Hindawi
Security and Communication Networks
Volume 2021, Article ID 1375954, 10 pages
https://doi.org/10.1155/2021/1375954

Research Article

Fast Software Implementation of Serial Test and Approximate Entropy Test of Binary Sequence

Xian-wei Yang,[1] Xue-qiu Zhan ,[1] Hong-juan Kang,[2] and Ying Luo[3]

[1]*Wuxi Institute of Technology, Wuxi, China*
[2]*Sichuan Changhong Electric Co., Ltd., Chengdu, China*
[3]*Sichuan Innovation Center of Industrial Cyber Security Co., Ltd., Chengdu, China*

Correspondence should be addressed to Xue-qiu Zhan; zhanxq@wxit.edu.cn

Received 18 June 2021; Revised 6 August 2021; Accepted 23 August 2021; Published 16 September 2021

Academic Editor: Stelvio Cimato

In many cryptographic applications, random numbers and pseudorandom numbers are required. Many cryptographic protocols require using random or pseudorandom numbers at various points, e.g., for auxiliary data in digital signatures or challenges in authentication protocols. In NIST SP800-22, the focus is on the need for randomness for encryption purposes and describes how to apply a set of statistical randomness tests. These tests can be used to evaluate the data generated by cryptographic algorithms. This paper will study the fast software implementation of the serial test and the approximate entropy test and propose two types of fast implementations of these tests. The first method is to follow the basic steps of these tests and replace bit operations with byte operations. Through this method, compared with the implementation of Fast NIST STS, the efficiency of the serial test and approximate entropy test is increased by 2.164 and 2.100 times, respectively. The second method is based on the first method, combining the statistical characteristics of subsequences of different lengths and further combining the two detections with different detection parameters. In this way, compared to the individual implementation of these tests, the efficiency has been significantly improved. Compared with the implementation of Fast NIST STS, the efficiency of this paper is increased by 4.078 times.

1. Introduction

In cryptography, random numbers and pseudorandom numbers are widely used in applications. For example, use a randomly generated key in a cryptographic system. There are also random or pseudorandom numbers required to be used at various points in cryptographic protocols, for example, for auxiliary data in digital signatures or challenges in authentication protocols.

The random bit sequence can be explained by the result of an unbiased "fair" coin flip, with the sides of the coin marked as "0" and "1." The probability that each flip produces a "0" or "1" is 1/2, and the results of each coin toss are independent of each other. Unbiased, fair coins are perfect random bitstream generators because 0 and 1 values will be randomly distributed, and all elements in the sequence are generated independently of each other. The value of any element in the sequence is unpredictable and has nothing to do with all previously generated elements.

The SP 800-22 [1] issued by the National Institute of Standards and Technology (NIST) discusses the randomness test of random number and pseudorandom number generators. These tests can be applied to fields such as cryptography, modeling, and simulation. In NIST SP800-22, the focus is on the need for randomness for encryption purposes and describes applying a set of statistical randomness tests. Germany released the BSI AIS 30 specification [2]. In 2009, the National Cryptography Administration (NCA) of China issued a randomness test specification [3]. In addition, research on random sequences is in full swing, and a large number of new statistical tests have been proposed [4]. There are two basic types of random sequence generators: random number generator (RNG) and pseudorandom number generator (PRNG) [5]. In cryptographic applications, both

generators produce zero and a stream divided into subsequences or blocks.

These tests can be used to evaluate the data stream generated by the cryptographic algorithm, thereby providing useful reference data for the theoretical analysis of the algorithm [6–8]. This approach can reduce the workload of theoretical analysis and detect security risks that cannot be found by other analytical methods. For example, in the competition of AES [9, 10], randomness detection is used to evaluate candidate cryptographic algorithms [11]. ZUC [12, 13] has officially become a cryptographic algorithm of LTE and executed many randomness tests. Parameter of these detections can be recommended or be adjusted [14–16].

Some researchers study the rapid implementation of all NIST STS tests and achieved interesting speedups in most of the tests [17]. Q-value is introduced, and the distribution of Q-value is closer to a uniform distribution than P value, which can reduce the false detection rate [18]. When the runs distribution test is applied on some well-known good deterministic random bit generators (DRBGs), the test results show apparent bias from randomness [19]. A new DFT test method for the long sequence is proposed, and this DFT test reconstructs the statistics to follow the chi-square distribution [20].

In this paper, we study the fast implementation of the serial test and the approximate entropy test and propose two types of fast implementation of these tests. The first fast implementation method is to follow the basic steps according to these tests. In this implementation, the efficiency of the serial test and the approximate entropy test is increased by 2.164 and 2.100 times, respectively, compared with the basic implementation. The second one is to merge these tests. Relative to the individual implementation of these tests, the efficiency has been improved in this implementation. Compared with the basic implementation, the best efficiency of this method is increased by 4.078 times, and the effect is significant.

This paper is organized as follows. Section 2 presents an introduction to these statistical tests. Section 3 discusses the serial test, the approximate entropy test, and the basic implementation. Sections 4 and 5 present two types of fast implementation of these tests. Section 6 presents the software implementation results of these methods. Section 7 concludes the paper.

2. Introduction of Statistical Tests

By performing various statistical tests on the sequences, it is possible to compare and evaluate sequences with random sequences. Both characterization and description of the properties of random sequences can be done through probability. There are countless possible statistical tests to assess the presence or absence of a pattern, which indicates that the sequence is nonrandom. These test methods are designed for the different characteristics of the sequence, mainly based on the different focus of the sequence characteristics, and there are also some test methods that have no significant differences in the principles. Therefore when choosing a randomness test method, it is necessary to consider all aspects of the random characteristics of the test sequence, but also to take into account the efficiency of the test. In addition, one must be careful when interpreting the results of statistical tests to avoid erroneous conclusions about specific generators [21].

The randomness of the test sequence is essential to test whether it is truly random or the gap between it and true randomness. Randomness testing usually uses hypothesis testing. Hypothesis testing is to propose certain assumptions about the population in order to infer certain properties of the population when the population distribution is unknown or only its form is known. However, its parameters are not known, and then, make judgments on the proposed hypotheses based on the sample. Random hypothesis testing means that a certain aspect of a truly random sequence conforms to a specific distribution. If the sequence to be tested is random, then the sequence to be tested should also conform to this specific distribution in this respect. Take a certain statistical value V of a random sequence that conforms to the chi-square distribution with n degrees of freedom as an example. Null hypothesis (null hypothesis) H_0: the sequence is random if the statistical value V of the sequence to be tested obeys the $\chi^2(n)$ distribution. Alternative hypothesis H_α: the sequence is not random if the statistical value V of the sequence to be tested does not obey the $\chi 2(n)$ distribution. The opposite of this null hypothesis is the alternative hypothesis, which is that the sequence is not random. By testing each application, a decision or conclusion is derived. Based on the generated sequence, determine whether to accept the null hypothesis H_0. Under the original hypothesis, the theoretical reference distribution of the statistical data is determined by mathematical methods, and the critical value is determined. During the test, the statistical value of the tested sequence is calculated and compared with the critical value. If the test statistic value does not exceed the critical value, the null hypothesis H_0 of randomness is accepted. Otherwise, accept the alternative hypothesis H_a.

There are two possible outcomes of statistical hypothesis testing, namely, accepting H_0 or accepting H_a.

Two types of errors may occur with this method. First, if the data is random, then we conclude that the data is nonrandom. This conclusion is called a type I error. Second, if the data is nonrandom, then we conclude that the data is random. This conclusion is called a type II error. The probability of a type I error is called the significance level of the test, and this probability is usually denoted as α. Typically, α is selected in the range of [0.001, 0.01], and the value of α in cryptography is about 0.01. In the test, α is the probability that when the test shows that the sequence is truly random, it is not a random sequence. The probability of a type II error is usually denoted as β. In the test, β is the probability that the test will indicate that the sequence is random when it is not. β is not a fixed value, which is different from α. There are many ways to select data, they can be nonrandom, and each different way can produce different β. One of the main goals of the test is to minimize the

possibility of making type II errors. Table 1 below correlates the true state of the data with the test conclusion.

In order to reflect the strength of the evidence against the null hypothesis, the P value can be calculated using test statistics. In all tests, the P value is the probability that the sequence generated by the perfect random number generator has less randomness than the sequence to be tested. When the sequence seems to have perfect randomness, the P value is equal to 1, and when the sequence seems to be completely nonrandom, the P value is equal to 0. When the P value $\geq\alpha$, the null hypothesis H_0 is accepted, which means that the sequence appears to be random. When the P value $<\alpha$, the alternative hypothesis H_a is accepted, which means that the sequence appears nonrandom.

In the test, the P value can be calculated with the test statistic. The P value represents the probability that the sequence generated by the perfect random number generator is less random than the sequence being tested. When the P value is 1, it means that the sequence seems to have perfect randomness, and when the P value is 0, it means that the sequence seems to be completely nonrandom. When $P \geq\alpha$, it means that the null hypothesis is credible, which reflects that the sequence seems to be random. When $P <\alpha$, it means that the null hypothesis is unreliable, which reflects that the sequence seems to be nonrandom. The P value is the strength of evidence for accepting the null hypothesis.

The NIST SP800-22 is a statistical test suite consisting of 16 tests, and the test suite of NCA consists of 15 tests. Table 2 lists all of the tests. Serial test and approximate entropy test are the tests of the two test suites.

Table 1: Conclusions of statistical hypothesis testing.

True situation	Accept H_0	Accept H_a
Data is random	No error	Type I error
Data is not random	Type II error	No error

Table 2: The tests of NIST SP800-22 and NCA randomness test specification.

Items	NIST	NCA
Monobit test	√	√
Frequency test within a block	√	√
Poker test	×	√
Serial test	√	√
Runs test	√	√
Runs distribution test	×	√
Test for the longest run of ones in a block	√	√
Binary derivative test	×	√
Autocorrelation test	×	√
Binary matrix rank test	√	√
Cumulative sums test	√	√
Approximate entropy test	√	√
Linear complexity test	√	√
Maurer's universal statistical test	√	√
Discrete Fourier transform test	√	√
Nonoverlapping template matching test	√	×
Overlapping template matching test	√	×
Lempel–Ziv compression test	√	×
Random excursions test	√	×
Random excursions variant test	√	×

3. Serial Test and Approximate Entropy Test

3.1. Definitions and Symbols. Table 3 lists all of the symbols and their meanings used in this paper.

The establishment of the incomplete gamma function is based on the approximate formula [22], which can be approximated by continuous fraction expansion or series expansion according to the values of its parameters a and x.

Gamma function and incomplete gamma function are defined as

$$\Gamma(x) = \int_0^{\infty} e^{-t}t^{x-1}\mathrm{d}t, \quad x>0,$$

$$P(a,x) = \frac{\gamma(a,x)}{\Gamma(a)} = \frac{1}{\Gamma(a)}\int_0^{x} e^{-t}t^{a-1}\mathrm{d}t,$$

$$Q(a,x) = 1 - P(a,x) = \frac{\Gamma(a,x)}{\Gamma(a)} = \frac{1}{\Gamma(a)}\int_x^{\infty} e^{-t}t^{a-1}\mathrm{d}t. \tag{1}$$

3.2. Serial Test. This section describes the serial tests, such as the test description, the technical details, the testing strategy, and the purpose of this test.

The frequency of all possible m-bit subsequences in the entire sequence is the focus of the serial test. From the results of this test, it can be determined whether the frequency of occurrence of the 2^m m-bit subsequences of the random sequence is roughly in line with expectations. The probability of all subsequent m-bit in the random sequence is the same, and when m is 1, the serial test has the same utility as the frequency test.

For different values of n, the sequence is expanded by appending the first m-1 bits to the end of the sequence to form an expanded sequence $\varepsilon\prime = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n, \varepsilon_1, \ldots, \varepsilon_{m-1})$. The serial test is based on testing the uniformity of distributions of subsequences of given lengths in the circularised string $\varepsilon\prime$. Set

$$\psi_m^2 = \frac{2^m}{n}\sum_{i_1\cdots i_m}\left(\nu_{i_1\cdots i_m} - \frac{n}{2^m}\right)^2 = \frac{2^m}{n}\sum_{i_1\cdots i_m}\nu_{i_1\cdots i_m}^2 - n. \tag{2}$$

Here, $\psi_0^2 = \psi_{-1}^2 = 0$. Thus, ψ_m^2 is a χ^2 type of statistic, but it is a common mistake to assume that ψ_m^2 has the χ^2 distribution. The corresponding generalized serial statistics for the testing of randomness are $\nabla\psi_m^2$ and $\nabla^2\psi_m^2$:

$$\begin{aligned}\nabla\psi_m^2 &= \psi_m^2 - \psi_{m-1}^2,\\ \nabla^2\psi_m^2 &= \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2.\end{aligned} \tag{3}$$

Table 3: Symbols.

Symbols	Meaning
ε	The original input string of zero and one bits to be tested
εi	The ith bit in the original sequence ε
n	The number of bits in the stream being tested
m	The number of bits in a substring (block) being tested
$v_{i_1}, v_{i2}, \ldots, v_{it}$	The frequency of the t-bit pattern $i_1 i_2 \ldots i_t$
$\nabla\Psi^2{}_m(\text{obs})$	A measure of how well the observed values match the expected value
$\nabla^2\Psi^2{}_m(\text{obs})$	A measure of how well the observed values match the expected value
H0	The null hypothesis; i.e., the statement that the sequence is random
Igamc	The incomplete gamma function
$\log(x)$	The natural logarithm of x: $\log(x) = \log_e(x) = \ln(x)$
A	The significance level
&	The and operator
|	The or operator

Then, $\nabla\psi_m^2$ and $\nabla\psi_{m-1}^2$ have the χ^2 distribution with $2m-1$ and $2m-2$ degrees of freedom, respectively. Thus, for small values of m, $m \le \lfloor \log_2(n) \rfloor - 2$, one can find the corresponding $2m$ P values from the standard formulas:

$$P - \text{value1} = igamc\left(2^{m-2}, \frac{\nabla\psi_m^2}{2}\right),$$
$$P - \text{value2} = igamc\left(2^{m-3}, \nabla\frac{\nabla\psi_m^{2\,2}}{2}\right). \tag{4}$$

Denote by $v_{i_1 v_{i_2}, \ldots, v_{i_m}}$ the frequency of m-bit pattern $i_1 i_2 \ldots i_m$, $v_{i_1 v_{i_2} \ldots v_{i_{m-1}}}$ the frequency of $(m-1)$-bit pattern $i_1 i_2 \ldots i_{m-1}$, and $v_{i_1 v_{i_2} \ldots v_{i_{m-2}}}$ the frequency of $(m-2)$-bit pattern $i_1 i_2 \ldots i_{m-2}$, respectively.

The steps of the serial test (Algorithm 1) are as follows.

Note that choose m such that $m \le \lfloor \log_2(n) \rfloor - 2$. The bit length of subsequence m and the sample length n are proposed $m = 2$ and 5 and $n = 1000000$ in specification [3], respectively.

3.3. Approximate Entropy Test. The frequency of all possible overlapping m-bit patterns in the sequence is the focus of the approximate entropy test. Through this test, the frequency of two adjacent lengths' (m and $m+1$) subsequences can be compared with the expected result.

The repetitive pattern in the string is a feature of approximate entropy. Set

$$C_i^m = \frac{v_{i_1 i_2 \cdots i_m}}{n},$$
$$\Phi^{(m)} = \sum_{i=1}^{2^m} C_i^m \ln C_i^m. \tag{5}$$

The approximate entropy $ApEn(m)$, $m \ge 1$, is defined as

$$ApEn(m) = \Phi^{(m)} - \Phi^{(m-1)}, \tag{6}$$

with $ApEn(0) = -\Phi^{(1)}$.

$ApEn(m)$ measures the logarithmic frequency with which blocks of length m that are close together. Thus, a smaller $ApEn(m)$ value indicates strong regularity in the sequence, while larger values indicate irregularities in the sequence.

For a fixed block length m, one should expect that, in long random strings, $ApEn(m) \sim \ln 2$. The limited distribution is consistent with the distribution of a random variable with $2m$ degrees of freedom. This fact provides the basis for statistical testing. Thus, with $\chi^2(\text{obs}) = n[\ln 2 - ApEn(m)])$, the reported P value is

$$P - \text{value} = igamc\left(2^{m-1}, \frac{\chi^2(\text{obs})}{2}\right). \tag{7}$$

The steps of the approximate entropy (Algorithm 2) test are as follows.

Note that choose m such that $m \le \lfloor \log_2(n) \rfloor - 2$. The bit length of subsequence m and the sample length n are proposed $m = 2$ and 5 and $n = 1000000$ in specification [3], respectively.

The second step of the serial test and the approximate entropy test is to determine the frequency of all possible m-bit subsequences of $\varepsilon\prime = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n, \varepsilon_1, \ldots, \varepsilon_{m-1})$. Algorithm 3 is a basic implementation of Step 2 based on bits' operations.

We note by LOAD the data loading operation. Left and right shifts will be denoted SHIFT. By CMP means the compare operation.

The computational complexity of Algorithm 3 is mn LOAD, $2mn$ ADD, mn SHIFT, and mn OR. The computational complexity of Algorithm 4 is $n/8$ LOAD, $n(m+7)/8$ ADD, n SUB, $n(m+7)/8$ SHIFT, n AND, $n(m+7)/8$ OR, and $E\prime = (E_1, E_2, \ldots, E_{n/8}, E_1)$ CMP.

4. Fast Implementation

4.1. Hotspots. After determining the key parts of the program, we can start to optimize the code. During the running of the program, there are two situations for time allocation. In some programs, more than 99% of the time is used for inner loop calculations. In other programs, 99% of the time is used to read and write data, and less than 1% of the time is used to calculate the data. It is important to optimize these parts of the code instead of the parts of the code that spend a small amount of time. Optimizing the less critical parts of the code not only wastes time but also makes the code more difficult to maintain.

We can use the profiler in the compiler to profile how much time each function costs. It can also be done with third-party analyzers, such as AQtime, Intel VTune, and AMD Code Analyst.

Due to the short time interval, time measurement requires very high resolution. Users can use the QueryPerformanceCounter or GetTickCount functions for millisecond resolution in Windows. Use the time stamp counter in the CPU to get a higher resolution RDTSC. This instruction counts at the CPU clock frequency.

We need to identify the most time-consuming hotspots of the basic implementation of the serial test and the approximate entropy test. The Hotspots analysis helps understand these tests and identify steps that take a long time to execute (hotspots). Intel VTune amplifier identifies that Step 2 of the serial test is the most time-consuming hotspot of this test. In the serial test, this step determines $v_{i_1}v_{i_2}\ldots v_{i_m}$, $v_{i_1}v_{i_2}\ldots v_{i_{m-1}}$, and $v_{i_1}v_{i_2}\ldots v_{i_{m-2}}$ and the frequency of all possible m-bit, $(m-1)$-bit, and $(m-2)$-bit subsequence, respectively. Then, the Intel VTune amplifier identifies that Step 2 of the approximate entropy test is the most time-consuming hotspots of this test. This step determines $v_{i_1}v_{i_2}\ldots v_{i_m}$ the frequency of all possible m-bit subsequences.

All the operations of Algorithm 3 are based on bit operations, which seriously reduce the performance of these tests.

In practice, a binary sequence is in the form of an octet string. This octet string needs to be converted into a bit string, and then, the bit string is sent into Algorithm 3 to determine the frequency of all possible m-bit subsequences.

Algorithm 3 is replaced by bit operations with byte operations, and the performance will be significantly improved.

4.2. Fast Implementation Based on Octet Operation. In practice, a binary sequence is almost in the form of the octet string, so we assume that n is divided by $(8|n)$ The random byte is a byte sequence formed by combining random bits by bit-by-bit splicing.

Denote $E = (E_1, E_2, \ldots, E_{n/8})$ the octet string of the binary sequence $\varepsilon = \varepsilon_1, \varepsilon_1, \ldots, \varepsilon_n$ of length n, with $E_i = \varepsilon_{8i-7}, \varepsilon_{8i-6}\ldots, \varepsilon_{8i}$, $i = 1, 2, \ldots, n/8$. Because the value of the parameter m is less than 8, the extended octets are not more than one octet (8 bits). Extend the sequence by appending the first octet to the end of the octet string and $E\prime = (E_1, E_2, \ldots, E_{n/8}, E_1)$ to form the augmented octet string $E\prime$. Algorithm 4 determines the frequency based on multiple bits' operations.

4.3. Analysis of Computational Complexity. This section analyses and compares the number of arithmetic operations of the above algorithms. Because Step 2 is the most time-consuming, so the main task is to analyze the computational complexity of Algorithm 3 based on bit string and bit operations and Algorithm 4 based on octet string and multibit operations.

Unroll Step 2.2 of Algorithm 4, which can reduce many operations. The CMP operation is clear, and the left and right shift number does not need to be calculated. The computational complexity is reduced to $n/8$ LOAD, n ADD, $n(m+7)/8$ SHIFT, n AND, and $n(m+7)/8$ OR.

The efficiency of the loop depends on the microprocessor's ability to predict the control branch of the loop. A loop with a small and fixed repeat count and no branches inside can be predicted perfectly. It is best to avoid loop unrolling on processors with microoperation cache because it is important to save the use of microoperation cache. The unrolled loop takes up more space in the code cache or microoperation cache. If there are specific advantages to be gained, such as eliminating the if branch, the programmer should manually unroll a loop.

Loop unrolling of Algorithm 4 has many advantages. Then, if the branch is eliminated, the CMP operation is removed, and the left and right shift numbers are cleared.

The computational complexity of Algorithm 3 (based on bit string) and Algorithm 4 (based on octet string) is provided in Table 4.

For example, when the value of m is 3, the execution numbers of Algorithms 3 and 4 are $15n$ and $3.625n$, respectively.

The serial test determines the frequency of all possible j-bit patterns, with $j = 1, 2, 3, 4,$ and 5. In Algorithm 3 (based on bit string), the sum of operations of the serial test (with $j = 1, 2, 3, 4,$ and 5) is

$$\sum_{m=1}^{5} 5mn = 75n. \tag{8}$$

In Algorithm 4 (based on octet string), the sum of operations of the serial test (with $j = 1, 2, 3, 4,$ and 5) is

$$\sum_{m=1}^{5} \frac{n(2m+23)}{8} = 18.125n. \tag{9}$$

In contrast to the number of arithmetic operations of the above two algorithms, it is shown that Algorithm 4 based on the octet string is superior to Algorithm 3 based on bit string, in the serial test. Table 5 shows the number of arithmetic operations of the above two algorithms.

The approximate entropy test determines the frequency of all possible j-bit patterns, with $j = 2, 3, 5,$ and 6. In Algorithm 3 (based on bit string), the sum of operations of the approximate entropy test is

$$\sum_{m=2,3,5,6} 5mn = 80n. \tag{10}$$

In Algorithm 4 (based on octet string), the sum of operations of the approximate entropy test is

$$\sum_{m=2,3,5,6} \frac{n(2m+23)}{8} = 15.5n. \tag{11}$$

In contrast to the number of arithmetic operations of the above two algorithms, it is shown that Algorithm 4 based on octet string is superior to Algorithm 3 based on bit string, in the approximate entropy test. Table 6 shows the number of arithmetic operations of the above two algorithms.

Input: a binary sequence $\varepsilon = \varepsilon_1, \varepsilon_1, \ldots, \varepsilon_n$ of length n. m is the bit length of subsequence.
Output: pass or not.
Step 1: extend the sequence by appending the first $(m-1)$ bits to the end of the sequence $\varepsilon\prime = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n, \varepsilon_1, \ldots, \varepsilon_{m-1})$
Step 2: determine $v_{i_1}v_{i_2}\ldots v_{i_m}$, $v_{i_1}v_{i_2}\ldots v_{i_{m-1}}$, and $v_{i_1}v_{i_2}\ldots v_{i_{m-2}}$ and the frequency of all possible m-bit, $(m-1)$-bit, and $(m-2)$-bit subsequence, respectively.
Step 3: compute ψ_m^2, ψ_{m-1}^2, and ψ_{m-2}^2. $\psi_m^2 = 2^m/n\sum_{i_1\cdots i_m} v_{i_1\cdots i_m}^2 - n$. $\psi_{m-1}^2 = 2^{m-1}/n\sum_{i_1\cdots i_m} v_{i_1\cdots i_m}^2 - n\psi_{m-2}^2 = 2^{m-2}/n\sum_{i_1\cdots i_m} v_{i_1\cdots i_m}^2 - n$.
Step 4: compute $\nabla\psi_m^2$ and $\nabla^2\psi_m^2$. $\nabla\psi_m^2 = \psi_m^2 - \psi_{m-1}^2 \nabla^2\psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2$.
Step 5: compute P values. $P-\text{value1} = igamc(2^{m-2}, \nabla\psi_m^2/2) P-\text{value1} = igamc(2^{m-3}, \nabla\psi_m^2/2)$.
Step 6: if $P-\text{value1} \geq \alpha$ and $P-\text{value2} \geq \alpha$, the sequence passes the test. Otherwise, return not pass.

Algorithm 1: Serial test.

Input: a binary sequence $\varepsilon = \varepsilon_1, \varepsilon_1, \ldots, \varepsilon_n$ of length n. m is the bit length of subsequence
Output: pass or not
Step 1: extend the sequence by appending the first m-1 bits to the end of the sequence $\varepsilon\prime = (\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_n, \varepsilon_1, \cdots, \varepsilon_{m-1})$.
Step 2: determine $v_{i_1}v_{i_2}\ldots v_{i_m}$ the frequency of all possible m-bit subsequences.
Step 3: compute C_i^m, for each value of i. $C_i^m = v_{i_1 i_2 \ldots i_m}/n$.
Step 4: compute $\Phi^{(m)}\Phi^{(m)} = \sum_{i=1}^{2_m} C_i^m \ln C_i^m$.
Step 5: repeat Steps 1–4, replacing m by m+1.
Step 6: compute $\chi^2 = n[\ln 2 - ApEn(m)]$ with $ApEn(m) = \Phi^{(m)} - \Phi^{(m-1)}$.
Step 7: compute P values. $P-\text{value} = igamc(2^{m-1}, \chi^2/2)$.
Step 8: if $P-\text{value} \geq \alpha$, the sequence passes the test. Otherwise, return not pass.

Algorithm 2: Approximate entropy test.

Input: A extended binary sequence $\varepsilon' = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n, \varepsilon_1, \ldots, \varepsilon_{m-1})$ of length $n+m-1$. m is the bit length of subsequence.
Output: v_w, for each value of $w = i_1 i_2 \cdots i_m$.
Step 1: $v_{i_1 i_2 \cdots i_m} = 0$, for each value of i.
Step 2: for $i = 1, 2, \ldots, n$, do: 2.1 $x = 0$. 2.2 for $j = 0, 1, \ldots, m-1$, do: $x = 2x + \varepsilon_{i+j}$. 2.3 $v_x = v_x + 1$.
Step 3: return v_w, for each value of $w = i_1 i_2 \ldots i_m$.

Algorithm 3: Determine the frequency of all possible m-bit subsequence (bit string).

However, different instructions cost different cycles. Integer operations are usually very fast. On most microprocessors, most simple integer operations (such as addition, subtraction, comparison, bit operations, and shift operations) only take one clock cycle. Multiplication and division require longer clock cycles. Usually, integer multiplication requires 3-4 clock cycles, and integer division requires 40–80 clock cycles. In addition, it may take longer to access data from RAM compared to the time required to perform calculations on the data. If it is cached, it only takes 2-3 clock cycles to read or write the variables in the memory, while if it is not cached, it takes hundreds of clock cycles [23]. So, the exact time consuming of these algorithms needs to do many different experiments.

5. Further Optimizing

5.1. Merging These Two Tests. As mentioned earlier, accessing data from RAM may take longer than the time it takes to perform calculations on the data. This is why all modern computers have a memory cache. Generally, there are a level 1 data cache, a level 2 cache, and a level 3 cache. If the total size of all data in the program is greater than the level 2 cache and the data is scattered in the memory or accessed in a nonsequential manner, memory access may be the largest time-consuming operation in the program. If it is cached, it only takes 2-3 clock cycles to read or write a variable in the memory, while if it is not cached, it takes hundreds of clock cycles.

The sequence size is 125000 bytes, which is bigger than most of the level-1 cache, so these data cannot be cached. Loading sequence data in memory costs many clock cycles, and memory access is the time-consuming operation (hotspots) in these tests. The size of the sequence cannot be reduced, and the cache of the CPU is fixed. The best thing we can do is reduce the loading times of data to improve the performance of these tests. Many specifications propose to perform both the serial test and the entropy test.

Step 2 of the serial test and Step 2 of the approximate entropy test have similar functions and call the same algorithm so that these tests may be merged. The values of m are proposed in specification [3]. In the serial test, the values are $m = 2$ and 5, which need to determine the frequency of 1-bit, 2-bit, 3-bit, 4-bit, and 5-bit pattern. In the approximate

Input: a extended octet string $E\prime = (E_1, E_2, \ldots, E_{n/8}, E_1)$ of octet length $n/8 + 1$. m is the bit length of subsequence.
Output: v_w, for each value of $w = i_1 i_2 \ldots i_m$.
Step 1: $v_{i_1 i_2 \ldots i_m} = 0$, for each value of i. $M = 2^m - 1$.
Step 2: for $i = 1, 2, \ldots, n/8$, do: 2.1 $x = 0$. 2.2 for $j = 0, 1, \ldots, m - 1$, do. $k = 8 - j - m$ if $k \geq 0$, then: $x = (E_i \gg k)$ & M if $k < 0$, then: $x = ((E_i \ll (-k)) | (E_{i+1} \ll (8 + k)))$ & M $v_x = v_x + 1$
Step 3: return v_w, for each value of $w = i_1 i_2 \ldots i_m$.

Algorithm 4: Determine the frequency of all possible m-bit subsequence (octet string).

entropy test, the values are $m = 2$ and 5, which need to determine the frequency of 2-bit, 3-bit, 5-bit, and 6-bit pattern so one can determine the frequency of i-bit patterns, with $i = 1, 2, \ldots, 6$.

There are many benefits to the merger of the six cases. Firstly, lots of the same operations of Algorithm 4 can be combined. For example, the large number of loading operations only need to execute one time with $m = 1, 2, \ldots, 6$, and the loading data operations reduce $5n/8$, which enhances the efficiency of data. Secondly, the number of calling algorithms can be greatly reduced, which further speeds up the test performance.

Algorithm 5 shows the emerged test based on multibit, which is the merger of the serial test and the approximate entropy test.

Algorithm 6 shows the merger of the six cases, which determine the frequency of the i-bit pattern, with $i = 1, 2, \ldots, 6$.

In Step 2.2, the conditions of j are $1 \leq j \leq 6$ and $8 < k + j \leq 16$, which ensure the validity of the results. For example, $k = 13$, $v2_y (j = 2)$ is valid, and $v5_y (j = 3)$ is invalid. It is recommended to unroll the loop of j and k in programming, reducing lots of operations.

The codes of Algorithm 6 are shown in Appendix, with loop unrolling.

5.1.1. Computational Complexity. This section analyses the number of arithmetic operations of the above algorithms.

Algorithm 5 shows the emerged test based on multibit, which is the merger of the serial test and the approximate entropy test. The most time-consuming step of this test still is Step 2, which calls Algorithm 6, to determine the frequency of all possible j-bit patterns, with $j = 1, 2, \ldots, 6$. In Algorithm 6, Step 1 and Step 3 do basic operations, and Step 2 is the crucial step and the most time-consuming step.

The computational complexity of Algorithm 6 is $n/8$ LOAD, $6n$ ADD, $7n/4$ SHIFT, $6n$ AND, and $n/8$ OR. The details are provided in Table 7.

In the implementation of these two tests, different algorithms have different performances. Table 8 lists the number of operations of Algorithms 3, 4, and 6 in the serial test and the approximate entropy test.

6. Experimental Results

This section shows the experimental results of the above algorithm. In contrast to the performance of these algorithms, it is shown that Algorithm 4 (based on octet string) and Algorithm 6 (based on the merger of these tests) are superior to Algorithm 1 (based on bit string).

Table 4: The computational complexity of Algorithms 3 and 4.

	Execution number	
Operations	Algorithm 3	Algorithm 3
LOAD	mn	$n/8$
ADD	$2mn$	n
SHIFT	mn	$n(m + 7)/8$
AND	0	n
OR	mn	$n(m - 1)/8$

At first, it shows the way to perform analysis of the above algorithms. The first step is to set a time counter T_s before the algorithm to be analyzed, and set another time counter T_e after the algorithm. Then, run the application and get the elapsed time $T_i = T_e - T_s$. Repeat Step 1 and Step 2 k times (k is odd), and get multiple values of the elapsed time $T_i (i = 1, 2, \ldots, k)$. By the descending order of these values, $T_1' T_2' \geq \ldots \geq T_k'$, and the intermediate value $T_{(k+1)/2}$ is the elapsed time of the algorithm.

We can use the command RDTSC to get the current time. RDTSC instruction loads the current value of the processor's time stamp counter (64-bit counter). The processor monotonically increments the timestamp counter every clock cycle and resets it to 0 every time the processor is reset. The RDTSC instruction is not serialized. It does not have to wait until all previous instructions have been executed to read the counter.

We measure our algorithms and these tests on a personal computer. This computer has one Intel Core i3-3240@ 3400 MHz CPU with four cores. Measurements use one core. The compiler used to compile our C code is Intel C++ Compiler XE 12.0.

The input data of Algorithm 3 is a bit string of length 1000000, and the input data of Algorithms 4 and 8 is the octet string of byte length 125000, which is converted from that binary string.

Algorithm 3 (based on bit string) and Algorithm 4 (based on octet string) can be used to determine the frequency of the m-bit pattern. Algorithm 6 (based on octet string) can be used to determine the frequency of the i-bit pattern with $m = 1, 2, \ldots, 6$.

In the serial test, one should determine the frequency of 1-bit, 2-bit, 3-bit, 4-bit, and 5-bit patterns. In the approximate entropy test, one should determine the frequency of 2-bit, 3-bit, 5-bit, and 6-bit patterns. Algorithm 6 can complete this task.

Table 5: The number of operations of Algorithms 3 and 4 in the serial test.

Operations	Execution number of serial test	
	Calling algorithm 3	Calling algorithm 4
LOAD	$15n$	$0.625n$
ADD	$30n$	$5n$
SHIFT	$15n$	$6.25n$
AND	0.5	n
OR	$15n$	$1.25n$

Table 6: The number of operations of Algorithms 3 and 4 in the approximate entropy test.

Operations	Execution number of approximate entropy test	
	Calling algorithm 3	Calling algorithm 4
LOAD	$16n$	$0.5n$
ADD	$32n$	$4n$
SHIFT	$16n$	$5.5n$
AND	0	$4n$
OR	$16n$	$1.5n$

Input: an octet string $E = (E_1, E_2, \ldots, E_{n/8})$ of octet length $n/8$.
Output: pass or not pass
Step 1: extend the sequence by appending the first octet to the end of the sequence $E\prime = (E_1, E_2, \cdots, E_{n/8}, E_1)$.
Step 2: call Algorithm 6 to determine $v_{i_1} v_{i_2} \ldots v_{i_m}$ the frequency of all possible j-bit pattern, with $j = 1, 2, \ldots, 6$.
Step 3: compute ψ_j^2, $j = 1, 2, \ldots, 5$. $\psi_j^2 = 2^j/n \sum_{i_1 \cdots i_j} v_{i_1 \cdots i_j}^2 - n$.
Step 4: compute $\nabla\psi_j^2$ and $\nabla^2\psi_j^2$, $j = 1, 2, \ldots, 5$. $\nabla\psi_j^2 = \psi_j^2 - \psi_{j-1}^2 \nabla^2\psi_j^2 = \psi_j^2 - 2\psi_{j-1}^2 + \psi_{j-2}^2$.
Step 5: compute P values of the serial test, $j = 2, 5 P1_{ST}(j) = igamc((2^{j-2}, \nabla\psi_j^2/2) P2_{ST}(j) = igamc((2^{j-3}, \nabla\psi_j^2/2)$.
Step 6: pass the serial test, if $P1_{ST}(j) \geq \alpha$, $P2_{ST}(j) \geq \alpha$, $j = 2$ and 5. Otherwise, not pass.
Step 7: compute $C_i^j = v_{i_1 i_2 \cdots i_j}/n$, for each value of i.
Step 8: compute $\Phi^{(j)}$, $j = 2, 3, 5, 6 \Phi^{(j)} = \sum_{i=1}^{2m} C_i^j \ln C_i^j$, $j = 2, 3, 5$, and 6.
Step 9: compute $\chi^2 = n[\ln 2 - ApEn(j)]$ with $ApEn(j) = \Phi^{(j)} - \Phi^{(j-1)}$, $j = 2, 3, 5$, and 6.
Step 10: compute P values of the approximate entropy test. $P_{AET}(j) = igamc(2^{j-1}, \chi^2/2)$, $j = 2$ and 5.
Step 11: pass the approximate entropy test, if $P_{AET}(2) \geq \alpha$, $j = 2$ and 5. Otherwise, not pass.

Algorithm 5: Merged test based on multibit.

Input: a extended octet string $E\prime = (E_1, E_2, \ldots, E_{n/8}, E_1)$ of octet length $n/8 + 1$.
Output: $v1_w, v2_w, v3_w, v4_w, v5_w$, and $v6_w$, for each value of $w = i_1 i_2 \ldots i_j$, $i = 1, 2, \ldots, 6$.
Step 1: $v1_w = v2_w = v3_w = v4_w = v5_w = v6_w = 0$, for each value of $w = i_1 i_2 \ldots i_j$, $i = 1, 2, \ldots, 6$. $M_j = 2^j - 1$, $i = 1, 2, \ldots, 6$.
Step 2: For $i = 1, 2, \ldots, n/8$, do: $2.1 Z = (E_i \ll 8) | E_{i+1}$ 2.2 for $k = 15, 14, \ldots, 3$, do $x = Z \gg k$ for $j = \mathrm{MAX}(9 - k, 1), \ldots, \mathrm{MIN}(16 - k, 6)$ do. $y = x \& M_j v j_y = v j_y + 1$.
Step 3: Return $v1_w, v2_w, v3_w, v4_w, v5_w$, and $v6_w$.

Algorithm 6: Determine the frequency of all possible m-bit subsequence with $m = 1, 2, \ldots, 6$.

Table 7: The number of operations of Algorithm 6.

Operations	Algorithm 6
LOAD	$0.125n$
ADD	$6n$
SHIFT	$1.75n$
AND	$6n$
OR	$0.125n$

On the same test platform, we use the NIST STS source code provided by NIST, the source code of Fast NIST STS from [17], and the implementation of this paper to test serial test and approximate entropy test. Table 9 shows the performance of the original implementation NIST STS, the implementation Fast NIST STS from [17], and our new implementation.

The original implementation of NIST STS is based on bits when performing serial tests and approximate entropy tests. This makes the detection time increase significantly when m increases. For example, when $m = 2$, it takes 32.844 milliseconds to perform the serial test, and when $m = 9$, the time increases to 169.163 milliseconds. In addition, the original implementations of NIST STS and Fast NIST STS both perform detection for a single parameter m when

Table 8: The number of operations of Algorithms 3, 4, and 6.

Operations	Serial test and approximate entropy test		
	Algorithm 3	Algorithm 4	Algorithm 6
LOAD	$31n$	$1.125n$	$0.125n$
ADD	$62n$	$9n$	$6n$
SHIFT	$31n$	$11.75n$	$1.75n$
AND	0	$9n$	$6n$
OR	$31n$	$2.75n$	$0.125n$

Table 9: The performance of the original implementation NIST STS, the implementation Fast NIST STS from [17], and our new implementation. Merged test is a combined test of the serial test ($m = 2$ and 5) and approximate entropy test ($m = 2$ and 5).

Tests	NIST STS [1] (ms)	Fast NIST STS [17] (ms)	Our (ms)	Speedup our vs. Fast NIST STS
Serial test ($m = 2$)	32.844	0.627	0.546	1.148
Serial test ($m = 5$)	64.610	0.630	0.575	1.096
Serial test ($m = 2$ and 5)	96.163	1.283	0.593	2.164
Approximate entropy test ($m = 2$)	36.046	0.636	0.553	1.150
Approximate entropy test ($m = 5$)	50.831	0.647	0.58	1.116
Approximate entropy test ($m = 2$ and 5)	124.506	1.262	0.601	2.100
Merged test	220.313	2.561	0.628	4.078

performing serial tests and approximate entropy tests. This makes $m = 2$ and $m = 5$ need to perform corresponding tests, respectively.

It can be seen from Table 9 that the efficiency of the serial test and the approximate entropy test are increased by 2.164 and 2.100 times separately, compared with the implementation of Fast NIST STS, and the execution efficiency of the merge test has been significantly improved. Its execution time is very close to that of the serial test or approximate entropy test alone. At the same time, the efficiency of the merge test reached 4.078 times that of Fast NIST STS.

In most applications, all the detection items of the test suit need to be executed. Therefore, the algorithm proposed in this paper has very important value in practical applications and can significantly reduce the execution time of these two detection algorithms.

7. Conclusions

In this paper, we study the fast implementation of the serial test and the approximate entropy test and propose two types of fast implementation of these tests.

The first fast implementation method is to follow the basic steps according to these tests and then call Algorithm 4 to determine the frequency of the m-bit pattern. In this implementation, the efficiency of the serial test and the approximate entropy test are increased by 2.164 and 2.100 times separately, compared with the implementation of Fast NIST STS. The second one is to merge these tests, combine the steps, and call Algorithm 6 to determine the frequency of all the possible values of m. In this implementation, the efficiency has been greatly improved relative to the individual implementation of these tests, and the improvement is much more significant compared with the basic implementation based on bits' operations. The best efficiency of this method is increased by 4.078 times, comparing to the implementation of Fast NIST STS.

In conclusion, we propose the fast implementation method based on merging tests and combining the frequency of the subsequences. This method not only can be used in the merging test but also can be used to only do the serial test or only do the approximate entropy test.

Data Availability

The raw/processed data required to reproduce these findings cannot be shared at this time as the data also form part of an ongoing study.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this article.

References

[1] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and M. Barker, "A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications," [EB/OL]. Version STS-2.1, NIST Special Publication 800-22rev1a, 2010.

[2] BSI AIS-20 and AIS-30, *Application Notes and Interpretation of the Scheme Functionality Classes and Evaluation Methodology for Deterministic and Physical Random Number Generators*, German Federal Office for Information Security, Berlin, Germany, 2008.

[3] National Cryptography Administration, *Randomness Test Specification*, National Cryptography Administration, Beijing, China, 2009.

[4] B. Y. Ryabko and A. I. Pestunov, ""Book stack" as a new statistical test for random numbers," *Problems of Information Transmission*, vol. 40, no. 1, pp. 66–71, 2004.

[5] E. Barker and J. Kelsey, "Recommendation for random number generation using deterministic random bit generators, [EB/OL]. revision 1, NIST special publication 800-90Ar1," 2015, http://csrc.nist.gov/publications/PubsDrafts.html#SP-800-90-A.

[6] H. Chen, *Security Test on Cryptographic Algorithms and Design of Key Cryptographic Components*, Institute of Software, Chinese Academy of Sciences, Beijing, China, 2004.
[7] E. Filiol, "A new statistical testing for symmetric ciphers and hash functions," in *Proceedings of International Conference on Information and Communications Security*, pp. 342–353, Singapore, December 2002.
[8] F. Pareschi, R. Rovatti, and G. Setti, "On statistical tests for randomness included in the NIST SP800-22 test suite and based on the binomial distribution," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, pp. 491–500, 2012.
[9] J. Daemen and V. Rijmen, *The Design of Rijndael: AES-the Advanced Encryption Standard*, Springer, Heidelberg, Germany, 2002.
[10] D. G. Feng and W. L. Wu, *Design and Analysis of Block Cipher*, Tsinghua University Press, Beijing, China, 2000.
[11] J. Soto and B. H. Lawrence, *Randomness Testing of the Advanced Encryption Standard Finalist Candidates*, Computer Security Division, NIST, Gaithersburg, ML, USA, 2000.
[12] ETSI/SAGE, *TS 35.222 Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3, Document 2: ZUC Specification [S/OL],* 2011, http://www.3gpp.org/DynaReport/35-series.htm.
[13] X. T. Feng, "ZUC algorithm: 3GPP LTE international encryption standard," *Information Security and Communications Privacy*, vol. 9, no. 12, pp. 45-46, 2012.
[14] W. W. Tsang, L. C. K. Hui, and K. P. Chow, "Tuning the collision test for power," in *Proceedings of the 27th Australasian Conference on Computer Science*, pp. 23–30, Australian Computer Society, New Zealand, Oceania, January 2004.
[15] K. Hamano and T. Kaneko, "Correction of overlapping template matching test included in NIST randomness test suite," *IEICE-Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 90, no. 19, pp. 1788–1792, 2007.
[16] F. Pareschi, R. Rovatti, and G. Setti, "Second-level NIST randomness tests for improving test reliability," in *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems*, pp. 1437–1440, New Orleans, LA, USA, May 2007.
[17] M. Sys and Z. Riha, "Faster randomness testing with the NIST statistical test suite," in *Proceedings of the International Conference on Security, Privacy, and Applied Cryptography Engineering*, pp. 272–284, Springer, Pune, India, October 2014.
[18] J. Zhuang, Y. Ma, S. Y Zhu, J. Q. Lin, and J. W. Jing, "Q_Value test: a new method on randomness statistical test," *Journal of Cryptologic Research*, vol. 3, no. 2, pp. 192–201, 2016.
[19] L. M. Fan, H. Chen, M. H. Chen, and S. Gao, "Corrected runs distribution test for pseudorandom number generators," *Electronics Letters*, vol. 52, no. 4, pp. 281–283, 2016.
[20] M. H. Chen, H. Chen, L. M. Fan, S. F. Zhu, and D. G. Feo, "A new discrete Fourier transform randomness test," *Science China(Information Sciences)*, vol. 62, no. 3, pp. 90–105, 2019.
[21] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, USA, 1997.
[22] W. Press, S. Teukolsky, and W. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, England, 2nd edition, 1993.
[23] I. Corporation, "Intel advanced vector extensions programming preference [EB/OL]," 2011, https://software.intel.com/sites/default/files/m/f/7/c/36945.