less than 384 bits suffices for a 256-bit security level for a DRBG with substantial data output. *Analysis: All generators in Appendix D fail to varying degree.*
- **Seeding.** The internal state must be seeded with sufficient entropy before it can be used to produce output. *Analysis: Not addressed by SP 800-22 or its generators.*
- **Implementation correctness and integrity.** *Analysis: Not addressed by SP 800-22 or its generators.*
- **Quantum Safety.** The use of primitives that rely on the hardness of problems for which fast quantum attacks exist (e.g. factoring and the elliptic curve discrete logarithm problem [Sho94]) are not suitable for use with quantum-safe cryptography. *Analysis: The BBSG and MSG generators explicitly fail in this aspect, possibly MODEXP too.*
- **Absence of trapdoors / backdoors.** The existence of any "public key trapdoor" function in a place where secure symmetric cryptography could be used is often an indication that the generator can be used to instantiate a covert channel or a backdoor. While a failure to protect the private state effectively broadcasts secrets, "unique-access" backdoors generally require public-key trapdoor functions. With such a backdoor, a third party can choose the public instantiation parameters so that a "private key" allows the secret state to be covertly determined from DRBG output. *Analysis: The BBSG and MSG generators allow this, possibly MODEXP too.*

## A.1 Linear Congruential Generator (LCG)

The first "reference generator" in Appendix D is a multiplicative congruential generator, attributed to Fishman and Moore in the text, but really proposed by D. H. Lehmer in 1949 [Leh51] – Lehmer even proposed the specific Mersenne prime modulus $p = 2^{31} - 1$ used. Multiplicative generators can be seen as a subset of linear congruential generators with the addition constant set to zero, although their analysis is slightly different.

**Note on the description and implementation of LCG.** Section D.1 offers a definition for a multiplicative generator as $z_{i+1} = a * z_i \bmod (2^{31} - 1)$, without stating what the $a$ value is. Instead, the text states that "$a$ is a function of the state," which is clearly incorrect. Examination of the code and outputs reveals it to be $a = 950706376$, which leads to a maximum $(2^{31} - 2)$ period.

The code (function `lcg_rand()` in `src/generators.c`) is a literal, line-by-line translation to C of the 1980s era Fortran code written by L. R. Moore of RAND corporation. The original can be found in [Fis96, Fig. 7.3, p. 604]. The reference code implements the Lehmer iteration (`seed = (seed * 950706376) % 0x7fffffff;`) with five double-precision multiplications, four full divisions, three calls to the `floor()` function, and some additional arithmetic. Perhaps this made sense with a specific 1980s computer without an integer multiplier, but we doubt it. We noticed the code does not work correctly under more aggressive floating-point optimization levels; it is dependent on specific IEEE 754 floating-point properties in a very "fragile" manner.

We note that the $(\bmod\ p)$ reduction can be easily implemented with a shift and an add since $2^{31} \equiv 1 \pmod{p}$. No division is required, just a single $31 \times 31$-bit multiply.

**Weaknesses.** Even if we ignore the cryptanalytically trivial 31-bit state size, a generator of this type can be rapidly attacked and distinguished from random [FHK$^+$88].

## A.2 Quadratic Congruential Generator I (QCG-I)

The quadratic generator QCG-I is characterized by iteration $x_{i+1} = x_i^2 \pmod{p}$, a 512-bit $p$, and starting point $x_0$. The values $x_i$ are used directly as 512-bit blocks.

**Weaknesses.** The generator outputs its entire state. From $x_i$ it is trivial to compute $x_{i+1}$ and also the two square roots $\pm x_{i-1}$. Hence the generator has neither forward nor backward security.

Statistically, the most significant bit of each 512-bit block will be biased by $2^{-512}p \approx 0.5956$ since $x_i < p$. The test suite does not seem to detect it, however. The bias leads to a trivial, low-complexity statistical distinguisher against the generator.

## A.3 Quadratic Congruential Generator II (QCG-II)

The QCG-II generator is based on iteration $x_{i+1} = 2x_i^2 + 3x_i + 1 \pmod{2^{512}}$.

**Weaknesses.** Entire 512-bit blocks are output, including the least significant bits. The generator allows both prediction and backtracking.

QCG-II does not have good statistical properties since there is no information flow from high-order bits towards the lower-order bits. The least significant bits $x_i \bmod 2^m$ will form a cycle with period of at most $x^m$. For example, the least significant bit of each 512-bit block alternates in each block. The property leads to a trivial, low-complexity distinguisher against the generator.

## A.4 Cubic Congruential Generator (QCG)

The iteration used in GCQ is $x_{i+1} = x_i^3 \pmod{2^{512}}$. We observe that this is equivalent to $x_i = x_0^{3^i} \pmod{2^{512}}$. The entire variable $\{x_i\}$ is output.

**Weaknesses.** The output blocks $x_i$ can be reduced to smaller modulus size $2^m$ for analysis. As a simple observation, we note that the least significant byte satisfies $x_i \equiv x_{i+16} \pmod{2^8}$; the cycle is only 16. As a result, there will be an equivalent byte repeating with distance $16 * (512/8) = 1024$ bytes, which should be detectable purely statistically – but is not. These properties lead to trivial, low-complexity distinguishers against the generator.

## A.5 Exclusive OR Generator (XORG)

The generated sequence starts with an initial value for $x_1, x_1, \ldots, x_{127}$ and applies the recurrence $x_i = x_{i-1} \oplus x_{i-127}$ for $i \geq 128$ to generate bits.

It is not explained that XORG implements an LFSR with an irreducible generator polynomial $x^{127} + x^{126} + 1$. Indeed its cycle $2^{127} - 1$ is prime too.

**Weaknesses.** There is no secret state with $x_i$, and hence the XORG / LFSR has no cryptanalytic security – despite attractive statistical properties and a long period. One can trivially both predict and backtrack outputs. A low-complexity attack models the LRSR as a system of linear equations over $\mathsf{GF}(2)$ and is able to distinguish it from random even if a continuous "block" of output bits are not available.

## A.6 Modular Exponentiation Generator (MODEXP)

The MODEXP generator is loosely based on the old Digital Signature Standard (DSS). One sets $x_1 = g^{\text{seed}} \mod p$ and $x_{i+1} = g^{y_i} \mod p$ for $i \geq 1$ where $y_i = x_i \bmod 2^{160}$. The output sequence consists of concatenated $x_i$ values.

**Weaknesses.** Since full output blocks $x_i$ are available, the sequence has no security. It can also be also trivially distinguished from random since the values satisfy $x_i < p$.

Some parameters are given, but not those that would be needed to assess the statistical quality of the generator. The $(p, g)$ values are the same as the $(p, x_0)$ values for QCG-1:

$$
\begin{aligned}
p = {} & \texttt{987b6a6bf2c56a97291c445409920032499f9ee7ad128301b5d0254aa1a9633f} \\
& \texttt{dbd378d40149f1e23a13849f3d45992f5c4c6b7104099bc301f6005f9d8115e1} \\
g = {} & \texttt{3844506a9456c564b8b8538e0cc15aff46c95e69600f084f0657c2401b3c2447} \\
& \texttt{34b62ea9bb95be4923b9b7e84eeaf1a224894ef0328d44bc3eb3e983644da3f5}
\end{aligned}
$$

We note that the order of the generator $g$ is not given; if it is very small, then only a small number of different $x_i$ values would be generated.

**Checking for backdoors.** To see how the backdoor could have been created into a generator of this type, we examine the factorization of the multiplicative order $p - 1$. Since the order of $g$ was not given, it was left to the author to discover the factorization[4]:

$$p - 1 = 2^5 * 11 * 47 * 151 * 175916087 * 22940963671 * p_{160} * p_{269},$$

where the two largest prime components are

$$
\begin{aligned}
p_{160} &= 1213137149285565671196618904624637288561542502557, \quad \text{and} \\
p_{269} &= 6529682639905403810339488131303178790370220143281 \\
& \quad\; 0974248631298387656923566051172 1.
\end{aligned}
$$

We find that $p_{160}$ is the order of the value $g$ presented: $g^{p_{160}} \equiv 1 \pmod{p}$.
It is also trivial to find weaker generators. For example,

$$
\begin{aligned}
h = {} & \texttt{31fbe4d542ad14935055b18465de2a19d261d3fd0625c565d1e17dceaebc479e} \\
& \texttt{860bcb11a6d6697a0b1fd7172567600a8808df985e2c88379bcb3a565db805e4}
\end{aligned}
$$

[4] The smaller factors were discovered on a personal laptop in a few minutes using GMP-ECM Elliptic Curve Method (ECM) factorization software. The remaining 429-bit composite was factored into $p_{160} * p_{269}$ in about 95 minutes of computation with a 2x16-core (64-thread) AMD EPYC 7302 server using CADO-NFS Number Field Sieve (NFS) software.

satisfies $h^{11} \equiv 1 \mod p$ and hence $g = h$ could generate at most 11 different 512-bit $x_i$ values. A shorter cycle is more likely, with the expected size being $O(\sqrt{n})$ where $n$ is the order of the multiplicative subgroup generated by $g$.

## A.7 Secure Hash Generator (G-SHA1)

SP 800-22 does not actually describe this generator but suggests looking at [MvOV96, Chap 5, P. 175]. Investigation of the code confirms that actual SHA-1 is not used, but a variant $G$ with no message padding. The comment in STS 2.1.2, line 370 of `src/generators.c` states: *"This is the generic form of the generator found on the last page of the Change Notice for FIPS 186-2"*.[5]. This 186-2 generation method is obsolete as it produced biased random numbers (rejection sampling was not used, just reduction mod $q$).

The "generic form" actually implemented does not use the "Xseed" variable provided in the document. It sets $y_0 = Xkey$ and iterates for $x \geq 1$:

$$\begin{aligned} x_i &\leftarrow G(t, y_{i-1}) \\ y_i &\quad y_i + x_i + 1 \bmod 2^{160} \end{aligned}$$

Where $G(\mathsf{IV}, M)$ is a SHA-1 variant with start state $t = \mathsf{IV}$ and message $M$ with (non-standard) zero padding. The $x_i$ values are output. This generator prevents the direct observation of the counter state $y_i$. Since it is not an actual counter mode, it will exhibit a cycle length of roughly $O(\sqrt{N}) \approx 2^{80}$. The security is also limited by the security of the SHA-1 compression function.

**Weaknesses.** The SHA-1 algorithm was cryptanalytically broken in 2005 [WYY05] and depreciated by NIST in 2011 [BR11]. Many types of full, practical attacks have been demonstrated [SBK+17].

## A.8 Blum-Blum-Shub (BBSG)

The name refers to [BBS86]. Here $p, q \equiv 3 \pmod 4$ are primes and $n = pq$ is their product. We set an initial seed $s_0$ and iterate $s_i \leftarrow s_{i-1}^2 \pmod n$ for $i \geq 1$. The least significant bit $x_i \quad s_i \bmod 2$ forms the random sequence.

The input parameters are fixed in the code; each $p, q$ is 512 bits, and $s$ is 509 bits.

**Weaknesses.** Since prime generation is slow and knowledge of the factorization, the composite $n$ is often a fixed system parameter. Such a generator allows access by those who know the secret factors $p$ and $q$ of $n$.

A backdoor of this type was included in the SP 800-90A random bit generator specification for a while[6]. The `Dual_EC_DRBG` method was based on the public key cryptography of elliptic curves.

The quadratic residuosity and factoring problems underlying BBS are, of course, vulnerable to Shor's quantum algorithm [Sho94]. Hence BBSG is not quantum-safe.

[5] An apparent reference to "FIPS Publication 186-2 (with Change Notice 1)", dated October 5, 2001. https://csrc.nist.gov/csrc/media/publications/fips/186/2/archive/2001-10-05/documents/fips186-2-change1.pdf

[6] Press release April 21, 2014, "NIST Removes Cryptography Algorithm from Random Number Generator Recommendations." https://www.nist.gov/news-events/news/2014/04/nist-removes-cryptography-algorithm-random-number-generator-recommendations

## A.9 Micali-Schnorr Generator (MSG)

The name refers to [MS88, MS91]. The MSG generator is not described in detail in [RSN+10, Appendix D.9]. The STS 2.1.2 code has fixed 512-bit prime factors $p$ and $q$, and sets the exponent as $e = 11$. A fixed 186-bit seed $x_0$ is set as well. The output sequence consists 837-bit lower-order chunks $z_i$, while the high 187 bits are used for the next exponentiation.

$$\begin{aligned} y_i &\quad x_{i-1}^{e} \bmod n \\ x_i &\quad \lfloor y_i / 2^{837} \rfloor \\ z_i &\quad y_i \bmod 2^{837} \end{aligned}$$

**Weaknesses.** The security of the Micali-Schnorr scheme relies on the difficulty of factoring $n$ and some additional statistical assumptions. It is vulnerable to Shor's quantum computing attack. The generator requires modular exponentiation and is therefore relatively slow.

A Micali-Schnorr generator `MS_DRBG` was proposed for standardization alongside the backdoored `Dual_EC_DRBG`. It is included in ISO 18031 [ISO11]. The standard text includes only the "default" composite moduli $n$, not their factors $p, q$. These unknown factors are likely to form a backdoor for inverting the random number generator.

## A.10 Bonus: Algebraic Irrational Numbers

Appendix F.3 of SP 800-22 [RSN+10] and the STS 2.1.2 test suite discuss binary expansions of four irrational numbers; constants $e$ and $\pi$, $\sqrt{2}$, $\sqrt{3}$. It is not explained why these particular numbers are included. From the perspective of "randomness testing", some of them have distinguishing properties.

**Weaknesses.** While all of these numbers are irrational (not expressible as a fraction), only $e$ and $\pi$ are transcendental numbers. The square roots $\sqrt{2}$ and $\sqrt{3}$ are algebraic numbers (roots of integer polynomials), and hence their automatic identification and distinguishing from random is relatively easy[7].

Algorithms based on the LLL (Lenstra-Lenstra-Lovász [LLL82]) method can rapidly find "minimal" polynomial coefficients from the binary expansion (of a part) of an algebraic number. Standard computer algebra systems readily provide access to these methods; they are well known to cryptanalysts. Hence algebraic number binary expansions should not be considered indistinguishable from random, and the process of computing them is not a "one-way function" either.

# Bibliography

[BBS86] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Comput.*, 15(2):364–383, 1986. doi:10.1137/0215025.

[BK15] Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators. NIST Special Publication SP 800-90A Revision 1, June 2015. doi:10.6028/NIST.SP.800-90Ar1.

[BKR+21] Elaine Barker, John Kelsey, Allen Roginsky, Meltem Sönmez Turan, Darryl Buller, and Aaron Kaufer. Recommendation for random bit generator (RBG) constructions. Draft NIST Special Publication SP 800-90C, March 2021.

[7] Many thanks to Sophie Schmieg for a helpful discussion about efficient solutions to this problem.

[BR11] Elaine Barker and Allen Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. NIST Special Publication 800-131A, January 2011. doi:10.6028/NIST.SP.800-131A.

[FHK$^+$88] Alan M. Frieze, Johan Hastad, Ravi Kannan, Jeffrey C. Lagarias, and Adi Shamir. Reconstructing truncated integer variables satisfying linear congruences. *SIAM J. Comput.*, 17(2):262–280, apr 1988. doi:10.1137/0217016.

[Fis96] George Fishman. *Monte Carlo*. Springer, 1996. doi:10.1007/978-1-4757-2553-7.

[ISO11] ISO. Information technology– security techniques – random bit generation. Standard ISO/IEC 18031:2011, International Organization for Standardization, 2011. URL: https://www.iso.org/standard/54945.html.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. doi:10.1201/b17668.

[KS11] Wolfgang Killmann and Werner Schindler. A proposal for: Functionality classes for random number generators. AIS 20 / AIS 31, Version 2.0, English Translation, BSI, September 2011. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.html.

[Leh51] Derrick H Lehmer. Mathematical methods in large-scale computing units. *Annu. Comput. Lab. Harvard Univ.*, 26:141–146, 1951.

[LLL82] H.W. jr. Lenstra, A.K. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982. doi:10.1007/BF01457454.

[MS88] Silvio Micali and Claus-Peter Schnorr. Efficient, perfect random number generators. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 173–198. Springer, 1988. doi:10.1007/0-387-34799-2_14.

[MS91] Silvio Micali and Claus-Peter Schnorr. Efficient, perfect polynomial random number generators. *J. Cryptol.*, 3(3):157–172, 1991. doi:10.1007/BF00196909.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. URL: http://cacr.uwaterloo.ca/hac/, doi:10.1201/9781439821916.

[NC21] NIST and CCCS. Implementation guidance for FIPS 140-3 and the cryptographic module validation program. CMVP, November 2021. URL: https://csrc.nist.gov/Projects/cryptographic-module-validation-program/fips-140-3-ig-announcements.

[NIA13a] NIAP. Clarification to the entropy documentation and assessment annex. National Information Assurance Partnership (NIAP), Common Criteria Evaluation and Validation Scheme, 2013.

[NIA13b] NIAP. Entropy submission and review process. National Information Assurance Partnership (NIAP), Common Criteria Evaluation and Validation Scheme, 2013.

[NIS19] NIST. Security requirements for cryptographic modules. Federal Information Processing Standards Publication FIPS 140-3, March 2019. doi:10.6028/NIST.FIPS.140-3.

[RSN$^+$10] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, JamesDray, and San Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication SP 800-22 Revision 1a, April 2010. doi:10.6028/NIST.SP.800-22r1a.

[SBK$^+$17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 570–596. Springer, 2017. doi:10.1007/978-3-319-63688-7_19.

[SCA12] SCA. Randomness test specification. Cryptography Industry Standard of the P.R. China GM/T 0005-2012, March 2012.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE, 1994. URL: https://arxiv.org/abs/quant-ph/9508027, doi:10.1109/SFCS.1994.365700.

[TBK$^+$18] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, and Mike Boyle. Recommendation for the entropy sources used for random bit generation. NIST Special Publication SP 800-90B, January 2018. doi:10.6028/NIST.SP.800-90B.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005. doi:10.1007/11535218_2.

## 3. Comments from Mario Lamberger, February 14, 2022

Dear NIST crypto team.

Technically, my colleagues from the entropy working group (Markku Saarinen and David Johnston) have summed up the arguments brilliantly and we just want to endorse their view:

https://mjos.fi/doc/saarinen_sts1a.pdf

https://cmuserforum.onlyoffice.com/Products/Files/DocEditor.aspx?fileid=7548477

We would like to see a much clearer disclaimer in the update to SP800-22 on what these tests actually accomplish for the user, in which cases they should be applied and to set the SP800-22 tests in perspective to SP800-90A, B, C.

Based on our experience with a large percentage of our customers, the SP800-22 suite of tests is still used by these customers as a simple PASS/FAIL criterion for entropy sources.

This is still representative of the anachronistic approach to PTRNG design based on “combining some easily available noisy circuits” and then sending the output data through some test suite as the final stamp of approval.

Since then, thankfully a lot of standards have been published which force the designers that they are able to explain (through stochastic modeling) the amount of noise in their entropy source and the associated min-entropy,

which is then combined with proven and well established cryptographic conditioning and DRBG constructions.

With kind regards,

Mario Lamberger

## 4. Comments from Jonas Almlöf, February 17, 2022

Dear review board,

With regard to our earlier discussions on adding new tests for non-randomness, I hereby re-acknowledge our wish to contribute to the NIST test suite with an additional test developed at Ericsson AB.

As mentioned in the email conversation from Nov-Dec 2021, our contribution is best represented in the full text located at http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-298980

Kind regards,

Jonas