This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security

On the unbearable lightness of FIPS 140-2 randomness tests

Darren Hurley-Smith[1], Constantinos Patsakis[2] (*Member, IEEE*), and Julio Hernandez-Castro[3]

***Abstract*—Random number generation is critical to many applications. Gaming, gambling, and particularly cryptography all require random numbers that are uniform and unpredictable. For testing whether supposedly random sources feature particular characteristics commonly found in random sequences, batteries of statistical tests are used. These are fundamental tools in the evaluation of random number generators and form part of the pathway to certification of secure systems implementing them. Although there have been previous studies into this subject [1], RNG manufacturers and vendors continue to use statistical tests known to be of dubious reliability, in their RNG verification processes.**

Our research shows that FIPS-140-2 cannot identify adversarial biases effectively, even very primitive ones. Concretely, this work illustrates the inability of the FIPS 140 family of tests to detect bias in three obviously flawed PRNGs. Deprecated by official standards, these tests are nevertheless still widely used, for example in hardware-level self-test schemes incorporated into the design of many True RNGs (TRNGs). They are also popular with engineers and cryptographers for quickly assessing the randomness characteristics of security primitives and protocols, and even with manufacturers aiming to market the randomness features of their products to potential customers. In the following, we present three *biased-by-design* RNGs to show in explicit detail how simple, glaringly obvious biases are not detected by any of the FIPS 140-2 tests. One of these RNGs is backdoored, leaking key material, while others suffer from significantly reduced unpredictability in their output sequences. To make our point even more straightforward, we show how files containing images can also fool the FIPS 140 family of tests. We end with a discussion on the security issues affecting an interesting and active project to create a randomness beacon. Their authors only tested the quality of their randomness with the FIPS 140 family of tests, and we will show how this has led them to produce predictable output that, albeit passing FIPS fails other randomness tests quite catastrophically.

I. Introduction

Random numbers are at the foundation of secure cryptography. From keys to authentication challenges, from nonces to initialisation vectors, crypto-systems and the secure infrastructures they underpin heavily rely on Random Number Generators (RNGs). As one of the most fundamental building blocks of any modern crypto-system, RNGs must be trustworthy, unbiased and unpredictable.

RNGs can be implemented in software or hardware. Software-implemented RNGs are called pseudo-random number generators (PRNGs) and are, fundamentally, deterministic. Non-deterministic RNGs, also known as True Random Number Generators (TRNGs), typically use physical sources of entropy. Both are employed in a variety of security-critical applications.

Dodis et al. [2] demonstrate that testing for *sufficient entropy* cannot ensure that sources of randomness are appropriate for use in cryptography. Secret-sharing, bit-commitment, encryption and zero-knowledge proofs are shown to be compromised by one or more parties using imperfectly random values.

Statistical tests of randomness attempt to identify *non-randomness* in a tested sequence [3]. While it is not technically possible to prove randomness beyond a shadow of a doubt, it is certainly feasible to collect enough statistical evidence to show that an RNG does not behave randomly. Marsaglia's [4] Diehard tests were one of the first examples of software implemented RNG tests. The National Institute of Standards and Technology (NIST) proposed a battery of tests, formalised in SP800-22 [5]. These two batteries were later consolidated into the Dieharder battery [6]. L'Ecuyer et al.'s [7] TestU01 is not as widely used as Dieharder and NIST SP800-22, but is nevertheless a powerful and very stringent set of tests that currently represent the state of the art in the area.

Motivation. Lightweight, hardware-friendly tests are used as health-checks in many TRNGs. Lee et al. [8] proposed an efficient hardware-implementation of the full Federal Information Processing Standard (FIPS) 140-2 randomness tests, while Suresh et al. [9] provided an on-chip reduced set of NIST 800-22 randomness suite. In fact, major manufacturers promote the FIPS 140-2 compliance of their products as a selling point and a proof of high security standards[1]. Some manufacturers have chosen to use a subset of FIPS 140-2, making the study of the limitations of this battery particularly relevant. These health-checks, also called online tests, require fast lightweight algorithms that do not significantly impact on the output speed of the hardware device. As a consequence of these requirements, the study of hardware-implemented statistical tests is a thriving area of research. For example, Hotoleanu et al. [10] discuss which of the NIST SP800-22 tests can be efficiently implemented in hardware, finding that 7 of the 15 tests are not suitable for lightweight hardware implementation.

Adversarial analysis of statistical tests can improve our general knowledge of randomness and testing, but also help modelling a range of realistic attack vectors [11]. Hardware trojans, implemented at the time of manufacture (by an appropriately well-provisioned agency), or created afterwards by enterprising (and malicious) resellers, pose a grave threat. By modelling biases that can pass statistical tests of randomness

[1]D. Hurley-Smith is with the Information Security Group, Royal Holloway University of London, TW20 0EX, Surrey, UK Darren.Hurley-Smith@rhul.ac.uk

[2]C. Patsakis is with the Department of Informatics, University of Piraeus, Greece and the Information Management Systems Institute of Athena Research Center, Greece. Email: kpatsak@unipi.gr

[3]J. Hernandez-Castro is with the School of Computing, University of Kent, Canterbury CT2 7NF, Kent, UK j.c.hernandez-castro@kent.ac.uk

[1]https://cloud.google.com/security/compliance/fips-140-2-validated/, https://aws.amazon.com/compliance/fips/, https://www.docker.com/docker-news-and-press/docker-awarded-fips-140-2-validation-nist

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security

and determining whether they have utility as a means of leaking privileged information (such as a hidden deterministic structure in the output of the RNG in question), we can come closer to understanding where the current gaps in randomness testing lie. Any attacker would ideally like that their trojanised generator pass all the common randomness tests batteries, with FIPS 140-2, SP800-22, and Dieharder being the main targets. FIPS 140-2 is the most common source of hardware self-tests, therefore the ability to bypass these is a requirement for compromised generators that attempt to seem legitimate to both internal and user-run statistical tests.

Main contributions. The main contribution of this work is to demonstrate that current methodologies for PRNG testing and verification, and in particular the FIPS 140-2 set of randomness tests, cannot detect a wide range of extremely poor generators and hence do not merit the continuous use and trust they receive from both the academic and the industrial communities. Many manufacturers continue to use the FIPS 140-2 tests as evidence of their compliance with the standards proposed by NIST, whether or not they have actually been certified by the organisations that establish those standards. We show this is sadly true even when confronted with unsophisticated biases. To prove this point, we define and analyse two extremely simple families of random generators (the σ-counter and ϵ-hole) and show that despite their simplicity they can fool the FIPS 140-2 tests. The ϵ-hole can also be implemented in a keyed format, wherein a key is leaked via the missing bytes in consecutive periods of the output sequence.

A slightly more sophisticated but also biased-by-design generator (the t-counter) is later introduced, which can remarkably avoid detection by FIPS 140-2 for any degree of bias. Sequences generated by a t-counter possess significantly lower entropy than would be expected of a true RNG, severely weakening any systems using their output.

To further shed light on the misleading results these tests can lead to, we compare their output on a real-world implemented RNG intended to provide randomness over the Internet, the *League of Entropy*[2] which uses the `drand` project as its basis. We are the first to report that this service, while claiming to offer random and unpredictable outputs, instead generates ones with a persistent and significant bias. This was unfortunately not detected by the developers, possibly because they only tested its output with FIPS 140-2.

To further demonstrate the limitations of the FIPS 140-2 battery of randomness tests, we finish by showing that we can bypass these tests with inputs that contain fully meaningful information, and hence are obviously non-random. Even more counter-intuitive is the fact that we have a great deal of freedom in choosing these contents. In particular, we show how images in the WEBP format can, typically after some simple transformations, successfully pass all the tests in the FIPS 140-2 battery.

Despite being deprecated FIPS 140-2 is still heavily in use. Non-experts may associate passing the FIPS 140-2 tests with a higher degree of randomness, or proof that the generator is appropriate for cryptographic use. This is clearly an erroneous assumption. Our motivation and why we believe it matters to focus primarily on FIPS 140-2, is to demonstrate exactly how easy it is to pass these tests and contribute to a wider academic and industrial awareness of their limitations. We hope works like this one, repeatedly exhibiting the many inadequacies of these tests can encourage the community to advance and adopt better practices.

Impact. Our work extends previous work by Becker et al. [1], regarding the failure of commonly used statistical tests of randomness as a means of detecting even primitive attempts to trojanise RNG output. We demonstrate how dangerous it is to trust sources of randomness implementing FIPS 140-2 as a means of hardware self-testing. This is achieved through rigorous statistical testing and comparison of sequences with known biases to a real-world trusted source. We achieve this by exploring extremely simple trojans, to identify the degree of technical sophistication required to successfully trojanise an RNG. While previous work focuses on the use of cryptographic algorithms or dopant-level hardware trojans, we identify that these tests can be fooled by significant less sophisticated trojans requiring little to no domain expertise.

Therefore, many sources of randomness currently in use, despite having credible certifications or even theoretical proofs backing their randomness, may be predictable at a minimal cost. For example, Dual-EC-DRBG has been shown to possess a backdoor which is difficult to detect with lightweight statistical tests, a fact discussed by Bernstein et al. [12]. Previous work undertaken by Checkoway et al. [13] demonstrates how this vulnerability can be exploited in TLS implementations, providing a practical way to compromise vital network security protocols.

Road map. The rest of the paper is organised as follows: In Section 2, we discuss generalities about the statistical testing of random number generators. Section 3 defines three biased-by-design generators, the σ-counter, t-counter, and ϵ-hole, with a backdoored variation of the latter. Their characteristics are outlined, followed by our experimental methodology. Section 4 discusses a real-world example of PRNGs. We conduct the first third-party analysis of the *League of Entropy* generator, identifying a consistent and significant bias, despite it successfully passing the FIPS 140-2 tests. Section 5 presents in more detail the methodology used to conduct these experiments. Section 6 provides the results of testing with FIPS 140-2 and `Ent` over all of the studied generators. Section 7 discusses the results in the context of validating statistical tests. Section 8 concludes the paper.

II. Statistical tests of randomness

In this section, two lightweight batteries of statistical tests (FIPS 140-1/2 and Ent) are discussed.

A. FIPS 140-1 and 140-2

The FIPS documents describe processes and algorithms related to civilian information technology security. FIPS 140-2 (security requirements for cryptographic modules) [14], is the successor of the deprecated FIPS 140-1 standard. It is

[2]https://leagueofentropy.com

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security

used to approve cryptographic modules in the U.S. FIPS 140-2, with errata added by NIST, continues to be an important element of many existing standards. Interestingly, the standard does not provide any description of secure processes for key generation. This lack can be detrimental for such certified products. Indeed, Cohney et al. [15] showed that hundreds of such certified products are vulnerable to attacks as, e.g. their manufacturers have hard-coded keys in their firmware.

FIPS 140-2 randomness tests are no longer officially supported, but they are still widely employed by many scientists and engineers around the world and, notably, they are still an important component of the largest battery of tests included in the Common Criteria evaluation of RNGs. Bundesamt für Sicherheit in der Informationstechnik (BSI) outlines the AIS-31 [16] evaluation methodology, which uses four of the FIPS 140-1 tests as part of its procedure *A offline tests*. As stated by Balasch et al. [17], BSI's methodology is considered to be the *de facto* European standard for evaluating RNGs.

The `rng-tools` module for Linux includes an implementation of FIPS 140-2 in `rngtest`. In addition, these tests are notably used as part of the random number generator daemon's (`rngd`) entropy-source procedure, checking 20,000-bit binary sequences. Although intended only as a test of the entropy being fed to the kernel entropy pool, many RNG developers still use them as evidence of unpredictability in their devices.

The test definitions outlined below, and may also be found in the NIST FIPS 140-2 [14] documentation, though they are no longer mandatory and have been struck out (but not replaced) in the most recent revision. FIPS 140-1's [18] less stringent set of parameters is also provided to show the developments made in this area.

Poker test. A 20,000-bit block is divided into 5,000 consecutive 4-bit chunks. The occurrences of each of the 16 possible chunks (15 degrees of freedom) are counted. With $f(i)$ representing the occurrences of each value, evaluate X as shown below. The FIPS 140-2 test is passed if $2.16 < X < 46.17$. FIPS 140-1 only requires that $1.03 < X < 57.4$. Both versions evaluate the following equation:

$$X = (16/5,000) \cdot \sum_{i=0}^{15} [f(i)]^2 - 5,000 \qquad (1)$$

Monobit test. This test simply counts the number of 1's in a 20,000 bit block. Let the number of 1's be X, FIPS 140-2 passes if $9725 < X < 10275$. FIPS 140-1 passes if $9,654 < X < 10,346$.

Length	FIPS 140-1	FIPS 140-2
1	2,267 - 2,733	2,343 - 2,657
2	1,079 - 1,421	1,135 - 1,365
3	502 - 748	542 - 708
4	223 - 402	251 - 373
5	90 - 223	111 - 201
6+	90 - 223	111 - 201

TABLE I: Runs intervals

Runs test. A run is defined as an unbroken sequence of 1's or 0's. Under *rngtest*, this test counts runs of length between 1 and 6. Table I specifies the passing intervals for both versions of the test.

Long run test. A long run is defined by FIPS 140-2 as any run of 1's or 0's larger than 25 bits. The probability of observing a 26-bit run is 0.000298 per block of 20,000 bits. If a long run occurs, the test is failed. FIPS 140-1 only fails this test if the number of identical consecutive values exceeds 34 bits.

Continuous run test. This test simply checks for consecutive identical 32-bit patterns. A 20,000-bit block fails this test if a consecutive repetition of any 32-bit sequence is detected.

FIPS 140-3's final draft[3] was published in March of 2019. FIPS 140-3 does not implement the FIPS 140-2 nor any other statistical tests. Instead, with regards to RNG testing, it focuses on entropy source modelling.
Despite their later redaction in FIPS 140-2 and removal from FIPS 140-3, these tests are frequently employed as a health check in RNGs by manufacturers. For example, Trezor [19] uses *rngtest* as part of their hardware cryptocurrency wallet evaluation process. Redhat [20] suggests that customers use *rngtest* to self-evaluate the quality of the randomness available for the generation of cryptographic keys. As we demonstrate in the following section, this is ill-advised.

B. Ent

`Ent` is a simple Linux utility created by John Walker [21]. It provides six output statistics: entropy, compression, χ^2, serial correlation, arithmetic mean and Monte-carlo estimated value for π. Unlike FIPS 140-2, it is not part of any official RNG evaluation scheme, but it has been used successfully to identify flaws in RFID card key generators, particularly the DESFire EV1 [22] and Mifare Classic. This makes it useful as a tool for cursory investigation of data, though it is unsuitable for the evaluation of cryptographic RNGs.

III. Constructing Biased Generators: the ϵ-hole, σ-counter & t-counter

Most current randomness tests focus on measuring either uniformity or independence, very few are capable of analysing both simultaneously. This leads to issues. A case in point are counters, which maximise uniformity but are horrendous in terms of independence so they can fool many tests only concerned by the flatness of the output. This inspired our σ-counter. On the other hand, tests specialised on measuring output independence can overlook if one value is significantly missing quite easily. This was our inspiration for the ϵ-hole generator. The t-counter is an attempt to hybridise these two approaches.

These RNGs are then tested extensively for different values of their parameter using FIPS 140-1 and FIPS 140-2. Additional analysis and insights are provided with the help of `Ent`.

The `/dev/urandom` PRNG is used as the source of random values for these biased generators. They all operate similarly: generating single bytes using `urandom` and then subject them to some conditional modification, as shown in the following sections. /dev/urandom) has been selected as it is a fast, efficient means of producing random numbers considered

[3]https://csrc.nist.gov/publications/detail/fips/140/3/final

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security

suitable for use as key material in non-critical systems. It has been chosen over `/dev/random` because it is non-blocking; it does not stop when the entropy buffer of the operating system is exhausted, allowing us to collect sufficient data in a timely manner. As `urandom` sequences pass without any trouble far more stringent tests (TestU01 and SP800-22), they are considered a suitable control sample for these experiments.

It is interesting to highlight that even unsophisticated approaches (such as the ones discussed below) offer a means of leaking information that could include keys, system state, or even exfiltrated files while passing many common tests of randomness. *AES* in counter mode will also pass these tests, but more insights is gained from studying which are the minimal, most simplistic types of flawed generators still able to fool these tests. Moreover, showing how these biased-by-design generators fool FIPS 140-2 will probably have a more impactful effect in spreading the message that these widely used generators are, for all intents and purposes, meaningless. For raising awareness, showing how these deeply flawed generators can bypass FIPS is more relevant than showing that *AES* in counter mode would do.

A. ϵ-hole

We can interpret ϵ as the degree to which an ϵ-hole is biased, and we write Eh_ϵ. X_i represents the values the Eh_ϵ generator outputs, R_i is a truly random byte (i.e. from a TRNG) and can be seen as the ϵ-hole input. The following expression defines this generator where b represents any fixed byte value, which in our experiments has been set to $0xff$:

$$X_{\mathrm{i}} = \begin{cases} R_i - \{b\}, & \text{with probability } \epsilon \\ R_i, & \text{with probability } 1-\epsilon \end{cases}$$

The intuitive concept of an ϵ-hole is very simple: It behaves identically to a random number generator with probability $1-\epsilon$, but with probability ϵ it suppresses a given value (in this paper 255). This means that with probability roughly $\frac{\epsilon}{256}$ the ϵ-hole will discard its input and output instead the first value in the sequence $R_{j+1}, R_{j+2}, ...$ which is different from 255. This makes Eh_ϵ produce an output that is seemingly random, particularly for small values of ϵ, but that in a histogram can be observed to produce slightly fewer values of the 255^{th} byte. The idea behind the Eh_ϵ is to measure how sensitive randomness tests are to partially missing values. The name for this transformation comes from the fact that for $\epsilon = 1$, the value 255 is never generated, hence creating a hole in the corresponding histogram.

Based in the definition above, the entropy of an epsilon-hole is: $H \approx 8(1-\epsilon) + \epsilon \log_2(255) \approx 8 - 0.0056465631\epsilon$

As a result, the actual entropy of an ϵ-hole varies from a maximum of 8 (for $\epsilon = 0$) to a minimum of $\log_2(255) \approx 7.994353$ (for $\epsilon = 1$) bits per byte.

a) Keyed ϵ-hole: A keyed version of this generator has been implemented to show how keys can be easily leaked using the output stream of a PRNG that still passes FIPS 140-2. By changing the 'missing' byte (set previously to $0xff$) to consecutive values in the series of bytes to leak, it is possible to perform a simple frequency analysis of 'missing' bytes and identify the secret information, all without FIPS noticing anything untoward. We have tested this idea by leaking 16-byte keys and the approach works very reliably and is unnoticed by the tests if we change this missing value to the next one in the key every 2048 outputs.

A randomly generated 16-byte value (analogous to a key) was embedded sequentially in output generated by a keyed ϵ-hole, and a χ^2 test was used to perform a frequency analysis of byte value. This allowed us to observe which values were absent or significantly less frequent than expected. By mapping these to a 16-byte repeating sequence we could identify the original embedded key. One-hundred 1MB sequences were generated using this backdoored RNG approach, each leading to a successful recovery. Listing 1 shows a Python script demonstrating such a generator.

```
def epsilon_hole(s_size, key, period):
  sequence = []
  random.seed(key)
  c = 0

  for i in range(0, int(s_size/period)):
      for j in range(0, period):
          temp = random.randint(0, 255)
          if temp == key[c]:
              while temp == key[c]:
                  temp = random.randint(0,
                   ↪  255)
          sequence.append(temp)

      if c >= len(key)-1:
          c = 0
      else:
          c += 1

  return sequence
```

Listing 1: A Python function for creating an ϵ-hole.

B. σ-counter

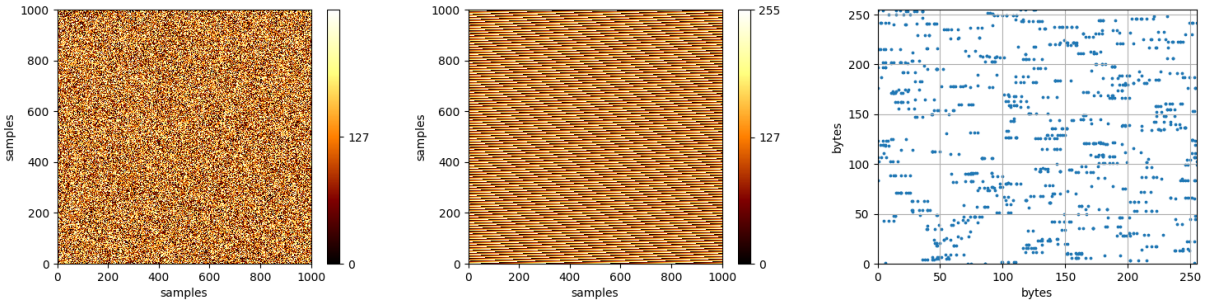
The value of the parameter σ can, also in this case, be intuitively interpreted as the degree to which this RNG is biased. A counter c is initialised to 0 when the RNG starts, and increased after each output. The definition of a σ-counter is then:

$$X_{\mathrm{i}} = \begin{cases} c \mod 256, & \text{with probability } \sigma \\ R_i, & \text{with probability } 1-\sigma \end{cases}$$

When the value in $c \mod 256$ is outputted, c is incremented. When $\sigma = 1$ the output will simply consist of a counter cycling upwards from 0 to 255 repeatedly, and hence its name. The intuitive idea behind the σ-counter is to explore how well randomness tests react to sequences that can exhibit, simultaneously, an almost perfectly uniform distribution but also a very high correlation, both characteristics of a counter.

The entropy of a sigma counter can be calculated based on its definition as $H \approx 8(1-\sigma) + 0\sigma = 8 - 8\sigma$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security



(a) $\epsilon = 1$ (b) $\sigma = 1$ (c) $t = 1$

Fig. 1: Visual representation of maximally biased sequences for σ, ϵ and t

This can take any value from 0 (for $\sigma = 1$) to 8 (for $\sigma = 0$) bits per byte.

C. t-counter

The previously discussed biased generators are relatively trivial examples. They possess a simplistic and unidimensional bias that is unlikely to pass the most rigorous inspection. However, they can deliver surprising results under less stringent analysis, as we will see. The t-counter is slightly more sophisticated.

The t-counter produces either a random number, with probability $1 - t$ or a value taken from the AES [23] S-box S indicated by index y. The value y is computed as $y = i + [-n, ..., 2n] \mod 256$, where i is the current index and n is a pre-defined value. This results in a locally random selection of values close to the target index; It is basically an attempt to obfuscate the bias of the generator without excessively reducing the predictability of the sequence.

$$X_{\mathrm{i}} = \begin{cases} S_y, & \text{with probability } t \\ R_i, & \text{with probability } 1 - t \end{cases}$$

In this work, the values $n \in \{4, 6, 8, 10, 16\}$ are investigated. Fig. 1 (c) provides a visualisation of a t-counter with varying t and $n = 4$, generated using the Infinite Noise test scripts [24]. This scatter plot is the result of taking consecutive, non-overlapping pairs of bytes from the extracted sequence, and using the bytes as X and Y coordinates respectively. Patterns begin to emerge only when $t = 0.5$ and become steadily more visible as t approaches 1.

The associated entropy of a t-n-counter is approximately $8(1 - t) + \log_2(3n)t = 8 - 8t + \log_2(3n)t$ bits per byte.

D. Initial observations

Fig. 1 provides a visualisation of the ϵ-hole (a), σ-counter (b), and t-counter (c) generated using the Infinite Noise scripts [24]. The period of the σ-counter can be observed clearly as can the underlying structure of $t = 1$ when $n = 4$. The ϵ generator is indistinguishable from noise even for this high value of ϵ. The purpose of this visualisation is to set initial expectations of how these generators should perform in randomness tests; if we can visually perceive the biases, surely the tests must provide a far more damning indication of their non-randomness. Unfortunately, this seems not to be always the case, at least with FIPS 140-2.

IV. League of Entropy

As the project claims[4]:

> "*The League of Entropy is a collaborative project between the founding members Cloudflare, École Polytechnique Fédérale de Lausanne, Kudelski Security, Protocol Labs, and University of Chile to provide a verifiable, decentralized randomness beacon. A decentralized randomness beacon combines randomness from multiple independent high entropy sources to generate a truly unbiased random number for anyone that may need a public source of randomness.*"

The project is based on `drand`, a distributed randomness beacon first reported by Syta et al. [25]. The core concept is that it operates in a "chained" mode so that each newly generated value is linked with the previous one. To this end, each time we want to produce a new value, we create a message m of the form $m = H(r||\sigma_{r-1})$, where r is an integer denoting the current round, H is a secure hash function, and σ_{r-1} is the previously produced output. To create the new random output we compute the BLS threshold signature $\sigma_r(m)$ proposed by Boneh et al. [26], therefore, every new generated random value depends on all the previous ones. Moreover, the use of BLS threshold signatures guarantees that no party can individually generate any values, nor manipulate them. To convert standard BLS signatures into threshold signatures Syta et al. exploit the secret sharing scheme of Feldman [27], using the Pedersen distributed key generation method [28]. As a result, there is no trusted third party to handle the shares of the involved parties, and the signature can only be made if at least τ parties cooperate to sign a message.

V. Results

In this section, we present the results of our statistical testing. The ability of statistical tests to detect existing and known biases is the focal point of this work.

[4]https://www.cloudflare.com/leagueofentropy/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security

All code associated with the biased generators presented in this section may be found in a BitBucket repository [5].

To provide an unbiased sample against which we could compare the three biased generators (σ, ϵ, t), 100 binary files of 2.5MB were generated using `urandom`. For the ϵ-hole and σ-counter, we generated 100 files of $2.5 \cdot 10^6$ bytes for every value of ϵ and σ in the interval [0,1) in increments of 0.01. This resulted in 9,900 files for each and a total of 19,900 files to analyse, including the `urandom` samples.

A. FIPS 140-1

FIPS 140-1 was used to test each file for 500 iterations, with each iteration requiring 20,000 bits of output. The Monobit, Poker, Run and Long-run tests were executed using the default parameters outlined in the FIPS 140-1 guidelines [18].

The total number of test failures for FIPS-140-1 is evaluated with a confidence threshold of 0.01. Individual test failures are analysed to characterise the tested sequences. It is unwise to reject a generator if it fails the tests once or twice, but consecutive sequences repeatedly failing the same test indicate systemic issues.

Fig. 2 shows the average failure rate for the Poker test out of 10 iterations of 500 runs each, for each value of σ in the interval [0,1] in increments of 0.1. Only the σ-counter is shown, as the ϵ-hole and t-counter pass all tests. We only show the Poker test in this graph because all other tests pass for all values of σ.

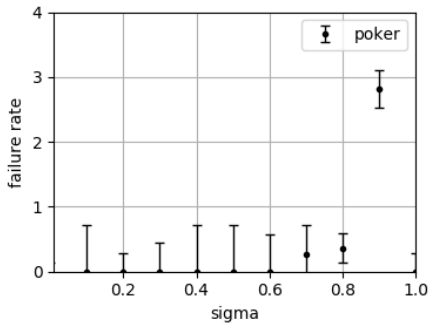


Fig. 2: σ-counter FIPS 140-1 poker test results

It is only at values of $\sigma \geq 0.9$ that one observes failures, and then only on the Poker test but the test is passed as the number of failures is very low. All other tests are passed. Compared with the results for FIPS 140-2, this demonstrates that the FIPS 140-1 requirements are too lax, allowing biased data to pass as if it was random. The t-counter passes this test for all tested values of n. Furthermore, the League outputs systematically pass these tests.

B. FIPS 140-2

FIPS 140-2 provides the same tests as 140-1, but with updated, more stringent pass conditions. For FIPS 140-2 evaluation, each file was tested using `rngtest` for 1,000 iterations. Each iteration takes 20,000 bits of input and uses the FIPS 140-2 Monobit, Poker, Run, Long Run and Continuous Run tests. We maintain the $a = 0.01$ confidence threshold (less than 1% of tests may fail) used to evaluate FIPS-140-1, for our analysis of FIPS-140-2.

Fig. 3 shows the mean FIPS 140-2 pass rate for the ϵ-hole (a), σ-counter (b), and t-counter (c). The red channel represents the standard deviation of `urandom` around its mean pass rate, to provide a direct comparison with unbiased data. In graph (b), this channel is very close to 1000 so is not easily distinguished. A snapshot offering greater detail can be seen in Fig. 4 (a).

The ϵ-hole passes `rngtest` (and hence FIPS 140-2) for all values of ϵ, despite being trivially non-random. Some individual FIPS 140-2 tests are failed, but these failures are incidental and non-consecutive, consistent with false negatives coming from a truly random source. There is no discernible pattern to the distribution of test failures. No value of ϵ fails more than 7 tests in total (out of 1,000), meaning it never exceeds our confidence threshold of 0.01 (that will need more than 10 failures).

The keyed ϵ-hole performs even better on these tests, failing a maximum of 4 tests when $\epsilon = 1$. This is due to the periodic change in the missing byte. The period selected for this shift was 2048 bytes, as 100% probability of recovering the secret values was not possible with smaller periods. This result demonstrates that ϵ-holes can easily be used to hide information recoverable by malicious actors by simple observation and analysis of the output stream while passing online tests of randomness.

The σ-counter consistently passes FIPS 140-2 tests within our confidence threshold, up until values around $\sigma = 0.6$. At this point, `rngtest` fails for σ in [0.59,...,0.65] before recovering for σ in [0.66,...,0.68]. From this point onward *rngtest* fails for all values of σ. Interestingly, the distribution of test failures is far from random. The Poker test is consistently failed for the σ values in the above-mentioned parameter intervals.

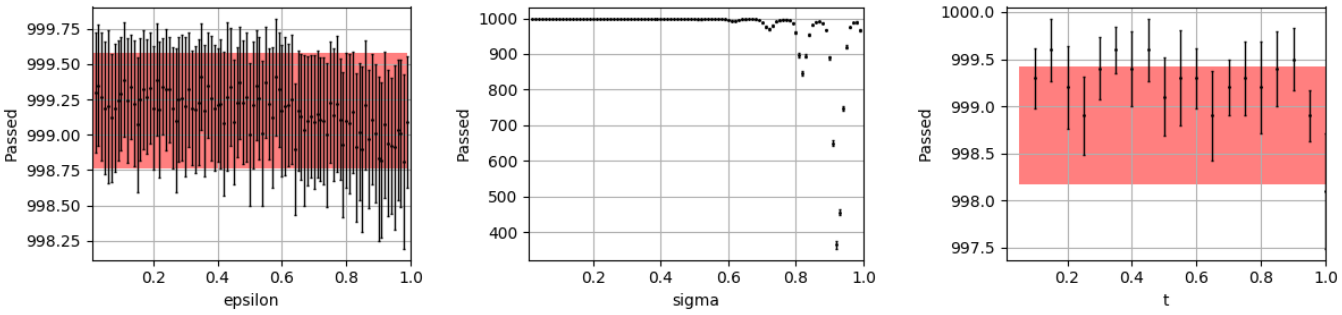
The t-counter results, observable in Figure 3 (c), show performance almost equivalent to the ϵ-hole for $n = 4$. This is the lowest value of n tested in these experiments, and it passes FIPS 140-2 for all values of t. When $t = 1$, the pass rate drops below that observed for `urandom`, but is still well within the 0.01 confidence interval.

It is quite surprising that the ϵ-hole can pass FIPS 140-2 tests regardless of the value of ϵ, i.e. of the degree of bias of its output. However, the σ-counter requires a further, more nuanced, analysis. Fig. 4 provides a closer look at the first area of interest from Fig. 3 (b). The t-counter passes all these tests.

Fig. 4 (a) shows the total pass rate for σ in $(0.5, \ldots, 0.75)$. The trend towards failure, with an unexpected recovery around $\sigma = 0.66$, is more evident at this scale. It is clear that this RNG does not pass the FIPS 140-2 tests in this interval, but is interesting to note it is only failing the Poker test to any significant degree. It is also clear that if the Poker test was not included, the flaw in this generator would not be identified

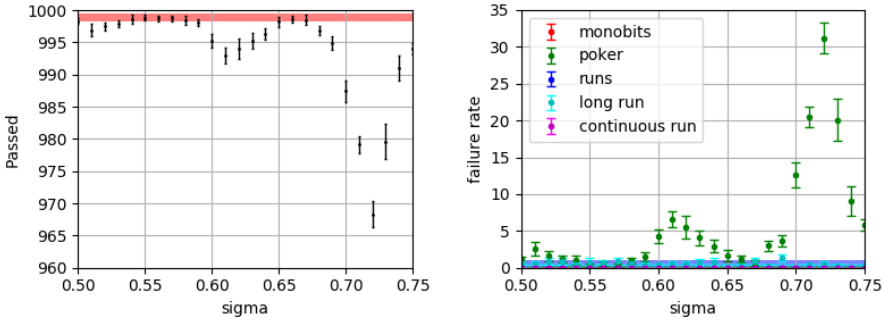
[5]https://highly_entropic@bitbucket.org/highly_entropic/dismantling_fips140-2

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security



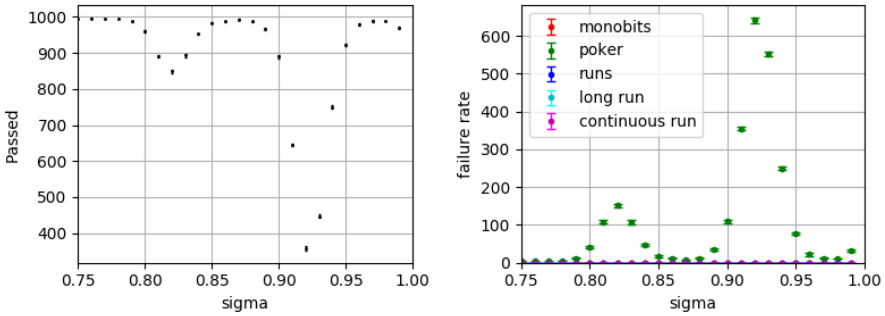
(a) ϵ-hole (b) σ-counter (c) t-counter (n=4)

Fig. 3: FIPS 140-2 aggregated test results



(a) FIPS-140-2 total pass rate (b) Individual FIPS-140-2 test results

Fig. 4: FIPS 140-2 results for σ =[0.5,0.75]



(a) FIPS-140-2 total pass rate (b) Individual FIPS-140-2 test results

Fig. 5: FIPS-140-2 detailed results for σ =[0.75,1.0]

at all. This makes sense because the Poker test is especially sensitive to the correlation between consecutive output bytes.

Fig. 4 (b) shows the failure rates for each test. As previously discussed, the Poker test can be seen to deviate significantly from the failure rate of the other tests. The coloured channels towards the bottom of the y-axis represent the standard deviation of failure rates for `urandom`, for comparison.

Fig. 5 focuses on the aggregated (a) and individual (b) test results for σ in $(0.75, \ldots, 1)$.

The unexpected recovery of the Poker test between periods of catastrophic failure is intriguing and, as yet, unexplained. This pattern also occurs in individual test results.

The ϵ-hole highlights the very real possibility of failing to recognise obvious biases despite employing FIPS 140-2 to validate RNG output. As we have seen, Eh_ϵ passes these tests with ease for any value of ϵ.

The σ-counter is spotted by the FIPS 140-2 Poker test. For $\sigma \leq 0.5$ it is only marginally worse than `urandom` or the ϵ-hole. The confidence threshold of 0.01 is not crossed until $\sigma \geq 0.59$. The Poker test operates on nybbles (4-bit sequences), testing the distribution of bits for 15 degrees of freedom. As a result, it can identify σ when its value rises

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security

to sufficiently bias the appearance of specific nybbles. The recovery at $\sigma => 0.95$ is explained by the return to an equal distribution as the counter reasserts itself, no longer acting to create duplicates of *random* bytes in the sequence. The Runs, Long Run, and Continuous Run tests will identify sufficient (though less dramatic) numbers of over-length bit-sequences to fail above the threshold $a = 0.1$. The ϵ-hole and t-counter are not detected, because they do not substantially alter the distribution of bits in nybbles, nor do they significantly modify the distribution of n-bit runs within the sequence.

C. Ent

The `Ent` tool was used to test each generator using the Approximate Entropy, χ^2, and Serial Correlation tests. Each output was tested for its full length of $2.5 \cdot 10^6$ bytes. A t-counter with parameters $n = 16$ and $t = 1$ was tested to allow for a reasoned comparison with the σ and ϵ generators.

Due to its simplicity and robustness, we use `ent` in this work to visualise and understand better the properties of the binary sequences under investigation.

Fig. 6 shows the results of the Serial Correlation test for ϵ-holes (a), σ-counters (b), and t-counters (c). The red channel represents the standard deviation plotted as error around the mean of `urandom` results, to indicate what a real random result should look like. This channel is highly compressed for (b), due to the scale of the y-axis.

The ϵ-hole once more proves difficult to distinguish from a random output, as it passes the serial correlation test with no issues. There is no discernible pattern in the graph, indicating that increasing values of ϵ have almost no effect on the test results. Removing a single byte-value from the stream has a minimal impact on the predictability of the sequence according to this test.

The σ-counter is, as expected, easily identified. It shows a steadily increasing Serial Correlation for larger values of σ. Even small values of σ (> 0.03) cause this test to fail. As a result, the serial correlation test is able to detect the σ-counter even when the FIPS 140-2 tests cannot. This should be a cause for further concern regarding FIPS 140-2.

The t-counter is identified almost as easily as the σ counter. The rapid decrease in $0xff$ bytes causes a rapid divergence from the parameters established using `urandom`. By $t = 0.4$, this generator consistently fails the serial correlation test. Further results in the appendices show that increasing n has a marginal, but positive, impact on the serial correlation results. The t-counter is, however, still detectable using this test.

Fig. 7 shows the χ^2 results for the ϵ-hole (a), σ-counter (b), and t-counter (c). The left y-axis shows the χ^2 score (log-scale for (b)). The right y-axis shows the corresponding p-value. The red channel is the standard deviation of χ^2 scores for 100 `urandom` files, plotted around their mean. This provides a useful comparison.

Both biased RNGs perform quite poorly on this test. The ϵ-hole rapidly approaches ($\epsilon \approx 0.2$) χ^2 scores on the order of 10^3, and reaches just short of 10^4 for $\epsilon = 0.99$. By $\epsilon \approx 0.09$ the p-value of the test is so low ($p < 0.01$) that the ϵ-hole fails.

The σ-counter actually decreases its χ^2 score as σ approaches 1. This is slightly counter-intuitive but not unexpected. It performs slightly better than the ϵ-hole, requiring values of $\sigma \approx 0.18$ before the p-value of the mean exceeds 0.99.

The t-counter passes the χ^2 test for all values of t when $n = 4$. Other results for greater values of n can be found in appendix A, where one can observe that increasing n brings the χ^2 results closer in line with `urandom` results.

The main limitation of the χ^2 test is that 20,000 bits of output are generally not sufficient to detect the ϵ-hole. We need to use 2.5MB of data (equivalent to the 1,000 blocks of 20,000 bits tested by `rngtest`) to achieve these results. A continuous, hardware implemented test, will generally have to run over significantly less output.

Fig. 8 shows the results of the approximate entropy test for the ϵ-hole (a), σ-counter (b), and t-counter (c).

The ϵ-hole shows an inverse correlation between entropy and ϵ. As ϵ increases, entropy decreases, but not by more than $5 \cdot 10^{-3}$. `urandom` provides a mean entropy of approximately 7.99992, whilst the mean entropy for $\epsilon = 0.99$ is approximately 7.9946.

AIS-31 [16] specifies an entropy of at least 0.997 per bit, which is far exceeded by both σ and t-counters, even for their worst parameter values. Values of ϵ above 0.8, however, fall below this guideline.

The σ-counter shows a direct correlation between entropy and σ values. As σ approaches 1, entropy approaches the maximum value of 8 bits per byte.

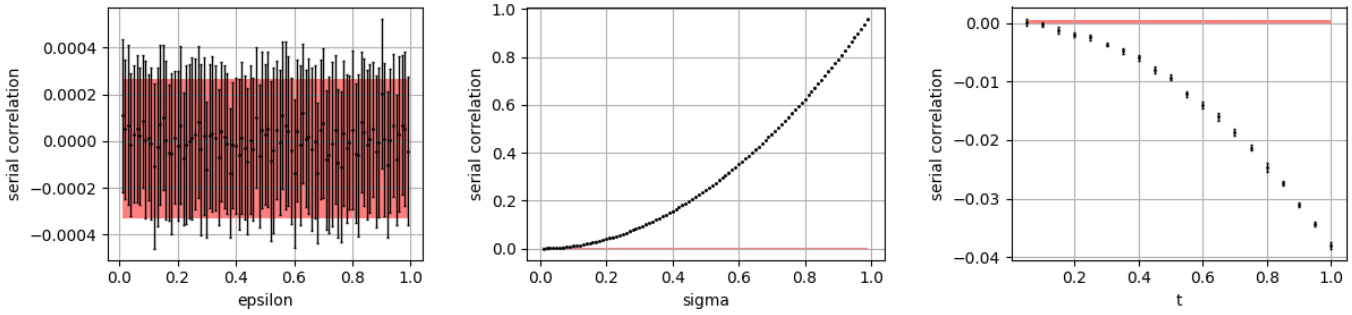
These results are a direct counterexample to the intuitive notion that higher Shannon entropy implies better randomness. For example, the sigma counter *increases* its entropy with higher σ values. It is important to keep in mind that multiple entropy tests exist, and any single one may not provide a definitive measure of the real entropy of a sequence.

Keyed ϵ-holes fail the χ^2 test significantly (with a score of 503.09), but pass all other `Ent` tests (serial correlation is $2.2 \cdot 10^{-5}$ and entropy is 7.99964 bits per byte). These results are the mean of 100 1MB samples with $\epsilon = 1$. Notably, the χ^2 result is significantly lower than that observed for a standard ϵ-hole. This is due to the changing byte values omitted in each period, which reduces (but does not wholly eliminate) the under-occurrence of specific bytes. The seed or key embedded in the output stream will have a direct effect on this test result; keys without repetition (sequential or otherwise) of bytes will achieve slightly better χ^2 scores than those with repeated values.

D. League of Entropy

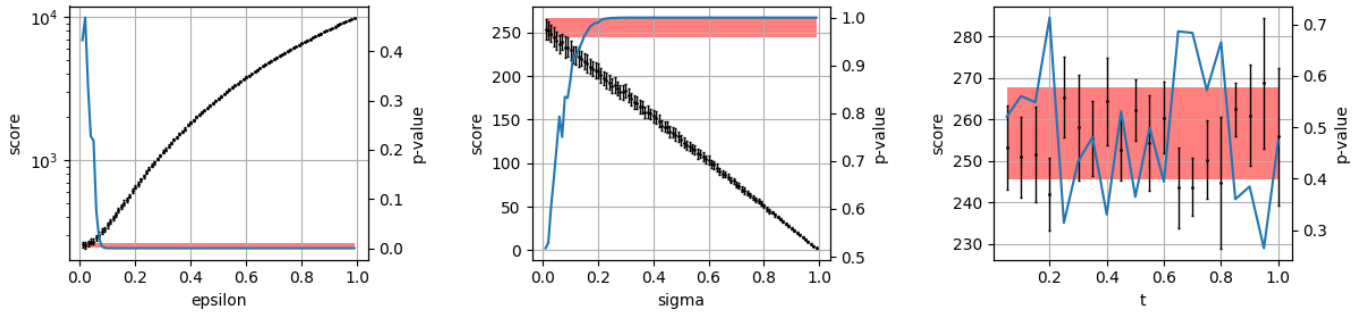
The *League of Entropy* is a publicly available generator based on elliptic curves. As this generator produces values very slowly, only a single file of 2.5MB has been tested at the time of writing. The *Ent* and FIPS 140-2 tests have been used over this file. Data collection from this source is an ongoing item of work.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security



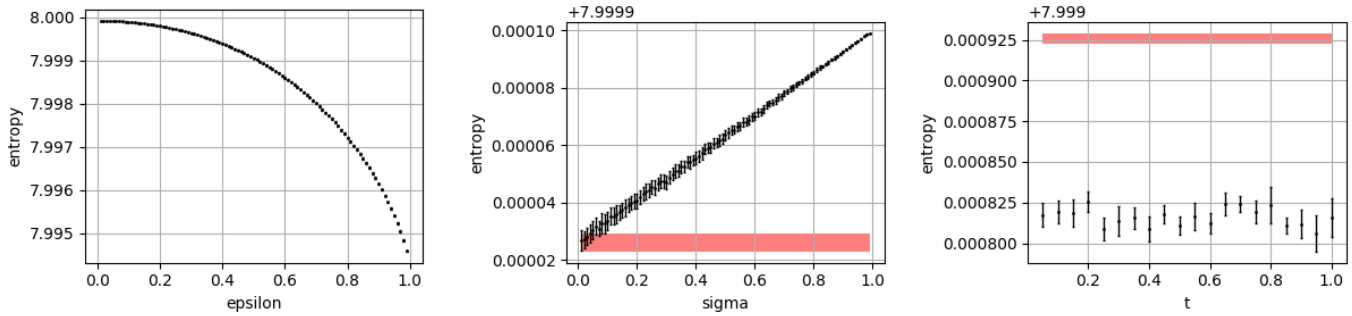
(a) ϵ-hole (b) σ-counter (c) t-counter (n=4)

Fig. 6: Serial correlation results



(a) ϵ-hole (b) σ-counter (c) t-counter (n=4)

Fig. 7: χ^2 results



(a) ϵ-hole (b) σ-counter (c) t-counter (n=4)

Fig. 8: Approximate entropy results

It passes[6] FIPS 140-2 without issues, but `ent` tells a different story. A model of the byte-level bias observed in *League* output has been developed to aid in further analysis of the issues observed.

Figure 9 shows the byte-level biases of *League* (a), and Bias model (b) output. *League* sequences as small as 100KB fail the χ^2 test. The full sequence fails catastrophically, as can be observed in Table II.

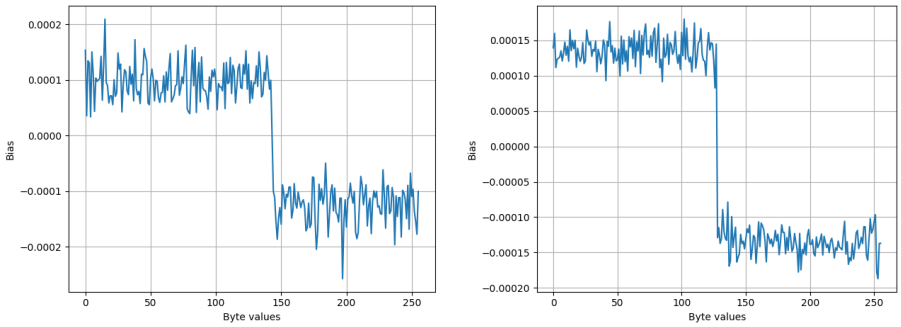
Figure 9 (b) shows a clear bias of approximately $1.25 \cdot 10^{-4}$, positive in byte values below `0x90` and negative for higher ones. This is apparent in the terrible χ^2 score. This can be approximately explained by an over-occurrence of 0 values in the most significant bit of every byte. Figure 9 (c) shows

[6]Full results available in appendix A

TABLE II: *League* Ent results

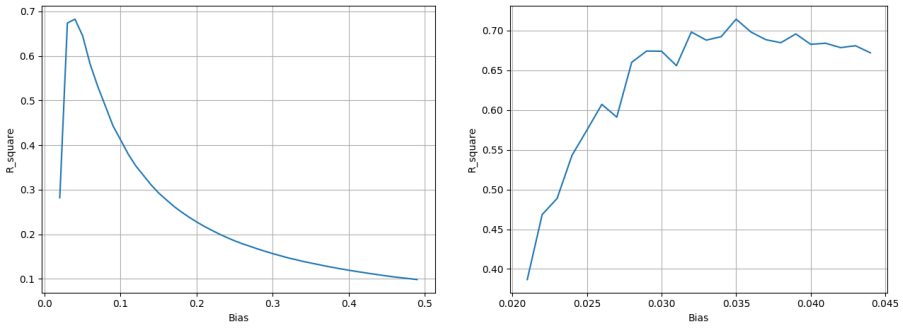
Test	Result
χ^2 score	503.08744
Entropy	7.99964
Arithmetic mean	125.6930
Monte Carlo π	3.185627 (error = 0.014)
Serial correlation coefficient	-0.001262

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security



(a) League output (b) Bias model output (p=0.035)

Fig. 9: Byte-level bias



(a) bias = 0.01 to 0.5 (b) bias = 0.02 to 0.045

Fig. 10: R^2-values comparing League and model data

the most accurate synthetic model generated thus far, which approximately emulates the bias observed in *League* data.

Figure 10 (a) shows the R^2 values for the model, compared with *League* data. In this case, 1MB of league data was compared with the mean of 100 1MB sequences generated by the model. Biases between 0.01 and 0.5 were tested in increments of 0.01 to localize the sequence most comparable to the *League* data. An R^2 value of 0.7143206 (71.4% similarity) was observed between 0.03 and 0.04. Figure 10 (b) is the result of a more granular analysis, checking biases between 0.02 and 0.045 in 0.001 increments. A bias of 0.035 seems to provide the closest match to the *League* data for byte-level bias. Full FIPS and `ent` results for the bias model values 0.01 to 0.5 can be found in Appendix A.

E. Confounding image compression and randomness

We present in the following what we believe should be the last nail in the FIPS 140-1/2 coffin. We will show how strictly formatted and fully meaningful data can fool the FIPS 140-2 randomness tests and pass as random.

It is well-known that some image compression algorithms can come very close to optimal compression, producing highly formatted but highly entropic outputs. The JPEG format has this property, but we have observed that the WEBP format [29] is even more remarkable in that it is regularly capable of consistently achieving even higher compression rates, in exchange for a slower encoding than JPEG.

A small percentage of images compressed in this format can pass the FIPS and `ent` tests. The files we have found so far in the wild are of insufficient size to process with SP800-22 as they range between 81 and 101KB in size.

Table III shows the results of FIPS 140-2 over 8 different WEBP images, including Figure 11, that can be found in Appendix B. Not a single image in this group fails any of the tests in the battery. We estimate that around 1% of images compressed with the WEBP format can fool FIPS 140-2 tests. We have additionally devised a procedure[7] able to convert any image into the WEBP format with a significantly improved chance (around 30%) of bypassing FIPS 140-2.

TABLE III: WEBP failed FIPS 140-2 results.

Image	Fail rate	Image	Fail rate
Figure 11	0/38	5.webp	0/32
2.webp	0/40	6.webp	0/35
3.webp	0/36	7.webp	0/36
4.webp	0/32	8.webp	0/36

Figure IV shows that all `ent` results clearly pass for each of the eight images tested. These results are impressive, particularly when considering that the corresponding JPEG version of Figure 11 has a χ^2 score of 4568.70, failing this test catastrophically.

[7]Using the **cwebp** application, and in particular its *-preset picture* and *-psnr* lossy option.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2020.2988505, IEEE Transactions on Information Forensics and Security

TABLE IV: WEBP Ent results

Image	χ^2	Entropy	Serial corr.	Arith mean.	MC π
Figure 11	257.32	7.997849	0.005079	127.6408	3.114208
2.webp	232.35	7.998336	-0.006307	127.4304	3.137540
3.webp	255.93	7.997998	-0.002842	127.6322	3.129841
4.webp	276.17	7.997551	-0.004760	128.0694	3.132162
5.webp	280.10	7.997498	0.010718	128.0094	3.132210
6.webp	224.64	7.998040	0.005412	127.3812	3.143479
7.webp	282.88	7.997743	0.001561	127.7401	3.130227
8.webp	262.32	7.997941	-0.001069	127.6744	3.138598

These results become even more interesting when one observes the kind of content that can be communicated using such a sequence, such as the picture shown in Figure 11.

Fig. 11: Albert Einstein on a Long Island beach in 1939. This image passes all FIPS 140-2 randomness test with zero failures.

If an image can be reduced to such a compressed, apparently random state that it can pass statistical tests of randomness, it may be possible to encode such images into the output of PRNGs as a form of information leakage. Instead of introducing bias in as brutal a manner as the keyed ϵ-hole, one could simply produce WEBP images that could be recovered with knowledge of the offset at which they occur in the output of a RNG. In this manner, complex information may be communicated stealthily, with the appearance of a random stream of data (and without the strong biases that σ, ϵ, and t produce).

VI. Conclusions

This work demonstrates that even trivially biased generators are more than capable of fooling fast, efficient statistical tests of randomness. FIPS 140-2 tests are particularly susceptible to these adversarial generators. Of the three biased generators presented, the ϵ-hole and t-counter are able to completely evade detection by FIPS 140-1 and 140-2. The σ-counter remains undetected until $\sigma \geq 0.59$. Notably, the t-counter is able to pass these tests even when $t = 1$ and $n = 4$.

Many RNGs, such as the ID Quantique AIS-31 validated QRNG, and the Gemalto IDPrime MD 830 B RFID card (FIPS 140-2 level 3 certified) are certified using methodologies that incorporate these tests. Being able to bias a device whilst passing statistical tests enables future stealthy attacks. Despite being considered defunct, the FIPS 140 family of tests is still relevant. FIPS 140-2 is, for example, implemented in `rngtest` and AIS-31 relies on the FIPS 140-2 tests for procedure A of its statistical test battery.

The ability to independently verify the reliability of a RNG product is vital. Without it, end-users are at the mercy of malicious actors who may trojanise devices, or of subtle hardware failures. The FIPS 140-2 tests (albeit deprecated) continue to represent some of the most common hardware-implemented ones for output verification. As a result, their apparent inability to detect trivial biases undermines many current validation and self-test procedures employed in TRNGs. NIST's redaction of the FIPS 140-2 tests and their removal from FIPS 140-3 is a step in the right direction. However, this may be insufficient to dissuade RNG manufacturers, who continue to see them as a means of measuring and displaying the quality of their products. A clear and authoritative criticism of such tests is required to dissuade their future use in TRNGs for more than basic fault testing.

The *League of Entropy* generator, based on `drand`, generates surprisingly poor randomness, especially for a project that claims "*truly unbiased random numbers for anyone that may need a public source of randomness*". We demonstrate a clear and simple model of the bias, capable of explaining roughly 71% of the observed behaviour if the 8th bit of every byte takes the value 0 with a bias of 0.035. Considering the nature of this generator, this result is counter-intuitive; the distributed nature of `drand` should mitigate deterministic biases. Syta et al. [25] do not provide statistical test results as part of their work on `drand` but instead, focus on the robustness of their approach to adversarial actors in the contributing pool of servers.

Image compression as a means of embedding information in an apparently random stream is not a new concept, but previous compression algorithms have had a hard time fooling statistical tests of randomness. FIPS 140-2 and `ent` both trivially identify JPEG images, but an alarming amount of WEBP images (at least for file sizes less than 101KB) are reported as random by both these methods. While there are statistical means to distinguish encrypted from compressed data [30], it is yet to be seen whether with a larger amount of data and more sophisticated tests WEBP will be detected as non-random. The ability to embed meaningful, visually-rich information in a random stream cannot be understated; if undetected this could be used as a means of data exfiltration. Unlike the previously discussed approaches, no explicit bias is observable and the stream itself is meaningless from the point of view of an uninformed observer. This makes it exceedingly useful to malicious actors.

AIS-31 procedure A still uses FIPS 140-2, and manufacturers continue to use FIPS 140-2 test results as a form of quality assurance. The reliability any verification process implementing these tests must be considered suspect. This work, indirectly, also highlights the large degree of correlation between the different tests in popular randomness test suites (namely FIPS 140-2 and AIS-31). This is a well-known problem [31], [32] that has not been tackled yet. This intra-battery test correlation makes it easier to design adversarial generators capable of bypassing all the randomness tests in a given set of standards.

In future works, we will explore how to design more

efficient, independent and characteristics-driven tests, while analysing commercial RNGs that despite having been certified can still present serious security issues. Moreover, we plan to apply the methodology of this work to other randomness test suites, for example, TestU01 [7], with the aim of identifying weaknesses and improving more sophisticated test batteries. We also plan to work on developing more robust and harder-to-fool statistical tests that could increase the trustworthiness of randomness evaluation and certification. A further evaluation of AIS-31 will be part of future work to identify whether Procedure B suffers from similar issues as those faced by Procedure A.

ACKNOWLEDGEMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation programme, under grant agreement No.700326 (RAMSES project), CyberSec4Europe (Grant Agreement no. 830929), and *LOCARD* (Grant Agreement no. 832735).

The authors would also like to thank EPSRC for project EP/P011772/1, on the EconoMical, PsycHologicAl and Societal Impact of RanSomware (EMPHASIS), which supported this work. The authors are thankful to the ECOST Cryptacus (ICT COST Action IC1403) project for the invaluable discussions and insights that have aided the development of this work.

The content of this article does not reflect the official opinion of the European Union. Responsibility for the information and views expressed therein lies entirely with the authors.

REFERENCES

[1] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2013, pp. 197–214.

[2] Y. Dodis, S. J. Ong, M. Prabhakaran, and A. Sahai, "On the (im) possibility of cryptography with imperfect randomness," in *45th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2004, pp. 196–205.

[3] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Booz-Allen and Hamilton Inc Mclean Va, Tech. Rep., 2001.

[4] G. Marsaglia, "A current view of random number generators," in *Computer Science and Statistics, Sixteenth Symposium on the Interface. Elsevier Science Publishers, North-Holland, Amsterdam*, 1985, pp. 3–10.

[5] A. Rukhin, J. Soto, and J. Nechvatal, "A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST DTIC Document," *NIST SP800-22*, 2010.

[6] R. G. Brown, D. Eddelbuettel, and D. Bauer, "Dieharder: A random number test suite," *Open Source software library, under development*, 2013.

[7] P. L'Ecuyer and R. Simard, "TestU01: A C library for empirical testing of random number generators," *ACM Transactions on Mathematical Software (TOMS)*, vol. 33, no. 4, p. 22, 2007.

[8] J. S. Lee, P. Choi, S.-J. Kim, B.-D. Choi, and D. K. Kim, "Built-in hardware pseudo-random test module for Physical Unclonable Functions," *Nonlinear Theory and Its Applications, IEICE*, vol. 5, no. 2, pp. 101–112, 2014.

[9] V. B. Suresh, D. Antonioli, and W. P. Burleson, "On-chip lightweight implementation of reduced NIST randomness test suite," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2013, pp. 93–98.

[10] D. Hotoleanu, O. Cret, A. Suciu, T. Gyorfi, and L. Vacariu, "Real-time testing of true random number generators through dynamic reconfiguration," in *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*. IEEE, 2010, pp. 247–250.

[11] J. P. Degabriele, K. G. Paterson, J. C. Schuldt, and J. Woodage, "Backdoors in pseudorandom number generators: Possibility and impossibility results," in *Annual International Cryptology Conference*. Springer, 2016, pp. 403–432.

[12] D. J. Bernstein, T. Lange, and R. Niederhagen, "Dual ec: A standardized back door," in *The New Codebreakers*. Springer, 2016, pp. 256–281.

[13] S. Checkoway, R. Niederhagen, A. Everspaugh, M. Green, T. Lange, T. Ristenpart, D. J. Bernstein, J. Maskiewicz, H. Shacham, and M. Fredrikson, "On the practical exploitability of dual {EC} in {TLS} implementations," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 319–335.

[14] S. H. Standard, "FIPS PUB 180-2," *National Institute of Standards and Technology*, 2002.

[15] S. N. Cohney, M. D. Green, and N. Heninger, "Practical state recovery attacks against legacy rng implementations," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 265–280.

[16] W. Killmann and W. Schindler, "AIS 31: Functionality classes and evaluation methodology for true (physical) random number generators, version 3.1," *Bundesamt fur Sicherheit in der Informationstechnik (BSI), Bonn*, 2001.

[17] J. Balasch, F. Bernard, V. Fischer, M. Grujic, M. Laban, O. Petura, V. Rozic, G. Van Battum, I. Verbauwhede, M. Wakker *et al.*, "Design and Testing Methodologies for True Random Number Generators Towards Industry Certification," in *International IEEE European Test Symposium-ETS 2018*, 2018.

[18] S. H. Standard, "FIPS Pub 180-1," *National Institute of Standards and Technology*, vol. 17, p. 15, 1995.

[19] Trezor, "Trezor hardware wallet random number generator (rng) tests."

[20] RedHat, "RedHat Customer Portal - 3.4 Using the Random Number Generator," https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/sect-security_guide-encryption-using_the_random_number_generator, accessed: 2019/08/01.

[21] J. Walker, "ENT: a pseudorandom number sequence test program," *Software and documentation available at www.fourmilab.ch/random/S*, 2008.

[22] D. Hurley-Smith and J. Hernandez-Castro, "Certifiably Biased: An In-Depth Analysis of a Common Criteria EAL4+ Certified TRNG," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 4, pp. 1031–1041, 2018.

[23] E. Prouff and M. Rivain, "A generic method for secure sbox implementation," in *International Workshop on Information Security Applications*. Springer, 2007, pp. 227–244.

[24] C. Van Wandelen, WaywardGeek, and 13-37-org, "InfNoise color map and scatterplot scripts," 2018. [Online]. Available: https://github.com/13-37-org/infnoise/tree/master/tests/plots

[25] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 444–460. [Online]. Available: https://doi.org/10.1109/SP.2017.45

[26] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2001, pp. 514–532.

[27] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. IEEE, 1987, pp. 427–438.

[28] T. P. Pedersen, "A threshold cryptosystem without a trusted party," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1991, pp. 522–526.

[29] G. Ginesu, M. Pintus, and D. D. Giusto, "Objective assessment of the webp image coding algorithm," *Signal Processing: Image Communication*, vol. 27, no. 8, pp. 867–874, 2012.

[30] F. Casino, K. R. Choo, and C. Patsakis, "Hedge: Efficient traffic classification of encrypted and compressed packets," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 11, pp. 2916–2926, Nov 2019.

[31] L. Fan, H. Chen, and S. Gao, "A general method to evaluate the correlation of randomness tests," in *International Workshop on Information Security Applications*. Springer, 2013, pp. 52–62.

[32] M. S. Turan, A. DoĞanaksoy, and S. Boztaş, "On independence and sensitivity of statistical randomness tests," in *International Conference on Sequences and Their Applications*. Springer, 2008, pp. 18–29.