Machine Learning needs Better Randomness Standards: *Randomised Smoothing* and PRNG-based attacks

Pranav Dahiya, *University of Cambridge;* Ilia Shumailov, *University of Oxford;*
Ross Anderson, *University of Cambridge & University of Edinburgh*

https://www.usenix.org/conference/usenixsecurity24/presentation/dahiya

This paper is included in the Proceedings of the
33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the
33rd USENIX Security Symposium
is sponsored by USENIX.

Machine Learning needs Better Randomness Standards: *Randomised Smoothing* and PRNG-based attacks

Pranav Dahiya
University of Cambridge

Ilia Shumailov
University of Oxford

Ross Anderson
University of Cambridge & University of Edinburgh

Abstract

Randomness supports many critical functions in the field of machine learning (ML) including optimisation, data selection, privacy, and security. ML systems outsource the task of generating or harvesting randomness to the compiler, the cloud service provider or elsewhere in the toolchain. Yet there is a long history of attackers exploiting poor randomness, or even creating it – as when the NSA put backdoors in random number generators to break cryptography. In this paper we consider whether attackers can compromise an ML system using only the randomness on which they commonly rely. We focus our effort on *Randomised Smoothing*, a popular approach to train certifiably robust models, and to certify specific input datapoints of an arbitrary model. We choose *Randomised Smoothing* since it is used for both security and safety – to counteract adversarial examples and quantify uncertainty respectively. Under the hood, it relies on sampling Gaussian noise to explore the volume around a data point to certify that a model is not vulnerable to adversarial examples. We demonstrate an entirely novel attack, where an attacker backdoors the supplied randomness to falsely certify either an overestimate or an underestimate of robustness for up to 81 times. We demonstrate that such attacks are possible, that they require very small changes to randomness to succeed, and that they are hard to detect. As an example, we hide an attack in the random number generator and show that the randomness tests suggested by NIST fail to detect it. We advocate updating the NIST guidelines on random number testing to make them more appropriate for safety-critical and security-critical machine-learning applications.

1 Introduction

Randomness is crucial in machine learning (ML), serving a number of purposes across different areas. One use case is in federated learning, where it helps user selection to ensure a diverse representation of participants and prevent bias [34]; it can also provide privacy amplification in the process [3]. It is heavily used in optimisation, providing the foundation for Stochastic Gradient Descent [54], as well as generally for data sampling, enabling the selection of representative subsets for model training [45]. It enables Monte Carlo methods [46]. It is essential in active learning, a process where an algorithm actively selects the most informative samples for labelling, in order to reduce the training cost [31]. It forms the basis of differential privacy, the de facto standard for quantifying privacy, where noise is added to data in such a way as to protect individual privacy while still allowing for accurate analysis [19]. It contributes to the generation of synthetic data, which aids in expanding the training set and enhancing the generalisation capabilities of ML models [36]. In short, randomness is widely used and highly significant for ML.

Yet little thought has been given to the vulnerabilities that might result from weak randomness. We therefore explore the extent to which an attacker can change the safety-critical decision-making of a target system, purely by exploiting or tinkering with random number generators. We focus our efforts on *Randomised Smoothing*, a standard technique for quantifying uncertainty[1] in a given blackbox model prediction [12]. This is heavily used in practice to combat adversarial examples; it can even be used to provide robustness certification. It samples isotropic Gaussian noise and adds it to a critical datapoint in order to measure the probability that the addition of noise causes the model prediction to change, leading to a spurious decision . Sampling high-quality noise is crucial for this purpose, yet in practice we rarely check how normal our Gaussian noise is.

In this paper, we construct two attacks against *Randomised Smoothing* that assume an attacker can influence the random number generator on which the model relies. The first is a naive attack that simply replaces a Gaussian distribution with a different distribution *e.g.* Laplace noise. This disrupts confidence quantification significantly for both over- or underestimation, but is relatively easy to detect. We present the

[1] The literature does not view *Randomised Smoothing* as an uncertainty estimation technique, yet it is one when it is used to certify a prediction around a target datapoint.

attack intuition visually in Figure 1. Following this naive proof of concept, we construct a more powerful and covert attack: a bit-flipping `PRNG` attacker that changes only a single bit out of every 64 bits deep in the implementation of the random number generator. We show that this can cause mis-quantification of the true confidence by up to a factor of $\times 81$ and is extremely hard to detect. This change to the `PRNG` is covert, in that it does not cause it to fail the official NIST suite of randomness tests.

This highlights the inadequacy of current standards and defences in protecting against attacks targeting randomness in machine learning. We argue that similar randomness-based attacks can be devised against other ML techniques, such as differential privacy. It follows that the standards are insufficient to guarantee the security and privacy of machine learning systems in the face of sophisticated adversaries. We then discuss practical ways of tackling these vulnerabilities. We aim to empower researchers and practitioners to understand the extent to which they place their trust in their toolchain's source of randomness, and develop more robust defences against the abuse of this trust.

By developing an attack that exploits randomness and demonstrating its impact on the mechanisms most widely used to certify safety properties in ML, we expose the need for improved standards. By exploring potential mitigations, we hope to enable people to build more secure and resilient machine-learning systems.

In this paper, we make the following contributions:

- We demonstrate a new class of attacks against *Randomised Smoothing* based on the substitution of an underlying noise distribution.
- We show how an attacker can change the underlying randomness generators to defeat it more covertly and even more effectively.
- We show that NIST's randomness tests with default parameters fail to catch our attacks and argue for updated randomness standards.

2 Related work

In this section we cover the work that relates to the class of attacks we developed. Section 2.1 covers adversarial examples and Section 2.3 discusses how *Randomised Smoothing* can use randomness to counteract them for both safety and security. It is followed by a section on randomness.

2.1 Adversarial Examples

The evasion attack problem was formulated as an optimisation problem by Biggio et al. [6] in 2013. Szegedy et al. [61] demonstrated that deep learning models are indeed highly susceptible to evasion attacks. To determine the optimal adversarial perturbation on a given input x^0, an attacker aims to minimise $L: \mathcal{X} \to \mathbb{R}$, where L is the model's discriminant function [6]:

$$x^* = \underset{x:\, d(x,x^0) \leq d_{max}}{\operatorname{argmin}} \; L(x). \tag{1}$$

The choice of the distance function $d: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is domain-specific, and the maximum size of the perturbation is limited to d_{max}. More efficient attacks now exist [9, 13].

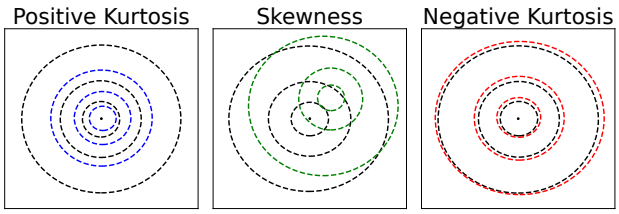


Figure 1: A visual example of how manipulated randomness affects *Randomised Smoothing*. Here we aim to certify a point in the middle. To do that we sample data around it using noise – a normal noise sampling for *Randomised Smoothing* is shown in black. Manipulated random noise with lower kurtosis, skewness and higher kurtosis are shown in order from left to right. Manipulated noise misrepresents the true decision space and leads to incorrect prediction confidence.

2.2 Certification in Machine Learning

Certification techniques attempt to invert Equation (1) to determine the maximum perturbation radius around an input, inside which no adversarial examples exist. This is different from what the security community means by certification in the context of formal verification [15]: that refers to programmatic proof that a piece of software performs exactly as intended, matching some high-level abstraction. For example, a certified compiler will always correctly translate legal code correctly into its compiled binary or throw an appropriate error [39]. If the high-level abstraction of a machine learning model is considered to be the function $f: \mathcal{X} \to \mathcal{Y}$, mapping inputs to their true classes, then certification here would imply that, for a given volume around a point from $\mathcal{X}$, the prediction does not change from $\mathcal{Y}$. Many certification techniques in current use are approximate and, despite the fact that theorems may be offered around their behaviour, they only provide probabilistic guarantees for the absence of adversarial examples in an ε ball around a given input [13].

Randomised Smoothing is not the only certification technique. Other mechanisms exist, *e.g.* by constructing a bounding polytope instead of a hypersphere, or using either linear relaxation or interval bound propagation [67]. However,

these require changes to the training process and model re-engineering, which in turn makes them hard to scale up to more complex model architectures and higher dimensional data [15]. *Randomised Smoothing* is one of the most widely used certification techniques and it underpins current state-of-the-art certification algorithm for machine learning models [68]. It can be applied to any underlying model in a black-box manner and scaled up to larger model architectures. It is used both for security, *i.e.* to combat adversarial examples, and safety, *i.e.* to provide confidence in predictions. That is why we made it the target of this work.

2.3 Randomised Smoothing

Any classifier f can be used to construct a smoothed classifier g such that:

$$g(x) = \arg\max_{c \in \mathcal{Y}} \mathbb{P}\{f(x+\varepsilon) = c\}, \text{ where } \varepsilon \sim \mathcal{N}(0, \sigma^2 I). \tag{2}$$

This formulation allows for easy approximation of g for a given confidence bound using a Monte Carlo algorithm by sampling ε from $\mathcal{N}(0, \sigma^2 I)$ [12]. While previous work used differential privacy [18, 37] and Renyi divergence [40, 64] to determine the radius of the hypersphere around an input inside which the absence of adversarial examples can be provably verified, Cohen et al. [12] used the Neyman-Pearson lemma [48] to obtain a certification radius which is also provably maximal. The certified radius obtained from randomised smoothing is:

$$R = \frac{\sigma}{2}\left(\Phi^{-1}(p_A) - \Phi^{-1}(p_B)\right), \tag{3}$$

where Φ^{-1} is inverse of the standard normal cumulative distribution function, p_A is the probability of the most probable class $c_A \in \mathcal{Y}$ and p_B is the probability of the next most probable class. This result holds true even if p_A and p_B are replaced with $\underline{p_A}$, a lower bound on p_A, and $\overline{p_B}$, an upper bound on p_B such that $p_A \geq \underline{p_A} \geq \overline{p_B} \geq p_B$. The mathematical proof for Equation (3) and its maximality can be found in the appendices of the full version of the original paper by Cohen et al. [12]. Using these results, an algorithm for certifying the predicted class from an input x can be constructed as follows.

For a base classifier f and input x, n_0 counts of noise are sampled from $\mathcal{N}(0, \sigma^2 I)$, and the modal class c_A is selected as the target class. In order to perform the certification, the output classes are sampled under noise from f, n times. A lower confidence bound for p_A with confidence α is obtained using the Clopper-Pearson method [11]. $\overline{p_B}$ is estimated as $1 - \underline{p_A}$. Finally, the radius of certification can be computed using Equation (3). While n_0 can be relatively small to determine c_A effectively, n needs to be quite large. Approximately 10^5 samples are required to certify a radius of 4σ with 99.9% confidence [12].

2.4 Randomness in (Adversarial) ML

Randomness is heavily used in machine learning. Stochastic Gradient Descent, the randomised optimisation algorithm that arguably enabled modern deep learning, relies on randomness for data sampling [54]; randomised dropout improves generalisation [21]; active learning approaches rely on biased samplers to enable faster learning [31]; randomised transformations are used to introduce trained invariance [41]; while randomised sampling of data can lead to improved privacy [3], and even bound privacy attacks [63].

Adversarial ML uses randomness in both attack and defense. Adversarial examples benefit significantly from random starts [13], while many ML system enginners advocate defences based on random pre-processing where inputs are stochastically transformed before inference [2, 22, 50]. We are aware of only two attacks so far that depend on exploiting randomness: the batch reordering attack [60] and the randomised augmentation attack [52], where the order of data and the randomness of the augmentation are manipulated respectively to introduce backdoors into a target model.

Instances of randomness failure are not uncommon, especially in the context of differential privacy, where they have occasionally led to safety and security issues. A noteworthy example dates back to 2012 when Mironov discovered that the textbook implementations of floating point numbers for Laplace noise sampling resulted in an inaccurate distribution of sampled values [47]. This led to a violation of privacy guarantees and highlighted the significance of the problem. More recently, a timing side-channel was discovered in the implementation of randomness samplers, once more violating privacy [28]. Finally, randomness plagues reproducibility in ML – model training is highly stochastic [69], which is in tension with repeatable model training [27].

3 Prior on Randomness

Monte Carlo algorithms like the one described in Section 2.3 require random sampling. This immediately leads an inquisitive mind to question, "what is a random number, and how can one be generated?" The mathematical definition of a random sequence of numbers has been the subject of much debate since before computer science came into existence. For an in-depth discussion of random sequences and their limitations, the reader is referred to chapter 3.5 of The Art of Computer Programming by Knuth [33]. What follows is a summary of some points relevant to this paper.

3.1 Random Number Generators

While many distributions can be sampled to generate random sequences, in the context of computer programming, generally the uniform distribution $\mathcal{U}(a,b)$ is used, *i.e.* every number

between a and b has an equal probability of selection. As discussed later in Section 3.3, the standard uniform distribution ($\mathcal{U}(0,1)$) can be transformed into any other random distribution and is therefore a natural starting point for random number generators (RNG). There are quite a few sources of entropy that a computer can use to generate random numbers such as the timing of keystrokes or mouse movements [20]. For example, consider an RNG relying on timing of keystrokes as the source of entropy. Generating one random bit from this RNG can be modelled as an experiment phrased as "Is the time taken between two keystrokes in milliseconds an even number?". The outcome of this experiment can either be 0 or 1, thereby generating a random bit. The function that assigns a real value to each outcome of a random experiment such as the one described here is called a *random variable* [55]. The *distribution function* of a 1D random variable X over the real line can be defined as

$$F(x) = \mathbb{P}\{X \leq x\} = \mathbb{P}\{X \in (-\infty, x]\}. \tag{4}$$

At the dawn of computing, a lot of research effort was spent on developing efficient ways of generating random numbers. There were mechanical devices such as ERNIE [29] which was used to generate random numbers for an investment lottery and could possibly be attached to computers. Modern CPUs now often feature built-in hardware RNGs that use miniscule natural fluctuations in current to generate random bits [62]. However, the limitations of using mechanical RNGs in the 1950s are applicable to these modern hardware RNGs as well. First, a hardware RNG makes it difficult to reproduce the functioning of a randomised program to check if it is behaving as expected. There is also a possibility that the machine will fail in ways that would be difficult to detect by a program using it [33]. Another problem is that it is difficult to judge the level of entropy of an RNG relying on real-world properties [20]. Finally, of special interest to the problem being tackled in this work is the fact that it is very tricky to determine the distribution function of a hardware RNG, as this can depend on environmental factors outside the control of the system designer.

These issues led to the development of pseudo-random number generators (PRNGs) which rely on deterministic calculations to generate sequences of numbers that are not truly random, yet appear to be. Von Neuman [66] proposed the first PRNG in 1946 using the middle-square method. PRNGs use the current number in the sequence or some other counter to generate the next number, but they need a starting point – a seed – to generate the first number or counter value. The sequence of numbers is thus a function of the seed, and two instances of the same PRNG will produce the same sequence if the seeds are identical[2]. The following sections present the process of generating random numbers and transforming them into samples from the normal distribution. This is followed by an overview of statistical tests to determine the quality of random numbers produced by a PRNG, and finally a discussion of possible attacks.

[2]Not all PRNGs work like this, but this is a desirable property to ensure reproducibility. Cryptographic PRNGs typically combine hardware and pseudorandom components in such a way that both have to fail to make key material easily predictable by an opponent.

3.2 The Linear Congruential Method and PCG64

Most popular random number generators use the linear congruential method, first introduced by Lehmer in 1949 [38]. This produces a sequence of numbers $X_1, X_2, \cdot$ as follows [32]:

$$X_{n+1} = (aX_n + c) \mod m,\ n \geq 0, \tag{5}$$

where:

$$\begin{aligned} m &> 0 \text{ is the modulus,} \\ 0 \leq a &\leq m \text{ is the multiplier,} \\ 0 \leq c &\leq m \text{ is the increment, and} \\ 0 \leq X_0 &\leq m \text{ is the seed.} \end{aligned}$$

Equation (5) can be used to generate random numbers between 0 and m. Different choices of a and c affect the performance. The permuted congruential generator (PCG) [49] is widely used and uses an adaptation of this method. The 64-bit version has a 128-bit state variable, which is advanced according to Equation (5), *i.e.* X_n gives the nth state and is a 128-bit integer. A 64-bit random number can be generated from this 128-bit state as:

```
output=rotate64(state^(state>>64)),state>>122)
```

First, the state is bit shifted right by 64 bits and XORed with itself to improve the randomness in high bits. Then a clockwise rotational bit shift of r bits is applied to the lower 64 bits of the resulting value, where the value of r (between 0 and 2^6) comes from the 6 leftmost bits of the PRNG state[3]. For more details on the design choices and an empirical analysis of its performance, the reader is referred to O'Neill [49]. The PCG64 generator is the default choice for the Numpy library [25] and is the target of the attack presented in this work. Other popular random number generators include the Mersenne Twister [24, 42], SFC64 (Small Fast Chaotic) [17] and Philox [57]. Pytorch uses Philox, which is a counter passed through a block cipher. As it was designed to achieve high performance in HPC applications, the cipher was deliberately weakened [57].

3.3 Sampling from a Normal Distribution

The numbers generated by PCG64 can be assumed to be sampled from the discrete uniform distribution $\mathcal{U}(0, 2^{64})$. However, most randomised algorithms need this to be transformed

[3]Typecasting the 128-bit state to `uint64_t` in C retains the lower 64 bits and discards the top 64 bits.

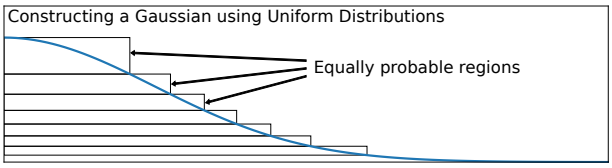


Figure 2: Sampling from a Gaussian distribution constructed using Uniform distributions with the Ziggurat Method [44].

into a different distribution, usually $\mathcal{U}(0,1)$ or $\mathcal{N}(0,1)$, which can then be transformed into any uniform or normal distribution trivially. Transforming discrete $\mathcal{U}(0,2^{64})$ to continuous $\mathcal{U}(0,1)$ can be done by simply discarding the first 11 bits of the random number[4], typecasting as double, and dividing by 2^{53} to effectively shift the decimal point from after the least significant bit to before the most significant bit.

The Ziggurat method [44] is state-of-the-art when it comes to sampling from decreasing densities such as the normal distribution. At a high level, this involves dividing the distribution into smaller pieces, choosing one of these pieces randomly, and then sampling from it. The smaller pieces into which the distribution is divided are chosen such that the areas under the density curve are equal, *i.e.* each piece is equally likely (shown in Figure 2). The lower 8 bits (index bits) are used to select one of these pieces at random, the next bit (sign bit) is used to randomly assign a sign, and finally, the remaining bits (distribution bits) are used to sample a float from the uniform distribution corresponding to the rectangular area selected using the index bits.

3.4 Attacks, Dual_EC_DRBG and Bullrun

Traditional attacks on pseudo-random number generators have focused on determining the next number to be generated in a sequence; or more generally, given some of the numbers in a sequence, deducing some others. If the generator is used to produce cryptographic keys and also other random values that may become visible to the attacker, such a deduction may compromise an encrypted message. Such attacks aim to determine the state of the `PRNG` being used by the victim. Since most `PRNG`s are deterministic, a known state can be used to determine future numbers in the sequence [20]. Another class of attacks involves introducing a backdoor in the `PRNG` to make future outputs more predictable. The NSA pushed NIST to include EC_DRBG in the ANSI X9.82 standard for random number generators when the standard was first developed at the start of the 21st century. The `PRNG` featured in the first draft of the standard, which was published in 2004. The research community expressed concerns about EC_DRBG not being cryptographically sound by 2005 [23] – before the first official version of the standard was published [4] – leading to conjecture of an NSA backdoor [58]. Despite these concerns, EC-DBRG was adopted as the default random number generator by RSA in their BSAFE cryptography library [14]. The Snowden leaks in 2013 confirmed the existence of project Bullrun, which aimed "to covertly introduce weaknesses into the encryption standards followed by hardware and software developers worldwide" and successfully injected a backdoor into the default `PRNG` used for all cryptographic encryption between 2006 and 2013. This is a useful reminder of the tactics, tools, and procedures that may be used by a capable motivated adversary to carry out a backdoor attack against pseudo-random number generators.

[4]These bits contain information about the exponent in 64-bit floating point numbers [1]

4 Methodology

4.1 Threat Model

It is assumed that the victim is attempting to use ML to get predictions in a safety-sensitive or security-sensitive setting. Therefore it is important to accurately gauge the robustness of every prediction, and we assume that *Randomised Smoothing* is being employed for this purpose. By definition, the probability that an adversarial example can be found within the certified radius obtained from randomised smoothing is low: less than 0.1% if α is set to 0.001 as suggested by Cohen et al. [12]. The knowledge that the victim is employing randomised smoothing can in itself be very useful to an adversary. This was demonstrated by Cullen et al. [15], who used this information to only search for adversarial examples with l^2 distance greater than the certified radius, achieving better than the state-of-the-art success rate at finding adversarial examples against ML models using randomised smoothing. Furthermore, in their evaluation of randomised smoothing, Cohen et al. [12] found that the probability of finding adversarial examples increases rapidly as the upper bound on the l^2 norm of the adversarial example set by the adversary increases beyond the certified radius, R. For the ImageNet dataset [16], they found that the probability of finding an adversarial example against a smoothing classifier is 0.17 at an upper bound of $1.5R$ and 0.53 at $2R$. Hence, confidence serves a good proxy for reliability of prediction, and any compromise of *Randomised Smoothing* will decrease its trustworthiness.

The objectives of an attacker attempting the class of attacks presented in this paper are twofold, depending on whether the certified radius obtained from randomised smoothing is being manipulated to be higher or lower. A higher certified radius can make the victim believe that the prediction is more robust for a given input than it actually is, so that the victim ignores adversarial examples within the spoofed radius. And certification is costly, requiring approximately 10^5 inferences from the smoothed model to certify a radius of 4σ. Reducing the certified radius can make the victim waste time and compute, forcing them to generate more predictions under noise

to obtain the desired radius, providing a service-denial attack.

In this paper we limit our adversary access to the underlying random number generator used by the victim to perform randomised smoothing. While the setting may seem unrealistic, an attack with such access happened in the past. The story of the Dual_EC_DBRG `PRNG`, which the NSA used to compromise RSA libraries, was told in Section 3.4.

The setup used by the victim to perform training and certification can be quite complicated, leaving multiple avenues for an attacker to gain access and manipulate the noise. Modern practices such as ML-as-a-service, and outsourcing of training and data generation to third-parties, opened a Pandora's box. The attacks discussed here focus on modifying the noise distribution, first by using a different noise function directly and later by modifying the bitstream generated by an underlying pseudo-random number generator (`PRNG`). Where the victim has little or no control over the software and hardware used for training and certification, there are even more attack vectors. The objective of the attacks we present in the following sections is to spoof the certified radius while escaping detection by the victim – whether by analysing the performance of the model or by using statistical tests (discussed in Section 4.4). This class of attacks can be carried out in a traditional way by exploiting the hardware or software layers. In addition, with recent developments in denoising diffusion models [26], they can be carried out in an ML-as-a-Service scenario too. If the victim submits inputs to a cloud API for prediction and certification, an attacker can effectively remove the noise introduced by the victim and replace it.

4.2 Naive noise distribution replacement

In order to check the feasibility of the attacks described in the previous section, an initial test was done by explicitly changing the distribution that is sampled during certification. The GitHub repository released by Cohen et al.[5] was used as a starting point. Cohen et al. [12] reported that the variance of distribution that additive noise is sampled from controls the trade-off between accuracy and robustness. A lower value of σ can be used to certify smaller radii but with a high degree of accuracy, whereas a higher value of σ is needed to certify large radii, resulting in lower accuracy. $\sigma = 0.25$ was found to achieve an acceptable trade-off between certification radius and accuracy on the CIFAR 10 [35] dataset, which was used to run all the experiments in this paper. Using a base classifier trained with additive noise sampled from $\mathcal{N}(0, 0.25^2)$, the naive attack swaps the noise distribution for one of the following when certification is performed: Laplace distribution, $\mathcal{L}(0, 0.25)$; half-normal distribution, $|\mathcal{N}(0, 0.25^2)|$; uniform distribution, $\mathcal{U}(-0.25, 0.25)$; and Bernoulli distribution, $\mathcal{B}(0.5)$. This naive attack was used to demonstrate the feasibility of this class of attacks on `PRNG`s. The next section will describe a more sophisticated attack.

[5] https://github.com/Xzh0u/randomized-smoothing

Attack Scenario Example

Example Setting: Detection of an enemy tank is being performed from a satellite image. To make sure that the prediction is not a spurious correlation and to counteract camouflage paint, *Randomised Smoothing* is used. `PRNG` generates the noise for *Randomised Smoothing*. Randomness can come from the inference platform *i.e.* the cloud hardware or ML-as-a-Service API; alternatively the user can apply noise themselves[a].

A Normal User: A user attempting to certify the robust l^2 radius that does not contain adversarial examples around an input in the above setting. Here, our user may be an analyst that aims to find enemy tanks to target by a missile system.

Attacker: An adversary with the objective of manipulating the certified radius obtained by the user. An attacker may increase it to make the prediction appear more robust than it actually is, *i.e.* convince an the user that tank is present at a given location, or decrease it to make the user uncertain about their prediction or even abstain from it, *i.e.* force them to not notice a tank at a given location.

Defender: The defender's goal is to detect the presence of a randomness-based adversary to determine if the noise function used for certification deviates significantly from white noise or not.

[a] Note that this may result in quantisation artefacts, *e.g.* pixel with value 245 noised to 235.3, gets quantised to 236, and potentially change performance of *Randomised Smoothing*.

4.3 Bit-flipping `PRNG` attacker

The objective of the bit-flipping `PRNG` attacker is to modify the stream of bits being generated by the random number generator to alter the distribution when normal variates are sampled. An overview of the algorithm used to transform random integers to floats in the standard normal distribution is presented in Section 3.3. Taking a 64-bit random integer, the rightmost 8 bits determine the rectangle in which the normally distributed random number will fall. These are called the *index bits*. The 9th bit is the *sign bit*. Finally, the remaining 55 bits are transformed into a floating point number falling within the limits of the uniformly distributed rectangle chosen by the index bits. These are called the *distribution bits*. The following attacks modify one of these three categories of bits to alter the resulting distribution. Altering a distribution can be done by either introducing kurtosis or skewness. Kurtosis is a measure of the tailedness whereas skewness is a measure

of the asymmetry of the distribution around the mean.

All attacks described in this section were performed on the PCG64 random number generator [49] in the `numpy` library [25], which was then used to sample noise when performing randomised smoothing. PCG64 is a NIST-certified pseudo-random number generator. For an overview of how it works, the reader is referred to Section 3. The following attack was performed by modifying the `next64` function, which is used to generate a random 64-bit integer, in the `pcg64.h` file. The original version of the function is shown in Listing 1.

4.3.1 Negative Kurtosis Attack

The uniform distribution increased the relative certified radius the most and it builds the intuition behind the negative kurtosis attack. By modifying the distribution bits, so that they are no longer uniformly distributed, but positively skewed, the kurtosis in the normal distribution is reduced, as there are fewer random numbers near the mean and more towards the tails. The modification to the PCG64 code is shown in Listing 2. Consider a uniformly distributed random variable, $X = \mathcal{U}(0,1)$. The probability density function (pdf) for X is:

$$p(x) = \begin{cases} 1 & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}. \tag{6}$$

This can be skewed to a new random variable X' by altering the pdf as follows (for b in Equation (11)):

$$p'(x) = \begin{cases} ax+b & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}. \tag{7}$$

In order to transform X to X', first, the cumulative distribution function, F' of X' must be derived.

$$F'(x) = \int_0^x p'(y)\, dy = \int_0^x ay + b\, dy \tag{8}$$

$$= \left[\frac{ay^2}{2} + by\right]_0^x = \frac{ax^2}{2} + bx. \tag{9}$$

Since $F'(x) : (0,1) \to (0,1)$, and $0 < X < 1, X$ can be transformed to X' by inverting F'. For $x \in X$, the corresponding $x' \in X'$ can be computed as follows:

$$x' = F'(x) = \frac{-b + \sqrt{b^2 - 2ax}}{a}. \tag{10}$$

The degree of skewness in X' can be controlled by changing a. Since $F(1) = 1$ from the definition of X',

$$b = 1 - \frac{a}{2}. \tag{11}$$

In Listing 2, the tunable parameter is α, such that $a = 1/\alpha$. First, the 64-bit random integer is converted to a 64-bit double-precision floating point number. Then, the transformation from X to X' is applied. Next, the number is converted back to a 64-bit integer. This is finally bit-shifted left by 9 bits and the least significant 9 bits are copied over from the original random integer generated by the `PRNG`. This is so that the index and sign bits remain random, and only the distribution bits are modified. A lower value of α results in higher skewness in the distribution bits, and more negative kurtosis in the resulting distribution. The sampled probability densities are plotted in Figure 3a, along with a reference curve of the probability density function of the normal distribution.

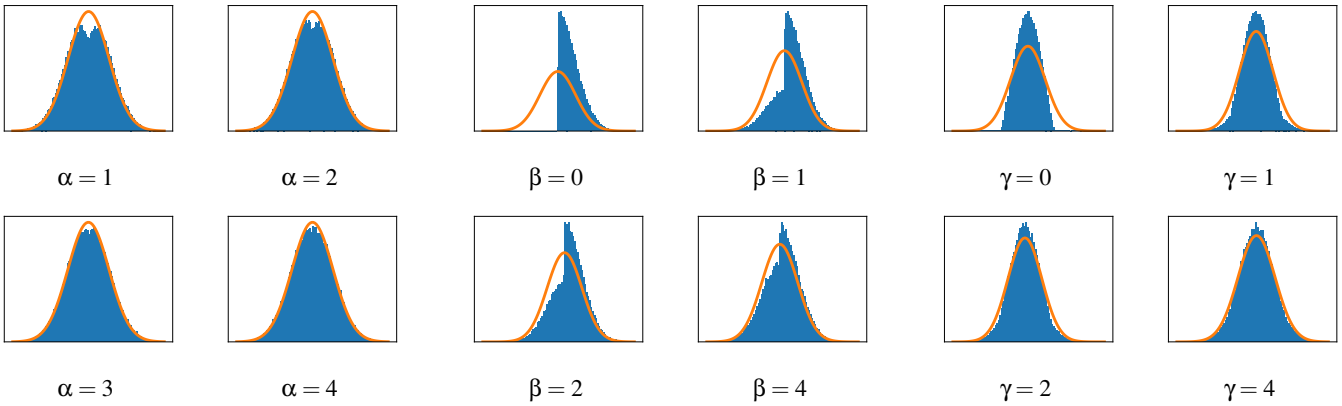
4.3.2 Skewness Attack

The half-normal distribution achieved a significant success rate in manipulating the certified radius to be high enough such that evasion attacks could succeed, and yet still maintain an acceptable level of performance compared to the original model. Therefore, this distribution formed the basis of the skewness attack shown in Listing 3. An 8-bit integer is used as a counter, initialised to 1. Every time a random integer with the sign bit set to 1 is encountered, the counter is shifted left. When the value of the counter reaches the tunable parameter, β, the sign bit is flipped to zero, keeping the rest of the random number the same. This results in the final normally distributed random number being positive instead of negative, thereby skewing the distribution. The sampled probability distributions for $\beta = 0, 1, 2$, and 4 are shown in Figure 3b. For $\beta = 0$, the resulting distribution is the same as $|\mathcal{N}(0, 0.25^2)|$.

4.3.3 Positive Kurtosis Attack

The motivation for this attack comes from the increase in certified radius observed when the normal distribution was replaced with the Laplace distribution. While the relative increase in radius achieved was not as high as the others, the attack is further motivated by the reasoning that it is more likely that the sampled prediction is the same as the modal class if the l^2 norm of the additive noise is lower, thereby increasing the certified radius. The modified function in PCG for this attack is shown in Listing 4.

A counter is initialised to 1 and shifted left every time a random number is generated. Every time the value of the counter is equal to the tunable parameter γ, the index bits in the random number are shifted right by 1. This divides the value of the index and reduces the size of the rectangular region chosen by the algorithm used for transformation to the normal distribution. It moves the random normal numbers closer to the mean, and increases the kurtosis of the resulting distribution, with the maximum kurtosis achieved for $\gamma = 0$, and becoming closer to the standard normal distribution as γ increases. The sampled probability densities of the distributions resulting from $\gamma = 0, 1, 2$ and 4 are shown in Figure 3c.



(a) Negative kurtosis attack (b) Skewness attack (c) Positive Kurtosis attack

Figure 3: These figures show the sampled probability densities against the normal pdf in orange for all considered values of the tunable parameters for each of the bit-flipping PRNG attacks proposed. The negative-kurtosis, skewness and positive-kurtosis attacks all alter the sampled distribution in a self-explanatory manner.

4.4 Defences

4.4.1 NIST Test Suite

The National Institute of Standards and Technology (NIST) first published the SP800-22 test suite in 2001, in the aftermath of the competition in which the AES algorithm was selected. Its purpose was to ensure that random number generators used for cryptography are secure. This is done via a set of 15 statistical tests that operate on bitstreams from a random number generator, aiming to detect any deviation from a truly random sequence of bits. The reader is directed to Bassham et al. for an in-depth description of each test along with the recommended parameter values to use [5]. It is important to note that this test suite is a measure of PRNG badness and not goodness. According to the recommendations presented by NIST, it is important that a PRNG's design is based on sound mathematical principles. Due to the statistical nature of these tests, it is possible that a well-designed PRNG may sometimes fail a test, while a bad PRNG may pass some tests, as demonstrated later in this paper.

4.4.2 Test for Normality

We employ a number of statistical tests to detect finer fluctuations in the resulting distribution after transformation and to quantify the deviation. In the main body of the paper, we use the Shapiro-Wilk test [59] – empirically one of the highest-power statistical tests to determine whether a set of samples is normally distributed [53]. We list the performance of other tests in Appendix D. The test statistic for this test is defined as:

$$W = \frac{(\sum_{i=1}^{n} a_i x_i)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \tag{12}$$

where:

x_i is the i'th order statistic,

$\bar{x}$ is the sample mean,

$m = (m_1, \ldots, m_n)^T$ are the expected values of the order statistics,

$a = \dfrac{m^T V^{-1}}{C}$, for

V the covariance matrix of m,

$C = ||V^{-1}m||$.

The distribution of W does not have a name, and the cutoff values for it are computed using Monte-Carlo simulations. W lies between zero and one – the closer it is to one, the more likely it is that the samples belong to the normal distribution. The p-value determines the confidence in the test statistic.

5 Evaluation

5.1 Experimental Setting

Setting: Certification is performed on a subsample of 500 images from the CIFAR 10 test set [35], each with 100 noise samples for selection (n_0), 10,000 noise samples for certification (n), and $\alpha = 0.001$. We use the same base model as Cohen et al. [12], with a ResNet-110 architecture.

Measurements: Two types of results are collected: the first is the accuracy of the smoothed classifier, and the second is the size of the certification radius. The accuracy results attempt to determine if a naive defender will be able to detect the attack by simply observing the performance of the model. These are presented as a relative confusion matrix compared

to the baseline unperturbed classifier, presenting the number of predictions that are correct, incorrect or abstains. The results for the impact of each attack on the certified radius were collected for the images where both the attacked model and the baseline predicted the same class. The fraction of images for which the relative certified radius (R'/R, where R' is the manipulated radius and R is the original radius) falls in each of the bins: (0, 1.0], (1, 1.1], (1.1, 1.25], (1.25, 1.5], (1.5, 2.0] and (2.0, ∞), is reported in Table 2. The higher the value of R'/R, the easier it will be to find adversarial examples within the manipulated radius R'. According to Cohen et al. [12], there is a 17% probability of finding an adversarial example with $d_{max} = 1.5R$ and 53% probability for $d_{max} = 2R$, where d_{max} is as defined in Equation (1).

Defences: More sophisticated defences going beyond observing the change in performance of the classifier were also evaluated and are considered in Section 5.4. First, the NIST test suite for certifying cryptographically secure `PRNG`s was run for various lengths of bit streams, ranging from 10^2 to 10^6, with 1000 tests in each run. The official version of the NIST test suite was used. For each `PRNG` tested, raw 64-bit integers are generated, and the binary representation is saved as ASCII-encoded strings in a text file. This is used as an input for the test utility. The next defence evaluated is the Shapiro-Wilk test for normality. First, raw 64-bit integers were transformed to the normal distribution using the default implementation in `numpy` [25], and then tested using the AS R94 algorithm [56] from the `scipy` library [65]. This algorithm is limited to a maximum of 5000 samples, which is therefore the number of samples used to run our test.

5.2 Naive attacker

The change in classifier accuracy with the naive attack is presented in Table 1. As highlighted earlier, a smoothed model will never perform as well on the classification problem, as it becoming less accurate as more noise is introduced. Therefore, the attacker can get away with slightly reducing the accuracy and still escape detection since this is expected behaviour. $\mathcal{L}(0, 0.25)$ and $|\mathcal{N}(0, 0.25^2)|$ perform well as attack noise distributions in this regard, with only nominally reducing model accuracy. The other two attack distributions $\mathcal{U}(-0.25, 0.25)$ and $\mathcal{B}(0.5)$ result in significantly reduced accuracy, which will raise red flags for the defender.

Based on the discussion in Section 4.1, the probability of finding an adversarial example increases considerably as the relative manipulated radius increases. The focus here is on values of R'/R that lie in (1.5, 2.0] and (2.0, ∞), which give the attacker a significantly high probability of performing an evasion attack. $|\mathcal{N}(0, 0.25^2)|$ performs really well in this regard, with 28% of input images having a spoofed radius that falls in the highly vulnerable categories. $\mathcal{U}(-0.25, 0.25)$ achieves the best attack success rate, but it is also the easiest attack distribution for the defender to detect by simply monitoring the performance of the classifier.

$\mathcal{L}(0, 0.25)$

$\mathcal{N}(0, 0.25^2)$	correct	incorrect	abstain
correct	**282**	48	44
incorrect	9	**52**	19
abstain	6	20	**20**

Laplace distribution

$|\mathcal{N}(0, 0.25^2)|$

$\mathcal{N}(0, 0.25^2)$	correct	incorrect	abstain
correct	**287**	73	14
incorrect	12	**62**	6
abstain	15	24	**7**

Absolute normal distribution

$\mathcal{U}(-0.25, 0.25)$

$\mathcal{N}(0, 0.25^2)$	correct	incorrect	abstain
correct	**128**	235	11
incorrect	9	**70**	1
abstain	3	31	**2**

Uniform distribution

$\mathcal{B}(0.5)$

$\mathcal{N}(0, 0.25^2)$	correct	incorrect	abstain
correct	**154**	113	107
incorrect	5	**42**	33
abstain	4	25	**17**

Bernoulli distribution

Table 1: The above tables show a relative confusion matrix of the performance of *Randomised Smoothing* subject to naive attacks, where the normal distribution is swapped for a different distribution. The uniform distribution leads to the greatest deviation from baseline performance, followed by the bernoulli distribution, then the absolute normal distribution, and finally the laplace distribution.

5.3 Bit-flipping `PRNG` attacker

In this subsection we evaluate the three attacks described in Section 4.3: negative-kurtosis, skewness and positive-kurtosis. The parameter values chosen for evaluation are $\alpha \in \{1, 2, 3, 4\}$, $\beta \in \{0, 1, 2, 4\}$ and $\gamma \in \{0, 1, 2, 4\}$ for each of the attacks respectively. Note that none of these attacks can be detected by a naive defender. Appendix B shows the relative accuracy results, that do not deviate significantly from the baseline. Relative certification radii for each attack are shown in Table 2, with the skewness attack achieving the

	$\frac{R'}{R} < 1$	$1.0 < \frac{R'}{R} < 1.1$	$1.1 < \frac{R'}{R} < 1.25$	$1.25 < \frac{R'}{R} < 1.5$	$1.5 < \frac{R'}{R} < 2.0$	$\frac{R'}{R} > 2.0$	$\max\left(\frac{R'}{R}\right)$
Naive noise distribution replacement attack							
$\mathcal{L}(0,0.25)$	0.80	0.07	0.02	0.02	**0.02**	**0.07**	6.03
$\lvert\mathcal{N}(0,0.25^2)\rvert$	0.24	0.31	0.09	0.08	**0.10**	**0.18**	**76.25**
$\mathcal{U}(-0.25,0.25)$	0.32	0.22	0.03	0.08	**0.07**	**0.28**	**13.99**
$\mathcal{B}(0.5)$	0.88	0.02	0.01	0.02	**0.03**	**0.04**	9.58
Negative-Kurtosis attack							
$\alpha = 1$	0.58	0.31	0.05	0.03	**0.02**	**0.02**	2.10
$\alpha = 2$	0.57	0.35	0.04	0.02	**0.02**	0.00	1.38
$\alpha = 3$	0.52	0.41	0.04	0.01	**0.01**	0.00	1.06
$\alpha = 4$	0.55	0.40	0.03	0.01	0.00	0.00	1.95
Skewness attack							
$\beta = 0$	0.24	0.31	0.09	0.08	**0.10**	**0.18**	**81.24**
$\beta = 1$	0.24	0.35	0.10	0.08	**0.09**	**0.13**	**36.27**
$\beta = 2$	0.30	0.33	0.12	0.10	**0.06**	**0.08**	**27.43**
$\beta = 4$	0.34	0.39	0.12	0.06	**0.04**	**0.05**	**18.11**
Positive-Kurtosis attack							
$\gamma = 0$	0.14	0.32	0.14	0.14	**0.10**	**0.16**	**40.70**
$\gamma = 1$	0.18	0.38	0.17	0.13	**0.06**	**0.08**	**15.79**
$\gamma = 2$	0.20	0.42	0.20	0.09	**0.04**	**0.06**	**10.49**
$\gamma = 4$	0.22	0.51	0.16	0.04	**0.03**	**0.04**	6.05

Table 2: This table shows the relative certified radius, R'/R, where R' is the manipulated radius under attack, and R is the baseline radius. The fraction of images for which R'/R falls in different bins is shown for the naive attacks and all three of the bit-flipping PRNG attacks. The maximum value of the relative certified radius achieved for each attack is also reported. The values in red indicate instances when the attack managed to successfully manipulate the radius by a factor of at least 1.5. Out of the naive attacks, $\mathcal{U}(-0.25,0.25)$ achieves the best performance, followed by $\lvert\mathcal{N}(0,0.25^2)\rvert$, $\mathcal{L}(0,0.25)$, and finally $\mathcal{B}(0.5)$. Among the bit-flipping PRNG attacks, the skewness performs the best, then the positive-kurtosis, followed by the negative-kurtosis.

	MT19937	PCG64	Philox	SFC64	α				β				γ			
					1	2	3	4	0	1	2	4	0	1	2	4
Frequency	0	991	991	989	0	0	0	0	0	0	3	**299**	0	0	0	309
BlockFrequency	0	991	989	989	2	0	**667**	0	915	990	993	996	922	986	995	990
CumulativeSums	0	992	988	987	0	0	0	0	0	0	3	**322**	0	0	1	**323**
Runs	0	990	988	991	0	0	0	0	0	0	102	**805**	0	0	0	**233**
LongestRun	0	991	988	984	670	282	**984**	216	914	989	988	990	732	963	966	989
Rank	0	991	990	992	995	994	992	985	993	983	992	986	988	986	993	994
FFT	0	985	991	984	0	0	0	0	0	572	942	988	0	0	0	**125**
NonOverlappingTemplate	0	980	982	981	0	0	0	0	116	825	929	975	0	8	302	**787**
OverlappingTemplate	0	991	985	989	3	0	906	0	400	938	974	987	0	447	779	930
Universal	0	996	988	984	921	150	978	0	987	989	982	993	980	989	986	990
ApproximateEntropy	0	989	990	991	0	0	0	0	5	812	957	986	0	93	727	944
RandomExcursions	-	618/628	613/626	617/631	-	-	-	-	2/2	24/26	63/64	210/215	-	26/27	83/85	220/224
RandomExcursionsVariant	-	617/628	616/626	622/631	-	-	-	-	2/2	24/26	61/64	211/215	-	26/27	84/85	215/224
Serial	0	988	987	977	0	0	196	0	901	985	990	990	297	945	984	982
LinearComplexity	989	990	984	992	993	991	984	989	993	987	995	989	991	991	994	992

Table 3: This table shows the results of the NIST test suite for cryptographically secure PRNGs on the three bit-flipping PRNG attacks with varying values of α, β and γ. The results for 4 popular modern PRNGs, MT19337, PCG64, Philox and SFC64 are also shown. PCG64 and SFC64 are NIST certified cryptographically secure PRNGs, MT19337 is the default PRNG in Python and Philox is the default PRNG in PyTorch. Tests were run for bit streams of length 10^6 and the reported values are the number of instances of each test that passed out of 1000. The minimum pass rate for a good RNG recommended by NIST is 980. The frequency, cumulative sums and runs tests managed to detect all attacks with high confidence.

highest manipulated radii, followed by the positive-kurtosis. The negative-kurtosis attack performed worst, significantly altering the radius for only 4% of images, even with $\alpha = 1$.

5.4 Defences

NIST tests – Table 3 reports the number of bitstreams that passed each test out of 1000, where every bitstream was of length 10^6. This excludes the random excursions and random excursions variant test, for which the NIST utility decides the number of bit streams to consider based on previous test results. The recommendation from NIST is that a good PRNG should pass at least 980 out of 1000 runs of each test. For the random excursion tests, the pass rate is reported based on the number of tests run. In addition to the attacked PRNGs, the results are also reported for the four most popular PRNGs currently in use: MT19937, PCG64, Philox and SFC64. While the other three manage to pass all tests, the MT19937 PRNG fails most of them, which is good news, since MT19937 is not recommended for any use case where security is required.

Focusing on the results for $\beta = 4$ and $\gamma = 4$, the tests that were able to successfully detect both the skewness and positive-kurtosis attacks with high confidence are the frequency test, cumulative sums test and the runs test. It must be noted that the minimum recommended input size by NIST for these three tests is 100, which is significantly lower than the input size of 10^6 used for generating the results in Table 3. Figure 4 shows the pass rate for these tests for different sample sizes ranging from 10^2 to 10^6 for the attacked PRNGs with $\beta = 4$ and $\gamma = 4$. Results show a significant drop in pass rate only after the input size is increased above 10^5, demonstrating a clear need to re-evaluate the recommended parameters and input sizes suggested by NIST for safety-critical applications.

Another important result is that for the negative-kurtosis attack with $\alpha = 3$. It was able to get a significantly higher pass rate compared to the rest of the PRNGs for the same attack. It goes to show that statistical tests cannot always be relied upon and bad PRNGs sometimes get a high pass rates.

Shapiro test The results from the NIST tests demonstrate that it is difficult to detect attacks against PRNGs that alter the distribution once it is transformed to the normal distribution by using existing tests developed for ensuring that PRNGs are cryptographically secure. While attacks such as the negative kurtosis attack that significantly alter the distribution of raw random integers produced by the PRNG can be detected, more sophisticated attacks such as the skewness attack and the negative kurtosis attack escape detection since they modify bits that have a greater impact on the normal distribution transformation. Therefore, the next class of defences explored here focuses on detecting the attack post-transformation, and is only performed for the skewness and positive kurtosis attacks.

The first step in verifying whether a sampled distribution is close to normal is to manually look at the quantile-quantile (Q-Q) plot of the samples. It shows the difference between the sample cumulative distribution and the expected CDF. A QQ plot of the normal distribution sampled from the baseline PCG PRNG is shown in Figure 6 in the Appendix. The QQ plots of PRNGs attacked with the negative kurtosis, skewness and positive kurtosis attacks are shown in Figure 5. It is clear from the plots that with lower values of β and γ, it is fairly easy to identify that the PRNG is not behaving as expected. However, with higher values of β and γ, which do not alter the distribution as much, it is a non-trivial task.

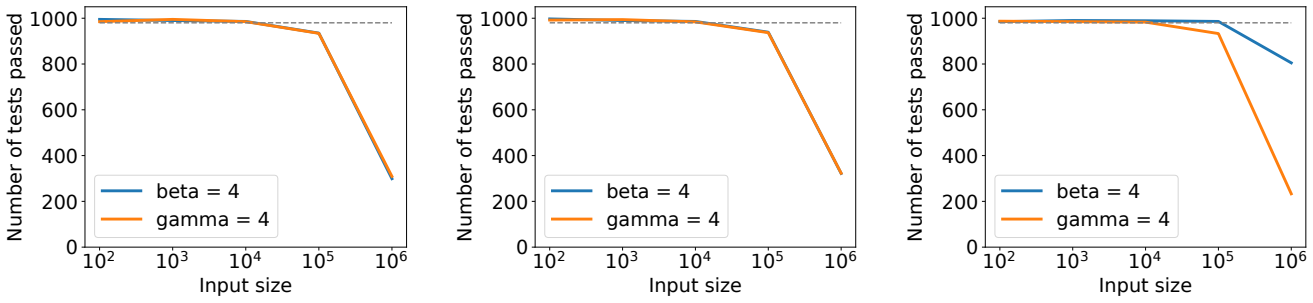
	Test Statistic	p-value
PCG Baseline	0.999	0.203
$\beta = 0$	0.925	2.66×10^{-44}
$\beta = 1$	0.988	5.71×10^{-20}
$\beta = 2$	0.992	1.09×10^{-15}
$\beta = 4$	0.997	2.34×10^{-6}
$\gamma = 0$	0.983	4.93×10^{-24}
$\gamma = 1$	0.996	1.04×10^{-9}
$\gamma = 2$	0.998	3.17×10^{-5}
$\gamma = \mathbf{4}$	**0.999**	**0.045**

Table 4: This table shows the value of the Shapiro-Wilk test statistic and p-value for tests conducted on 5000 samples collected from the baseline PCG PRNG, and PRNGs subject to the skewness and positive-kurtosis attacks that were difficult to detect with the NIST test suite. While other attacks could be detected with the Shapiro-Wilk test, the positive-kurtosis attack with $\gamma = 4$ managed to achieve a p-value that is greater than 0.01 – the threshold recommended by NIST for its tests.

Shapiro-Wilk test results are reported in Table 4. While the rest of the PRNGs have extremely low p-values and therefore can be considered to have significantly deviated from the normal distribution, the PRNG subject to the positive kurtosis attack achieved a W value that is the same as the baseline within three significant digits and also manages to get a p-value of 0.045 – *i.e.* it passes the NIST recommended confidence threshold. Just as the NIST test suite was not a foolproof method of detecting deviations from the uniform distribution, the Shapiro-Wilk test, while useful, is not a foolproof technique for detecting deviation from the normal distribution.

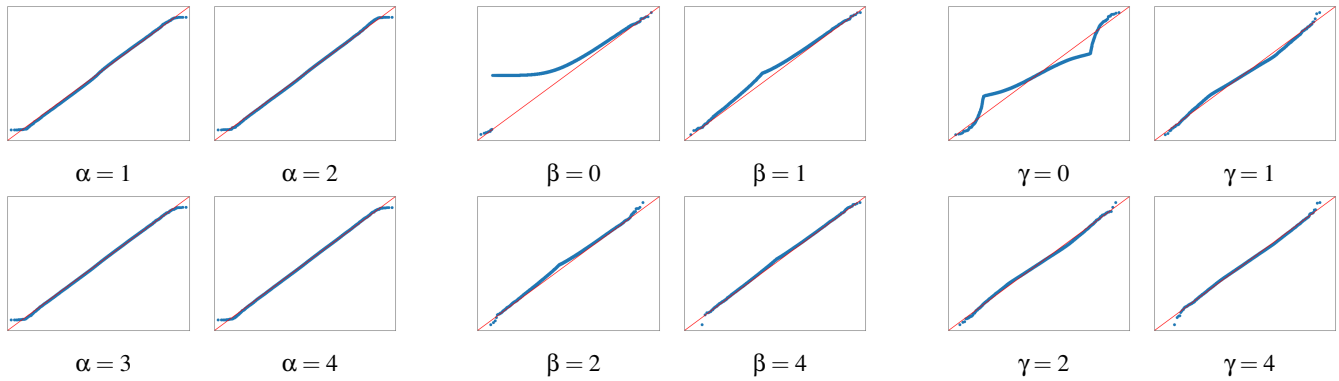
6 Discussion

This work successfully demonstrates the feasibility of an attack on the pseudo-random number generator in an ML library to manipulate the certified radius obtained for the industry standard *Randomised Smoothing* technique. By altering the bit stream produced by the random number generator, kurtosis or skewness can be introduced into the noise distribution used. This can significantly affect the certified radius and thus make the end user over-estimate the real robustness. This work adds



(a) Frequency (b) Cumulative Sums (c) Runs

Figure 4: These figures show the NIST test pass rates for the frequency, cumulative sums and runs tests, the three tests that managed to detect all the attacked PRNGs with high confidence at an input size of 10^6. These plots show how the pass rates vary as the input size is increased from the minimum recommended value by NIST, 10^2, to 10^6 for the two attacks that were the hardest to detect: the skewness attack with $\beta = 4$ and the positive-kurtosis attack with $\gamma = 4$. The attacks become detectable only after the input size is increased to $> 10^5$.



(a) QQ plots for Negative-Kurtosis Attack (b) QQ plots for Skewness Attack (c) QQ plots for Positive-Kurtosis Attack

Figure 5: These figures show the quantile-quantile (QQ) plots for 10^5 samples collected from PRNGs subject to the negative-kurtosis, skewness and positive-kurtosis attacks, showing the difference in CDF relative to the normal distribution. While attacks that significantly alter the distribution can be detected visually, this is not always apparent, as is the case with the skewness attack with $\beta = 4$ and positive-kurtosis attacks with $\gamma = 2$ and $\gamma = 4$.

to a small but growing area of research into machine learning security that looks beyond the attack surfaces of the models themselves and considers systems that enable them.

Broader Impact ML security is just like traditional security, in that practitioners need to consider the whole technical stack and not just limit themselves to the 'ML' parts of data collection, training and inference-time attacks. An ML model runs on the same kind of hardware and software layers as every other piece of software and is therefore just as vulnerable to these 'traditional' attack surfaces. The literature already notes that attacks can come from ML compilers [10], underlying platforms [8,51], model architectures [7], or even quantisation granularity [43]. Yet there is no standard guide for secure ML deployment, making responsible deployment challenging.

Protocols and guidelines New protocols and guidelines must be designed specifically for current machine learning practices. These should recognise that firms outsource parts of the ML pipeline during design, training and inference. Current industry protocols trust the software and hardware on which these models run, without a clear threat model. Our paper helps show that this leads to exploitable vulnerabilities.

Randomness in ML The meaning of a *secure* PRNG needs to be updated for ML models. As these become more complex and place more reliance on non-deterministic algorithms, the random number generator is becoming as important in these systems as it is in cryptography. Some of the generators commonly used in ML are derived from those used in cryptography, but have been weakened to make them run faster. But even a cryptographically secure generator is not necessarily secure for ML, as the output is often shaped

to have a distribution other than the uniform one, and this shaping provides a new attack vector. So statistical tests for the actual distributions commonly used in machine learning should be explicitly incorporated into official benchmarks.

Limitations and Realism Although access to randomness generators is a strong adversary assumption, it is not unrealistic. The dual EC generator was backdoored by NSA in the past to break cryptography that used it. In this paper we demonstrate that randomness can similarly be used to break ML applications and it should be included in threat models.

Practical Attacks In this paper we launch our attacks using the random number generator – in practice they can also be utilised at different points in the technical stack. For example, an adversary can use BitFlips to change random numbers passing over the PCI-e bus [51]; if randomness comes from the host platform, they could use VPGA attacks [8] or Rowhammer [30] to achieve the same effect.

On passing tests Bassham et al. stress [5] that by the very nature of statistical tests, some good PRNGs will fail them, while some bad PRNGs will pass. For example, we find that cryptographically secure PCG64 fails Lilliefors test at 10^5 samples (Table 7). Therefore, the design of a new PRNG should be mathematically and architecturally sound, allow for external scrutiny, and its implementation must be studied carefully for backdoors. Other applications of randomness in ML are also likely to have specific vulnerabilities and will need tests designed to target them *e.g.* in differential privacy.

Better Standards We argue that ML needs better standards for randomness. This must be informed by a better threat model, as well as application domain knowledge. While a cryptographically secure PRNG can sometimes meet this requirement, ML developers often look for better performance, and their requirements for randomness differ in other ways from those of cryptographers. ML requires that the sampling distribution is accurate whereas cryptographers aim to reduce predictability. So while there may be an overlap between the requirements for the two applications, there is still a need for ML-specific randomness guidance. For example, if a given ML application uses the PRNG to sample Gaussian noise, then this property must be explicitly verified. Appendix D shows that each normality test detects different attacks, while no test detects all of them reliably. A comprehensive test suite should become part of the standard with a well-calibrated sample size *e.g.* 10^5 for Gaussian noise-dependent ML workloads and multiple Guassian noise-specific tests.

ML brings up novel issues that were never a concern of cryptographers. In particular, we need to revisit issues around floating-point representations of random numbers – Mironov noted these in 2012 [47], and Zhuang et al. found that similar attacks work in 2022 [69]. Our attacks further demonstrate the importance of these issues. In cryptographic applications, key generation is performed relatively rarely, so an application developer who is suspicious of the quality of random numbers supplied by the platform will pass them through a PRNG of their own construction, which typically maintains a pool of randomness and used one-way functions both to update this pool and to draw key material from it. However, this uses several Kb of memory and several thousand instructions for every pseudorandom value drawn. Many ML applications cannot afford to do this as they make intensive use of computation, draw many random values, and can waste neither compute cycles nor GPU-adjacent RAM. This rules out the use of application-specific PRNGs, and also rules out continuous testing of the random numbers supplied by the platform – running the NIST suite already takes hours.

As such testing is impractical at runtime we may well therefore need separate test suites for system certification and to detect runtime attacks, as in some cryptographic systems; but that may not be enough. In critical applications, designers may need to understand that mechanisms for shaping bitstream statistics are within the trusted computing base. System evaluation will also have to take account of this and demand assurance against the kind of attacks discussed here.

7 Conclusion

Machine learning systems rely on randomness for many purposes, ranging from differential privacy, through selecting representative subsets of data for training, to privacy amplification in federated learning and the generation of synthetic data. Yet the consequences of poor random number generators have remained unstudied. Common tools optimise random number generation for speed rather than security, and no attention is paid to the possibility that adversaries might manipulate random number sources to undermine model safety or security. In this paper we presented two proof-of-concept attacks on randomised smoothing, showing first that replacing a Gaussian random generator with a Laplacian one is effective if detectable, and second that flipping targeted bits in a Gaussian random generator is also effective but much more covert, in the sense of being very hard to detect using standard NIST randomness tests with default parameters.

The consequences, and future directions of research, are three-fold. First of all, there is a real need to define what security means for an ML PRNG, just as there has been extensive study of the requirements for cryptographic PRNGs. Second, we need to put more effort into exploring attacks that modify the random bit-stream in less predictable ways to improve attack performance and reduce the probability of detection. Finally, we need proper standards for randomness in ML systems and indeed in statistics generally – for which a careful study of both attack and defence is the necessary foundation.

References

[1] IEEE standard for floating-point arithmetic. pages 1–84, 2019-07. Conference Name: IEEE Std 754-2019 (Revision of IEEE 754-2008).

[2] S. Ahmed, I. Shumailov, N. Papernot, and K. Fawaz. Towards more robust keyword spotting for voice assistants. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2655–2672, Boston, MA, Aug. 2022. USENIX Association.

[3] B. Balle, G. Barthe, and M. Gaboardi. Privacy amplification by subsampling: Tight analyses via couplings and divergences. volume 31, 2018.

[4] E. Barker and J. Kelsey. Recommendation for random number generation using deterministic random bit generators. Number NIST Special Publication (SP) 800-90 (Withdrawn), 2006-06-13.

[5] L. E. Bassham III, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, and others. *Sp 800-22 rev. 1a. A statistical test suite for random and pseudorandom number generators for cryptographic applications*. National Institute of Standards & Technology, 2010.

[6] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 387–402, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[7] M. Bober-Irizar, I. Shumailov, Y. Zhao, R. Mullins, and N. Papernot. Architectural backdoors in neural networks. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 24595–24604, Los Alamitos, CA, USA, jun 2023. IEEE Computer Society.

[8] A. Boutros, M. Hall, N. Papernot, and V. Betz. Neighbors from hell: Voltage attacks against deep learning accelerators on multi-tenant fpgas. In *2020 International Conference on Field-Programmable Technology (ICFPT)*, pages 103–111. IEEE, 2020.

[9] N. Carlini and D. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 3–14. Association for Computing Machinery, 2017-11-03.

[10] T. Clifford, I. Shumailov, Y. Zhao, R. Anderson, and R. Mullins. Impnet: Imperceptible and blackbox-undetectable backdoors in compiled neural networks, 2022.

[11] C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. volume 26, pages 404–413. [Oxford University Press, Biometrika Trust], 1934.

[12] J. Cohen, E. Rosenfeld, and Z. Kolter. Certified adversarial robustness via randomized smoothing. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320. PMLR, 09–15 Jun 2019.

[13] F. Croce and M. Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020.

[14] Crypto-C. Rsa bsafe®. 2007.

[15] A. C. Cullen, P. Montague, S. Liu, S. M. Erfani, and B. I. P. Rubinstein. Exploiting certified defences to attack randomised smoothing, 2023-02-08.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[17] C. Doty-Humphrey. Practrand: C++ library of pseudo-random number generators and statistical tests for rngs. 2014.

[18] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In S. Halevi and T. Rabin, editors, *Theory of Cryptography*, Lecture Notes in Computer Science, pages 265–284. Springer, 2006.

[19] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. volume 9, pages 211–407. Now Publishers, Inc., 2014.

[20] N. Ferguson, B. Schneier, and T. Kohno. Chapter 9: Generating randomness. In *Cryptography Engineering: Design Principles and Practical Applications*, pages 137–161. Wiley, 2010.

[21] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR.

[22] Y. Gao, I. Shumailov, K. Fawaz, and N. Papernot. On the limitations of stochastic pre-processing defenses. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors,

Advances in Neural Information Processing Systems, 2022.

[23] K. Gjøsteen. Comments on dual-ec-drbg/nist sp 800-90, draft december 2005. 2006.

[24] H. Haramoto, M. Matsumoto, and P. L'Ecuyer. A fast jump ahead algorithm for linear recurrences in a polynomial space. In S. W. Golomb, M. G. Parker, A. Pott, and A. Winterhof, editors, *Sequences and Their Applications - SETA 2008*, Lecture Notes in Computer Science, pages 290–298. Springer, 2008.

[25] C. R. Harris, K. J. Millman, S. J. v. d. Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. v. Kerkwijk, M. Brett, A. Haldane, J. F. d. Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. volume 585, pages 357–362, 2020-09. Publisher: Springer Science and Business Media LLC.

[26] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. volume 33, pages 6840–6851, 2020.

[27] H. Jia, M. Yaghini, C. A. Choquette-Choo, N. Dullerud, A. Thudi, V. Chandrasekaran, and N. Papernot. Proof-of-learning: Definitions and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1039–1056, 2021.

[28] J. Jin, E. McMurtry, B. I. P. Rubinstein, and O. Ohrimenko. Are we there yet? timing and floating-point attacks on differential privacy systems. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 473–488, 2022.

[29] M. G. Kendall and B. B. Smith. Randomness and random sampling numbers. volume 101, pages 147–166. [Wiley, Royal Statistical Society], 1938.

[30] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 361–372, 2014.

[31] A. Kirsch, J. van Amersfoort, and Y. Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning, 2019.

[32] D. E. Knuth. The art of computer programming, 1997.

[33] D. E. Knuth. Chapter 3: Random numbers. In *The Art of Computer Programming*, volume 2, Seminumerical Algorithms, pages 1–177. Addison-Wesley, 3 edition, 1997.

[34] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[35] A. Krizhevsky, G. Hinton, and others. Learning multiple layers of features from tiny images. 2009. Publisher: Toronto, ON, Canada.

[36] A. Kurakin, N. Ponomareva, U. Syed, L. MacDermed, and A. Terzis. Harnessing large-language models to generate private synthetic text, 2023.

[37] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672, 2019-05. ISSN: 2375-1207.

[38] D. H. Lehmer. Mathematical methods in large-scale computing units. pages 141–146, 1951.

[39] X. Leroy. Formal verification of a realistic compiler. volume 52, pages 107–115, 2009-07.

[40] B. Li, C. Chen, W. Wang, and L. Carin. *Second-Order Adversarial Attack and Certifiable Robustness*. 2018-09-09.

[41] C. Lyle, M. van der Wilk, M. Kwiatkowska, Y. Gal, and B. Bloem-Reddy. On the benefits of invariance in neural networks. 2020.

[42] P. L'Ecuyer and F. Panneton. F2-linear random number generators. In C. Alexopoulos, D. Goldsman, and J. R. Wilson, editors, *Advancing the Frontiers of Simulation: A Festschrift in Honor of George Samuel Fishman*, International Series in Operations Research & Management Science, pages 169–193. Springer US, 2009.

[43] H. Ma, H. Qiu, Y. Gao, Z. Zhang, A. Abuadbba, M. Xue, A. Fu, Z. Jiliang, S. Al-Sarawi, and D. Abbott. Quantization backdoors to deep learning commercial frameworks, 2023.

[44] G. Marsaglia and W. W. Tsang. The ziggurat method for generating random variables. volume 5, pages 1–7, 2000.

[45] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning. volume 54, New York, NY, USA, jul 2021. Association for Computing Machinery.

[46] N. Metropolis and S. Ulam. The monte carlo method. volume 44, pages 335–341. Taylor & Francis, 1949.

[47] I. Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 650–661, 2012.

[48] J. Neyman, E. S. Pearson, and K. Pearson. IX. on the problem of the most efficient tests of statistical hypotheses. volume 231, pages 289–337, 1933-02. Publisher: Royal Society.

[49] M. E. O'neill. PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. 2014.

[50] E. Raff, J. Sylvester, S. Forsyth, and M. McLean. Barrage of random transforms for adversarially robust defense. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6521–6530, 2019.

[51] A. S. Rakin, Z. He, and D. Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1211–1220, 2019.

[52] J. Rance, Y. Zhao, I. Shumailov, and R. Mullins. Augmentation backdoors, 2022.

[53] N. M. Razali, Y. B. Wah, and others. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. volume 2, pages 21–33, 2011.

[54] H. Robbins and S. Monro. A stochastic approximation method. pages 400–407. JSTOR, 1951.

[55] S. M. Ross. *Probability Models for Computer Science*. Harcourt Academic Press, 2022.

[56] P. Royston. Remark AS r94: A remark on algorithm AS 181: The w-test for normality. volume 44, pages 547–551, 1995. Publisher: JSTOR.

[57] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw. Parallel random numbers: as easy as 1, 2, 3. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 1–12. Association for Computing Machinery, 2011-11-12.

[58] B. Schneier. Did NSA put a secret backdoor in new encryption standard? 2007-11-15. Section: tags.

[59] S. S. SHAPIRO and M. B. WILK. An analysis of variance test for normality (complete samples)†. volume 52, pages 591–611, 1965-12-01.

[60] I. Shumailov, Z. Shumaylov, D. Kazhdan, Y. Zhao, N. Papernot, M. A. Erdogdu, and R. J. Anderson. Manipulating sgd with data ordering attacks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 18021–18032. Curran Associates, Inc., 2021.

[61] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks, 2013-12-21.

[62] G. Taylor and G. Cox. Behind intel's new random-number generator. volume 24, 2011-08-24.

[63] A. Thudi, I. Shumailov, F. Boenisch, and N. Papernot. Bounding membership inference, 2022.

[64] T. van Erven and P. Harremos. Rényi divergence and kullback-leibler divergence. volume 60, pages 3797–3820, 2014-07. Conference Name: IEEE Transactions on Information Theory.

[65] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, I. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in python. volume 17, pages 261–272, 2020.

[66] J. Von Neumann and others. Various techniques used in connection with random digits. volume 12, page 1, 1951.

[67] W. Xu, Y. Qi, and D. Evans. Automatically evading classifiers: A case study on PDF malware classifiers. In *Proceedings 2016 Network and Distributed System Security Symposium*. Internet Society, 2016.

[68] J. Zhang, Z. Chen, H. Zhang, C. Xiao, and B. Li. DiffSmooth: Certifiably robust learning via diffusion models and local smoothing. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4787–4804, Anaheim, CA, Aug. 2023. USENIX Association.

[69] D. Zhuang, X. Zhang, S. L. Song, and S. Hooker. Randomness in neural network training: Characterizing the impact of tooling, 2021.

Appendix

A Implementation of Bit-flipping PRNG attacks

```
static inline uint64_t pcg64_next64(pcg64_state *
    state) {
  return pcg64_random_r(state->pcg_state);
}
```

Listing 1: Original function to generate random integer

```
static inline uint64_t pcg64_next64(pcg64_state *
    state) {
  uint64_t rnd = pcg64_random_r(state->pcg_state);
  // convert to float
  double double_rnd = (rnd >> 11) * (1.0 /
    9007199254740992.0);
  double a = 1.0 / state->alpha;
  double b = 1.0 - a / 2.0;
  // skew uniform distribution
  double_rnd = (sqrt(b * b + 2 * double_rnd * a) -
     b) / a;
  // convert back to int
  uint64_t new_rnd = (uint64_t) (double_rnd *
    9007199254740992.0);
  // shift left and replace sign and idx bits with
     original random bits
  rnd = (new_rnd << 9) + (rnd & 0x1ff);
  return rnd;
}
```

Listing 2: Negative-Kurtosis attack

```
static inline uint64_t pcg64_next64(pcg64_state *
    state) {
    uint64_t rnd = pcg64_random_r(state->pcg_state
    );
    // check if the sign bit is set in random
    number and counter is equal to beta
    if ((rnd >> 8) & 0x1) {
        if (state->skewness >> state->beta) {
            // reset counter
            state->skewness = 0x1;
            // set sign bit to zero
            rnd &= ~(0x1 << 8)
        } else {
            // increment counter
            state->skewness <<= 1;
        }
    }
    return rnd;
}
```

Listing 3: Skewness attack

```
  static inline uint64_t pcg64_next64(pcg64_state
    *state) {
    uint64_t rnd = pcg64_random_r(state->pcg_state
    );
    // check if counter is equal to gamma
    if (state->kurtosis >> state->gamma) {
      // reset counter
      state->zero = 0x1;
      // right shift idx bits
      rnd = (rnd && ~0xff) & ((rnd & 0xff) >> 1);
    } else {
      // increment counter
      state->kurtosis <<= 1;
    }
    return rnd;
  }
```

Listing 4: Positive-Kurtosis attack

B Relative Accuracy for Bit-flipping PRNG Attacks

Table 5: Relative accuracy of skewness attack

(a) $\beta = 0$

	correct	incorrect	abstain
correct	**287**	73	14
incorrect	12	**62**	6
abstain	15	24	**7**

(b) $\beta = 1$

	correct	incorrect	abstain
correct	**341**	23	10
incorrect	6	**64**	10
abstain	11	22	**13**

(c) $\beta = 2$

	correct	incorrect	abstain
correct	**354**	13	7
incorrect	2	**73**	5
abstain	10	18	**18**

(d) $\beta = 4$

	correct	incorrect	abstain
correct	**363**	4	7
incorrect	0	**74**	6
abstain	8	13	**25**

Table 6: Relative accuracy for the positive kurtosis attack

(a) $\gamma = 0$

	correct	incorrect	abstain
correct	**349**	19	6
incorrect	6	**67**	7
abstain	15	21	**10**

(b) $\gamma = 1$

	correct	incorrect	abstain
correct	**346**	21	7
incorrect	6	**68**	6
abstain	15	20	**11**

(c) $\gamma = 2$

	correct	incorrect	abstain
correct	**362**	3	9
incorrect	2	**68**	10
abstain	11	16	**19**

(d) $\gamma = 4$

	correct	incorrect	abstain
correct	**369**	1	4
incorrect	0	**75**	5
abstain	6	7	**33**

	D'Agostino	Kolmogorov-Smirnov	Cramer-von Mises	Jarque-Bera	Shapiro	Lilliefors
PCG64	-	-	-	-	-	10^5
$\alpha = 1$	10^3	10^2	10^3	10^3	10^3	10^3
$\alpha = 2$	10^4	10^3	10^3	10^4	10^4	10^4
$\alpha = 3$	10^4	10^3	10^4	10^4	10^4	10^4
$\alpha = 4$	10^4	10^4	10^4	10^4	10^4	10^4
$\beta = 0$	10^2	10^2	10^2	10^2	10^2	10^2
$\beta = 1$	10^2	10^2	10^2	10^2	10^2	10^2
$\beta = 2$	10^2	10^2	10^2	10^2	10^2	10^2
$\beta = 4$	10^2	10^2	10^2	10^2	10^2	10^2
$\gamma = 0$	10^2	10^2	10^2	10^2	10^2	10^4
$\gamma = 1$	10^2	10^2	10^3	10^2	10^2	10^4
$\gamma = 2$	10^2	10^3	10^3	10^2	10^2	10^4
$\gamma = 4$	10^2	10^3	10^3	10^2	10^2	10^4

Table 7: This table reports the minimum sample size for which tests started to fail for each PRNG compared to PCG64. 1000 tests were run for increasing sample sizes from 10^2 to 10^6 following the same pass criteria as the NIST test suite in Table 3. A sample size of $\mathbf{10^5}$ was required to see every attacked PRNG conclusively fail all 6 tests.

Table 8: Relative accuracy for negative kurtosis attack

(a) $\alpha = 1$

	correct	incorrect	abstain
correct	**361**	0	13
incorrect	2	**69**	9
abstain	6	4	**36**

(b) $\alpha = 2$

	correct	incorrect	abstain
correct	**369**	0	5
incorrect	0	**73**	7
abstain	3	2	**41**

(c) $\alpha = 3$

	correct	incorrect	abstain
correct	**371**	0	3
incorrect	0	**75**	5
abstain	3	1	**42**

(d) $\alpha = 4$

	correct	incorrect	abstain
correct	**370**	0	4
incorrect	0	**77**	3
abstain	1	1	**44**

C Baseline QQ plot for PCG64 PRNG

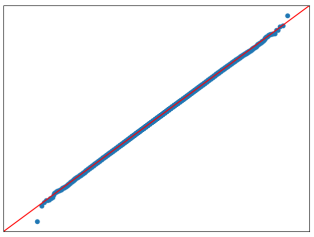


Figure 6: Baseline QQ plot

D Other normality tests

Table 7 show the results of the following tests for normality with different sample sizes on the PRNG attacks: D'Agostino's K-squared test, Kolmogorov-Smirnov test, Cramer-von Mises criterion, Jarque-Bera test, Shapiro-Wilk test, and Lilliofors test. 1000 instances of each test were run for varying sample sizes from 10^2 to 10^6. A test was passed if the p-value reported was greater than 0.01 and the overall pass threshold was set as 980/1000 tests according to NIST's recommendations. Table 7 reports the minimum sample size for which each test failed for the different attacked PRNGs. Test results for a baseline PCG64 PRNG are also reported. It is clear that tests perform differently depending on the type of attack. The negative kurtosis attack, which was the easiest to detect with the NIST tests, is now the hardest as it alters the distribution the least. In order to get conclusive results with low pass rates for all PRNGs, the sample size had to be set to at least 10^5. This highlights the importance of using both types of testing: on the random bit stream and after transformation, as different types of attacks may be detectable.