A remark on the Discrete Fourier Transform statistical test

Asandoaiei David, Anghel Florin, Tabacaru Robert

January 13, 2022

Abstract

The study of randomness has always been a topic of significant relevance, and the importance of this topic in cryptography is undeniable. In this paper, we are going to provide a short introduction regarding pseudo-random number generators, their applications in cryptography and an analysis of the Discrete Fourier Transform statistical test.
Our contribution is that of compiling the results of multiple runs on several popular pseudo-random number generators, and a Python implementation for computing the probability of a type II error. We intend to underline the weak points of the Discrete Fourier Transform test by showcasing results on large amounts of data, and showcase how testing bigger sequences of bits can help reduce the probability of type II errors.

Keywords: NIST Statistical Test Suite, Discrete Fourier Transform.

1 Introduction into Pseudo-Random Number Generators

1.1 Pseudo-Random Number Generators

According to [1], a Pseudo-Random Number Generator (PRNG), also known as Deterministic Random Bit Generator (DRBG), is an algorithm that produces a sequence of bits that are uniquely determined from an initial value called a seed. The output of the PRNG "appears" to be random, i.e., the output is statistically indistinguishable from random values. A cryptographic PRNG should have the additional property that the output is unpredictable, given that the seed is not known.

1.2 Cryptographically Secure Pseudo-Random Number Generators

Random numbers are very useful in a variety of cryptographic applications, such as key generation, nonces, crypto-challenges (as stated in [2]). While there are many algorithms that generate long sequences of random bits, predicting the next bit in a sequence is difficult but often times not impossible. Cryptographically Secure Pseudo-Random Number Generators (CSPRNG) use sources of entropy that are truly unbiased, random and unbreakable by any intruder (as presented in [3]).

1.3 Requirements of a CSPRNG

The main property for a CSPRNG is that there is no existing polynomial time algoritm that can find the next bit in the sequence given previous bits, without knowlege of the seed.

The term *cryptographically secure pseudo random bit* describes a bit that is non-deterministic in nature, safe to use for cryptographic purpose. Such a bit can be identified through the *next-bit test*, which consists of finding an algorithm capable of predicting the$(k+1)^{th}$ bit given the previous k bits with probability higher than $1/2$ in polynomial time (according to [3]). Any CSPRNG should

be able to produce sequences that are statistically indistinguishable from true randomness. This means that probabilities for *zero* and *one* are equal at any time and are statistically independent (stated in [4]).

2 The Discrete Fourier Transform (Spectral) Test

The NIST Test Suite is a statistical package consisting of 15 tests that were developed to test the randomness of (arbitrary long) binary sequences produced by either hardware or software based cryptographic random or pseudo-random number generators. These tests focus on different types of non-randomness that could exist in a sequence. In this section we will discuss the Discrete Fourier Transform (Spectral) test.

The focus of this test is the peak heights in the Discrete Fourier Transform of the sequence. The purpose of the DFT test is to detect periodic features such as repetitive patterns that are near each other, in the tested sequence that would indicate a deviation from the assumption of randomness. We can try to achieve this purpose by detecting whether the number of peaks exceeding the 95% threshold is significantly different than 5%.

In the following part we will introduce the mathematical concept behind DFT, and after we will present according to [5] the function call, the test statistic and reference distribution and the test description.

DFT method
The Discrete Fourier Transform is a method for converting a sequence of n complex numbers $x_0, x_1, ..., x_{n-1}$ to a new sequence of n complex numbers $X_0, X_1, ..., X_{n-1}$, where $X_k = \sum_{j=0}^{n-1} x_j e^{-2\pi ikj/n}$, for each $0 \leq k \leq n-1$. The x_i are thought of as the values of a function, or signal, at equally spaced times $t = 0, 1, ..., n-1$. The output X_k is a complex number which encodes the amplitude and phase of a sinusoidal wave with frequency $\frac{k}{n}$ cycles per time unit [6].(This comes from Euler's formula $e^{-2\pi ikj/n} = \cos(2\pi kj/n) + i\sin(2\pi kj/n)$). The effect of computing the X_k is to find the coefficients of an approximation of the signal by a linear combination of such waves. Since each wave has an integer number of cycles per n time units, the approximation will be periodic with period n. This approximation is given by the $Inverse\ Fourier\ Transform\ x_j = \frac{1}{n}\sum_{k=0}^{n-1} X_k^{e^{-2\pi ikj/n}}$.

As mentioned in [6], the DFT is useful in many applications, including the simple signal spectral analysis outlined above. Knowing how a signal can be expressed as a combination of waves allows for manipulation of that signal and comparisons of different signals: digitals files (jpg, mp3, etc.) can be shrunk by eliminating contributions from the least important waves in the combination, different sound files can be compared by comparing the coefficients X_k of the DFT, radio waves can be filtered to avoid "noise" and listen to the important components of the signal.

Function Call
The function is called $DiscreteFourierTransform(n)$, where n represents the length of the bit string. The function could take an additional parameter ϵ representing the sequence of bits generated by the RNG or $PRNG$ being tested; this exists as a global structure at the time of the function call: $\epsilon = \epsilon_1, \epsilon_2, ..., \epsilon_n$.

Test Statistic and Reference Distribution
We will denote d as the normalized difference between the observed and the expected number of frequency components that are beyond the 95% threshold. Also, the reference distribution for the test statistic is the normal distribution.

Test Description

1. The zeros and ones of the input sequence (ϵ) will be converted into -1 and 1, obtaining the sequence $X = x_1, x_2, ..., x_n$, where $x_i = -1$ if $\epsilon_i = 0$ or $x_i = 1$ if $\epsilon_i = 1$.
2. A Discrete Fourier Transform (DFT) is applied to X to produce $S = DFT(X)$.
3. M is calculated as $M = modulus(S')$, where S' represents the sequence of the first $n/2$ elements in S and $modulus$ function produces a sequence of peak heights.
4. Calculate $T = \sqrt{(\log \frac{1}{0.05})n}$, which represents the 95% peak height threshold value. Under the assumption of randomness, 95% of the values obtained from the test should not exceed T.
5. Calculate $N_0 = 0.95n/2$, which represents the expected theoretical (95%) number of peaks that are less then T (under the assumption of randomness).
6. Calculate N_1 which is the actual number of peaks in M that are less than T.
7. Compute $d = \frac{N_1 - N_0}{\sqrt{n(0.95)(0.05)/4}}$.
8. Compute P-value $= erfc(\frac{|d|}{\sqrt{2}})$, where $erfc$ represents the complementary error function: $erfc(z) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-u^2} du$.

With the final P-value resulted, we can conclude if a sequence is or is not random. If the P-value< 0.01 then the sequence is non considered random, otherwise the sequence is considered random. It is recommended that each sequence to be tested should consist of a minimum of 1000 bits ($n \geq 1000$).

3 Experiment and Results

This section contains the results of our experiments on the Discrete Fourier Transform (DFT) statistical test.
The source code for the type II error implementation and the results presented in this paper can be found at [7].

3.1 Experimental setup and Preliminaries

In order to generate input vectors and perform the statistical tests we used NIST's statistical test suite, and various tools that provide access to its features through a graphical user interface. One can download their STS implementation from [8]. (or use [9], as it also provides a good UI)

NIST's statistical test suite is designed to test that a specific input sequence is random. This can also be referred to as the null hypothesis (or H_0). Complementary to this is the alternative hypothesis (H_a), which refers to the case in which the input sequence is not random.

Before we move on to the results, it is important to understand some preliminary information about statistical hypothesis testing, and how we can interpret the results from the NIST statistical suite.

In practice, we find ourselves in one of the following situations:

1. The input data is random.
   (a) And our statistical test confirms that it is (accepts H_0). (desired behaviour)
   (b) And our statistical test fails, accepting H_a. (Type I error)
2. The input is not random.
   (a) And our statistical test states that it is random. (Type II error)
   (b) And our statistical test confirms that it isn't. (desired behaviour)

NIST's suite focuses on determining the probability of type I errors. This probability is often denoted by α and its value is usually set to 0.01.
The probability of type II (denoted by β) errors is more complex to co calculate, and it is not a fixed value. This is because in practice, non-randomness can come in many forms, resulting in different values for β.
Our contribution also includes a Python implementation for computing β according to the length of the bitstream that is provided as input, and we will showcase further on in this chapter.

Notations:

- $PRNG$ - Pseudo-Random Number Generator.
- n - number of bits in a sequence
- s - number of sequences
- $p-value$ - the main result of each statistic test comes in the form of a $p-value$, the probability that a perfect $PRNG$ would have produced a sequence that is less random than the one that was tested, based on the feature explored by the current statistical test. A $p-value$ equal to 1 means that the given input is perfectly random. For DFT, if $p-value \geq \alpha = 0.01$, the sequence passes the test and is accepted as random.
- $C1-C10$ - frequency classes for $p-values$. For example, $C1$ represents the number of $p-values$ that fall within the interval $[0.0, 0.1)$, $C2$ corresponds to $[0.1, 0.2)$ and so on.
- **Others:** LCG - Linear Congruential, CBG - Cubic Congruential, QDR - Quadratic, BBS - Blum Blum Shub, $1k = 1000$, $10k = 10000$, etc..

When working with empirical data in order to evaluate randomness, choosing the right sample size is critical in order to ensure the validity of the results. For DFT, it is important that each sequence tested is at least 1000 bits long. To gain a comprehensive overview of the DFT test, we ran tests for $n \in \{1000, 10000, 100000\}$, and also tried to experiment with a wide range of PRNG's.

3.2 NIST STS Results

Despite running multiple tests with varying parameters, we will showcase the results we believe to be most significant. The results on display were ran by setting $n = 100000$ ($100k$) (the only exception being BBS), and observing the results when increasing the number of sequences s.

PRNG	s	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	$p - value$	Proportion	Result
LCG	$1k$	116	99	112	90	95	108	113	191	80	86	0.13	0.984	✓
LCG	$10k$	1098 93 1064 ... 977										0.0	0.987	××
LCG	$100k$	11196 9701 10709 ... 10034										0.0	0.987	××
CBG	$1k$	127	90	117	78	88	104	95	121	76	104	0.0008	0.981	✓
CBG	$10k$	1280 1025 1099 ... 982										0.0	0.983	××
CBG	$100k$	12611 10138 10801 ... 9796										0.0	0.984	××
GSHA1	$1k$	102	90	122	77	113	89	93	102	117	95	0.038	0.989	✓
GSHA1	$10k$	1085 963 1103 ... 967										0.0	0.988	×
GSHA1	$100k$	11075 9760 10772 ... 10115										0.0	0.987	××
Micali	$1k$	118	100	99	76	117	105	92	93	87	113	0.0543	0.981	✓
Micali	$10k$	1150 967 1050 ... 1031										0.0	0.987	×
Micali	$100k$	11019 9675 10749 ... 10274										0.0	0.987	××
QDR1	$1k$	111	97	108	84	106	90	107	102	88	107	0.502	0.981	✓
QDR1	$10k$	1108 965 1014 ... 1002										0.0	0.986	××
QDR1	$100k$	11250 9819 10811 ... 10010										0.0	0.987	××
QDR2	$1k$	113	81	112	86	105	94	104	109	94	102	0.298282	0.987	✓
QDR2	$10k$	1166 935 1055 ... 1043										0.0	0.986	××
QDR2	$100k$	11296 9761 10687 ... 9984										0.0	0.987	××
XOR	$1k$	732	42	46	30	28	31	26	29	15	21	0.0	0.375	××
XOR	$10k$	3895 874 787 ... 635										0.0	0.737	××
XOR	$100k$	4942 1220 1154 ... 988										0.0	0.968	××
BBS*	100	13	6	6	7	9	14	12	13	4	16	0.0855	1	✓
BBS*	$1k$	111	117	76	100	102	123	67	132	74	98	0.000002	0.993	×

Table 1: Test results

To give a better insight into the results on display, for a number of sequences of $1k$, the results are conistently positive. It is worth mentioning that the treshold value that the proportion of passed tests needs to have depends on the number of sequences s: $s = 1k \rightarrow 0.980$, $s = 10k \rightarrow 0.987$, $s = 100k \rightarrow 0.989$. The $p - value$ column points out the uniformity of the computed $p - values$. Rows that have $\times\times$ in the result column fail both the uniformity test and the proportion test, while one $\times$ symbol suggests that only the uniformity test has failed.
It is clear to see that when we generate an input with a size the range of $100MB$ to $1GB$, the uniformity of the p-values and the proportion of sequences that have a $p - value \geq 0.01$ seems to increase, but not at a desired pace.
However, it is interesting to observe that the proportion of sequences that pass the test still maintains a high value for most generators.
One of the biggest problems with the NIST STS however, is that it does not account for the probability of a type II errors. If a sequence of bits passes the STS, it means that we can consider it to be random with a confidence of 99%, given $\alpha = 0.01$.
However, in some cases it is critical to also assess the probability that we have accepted a sequence to be random, when in reality it is not.

3.3 Computing the probability of type II errors.

The mathematical calculations required for this section were selected from [10], our contribution being the python implementation of the final formula.

$$\beta \ \ (p_1) = P\left(u_{\frac{\alpha}{2}} \leq \frac{N_1 - 0.95 \cdot np_0}{\sqrt{np_0q_0 \cdot 0.95 \cdot 0.05}} \leq u_{1-\frac{\alpha}{2}} \middle| p = p_1\right) =$$

$$= P\left(u_{\frac{\alpha}{2}}\sqrt{\frac{p_0q_0}{p_1q_1}} + \frac{0.95 \cdot n(p_0 - p_1)}{\sqrt{np_1q_1 \cdot 0.95 \cdot 0.05}} \leq \frac{N_1 - 0.95 \cdot np_1}{\sqrt{np_1q_1 \cdot 0.95 \cdot 0.05}} \leq u_{1-\frac{\alpha}{2}}\sqrt{\frac{p_0q_0}{p_1q_1}} + \frac{0.95 \cdot n(p_0 - p_1)}{\sqrt{np_1q_1 \cdot 0.95 \cdot 0.05}}\right)$$

$$\simeq \Phi\left(u_{1-\frac{\alpha}{2}}\sqrt{\frac{p_0q_0}{p_1q_1}} + \frac{0.95 \cdot n(p_0 - p_1)}{\sqrt{np_1q_1 \cdot 0.95 \cdot 0.05}}\right) - \Phi\left(u_{\frac{\alpha}{2}}\sqrt{\frac{p_0q_0}{p_1q_1}} + \frac{0.95 \cdot n(p_0 - p_1)}{\sqrt{np_1q_1 \cdot 0.95 \cdot 0.05}}\right)$$

Figure 1: A formula for computing β, as seen in [10]

Notations:

- p_0 - this value stands for the null hypothesis test $H_0 : p = p_0$, and $q_0 = 1 - p_0$.
- p_1 - in this scenario, we are seeking proof for the alternate hypothesis $H_a : p \neq p_0$, or $H_a : p = p_1$.
- ϕ represents the cumulative distribution function, which, when given an input x, outputs the probability that a random variable, which in our case is a normally distributed random variable, takes a value $\leq x$.
- $u_{1-\frac{\alpha}{2}}$ and $u_{\frac{\alpha}{2}}$ are quantiles of the standard normal distribution.

For a short introduction regarding statistical hypothesis tests one can view [11], and for more information regarding the probability functions used can be found at [12] and [13].

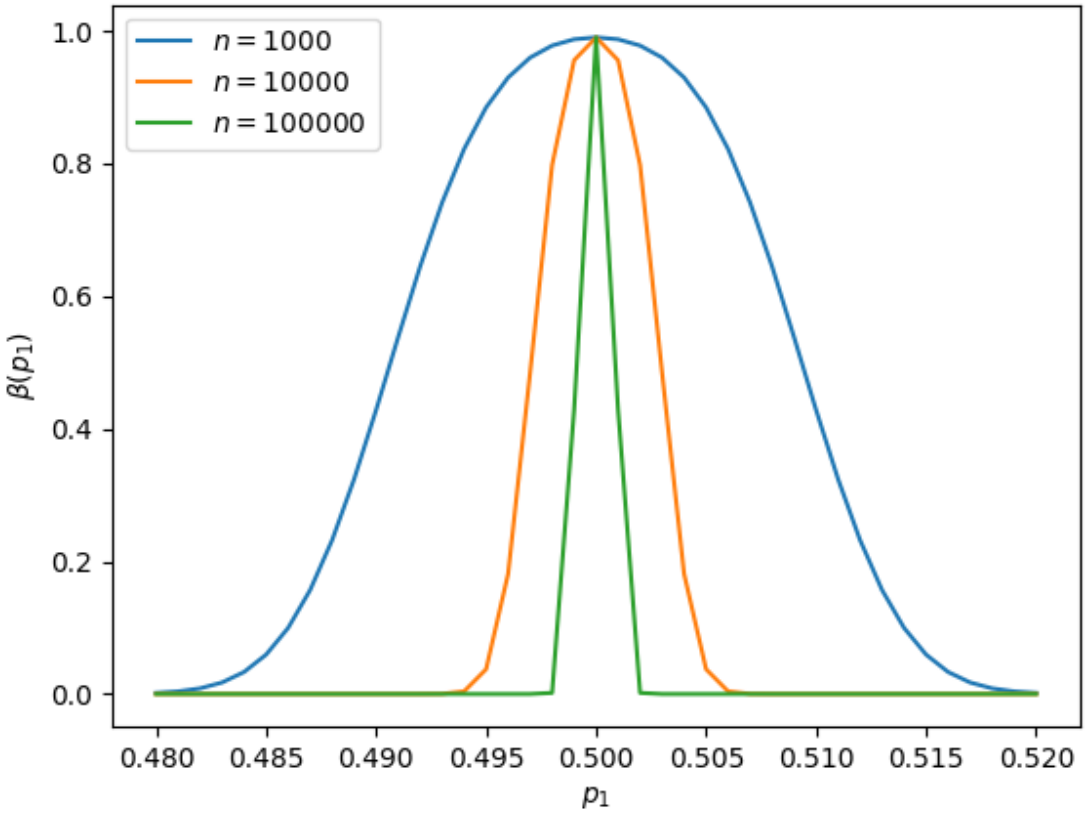

Figure 2: Results of β estimation

As we can observe from figure 2, when we increase the length of the sequence we test, the range of values of p_1 for which there exists a probability of a type II error is significantly lower, but not nonexistent. Outlining the importance of performing tests on large sequences of numbers, and the need for introducing a type II error calculation into determining the result of every NIST statistical test.

4 Conclusions and future work

Our research shows that the current version of the DFT statistical test can still be improved, and one of the first steps in that direction can be the inclusion of the type II error probability as a meaningful metric that influences the result of the test in the statistical suite.
Our results also show that the test is not as reliable (or efficient, for that matter) when we increase the sequence size and the number of sequences tested, and several improvements to the version proposed in the NIST STS have emerged in literature, that tackle just the problem that we raise in this article. A good example is [14], which aims to reduce the memory consumption and result accuracy for tests on large sequences ($10^6 - 10^7$ bits).

A great direction for future work would be exploring these open issues, as well as testing the behaviour of the test in relation to the other statistical tests present in the current NIST STS suite.

References

[1] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management – part 1: General (revision 3). *NIST Special Publication Revision*, 3, 01 2005.

[2] Amalia Beatriz Orúe López, Luis Hernández Encinas, Veronica Fernandez, and Fausto Montoya Vitini. A review of cryptographically secure prngs in constrained devices for the iot. In Hilde Pérez García, Javier Alfonso-Cendón, Lidia Sánchez-González, Héctor Quintián, and Emilio Corchado, editors, *International Joint Conference SOCO'17-CISIS'17-ICEUTE'17, León, Spain, September 6-8, 2017, Proceedings*, volume 649 of *Advances in Intelligent Systems and Computing*, pages 672–682. Springer, 2017.

[3] Divyanjali, Ankur, and Vikas Pareek. Article: An overview of cryptographically secure pseudorandom number generators and bbs. *IJCA Proceedings on International Conference on Advances in Computer Engineering and Applications*, ICACEA(2):19–28, March 2014. Full text available.

[4] Peter Kietzmann, Thomas C. Schmidt, and Matthias Wählisch. A guideline on pseudorandom number generation (prng) in the iot. *ACM Comput. Surv.*, 54(6), jul 2021.

[5] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. *NIST Special Publication 800-22, Gaithersburg, MD, US,*, 800:163, 05 2001.

[6] Discrete fourier transform. brilliant.org.

[7] https://github.com/anghelflorinm/dft-nist-tests.

[8] https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software.

[9] https://github.com/sovist/NIST-statistical-test-suite-UI.

[10] Carmina Georgescu and Emil Simion. New results concerning the power of nist randomness tests. *Proceedings of the Romanian Academy - Series A: Mathematics, Physics, Technical Sciences, Information Science*, 18:191–198, 11 2017.

[11] https://www.colorado.edu/amath/sites/default/files/attached-files/lesson9_hyptests.pdf.

[12] https://www.itl.nist.gov/div898/handbook/eda/section3/eda364.htm.

[13] https://www.itl.nist.gov/div898/handbook/eda/section3/eda362.htm.

[14] Meihui Chen, Hua Chen, Limin Fan, Shaofeng Zhu, Wei Xi, and Dengguo Feng. A new discrete fourier transform randomness test. *Sci. China Inf. Sci.*, 62(3):32107:1–32107:16, 2019.