# The Effectiveness of Randomness Test Suites for Block Ciphers

Luke Y. Tao

Massanutten Regional Governor’s School

Research

Ms. Klus

April 16, 2021

# Abstract

This study aimed to compare the effectiveness of different randomness test suites when used on block ciphers. AES256 with various modes of operation including Electronic Codebook (ECB), Cipher Block Chaining (CBC), Counter (CTR), and Output Feedback (OFB) was used to encrypt 32GB of binary zeroes to ensure randomness came from encryption and not the plaintext. The NIST and Dieharder randomness test suites were run on this ciphertext. The CryptoStat test suite was run on the included AES256 implementation. The results for NIST and Dieharder showed that CBC, CTR, and OFB were random whereas ECB failed almost every single test. AES256 passed every CryptoStat test. CryptoStat gave results consistent with NIST and Dieharder when the latter tests were run on the more cryptographically secure modes of operation: CBC, CTR, and OFB. CryptoStat may be used as an alternative to NIST and Dieharder that is unaffected by the effectiveness and randomness of different modes of operation. Further research should be conducted to replicate these results across other encryption methods.

*Keywords:* encryption, randomness, block ciphers, modes of operation, test suites, NIST, Dieharder, CryptoStat, AES

# The Effectiveness of Randomness Test Suites for Block Ciphers

In the modern age, everything from banking to taking a class can be done online. Therefore, it is important to protect the privacy and personal information of users. This is accomplished through encryption: a process in which the original information, known as plaintext, is converted into an unreadable form called ciphertext using a unique and random string called a key. One of the many properties that help determine the strength of an encryption algorithm is the randomness of the ciphertext. This property is called entropy and increases the strength of an encryption algorithm because it makes the relationship between the key and ciphertext more complex (Patil et al., 2016). Testing randomness may initially seem impossible. However, randomness, somewhat paradoxically, has certain properties and expected behavior (Gevorkyan et al., 2020). In order to test each of these properties, test suites are typically used. The randomness of a sequence or generator is generally determined by the number of tests that the sequence or generator passes (Gevorkyan et al., 2020). Each experimental result is compared to its theoretical counterpart to see how close it is to the expected result. There is little agreement among researchers when it comes to which randomness test suite to use when testing block ciphers. Popular test suites such as NIST and Dieharder were initially designed to test pseudorandom number generators and not the output of block ciphers. Consequently, their designs present a few fundamental issues when it comes to testing the randomness of block ciphers. This study aims to compare the results between them and CryptoStat, a new test suite designed specifically for block ciphers.

Although the NIST test suite, Dieharder and other randomness tests are frequently used to test the randomness of block ciphers, they are actually rather unsuited due to being originally designed for the purpose of testing pseudorandom number generators that could generate arbitrarily long sequences. This is because it is ambiguous whether their results indicate the randomness of the block cipher itself or the mode of operation that used to combine the chunks and they produce a large number of results that are difficult to interpret and compare. It is uncertain why the NIST test suite and other tests designed for PRNGs continue to be used. It may be because many people are simply unaware of the issues that modes of operation bring or that test suites designed specifically for block ciphers are relatively new and unknown.

## Literature Review

### NIST Statistical Test Suite

From 1997 to 2000, the National Institute of Standards and Technology (NIST) held a competition to decide on a new Advanced Encryption Standard (AES) to replace the deprecated Data Encryption Standard (National Institute of Standards and Technology, 2016). One of the criteria used to evaluate the encryption algorithms was the randomness of the ciphertext they produced (Soto, 1999). To help evaluate the submitted encryption algorithms, NIST created the NIST Test Suite which has been updated twice, most recently in April of 2010 (Rukhin et al., 2010). This test suite was actually designed to test random and pseudorandom number generators (Rukhin et al., 2010). However, the test suite was still used. This is likely because NIST reasoned that how the input was generated should not matter, especially since their ideal ciphertext was indistinguishable from truly random text (Soto, 1999).

The test suite asks the user to specify the number and length of the bitstreams or samples that will be tested. These bitstreams are then randomly selected from the dataset provided by the user. Each of these bitstreams then has the full gamut of tests run on it. The test calculates a p-value which is the probability that the observed value would occur by chance even if the null-hypothesis, that the sequence is random, is true. This value is used along with a critical value to determine whether or not the bitstream passes or fails the test. The p-value is also sorted into one of ten different ranges. The number of p-values in each range is expected to be uniform in a truly random sequence. Another test is performed on these ranges to determine the p-value or likelihood of this distribution occurring assuming that the dataset is truly random (Rukhin, 2011). Given the large number of tests included in the NIST test suite, as well as the number of times each test is run, test sequences will likely fail some of them. However, this does not mean that the sequence is non-random.

This test suite is still rather popular: in 2014, Ahmad and Younis used the NIST test suite in their study on the viability of using non-cryptographic hashes as random number generators. Hashes are algorithms that take an input of any size and output a string of fixed size. Similar to encryption algorithms, the randomness of its output is an important property of a hash function (Ahmad & Younis, 2014). In their study, they combined the hashed output of 2,744,529 URLs in ASCII format. Although they did not specify how they were combined to create sufficient data, it is likely that they were simply concatenated without any special procedures. This is very similar to how block ciphers encrypt the plaintext in chunks before combining them in some way.

### Dieharder

Dieharder is a random number generator testing suite. It includes a large variety of tests from many sources such as the now deprecated Diehard battery of tests, the NIST Statistical Test Suite, as well as new tests developed by Robert G. Brown himself (Brown, 2021). These tests are run on a variable number of samples until an unambiguous pass/weak/fail decision is reached for each (Brown, 2021). Similar to the NIST test suite, simply due to the large number of tests run, it is likely that some will be weak even when testing a truly random generator. It is important to note that although the program supports large file input, it is intended to test generators themselves (Brown, 2021). This study ignores this warning, which could be viewed as problematic.

### Multiple Scores

NIST and Dieharder both run a large number of different tests on a certain number of samples, either determined by the user or automatically by the suite itself. They both provide a secondary p-value that evaluates the distribution of the primary p-values for each test which makes interpretation and comparison of the results even more difficult (Rukhin, 2011). This means that the test suite outputs many statistics based on short subsequences instead of a single summary statistic from one long sequence which is undesirable for statistical inference (Rukhin, 2011). It also makes it difficult to accurately compare test results across multiple encryption algorithms because the many different results each represent different tests with different levels of reliability, accuracy and importance all run on different parts of the data set.

### Modes of Operation

AES, 3DES, and Blowfish, some of the most popular encryption algorithms in use today, are all block ciphers (Patil, et al., 2016). Unlike a stream cipher which encrypts continuously, a block cipher encrypts in chunks typically 128-bits in size. This causes a few issues for both the NIST test suite and Dieharder which both prefer arbitrarily long, continuous streams of 1s and 0s. To solve this issue, modes of operation are used to connect the different chunks.

In Electronic Codebook (ECB), the plaintext is split into chunks and then each chunk is encrypted before the ciphertext is reassembled without changing the order of the chunks (Dworkin, 2001). This mode of operation was used on eight out of nine datasets during the NIST competition (Soto, 1999). While this method is very fast, it raises quite a few issues when it comes to security and randomness testing. Since each chunk of the plaintext is encrypted independently, chunks that contain identical information will be encrypted to create identical ciphertext (Dworkin, 2001). A classic example of this is the ECB penguin, in which a simple image of a penguin is encrypted, but because many of the chunks contain identical colors, the encrypted image did not conceal enough information and still shows the pattern of a penguin.

Another common mode of operation is Cipher Block Chaining Mode (CBC) which also splits the plaintext into chunks before encrypting them, however the ciphertext for each chunk is used to perform an exclusive or (XOR) operation with the plaintext of the next chunk before it is encrypted (Dworkin, 2001). The first chunk is XORed with an initialization vector (IV). This makes encryption much slower as each chunk’s encryption is dependent on the one before it.

Output Feedback Mode (OFB) makes the block cipher into a stream cipher. The IV is encrypted using the cipher to create an output block. This output block is then XORed with the plaintext to create the ciphertext (Dworkin, 2001). The output block is also encrypted again to create a new output block which is then XORed with the next part of the plaintext (Dworkin, 2001). This process is repeated until the entire plaintext is encrypted.

Counter Mode (CTR) also makes a block cipher into a stream cipher. Instead of an IV it uses a counter. The counter is encrypted and then XORed with the plaintext to create the ciphertext (Dworkin, 2001). The counter is then incremented before being encrypted again and then XORed with the next portion of the plaintext (Dworkin, 2001). This process is repeated until the entire plaintext is encrypted. This mode allows for encryption or decryption in parallel because each part is independent of the others. This makes the speed basically on par with ECB which also allows for encryption or decryption in parallel, but is much less secure.

### CryptoStat

When a block cipher is converted into a pseudorandom number generator using these methods, it is unclear whether the randomness of the block cipher is being tested or the randomness of the mode of operation (Kaminsky, 2019). To solve this problem, Kaminsky, a professor at the Rochester Institute of Technology, created CryptoStat, which instead of converting block ciphers into pseudorandom number generators, directly tests the mapping of the block cipher. A map in the context of computer science can be thought of using IDs. Each ID corresponds to a certain person. In this case, the ID is the plaintext and the person is the ciphertext. By directly testing the block cipher mapping, Kaminsky aims to circumvent the issue that modes of operation, especially ECB, bring. CryptoStat also avoids the issue of multiple scores, as it combines all of its results into one “randomness margin” at the end (Kaminsky, 2019). CryptoStat uses odds ratio tests which apply Bayseian thinking to binomial distributions (Kaminsky, 2019). Bayes’ theorem states that the probability of event A given event B is true is equal to the probability of B given A is true multiplied by the probability of A divided by the probability of B or P(A|B) = P(B|A)\*P(A)/P(B). This allows for the results of many different tests to be easily combined. While Kaminsky’s logic and research appear to be rather sound, it is important to note that his article is only published in the Cryptology ePrint Archive which is meant for recent research on cryptology, but only has a brief review process and is therefore somewhat unreliable.

# Methods

## Setup

/path/to/ indicates that the location of the specified file or folder may vary and should be customized by the user.

If permission is denied for any of the shell scripts, use sudo chmod 755 /path/to/file.

This study was conducted on the Ubuntu operating system, although it could have been performed on any Linux system. Ubuntu 20.04 LTS was installed alongside Windows 10 on a separate drive in a dual boot setup. An NVIDIA graphics card that supports CUDA is required to run CryptoStat. This experiment used an NVIDIA GEFORCE GTX 1660 SUPER.

Execute dd bs=4000000 count=8000 if=/dev/zero of=/path/to/plaintext in the termina to create a plaintext file of 32GB of binary zeroes using dd which ensures all randomness is from encryption and not the plaintext.

## Ciphertext

Execute sudo apt install openssl in the terminal

The file was encrypted with AES paired with four different modes of operation using OpenSSL. The key used for each encryption was 8d19c708c2fa60012c558d98454a2018e5c7bb0e62d05ff60972f10750279323 and the initialization vector used for those that required one was 00000000000000000000000000000000.

Salt was removed to ensure that randomness only came from encryption.

### Electronic Codebook

openssl enc -aes-256-ecb -pbkdf2 -nosalt -K 8d19c708c2fa60012c558d98454a2018e5c7bb0e62d05ff60972f10750279323 -in /path/to/plaintext -out /path/to/ciphertextECB

### Cipher Block Chaining

openssl enc -aes-256-cbc -pbkdf2 -nosalt -K 8d19c708c2fa60012c558d98454a2018e5c7bb0e62d05ff60972f10750279323 -iv 00000000000000000000000000000000 -in /path/to/plaintext -out /path/to/ciphertextCBC

### Output Feedback

openssl enc -aes-256-ofb -pbkdf2 -nosalt -K 8d19c708c2fa60012c558d98454a2018e5c7bb0e62d05ff60972f10750279323 -iv 00000000000000000000000000000000 -in /path/to/plaintext -out /path/to/ciphertextOFB

### Counter

openssl enc -aes-256-ctr -pbkdf2 -nosalt -K 8d19c708c2fa60012c558d98454a2018e5c7bb0e62d05ff60972f10750279323 -iv 00000000000000000000000000000000 -in /path/to/plaintext -out /path/to/ciphertextCTR

## NIST Statistical Test Suite

### Installation

Download and extract the file from <https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software>. Type make in the terminal while inside the sts-2.1.2 directory to compile the program. This should create an executable named “assess” inside the sts-2.1.2 directory.

### Usage

Execute ./assess 100000 in the terminal while inside the sts-2.1.2 directory. This runs the program and tells it that samples or bitstreams should be 100,000 bits in length. Next type 0 for generator options, /path/to/ciphertextECB for user prescribed input file, 1 for statistical tests, 0 for parameter adjustments, 100 for how many bit streams should be generated, and 1 for input file format. This uses default options and tells the program to use 100 different samples or bitstreams and tells the program that the file is in a binary format. Results will appear in /path/to/sts-2.1.2/experiments/AlgorithmTesting/finalAnalysisReport.txt. Repeat this process for each mode of operation by changing /path/to/ciphertextECB.

## Dieharder

### Installation

Execute sudo apt install dieharder in the terminal.

### Usage

Execute dieharder -a -g 201 -f /path/to/ciphertextECB in the terminal.

Copy paste displayed results into a file.

Repeat this process for each mode of operation by changing /path/to/ciphertextECB.

## CryptoStat

### Installation

Execute sudo apt install default-java in the terminal.

Execute sudo apt install nvidia-cuda-toolkit-gcc in the terminal.

Execute sudo apt install gcc-8 in the terminal.

These install prerequisites for the following programs.

Download the source distribution of Parallel Java 2 Library (PJ2) from

<https://www.cs.rit.edu/~ark/pj2.shtml>

Download the source distribution of CryptoStat from <https://www.cs.rit.edu/~ark/parallelcrypto/cryptostat/>

Although both of these files are .jar files, they should still be extracted and placed in convenient locations.

Execute the following commands in the terminal each time you reopen it.

export CLASSPATH=/path/to/CryptoStat/lib:/path/to/pj2/lib

export LD\_LIBRARY\_PATH=/path/to/pj2/lib/

Install PJ2’s GPU capability

export GCCBINDIR=/usr/bin/

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 80 --slave /usr/bin/g++ g++ /usr/bin/g++-8 --slave /usr/bin/gcov gcov /usr/bin/gcov-8

Navigate to /path/to/pj2/lib/ and change the ccompile shell script to

#!/bin/bash

echo "gcc -I/usr/lib/jvm/default-java/include -I/usr/lib/jvm/default-java/include/linux -I/usr/lib/cuda/include -I/usr/local/dcs/jdk/include -I/usr/local/dcs/jdk/include/linux -I/usr/local/cuda/include -shared -fPIC -o libEduRitGpuCuda.so edu\_rit\_gpu\_Cuda.c -lcuda"

gcc -I/usr/lib/jvm/default-java/include -I/usr/lib/jvm/default-java/include/linux -I/usr/lib/cuda/include -I/usr/local/dcs/jdk/include -I/usr/local/dcs/jdk/include/linux -I/usr/local/cuda/include -shared -fPIC -o libEduRitGpuCuda.so edu\_rit\_gpu\_Cuda.c -lcuda

Execute ./ccompile in the terminal to compile the edu\_rit\_gpu\_Cuda.c source file and create the libEduRitGpuCuda.so native code library file.

Navigate to /path/to/pj2/lib/edu/rit/gpu/example/ and edit the ccompile file.

Replace every instance of compute\_20 with compute\_30.

Execute ./ccompile in the terminal.

Navigate to /path/to/pj2/lib/edu/rit/gpu/test/ and execute ./ccompile in the terminal.

Recompile CryptoStat

export CRST\_HOME=/path/to/CryptoStat

Navigate to /path/to/CryptoStat/ and edit the ccompile file.

Replace every instance of compute\_20 with compute\_30.

Navigate to /path/to/CryptoStat/ and edit the jcompile file.

Edit SOURCE\_HOME=/path/to/CryptoStat/lib

Edit PJ2\_HOME=/path/to/pj2/lib

Edit JDK\_HOME=/lib/jvm/default-java

Navigate to /path/to/CryptoStat/ and execute ./compile in the terminal.

### Usage

Execute java pj2 Analyze -v "edu.rit.aes.AES256()" "edu.rit.crst.Single(00000000000000000000000000000000)" "edu.rit.crst.Rand(100,123456789)" "edu.rit.crst.Direct()" "edu.rit.crst.Adjacent(8)" in the terminal. Copy paste displayed results into a file.

# Results

**Table 1**

*NIST Statistical Suite*

| Mode of Operation | Passed | Failed |
| --- | --- | --- |
| ECB | 1 | 187 |
| CBC | 187 | 1 |
| OFB | 187 | 1 |
| CTR | 185 | 3 |

*Note*.

**Table 2**

*Dieharder*

| Mode of Operation | Passed | Weak | Failed |
| --- | --- | --- | --- |
| ECB | 0 | 0 | 114 |
| CBC | 111 | 3 | 0 |
| OFB | 110 | 4 | 0 |
| CTR | 114 | 0 | 0 |

*Note*.

**CryptoStat**

Nonrandom rounds = 0/14

Randomness margin = 1.00000

Randomness margin sounds may seem to be a rather complex term but is actually the result of the number of random rounds divided by the number of total rounds. In this case, the number of random rounds is 14 out of 14, so the randomness margin is 1.

# Discussion

# 

# References

Ahmad, T., & Younis, U. (2014). Randomness testing of non-cryptographic hash functions for real-time hash table based storage and look-up of URLs. *Journal of Network and Computer Applications*. 41, 197-205. [doi.org/10.1016/j.jnca.2013.11.007](https://doi.org/10.1016/j.jnca.2013.11.007)

Brown, R. G. (2021). Dieharder: A Random Number Test Suite. *Duke University Physics Department*. <https://webhome.phy.duke.edu/~rgb/General/dieharder.php>

Dworkin, M. (2001). *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. United States Department of Commerce, National Institute of Standards and Technology. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf>

Gevorkyan, M.N., Demidova, A.V., Korol’kova, A.V. & Kulyabov, D.V. (2020). A Practical Approach to Testing Random Number Generators in Computer Algebra Systems. *Comput. Math. and Math. Phys*. 60, 65–73. [doi.org/10.1134/S096554252001008X](https://doi.org/10.1134/S096554252001008X)

Kaminsky, A. (2019). Testing the randomness of cryptographic function mappings. *Cryptology ePrint Archive*. [eprint.iacr.org/2019/078.pdf](https://eprint.iacr.org/2019/078.pdf)

National Institute of Standards and Technology. (2016). *AES Development - Cryptographic Standards and Guidelines*. United States Department of Commerce. [csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development](https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development)

Patil, P., Narayankar, P., Narayan, D.G., & Meena S.M. (2016). A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish. *Procedia Computer Science*, 78, 617-624. [doi.org/10.1016/j.procs.2016.02.108](https://doi.org/10.1016/j.procs.2016.02.108)

Rukhin, A. (2011). Statistical Testing of Randomness: New and Old Procedures appeared as Chapter 3 in *Randomness through Computation,* H. Zenil ed. World Scientific, 2011, 33-51. [doi.org/10.1142/9789814327756\_0003](https://doi.org/10.1142/9789814327756_0003)

Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., & Vo, S. (2010). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. United States Department of Commerce, National Institute of Standards and Technology. [tsapps.nist.gov/publication/get\_pdf.cfm?pub\_id=906762](https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=906762)

Soto, J. (1999). *Randomness Testing of the Advanced Encryption Standard Candidate Algorithms* (NISTIR 6390). United States Department of Commerce, National Institute of Standards and Technology. [tsapps.nist.gov/publication/get\_pdf.cfm?pub\_id=151193](https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=151193)