Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 3, Issue 11, 2025

AN IMPROVED STATISTICAL TEST SUITE FOR CRYPTOGRAPHIC ALGORITHMS

Rashid Ali

Riphah Institute of Systems Engineering, Riphah International University,Islamabad, Pakistan

engr.rashid.ali91@gmail.com

DOI: https://doi.org/10.5281/zenodo.17797482

Keywords
Cryptography, NIST Statistical Test Suites (STS), Advanced Encryption Standard (AES), User Interface (UI), Randomness Testing (Entropy), cumulative sums (Cusum), Linear Feedback Shift Register (LFSR), Man in the Middle (MIM)

Article History
Received: 01 October 2025
Accepted: 19 November 2025
Published: 29 November 2025

Corresponding Author: *
Rashid Ali

Abstract
Information Security is one of the biggest challenges of today's world and without cryptography Information Security cannot meet the current challenges. In Information security, cryptography is practiced to accomplish numerous goals together with CIA triad e.g. confidentiality, integrity, and authentication of data we sent via internet. One of the most important properties of a cryptographic system is a proof of its security. There are multiple cryptographic algorithms and applications that can be used. Amongst them, a user needs one of high performance with low cost. Similarly, there are different techniques to evaluate strength of ciphertext and to assess strength of these encryption algorithms and applications. These Evaluation parameters are Encryption time, Decryption time, Memory used by cryptographic application to generate ciphertext, Avalanche effect and Entropy. Entropy, Randomness testing plays a vital role in cryptography. Due to severe limitation in existing NIST STS and other software used for randomness testing this paper has proposed a better an improved statistical test suite. All the codes are rewritten in python. This software is tested with the same test data that is provided in the NIST document and satisfactory results have been found. The codes have been put together in a software package that can be deployed on linux server. In the development process much help was derived from the C implementation of the NIST test suite by NIST as well as an earlier code of the NIST suite in r4nd0m by StuartGordonReid.

INTRODUCTION

Modern world faces lot of challenges and one of major challenge amongst them is information security.The internet is the essential part of any organization and individual person daily life. Cryptography plays key role to overcome these challenges. Foreample there are different cryptographic algorithms used to encrypt private/personal data or information. After the invention of these algorithms many attacks are launched to break these algorithms to access personal/confidential data. cyber attacks include brute force, linear cryptanalysis, differential cryptanalysis, related key cryptanalysis and MIM known as man in the middle attack but are not limited to only these attacks. To avoid such attacks, it is required to assess these algorithms by evaluating strength of ciphertext generated after applying technique/algorithm It is required that strength as well as performance of each algorithm is known in order to put on certain cryptographic algorithm to any application. There are some features grounded on which these algorithms are examined which are Encryption time, Decryption time, Memory used, Avalanche effect and Entropy [1][19]. Entropy, Randomness testing has key role amongst all motioned features. Entropy is measure of randomness in any bit string. It is used to calculate randomness in cipher text. All randomness tests verify assumptions, that are considered true for a truely random bitstream. These assumptions are.

A truely random number has the same number of 1s and 0s.

One can not predict the n+1 bit by knowing the 0..n's bits.

Randomness of information is typically examined by NIST STS [2], TestU01 [3], Diehard [4], ENT [5] and CryptX [6] e.g. NIST convert the AES block ciphers into PRNGs and and then the NIST test suite tests are applied[20]. Most prevalent among all is NIST STS since it is used for testing and certification of random number generators in cryptographic applications. For randomness analysis NIST STS is considered to be a significant testing suit. But still lot of challenges are faced by users while using NIST STS for randomness testing of ciphertext generated by different cryptographic applications [9]. All these challenges are written in details in section 3. This paper proposes an improved version of Statistical test suites by solving the problems that are found in NIST STS. The improved STS has given facility to the end user to choose a test or few tests together or all tests by providing a clear interface to user. The results are also shown to the user on UI which are much clear as compared to NIST STS. For all fifteen tests NIST STS has some criteria as expounded in table 1 [10]. Implementation is done for all fifteen tests according to standards set by NIST NTS. An improved STS code perform properly producing the results almost equal with those of the NIST results.

Table 1: Minimum length required by NIST STS

Test No.	Test Name	Minimum Requirement
1	Frequency Test	100
2	Frequency Test within a block	9,000
3	Runs Test	100
4	Longest Run of ones in a block	128
5	Binary Matrix Rank Test	38,912
6	Discrete Fourier Transform Test	1,000
7	Non-overlapping Template Test	10,48,576
8	Overlapping Template Test	10,00,000
9	Maurer's "Universal Statistical" Test	13,42,400
10	Linear Complexity Test	10,00,000
11	Serial Test	10,00,000
12	Approximate Entropy Test	100
13	Cumulative Sums Test	100
14	Random Excursions Test	10,00,000
15	Random Excursions Variant Test	10,00,000

Overvie Of NIST STS

NIST STS is published in April 27, 2010. The project is developed in C language. This software has fifteen tests which are briefly described below. As this paper proposed an improved version of NIST STS, so it is necessary to have clear understanding of all the fifteen tests of the NIST STS. Figure 2 shows the default interface of NIST STS showing all of its fifteen tests which are applied for randomness test over bitstream.

Fig.2. Default view of NIST STS

Here all the tests are briefly described. To understand its computation, we suggest reading NIST publication 800-22 [1]. We have focused on types of defects for which these statistical tests were designed. Table 2 explains general characteristics of each statistical test of NIST suite.

Frequency (Monobits) Test

Monobits demonstrates if the frequencies of 1 and 0 over the whole n bit succession are around equivalent that is the extent of 0s is near ½. In a genuine irregular arrangement, the number of zeros out of a bit stream is around equivalent to number ones.

Frequency Test within a Block

This Test inside a Block is intended to ensure that frequencies of 1 and 0 are similarly disseminated over the entire n-bit succession.

The tried stream is isolated into different hinders, every one of which has length M. The rest part, if exist, is disposed of. Along these lines, there will be completely N = $\left[\frac{n}{m}\right]$ non-covering squares.

Runs Test

The reason for the runs test is to test whether the conduct of changes in a grouping meets the paradigm of irregular succession. This test centers the check of runs where continuous indistinguishable bits in a succession is called run.

Longest Run of Ones in a Block

This test for the most part centers on the longest keeps running of 1s in M-bits squares. For each square with a length of M, the test will execute in excess of multiple times, at that point record the longest keeps running of 1s and use measurement to assess the P-esteem, so the length of the stream ought to be a lot greater than the square size. The reason for this test is to distinguish whether the length of the longest keeps running of 1s in the tried stream is predictable with the length of that would be normal in a genuinely irregular piece stream.

Binary Matrix Rank Test

The reason for the double framework rank test is whether the circulation of the position of disjoint sub-lattices, 32x32 piece networks concurs with the hypothetical probabilities. In this test direct reliance of fixed length substrings of the first piece, stream is checked. Each Block by a framework of M lines and Q segment to such an extent that N= [n/MQ], the staying unused bits are disposed of its calculation incorporates the accompanying strides to compute p-value.

Discrete Fourier Transform (Spectral)

This test is utilized to discover intermittent highlights, occasional highlights are additionally called redundant examples that are much closed to each other. Its computation includes the following steps to calculate p-value.

Non-overlapping Template Matching Test

The motivation behind the test is to recognize number of events for a predetermined no occasional layout concurs with the number expected genuinely arbitrary grouping. This test serves to distinguish application that produce such a large number of cases of a given non-intermittent (aperiodic) design. Its calculation incorporates the accompanying strides to compute p-esteem.

Overlapping Template Matching

Through this test one expects to recognize layout coordinating in a covering way, that is, it searches for events of the pre-specified bit string and to check whether the number of such events as against an arrangement under the suspicion of arbitrariness portrayed in NIST production.

Maurer's "Universal Statistical"

The widespread test centers to gauge separates regarding L-bit square numbers between L-bit coordinating examples. For measurement conveyance whole of log 2 separates between L-bit coordinating examples is determined. This test chooses whether a succession could be essentially compacted or not. An essentially compressible arrangement is non-irregular.

Linear Complexity Test

A long piece stream is typically acquired from a Feedback Shift Register (LFSR). The bit stream from which the more LFSR is gotten can be considered as irregular, while the shorter LFSR shows non-haphazardness. A calculation called berlekamp-Massey is utilized to acquire an LFSR. This test looks for the length of LFSR and finds if the bit succession from which the LFSR is acquired is arbitrary or not. Its calculation incorporates the accompanying strides to figure p-esteem.

Serial Test

In a long n-bit arbitrary grouping, each m-bit design has a similar possibility of showing up as each other m-bit designs. In n-bit succession, every one of all m-bit designs is relied upon to happen A m times, where A m = n/2m. The sequential test checks the recurrence of all conceivable covering m-bit designs everywhere throughout the n-bit grouping and upon the deviations of every one of all tallies together one means to check whether the arrangement can be named as arbitrary or not.

Approximate Entropy

Entropy is haphazardness which dependent on rehashing designs [1]. Bigger is the haphazardness, bigger is entropy. This test looks at the cycle of covering squares of two adjoining/back to back lengths (m and m+1) against the normal outcomes for an ordinarily conveyed bit stream. For n-bit string, the entropy is estimated by looking at the recurrence of covering examples of all conceivable m-bit designs with that of (m+1)- bit designs. The correlation between entropies of m-bit and (m+1)- bit designs is named as rough entropy, ApEn (m), which is thought about against the normal aftereffect of an irregular arrangement. To put it plainly, this test decides if a succession seems more ordinary than anticipated from a genuine irregular grouping.

Cusum

This test looks whether 1s or 0s are produced in the huge arrangement at starting stages or at later stages or 0s are intermixed equally over the whole grouping. In an irregular number stream, the number of 0s ought to be generally equivalent. This is a fundamental agreement and has been connected into tests by the remarkable way. This test reaffirms this standard. It gives two filtering requests and utilizes a parameter mode to control: mode = 0 implies sweep forward and mode = 1 for in reverse.

Random Excursions

The Random Excursions Test means to look if the quantity of visits to an aggregate wholes state in a cycle falls into a classification that is expected of arbitrary arrangement. Eight states, for example - 4, - 3, - 2, - 1 and +1, +2, +3, +4 are explored – visits to states more noteworthy than +4 are clubbed inside the visits to +4 state and visits to states lesser at that point - 4 are clubbed inside the visits to - 4 states. Its calculation incorporates the accompanying strides to ascertain p-esteem.

Random Excursions Variant

This test searches for check of visits to a state in total wholes of arbitrary example over the whole piece stream and compute deviations from expected number of visits in the irregular example thinking about haphazardness. There are 18 states, for example s = – 9, – 8, – 7, – 6, – 5, – 4, – 3, – 2, – 1, +1, +2, +3, +4, +5, +6, +7, +8, +9 which are considered.

Table 2: Characteristics of the NIST STS Tests

#	Test (Statistical)	Error Detected
1	Monobit	Lots of ones or zeroes
2	Monobit Test inside Block	Lots of ones or zeroes at earlier or later stage of the sequence.
3	Longest Runs of Ones	Deviation of the distribution of long runs of the bitstream
4	Runs	Large (small) total number of runs indicates that the oscillation5 in the bit stream is too fast (too slow).
5	Rank	Deviation of the rank distribution from a corresponding random sequence, due to periodicity.
6	Spectral	Periodic features in the bit stream
7	Non-overlapping Template Matchings	Too many occurrences of non-periodic templates.
8	Overlapping Template Matchings	Too many occurrences of m-bit run of ones.

9	Universal Statistical	Compressibility (Regularity)
10	Linear Complexity	Deviation from the distribution of the linear complexity for finite length (sub)strings.
11	Serial	Non-uniform distribution of m-length words. Like Approximate Entropy.
12	Approximate Entropy	Non-uniform distribution of m-length words. Small values of ApEn(m) imply strong regularity.
13	Cumulative Sums	Too many zeroes or ones at the beginning of the sequence
14	Random Excursions	Deviation from the distribution of the number of visits of a random walk to a certain state.
15	Random Excursions Variant	Deviation from the distribution of the total number of visits (across many random walks) to a certain state.

NIST STs Challaenges

As we know previously, for entropy testing NIST STS is considered to be finest tool and for assessment of the candidates for AES (FIPS PUB 197), NIST 800-22 was one of the tool of cryptography. While working with this software users still face some contests which are deliberated underneath briefly.

As there are two generator for random number, RNG and PRNG, NIST STS deals with binary data either generated by RNG or PRNG. This software does not have support to direclty execute entropy tests on ciphers generated by different cryptographic applications. Either changes are required in the implementation of cryptographic applications to produce cipher texts in form of binary or to convert cipher texts into binary form which will be then feeded to NIST STS for entropy test. As this tool takes data only in binary format. In short NIST STS test are performed only when ciphers are translated into binary first.

NIST STS can be installed on Linux only as most of users don't find Linux much user friendly so they find it grim to work in Linux Environment so here is space for improvement in NIST SIS software.

AS NIST STS is text-based software poorly handle exceptions and errors. It is hard for end user to use this software as most of them are not familiar with the technical errors shown to them. Am improvement is needed by providing User interface of this software.

End User found difficulties while understanding the results generated by NIST STS. User must go to results folder to observe result of each test. User finds it difficult to decide whether there is randomness on the sequence tests are executed. Interprets results on testers own understanding.

This is time consuming software as NIST STS consider randomness in data if all of its tests are passed. But even for a truly random data there is more than 80% probability of failing few NIST STS tests. Incase if few tests are failed for data, NIST STS recommends the analysis of additional samples. So in this case it required more time in executing all tests for different samples of data. Hence it is very hard to practice this software in case of errors and exceptions for real time usage.

Related work and their limitations

In cryptographic applications based on some algorithm like AES or a block cipher, there are three main algorithms which are encryption, decryption, and schedule algorithm used for key. The encryption/decryption applications and algorithms are evaluated through statistical tests [9]. Whereas evaluation of Key Schedule Algorithm (KSA) is out of our scope. The security of any block cipher also based on the strength of KSA. If the KSA is weak then the security of any strong encryption algorithm might be compromised [10]. Other evaluation features for block cipher includes Encryption time, Decryption time, Memory used, Avalanche effect which are also ignored in this literature work. Few known statistical software packages are available for randomness testing of the cipher text produced by different cryptographic applications. Evaluating the quality of randomness of a given data stream is a challenging task. In practice, randomness assessment heavily relies on empirical tests of randomness. Each test has its own purpose. However, even a good random number generator sometimes produces sequences with characteristics significantly different from the values expected in tests. Therefore, we are unable to distinguish with certainty whether a generator is suitable or not. In such scenario NIST suggest increasing number of samples to be tested for a generator. Statistical randomness is evaluated from many points of view, for that purpose statistical tests are put into group in different testing suites to provide more comprehensive randomness analysis. One of the first compact sets of randomness tests was the Diehard computer package of statistical Tests of Randomness developed by G. Marsaglia. Soon after, the Statistical Test Suite by the NIST and the theoretical summary by D. Knuth followed. The NIST compute package of statistical tests has a special importance since it was published as a NIST standard and is still used for the preparation of many formal certifications or approvals (for example for selecting AES) [11]. As these suites ceased to be maintained and improved, new efforts for a modular and extensible testing framework arose. These included Die harder, TestU01 and ENT to name just a few. Table 3 has been taken from a research paper "Statistical Testing of Cryptographic Randomness" [12] in which all the available statistical batteries are summarized.

Table 3: Randomness Testing Batteries and Tests

Knuth (11)	frequency, serial, gap, poker, coupon collector's, permutation, run, maximum-of-t, collision, birthday spacings, and serial correlation
Diehard (16) Die harder (26)	birthday spacings, overlapping permutations, ranks of matrices, monkey tests, count the 1s, parking lot test, minimum distance test, random spheres test, squeeze test, overlapping sums test, runs test and craps test
NIST (15)	frequency, frequency within a block, runs, longest run of ones in a block, binary Matrix rank, spectral, non-overlapping template matching, overlapping template matching, universal Statistical, linear Complexity, serial, approximate entropy, cusum, random excursions, random Excursions variant
Helsinki (4)	Autocorrelations, cluster, n-block and random walk
ENT (2)	arithmetic mean, chi-square test, entropy test, monte-carlo pi estimation and serial correlation
Crypt-X (6)	binary derivative, change point, frequency, linear complexity, runs and sequence complexity
SPRNG (13)	blocking, collisions, coupon collector, fourier, frequency transform, gab, lsing model, maximum-of-t, permutations, poker, random walk, runs and serial
TestU01	small crush (96), crush (144) and big crush (106)

As shown in Table 4 all the existing technologies is lacking User friendly interface, Also, they do not provide a method to be used for custom cryptographic application based on some encryption algorithm. Because of such limitation it is need of the day to have an improved version on the best computer package from table 2.3 with new features. Based on the other feature and importance that NIST Statistical suite has we proposed its improved version.

Table 4: Summary Table of Related work

Features	User Friendly	Configuration complexity	Accuracy	Open source	Support for Hex
NIST STS	×	×	✓	×	×
SP800_22_tests	×	×	×	×	×
sts2.1.1 statistical tests	×	×	×	×	×
Python 2.6 by Ilja	×	×	×	×	×
R4nd0m	×	×	×	×	×
TestU01	×	×	N/A	×	×
Diehard	×	×	N/A	×	×
ENT	×	N/A	N/A	×	×
Crypt-XS	×	N/A	N/A	×	×
Improved STS	✓	×	✓	✓	✓

Proposed Solution and Methodology

As discussed in section (III) that NIST STS has some area where improvement is required. This paper has enlisted all those areas and has proposed solution that helped us to develop better version of STS with a title of an improved statistical test suites for cryptographic applications. Such type of efforts is done previously, code written in python for NIST statistical test suite with same objective, but we have found compilation and accuracy errors in their code. We have reviewed that existing code solved its code and added new features in it. Our improved version of statistical test suite is user friendly and easy to use. It can be easily deployed on server and end user can access it via any browser, because of this approach it has resolved configuration efforts that are required for most of statistical test suites to be setup. All the tests are tested with test data provided in NIST publication and results are obtained with 100 percent accuracy. Most of the statistical test suites does not provide direct support for encrypted data in hex format. An improved statistical test suite has a module which can convert cipher texts into binary form upon which entropy tests are computed. So, this software has a support to execute NIST STS directly on cipher texts generated by different cryptographic applications.

An improved statistical test suite will ask user for input a sequence. There are two ways to provide that sequence either by generating it using AES module of the program or by providing a path of ciphertext produced by any other cryptographic algorithm. This program supports input in two formats i.e. binary format and hex format. Data validator will ensure that user has provided binary sequence in correct format, program stops its execution on wrong input and will show prompt to the user. With valid sequence program will call NIST_STS module and will execute statistical tests to compute p value/values for each test. At the End Results is shown to the user that can be saved to a file.

Following are different components of the improved statistical test suite.

Python

Python is a translated, abnormal state, broadly useful programming language. Made by Guido van Rossum and first discharged in 1991. It has 2 noteworthy discharges 2.6 and 3.6. An improved measurable test suite has been created in python in light of the fact that SciPy and NumPy are two open source libraries of python that has every one of the capacities need to compute measurements dependent on which p esteem is determined in the various trial of the NIST factual test suite. Additionally, we have discovered code on GitHub of the NIST factual tests. This code helped us being developed of an improved measurable test suite.

We have made it portable meaning that this code will be executed on both releases of python without any error.

Flask

This is a framework used in web application suited for most of the python program. It is classified as a micro framework because it does not require particular tools or libraries. Based on our requirement we have integrated our statistical tests code with Flask for to make it web based by providing a user-friendly interface to end user.

Advance Encryption Standard (AES) Module

AES, is encryption decryption algorithm [18]. This module contains a program that produce bit stream that is required to execute statistical tests. This program is based on Crypto library of python.

NIST_STS Module

All statistical test of NIST are written in this module, code is function based meaning that separate function is written for each test. Each function accepts a sequence, do statistical calculation on it and return p value.

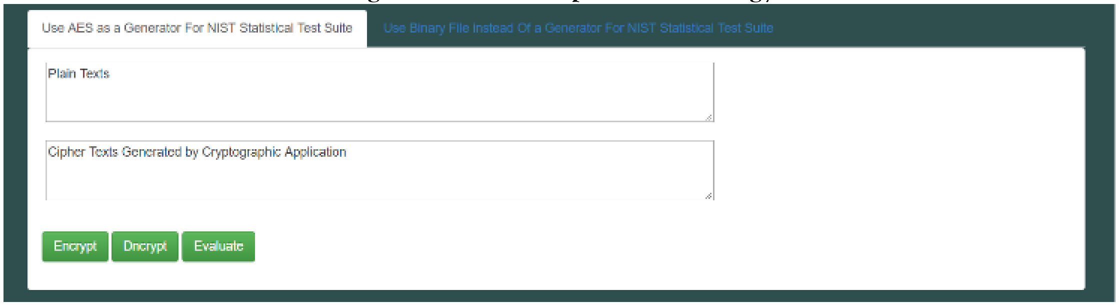
Implementation and Results

Following are different component of the improved statistical test suite. Each component of the system is explained in detail along with appropriate figure.

Data Input

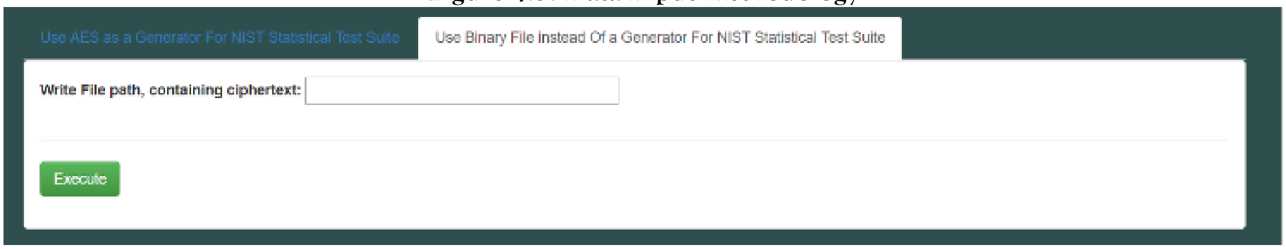
There are two ways for the input. In first method user is asked to choose AES cryptographic algorithm for evaluation. In Figure 4.1 the text area control is provided to user which takes required amount of data which are then encrypted via AES and bit of stream is generated by clicking Encrypt button from the interface shown below, that is used in statistical tests.

Figure 4.2: Data Input methodology



In second method user is asked to provide path of the file. That file must contain string of binary or ciphertext in form of hexadecimal generated by some cryptographic algorithm. If ciphertext is given in hexadecimal form, then the contents are first given to bit convertor module of the program which make binary sequence from hexadecimal form this sequence is used in statistical tests. This whole functionality is done by clicking Execute button as shown in Figure 4.2.

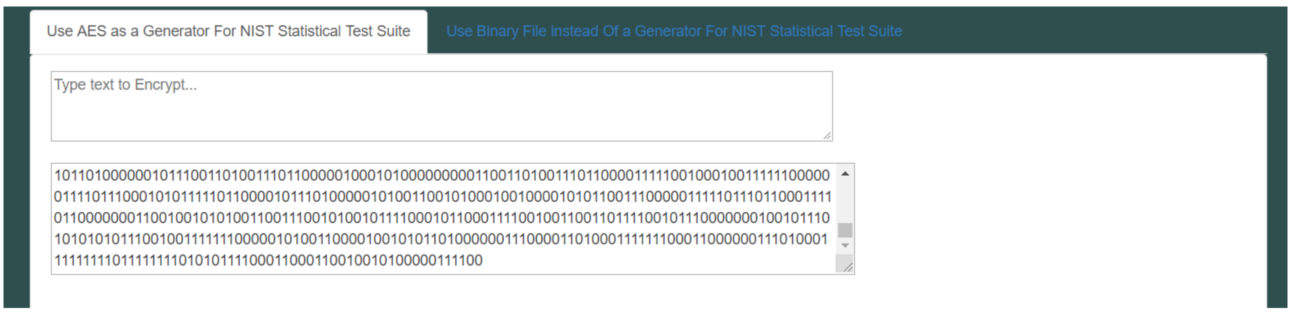
Figure 4.3: Data Input methodology



Data Output

From the input data a bit stream is produce that is then given to statistical tests for further process. In Figure 4.4 shows that user can view that data on interface.

Figure 4.4: Data Output inform of Bit Stream



Statistical test to Evaluate Cryptographic Algorithm

Once binary sequence is provided or generated via AES built-in program it is then pass the NIST_STS module which calculate p values for all defined statistical tests on provided binary sequence. The compiled results are then given to manager module of the program. This functionality is done when user click on Evaluate button as shown in Figure 4.3.

Output of Statistical Tests

Manager module of the program is responsible for collecting results from NIST_STS module of the program and showing these results to user in tabular form with appropriate status either pass/fail. See Figure 4.5.

Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 3, Issue 11, 2025

Figure 4.5s: Data Statistical Tests Output

Report Generated by an Improve Statistical test suite

#	TEST NAME	P Value	STATUS
1	Monobit	0.6198944109453393	Pass
2	Frequency Block	0.7065329294567547	Pass
3	Runs	0.8967610452310607	Pass
4	Longest Run	0.5101194804270734	Pass
5	Binary Matrix	0.9993060149755448	Pass
6	Spectral	0.1496607474939273	Pass
7	Non Overlapping Template	0.3235747588539187	Pass
8	Overlapping Template	0.3305051386130132	Pass
9	Universal Statistical	0.5098012903985121	Pass
10	Linear Complexity	0.93487252967938	Pass
11	Serial	1.0	Pass
12	Entropy	0.530925287678524	Pass
13	Cusum	0.4504709553306088	Pass
14	Random Excursions	0.7287447923400823	Pass
15	Random Excursions Variant	0.25490304364305605	Pass

For all fifteen tests NIST statistical test suite has some criteria as expounded in table 1 [20]. Implementation is done for all fifteen tests according to standards set by NIST. An improved statistical test suite which contain 15 tests which are used to check a long bit stream that is produced by its built-in generator, AES from dummy tests. Results obtained for the sequences are observed and it is found that p values are identical to the values gained by NIST statistical test suite. Both results are compared which are given in Appendix A. Our Observation are as follows:

Although new features are added which has made it very comprehensive to be used but we have found that Binary Matrix Rank Test and Serial Test are taking too much time in computational process to calculate p value, this time can be reduced by reviewing the code. Also, an improved statistical test suite calculating one P-value for this test while NIST statistical test suite calculate p value for all non-periodic (aperiodic) templates.

Conclusion and future work

An Improved STS is proposed in this research paper which has been tested to evaluate randomness of a long bit sequence generated cryptographic applications and verified that the P-values results are same to the corresponding values of NIST STS. So, the output of the paper is to develop an improved statistical test suite which is suitable for different purposes like pseudo-randomness, cryptographic design and cryptanalysis techniques. The aim was to overcome the challenges of the NIST STS software and to provide an improved statistical test suite which is more comprehensive, easy to use and accurate in the results it provides.

Future contribution will focus on making it more comprehensive by adding other pseudocode and open source tests proposed for randomness testing in an improved statistical test suite. An improved STS provides clear results interpretation and performance are much better as compared to NIST STS but optimum level yet to be achieved, so more efforts are require also required in this regard. Performance of the software can be improved further by reviewing code.

REFERENCES

[1] Patil, P., Narayankar, P., Narayan, D. G., and Meena, S. M., “A comprehensive evaluation of cryptographic algorithms: DES, 3DES, AES, RSA and Blowfish”, Procedia Computer Science, Volume 78, pp. 617-624, 2016.

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

[3] McCullough, B. D., "A review of TESTU01", Journal of Applied Econometrics Volume 21, No. 5, pp. 677-682, 2006.

[4] Alani, M. M., "Testing randomness in the ciphertext of block-ciphers using DieHard tests", Int. J. Comput. Sci. Netw. Secure, Volume 10, No. 4, pp. 53-57, 2010.

[5] Soto, J., “Statistical testing of random number generators”, In Proceedings of the 22nd National Information Systems Security Conference, Gaithersburg, MD: NIST, Volume 10, No. 99, p. 12, October, 1999.

[6] Caelli, W., “Crypt x package documentation”, Information Security Research Centre and School of Mathematics, Queensland University of Technology, 1992.

[7] Marton, K., and Suciu, A., “On the interpretation of results from the NIST statistical test suite”, Science and Technology, Volume 18, No. 1, pp. 18-32, 2015.

[8] Li, L., “Testing Several Types of Random Number Generator”, Doctoral dissertation, Florida State University, 2012.

[9] Spain, M., and Vira, M. H., “Evolving s boxes with reduced differential power analysis susceptibiltiy” MIT Lincoln Laboratory Lexington United States, 2016.

[10] Salameh, J. N. B., “A New Technique for Sub-Key Generation in Block Ciphers” World Applied Sciences Journal, Volume 19, No.11, pp. 1630-1639, 2012.

[11] Zaman, J. K. M., and Ghosh, R., “A review study of NIST Statistical Test Suite: Development of an indigenous computer package”, arXiv preprint arXiv:1208.5740, 2012.

[12] Demirhan, H., and Bitirim, N., “Statistical testing of cryptographic randomness” Journal of Statisticians: Statistics and Actuarial Sciences, Volume 9, No. 1, pp. 1-11, 2016.

[13] Ukrop, M., “Randomness analysis in authenticated encryption systems”, Doctoral dissertation, Masarykova univerzita, Fakulta informatiky, 2016.

[14] Bellare, M., Brakerski, Z., Naor, M., Ristenpart, T., Segev, G., Shacham, H., and Yilek, S., “Hedged public-key encryption: How to protect against bad randomness”, In International Conference on the Theory and Application of Cryptology and Information Security, Springer, Berlin, Heidelberg, pp. 232-249, December, 2009.

[15] Yilek, S., “Resettable public-key encryption: How to encrypt on a virtual machine”, In Cryptographers’ Track at the RSA Conference, Springer, Berlin, Heidelberg, pp. 41-56, March, 2010.

[16] Johnsto. D., "sp800_22_tests," https://github.com/dj-on-github/sp800_22_tests, Visited 24-12-2018.

[17] StuardGordonReid, "R4nd0m," https://gist.github.com/StuartGordonReid, Visited 20-12-2018.

[18] Daemen, J., and Rijmen, V., “AES proposal: Rijndael”, 1999.

[19] Simion, E., and Burciu, P., "A Note On the Correlations Between NIST Cryptographic Statistical Tests Suite." UNIVERSITY POLITEHNICA OF BUCHAREST SCIENTIFIC BULLETIN-SERIES A-APPLIED MATHEMATICS AND PHYSICS Volume 81, No. 1, pp. 209-218, 2019.

[20] Kaminsky, A., "Testing the Randomness of Cryptographic Function Mappings." IACR Cryptology ePrint Archive, pp. 78 2019.