Evaluation of Whitenoise Cryptosystem
Part 1: Encryption Algorithm[1]

Information Security and Object Technology (ISOT) Group

University of Victoria,

Department of Electrical and Computer Engineering

Technical Report No ECE03-3

February 2003

Authors: Dr Issa Traore,
Michael Yanguo Liu

[1] This report describes the first part of the evaluation of the Whitenoise cryptosystem. The second part of the evaluation is related to the evaluation of the key management facility.

Table of Contents

Table of Contents……..2

1. Introduction…….……..4

2. Randomness Evaluations…….……..6

2.1 Description of the Randomness Tests……..……6

2.2 Data Selection…….…8

2.2.1 Plaintext/Ciphertext Correlation……8

2.2.2 Random Plaintext/ Random Keys…...……..8

2.2.3 Low Density Plaintext……..……8

2.2.4 High density Plaintext……..……8

2.3 Testing Strategy……..…….…9

2.4 Statistical Analysis on Empirical Results……..12

2.5 Comparison with AES Candidate Algorithms……...13

2.6 Conclusion…….13

3.Attack Evaluations…….13

4. Performance Evaluation……..……..15

4.1 Key generation speed with different key size……...……...….16

4.2 Encryption speed with different key size……..…….…..…….17

4.3 Encryption speed with different plaintext size……..19

4.4 Encryption speed with different plaintext format….…..……..20

5. Summary……..…….21

Reference……..……22

Appendix……..23

1. Introduction

Executive Summary

Whitenoise is a new generation stream cipher invented and developed by BSB Utilities Inc [1]. Stream ciphers convert plaintext to ciphertext one bit at a time. They are useful to encrypt never-ending streams of communications traffic. From our evaluation it appears that whitenoise carries all the characteristics of a one-time pad cryptosystem. To support this argument, first of all, whitenoise uses a key sequence as large as the plaintext message. The key has an extremely high level of randomness, and an extremely low level of repeatability. A one-time pad is considered as a perfect encryption scheme, providing unbreakable security, in which the key sequence has the same size as the plaintext, and is used only once. The security of one-time pad then relies on the randomness and non-repeatability of the key. Our evaluation has focused on verifying these properties for whitenoise. To verify randomness of whitenoise key generation, we used the statistical randomness test set designed by the National Institute of Standard and Technology (NIST) for the Advanced Encryption Standard (AES) competition [1]. To verify the nonrepeatability of the key generation, we engineered and executed a brute force attack using several Unix workstations running in parallel during a week. It is our intention in the future to re-execute the same attack using a 128 nodes supercomputer, and for a longer period. Whitenoise survived all these tests.

Traditional one-time pad ciphers are too expensive for most applications, since they use as much key materiel as there is traffic. Hence, they are used currently mostly for high-level diplomatic and intelligence traffic, which require undoubtedly, the highest level of security possible. The goal of the inventors of whitenoise is to shift from this paradigm and open whitenoise to a wider range of applications. For that to happen, in addition to security, performance of the encryption process and the key management facility becomes central. The second focus of our validation process was the evaluation of the performance of the encryption/decryption software. We have left out the security and performance and security analysis of the key management facility for the second part of this study. Hence, the first part of the study focused solely on the evaluation of the Whitenoise algorithm. The second part related to the key management facility, which is also very important will be presented in a second report, which will summarize our final conclusions about the whitenoise cryptosystem. The following results were obtained for the first part of the validation:

- Key generation time increases linearly as the number of subkeys increases. But the speed degradation is so slow, that it can be considered negligible; for instance generating a key consisting of 10 subkeys takes only 1 millisecond more than a key involving 5 subkeys.

- Increasing the key size doesn't affect significantly the encryption speed. The encryption speed remains almost the same using either 1 subkey or 10 subkeys.
- The encryption time remains constant for plaintext file of 10k and less, and over 10 k increases linearly as the size of the plaintext file grows.
- The magnitude of the encryption speed is the same for different file formats including pdf, text, image, and sound.

From these results, it appears that whitenoise encryption process is very fast compared to available cryptosystems, and that it can be used equally to encrypt various kinds of data formats, and hence can be used for various kinds of applications. The outcome of this evaluation is that Whitenoise is an efficient and cost-effective algorithm for securing direct communications from point-to-point over different media and processes including wireless, microwave, radio waves, remote controls, cables, wires, disks, etc. It is also ideal to be utilized to create a secure network layer for the Internet.

Whitenoise Overview

Whitenoise cryptographic system consists of a unique algorithm for encryption /decryption and an associated key exchange protocol. The encryption algorithm addresses the principle that if the key used to encrypt plaintext is random and the length of the key exceeds the amount of plaintext to be encrypted, the resultant cipher text is unbreakable. The key exchange protocol provides an alternative to the prevalent Public Key Infrastructure that is expensive and cumbersome to implement.

As part of the ongoing evaluation work, this report presents the evaluation results particularly on the Whitenoise encryption/decryption algorithm. The Whitenoise encryption process is a Multi-key-Super-key hierarchical cryptographic process that can be described conceptually as an algorithmic representation of a multi-dimensional encrypting cipher matrix [1]. More specifically, the Whitenoise encryption algorithm takes several subkeys and then creates a very long non-repeating super key by combining subkeys. Subkeys, whose lengths are prime numbers, are generated from two seeds that are two non-perfect square numbers taken from the computer random number generators. A super key is constructed bit by bit using XOR operation on each corresponding subkey bit [1]. In Whitenoise algorithm, the length of a super key is never less than the length of the plaintext. When generating super keys using Whitenoise software, only templates of super keys are created and they can be generated continuously as needed during the encryption process.

Validation Process

Several criteria can be used to evaluate encryption algorithms including security (i.e., the

effort required for cryptanalysis), computational efficiency (speed), memory requirements, hardware and software suitability, simplicity, flexibility, etc [2]. Security is the most important factor in the evaluation of diverse encryption algorithms, which includes resistance of the algorithm to cryptanalysis, soundness of its mathematical basis, randomness of the algorithm output, and possible attack analysis, etc. Algorithm speed is a second important area of evaluation. As mentioned above, in this work, we evaluate the security of the Whitenoise cryptosystem by testing the randomness and nonrepeatability of the key generation system. We evaluate the performance of the encryption/decryption process by testing several speed characteristics.

In section 2, the randomness of the algorithm output is statistically analyzed. In section 3, a possible attack on the Whitenoise encryption algorithm is presented. In section 4, the performance evaluation of Whitenoise software is described. Lastly, concluding remarks are made in section 5.

2. Randomness Evaluations

One of the criteria used to evaluate security of encryption algorithms consists of evaluating the **randomness** of their outputs using statistical tests. A **random** bit sequence could be interpreted as the result of the flips of an unbiased “fair” coin with sides that are labeled “0” and “1,” with each flip having a probability of exactly ½ of producing a “0” or “1.” Furthermore, the flips are independent of each other: the result of any previous coin flip does *not* affect future coin flips. The unbiased “fair” coin is thus the perfect random bit stream generator, since the “0” and “1” values will be randomly distributed (and [0,1] uniformly distributed). All elements of the sequence are generated independently of each other, and the value of the next element in the sequence cannot be predicted, regardless of how many elements have already been produced. Generally, the outputs of encryption algorithms should be computationally indistinguishable from a truly random source.

In order to compare the Whitenoise algorithm with the Advanced Encryption Standard (AES) algorithm, we conducted the randomness testing of Whitenoise algorithm using the approaches adopted by NIST. NIST statistical tests used up to 16 approaches to evaluate the randomness of 15 AES candidate algorithms [3].

2.1 Description of the Randomness Tests

A brief description of the 16 randomness tests is given in the following, more details can be found in reference [3].

Frequency Test: The purpose of this test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence.

Block Frequency Test: The purpose of this test is to determine whether the frequency of *m*-bit blocks in a sequence appears as often as would be expected for a truly random sequence.

Cumulative Sums Forward (Reverse) Test: The purpose of this test is to determine whether the maximum of the cumulative sums in a sequence is too large or too small; indicative of too many ones or zeroes in the early (late) stages.

Runs Test: The purpose of this test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such sub strings is too fast or too slow.

Long Runs of Ones Test: The purpose of this test is to determine whether the distribution of long runs of ones agrees with the theoretical probabilities.

Rank Test: The purpose of this test is to determine whether the distribution of the rank of 32x32 bit matrices agrees with the theoretical probabilities.

Spectral (Discrete Fourier Transform) Test: The purpose of this test is to determine whether the spectral frequency of the binary sequence agrees with what would be expected for a truly random sequence.

Non-periodic Templates Test: The purpose of this test is to determine whether the number of occurrences for a specified non-periodic template agrees with the number expected for a truly random sequence.

Overlapping Template Test: The purpose of this test is to determine whether the number of occurrences for a template of all ones agrees with what is expected for a truly random sequence.

Universal Statistical Test: The purpose of this test is to determine whether a binary sequence does not compress beyond what is expected of a truly random sequence.

Approximate Entropy Test: The purpose of this test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and $m+1$) against the expected result for a normally distributed sequence. In short, it determines whether a sequence appears more regular than is expected from a truly random sequence.

Random Excursion Test: The purpose of this test is to examine the number of cycles within a sequence and determine whether the number of visits to a given state, [-4, -1] and [1, 4], exceeds that expected for a truly random sequence.

Random Excursion Variant Test: The purpose of this test is to determine if the total number of visits to states, between [-9, -1] and [1, 9] exceeds that expected for a truly random sequence.

Lempel-Ziv Complexity Test: The purpose of this test is to determine whether or not the sequence compresses no more than a truly random sequence.

Linear Complexity Test: The purpose of this test is to determine whether or not the

sequence is complex enough to be considered truly random.

2.2 Data Selection

NIST harnessed and analyzed several sets of data for each of AES candidate algorithms to evaluate randomness. For comparison purpose, we chose similar data categories for the Whitenoise encryption algorithm. However, since Whitenoise is a stream cipher and some of the NIST data sets are selected particularly for the randomness evaluation of block ciphers, we discarded some date categories that are not suitable for stream ciphers. The data categories considered are decribed in the following sections.

2.2.1 Plaintext/Ciphertext Correlation

In order to study the correlation of plaintext/ciphertext pairs, NIST constructed 128 binary sequences (1,048,576 bits per sequence) that is the result of the XOR operator on the plaintext and its corresponding ciphertext. Based on the Whitenoise algorithm, the correlation binary sequence of plaintext/ciphertext pairs can be expressed using Whitenoise super keys. Therefore, 128 super key templates with the size of 1,048,576 bits are randomly generated as a data category of the statistical randomness testing.

2.2.2 Random Plaintext/ Random Keys

In order to examine the randomness of Whitenoise ciphertexts, 128 sequences (1,048,576 bits per sequence) were constructed in this category. Each sequence was a ciphertext encrypted using a random plaintext and a random key.

2.2.3 Low Density Plaintext

128 sequences (1,048,576 bits per sequence) were constructed in this category. Each sequence was a Whitenoise ciphertext encrypted using a random key and an all zeros plaintext.

2.2.4 High density Plaintext

128 sequences (1,048,576 bits per sequence) were constructed in this category. Each sequence was a Whitenoise ciphertext encrypted using a random key and an all ones

plaintext.

2.3 Testing Strategy

In practice, there are many distinct strategies employed in the statistical analysis of a random number generator. NIST has adopted the strategy outlined in Figure 5. Figure 5 provides an architectural illustration of the five stages involved in the statistical testing of a random number generator.

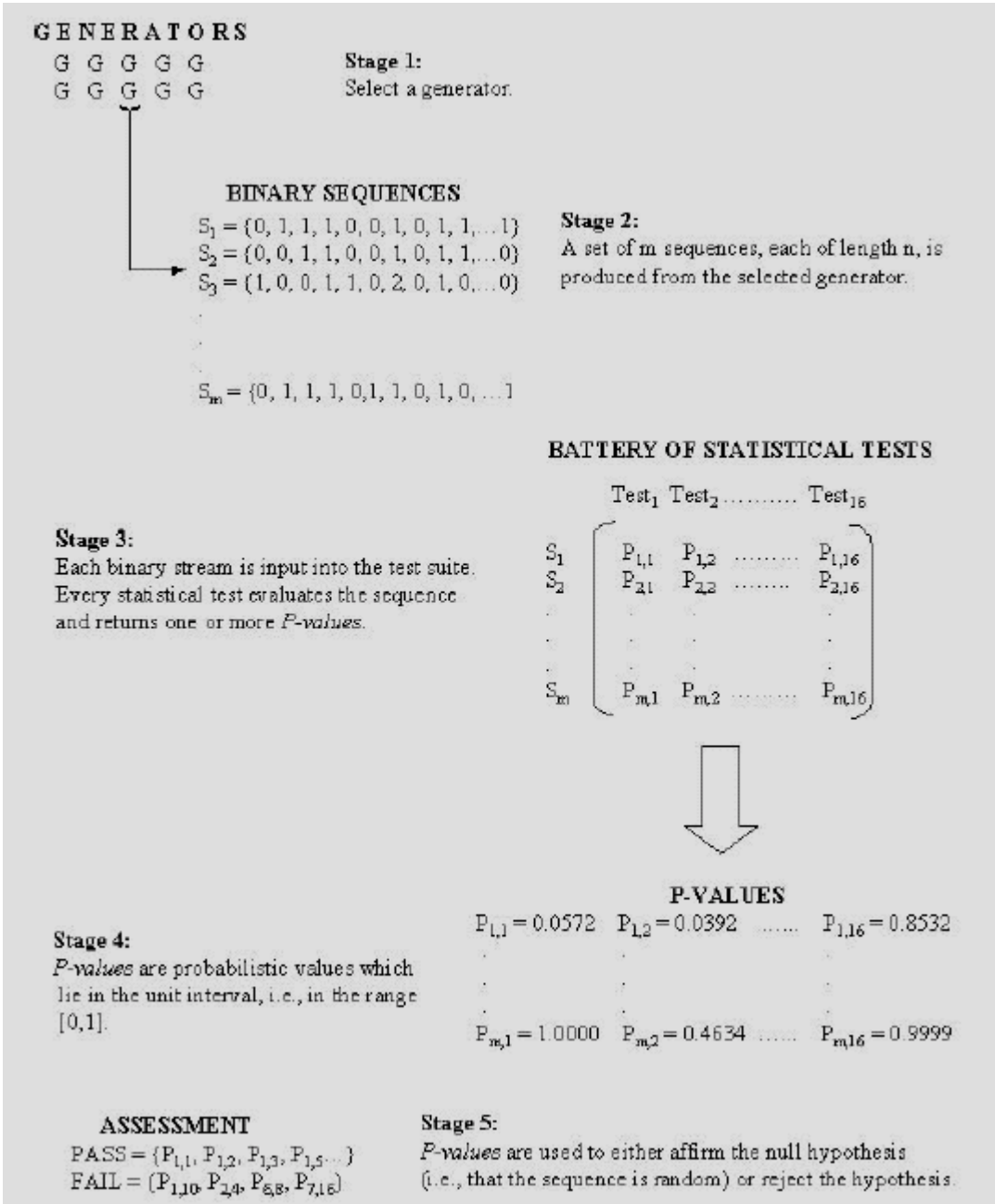

Figure 5. Architecture of the NIST Statistical Test Suite

Stage 1: Selection of a Generator

Select a hardware or software-based random number generator (RNG) for evaluation. The generator should produce a binary sequence of 0's and 1's of a given length *n*. In our evaluation, we used the Whitenoise encryption software as the required RNG.

Stage 2: Binary Sequence Generation

For a fixed sequence of length *n* and the pre-selected generator, construct a set of *m* binary sequences and save the sequences to a file. In our evaluation, four data files that

contain binary sequences were generated using Whitenoise encryption software according to the description of section 2.2.

Stage 3: Execute the Statistical Test Suite

Invoke the Statistical Test Suite using the file produced in Stage 2 and the desired sequence length. Select the statistical tests and relevant input parameters required by individual testing method.

Table 1 shows the relevant input parameters with the specific values applied in the randomness testing of each Whitenoise data file. The values are selected based on NIST recommendations.

Table 1 Input Parameters of Whitenoise Randomness Testing for the NIST statistical Test Suite

Parameter	Value
Number of Binary Sequences	128
Binary Sequence Size (bit)	1,048,576
Block Frequency Test block length	10000
NonOverlapping Template Test Block Length	9
Overlapping Template Test Block Length	9
Universal Test Block Length	7
Universal Test Number Of Initialization Steps	1280
Approximate Entropy Test Block Length	5
Serial Test Block Length	5
Linear Complexity Test Subsequence Length	500

Stage 4: Examine the *P-values*

An output file will be generated by the test suite with relevant intermediate values, such as test statistics, and ***P-values*** for each statistical test. For these tests, each ***P-value*** is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested, given the kind of non-randomness assessed by the test. If a *P-value* for a test is determined to be equal to 1, then the sequence appears to have perfect randomness. A *P-value* of zero indicates that the sequence appears to be completely nonrandom. A significance level (a) can be chosen for the tests. If $P\text{-}value >= a$, then the sequence appears to be random. If $P\text{-}value < a$, then the sequence appears to be non-random. Based on these *P-values*, a conclusion regarding the quality of the sequences can be made. (More details about *P-value* can be found in reference [3])

Stage 5: Assessment: Pass/Fail Assignment

Based on *P-values* and the significance level a, each statistical test can be assigned to pass or fail, which indicates whether the corresponding binary sequence is considered to be random or non-random.

2.4 Statistical Analysis on Empirical Results

The analysis of empirical results can be conducted in any number of ways. Two approaches NIST has adopted include (1) the examination of the proportion of sequences that pass a statistical test and (2) the distribution of *P-values* to check for uniformity.

Given the empirical results for a particular statistical test, compute the proportion of sequences that pass. For example, if 1000 binary sequences were tested (i.e., *m = 1000*), *a = 0.01* (the significance level), and 996 binary sequences had *P-values >=0 .01*, then the proportion is *996/1000 = 0.9960*. The range of acceptable proportions is determined using the confidence interval defined as, $\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$ where ^*p* = 1- *a*, and *m* is the sample size. If the proportion falls outside of this interval, then there is evidence that the data is nonrandom. For the example above, the confidence interval is $.99 \pm 3\sqrt{\frac{.99(.01)}{1000}} = .99 \pm 0.0094392$ (i.e., the proportion should lie above 0.9805607.

The distribution of *P-values* is examined to ensure uniformity. Uniformity may be determined via an application of a χ^2 test and the determination of a *P-value* corresponding to the Goodness-of-Fit Distributional Test on the *P-values* obtained for an arbitrary statistical test (i.e., a *P-value* of the *P-values*). This is accomplished by computing $\chi^2 = \sum_{i=1}^{10} \frac{(F_i - s/10)^2}{s/10}$, where *Fi* is the number of *P-values* in sub-interval *i*, and *s* is the sample size. A *P-valueT* is calculated such that *P-valueT* = $\mathbf{igamc}(9/2, \chi^2/2)$. If *P-valueT >= 0.0001*, then the sequences can be considered to be uniformly distributed.

Tables 7 ~10 in the Appendix give the statistical analysis reports of the four Whitenoise data categories considered. The statistical results are derived from the empirical results obtained from the NIST Statistical Test Suite. The comments based on the statistical analysis results are provided in the following.

Comments: The data set of *Plaintext/Ciphertext Correlation* (super keys) didn't demonstrate deviation from randomness.

The data set of *Random Plaintext/ Random Keys* didn't demonstrate deviation from randomness.

The data set of *Low Density Plaintext* didn't demonstrate deviation from randomness.

The data set of *High Density Plaintext* didn't demonstrate deviation from randomness.

2.5 Comparison with AES Candidate Algorithms

Based on the current version of the NIST statistical tests, each of the algorithms was evaluated [4]. Those algorithms that did not demonstrate deviation from randomness include **CAST-256**, **DFC**, **E2**, **LOKI-97**, **MAGENTA**, **MARS**, **RIJNDAEL**, **SAFER+**, and **SERPENT**. Other algorithms such as **CRYPTON, DEAL, FROG, HPC** are more or less demonstrated problems for some randomness tests.

In our evaluation, using the significance level a=0.001 and 1000 test sequences, the Whitenoise has no failure for any test cases. The Whitenoise algorithm exhibits excellent randomness level.

2.6 Conclusion

The Whitenoise algorithm survived from the statistical testing methods of NIST. Based on the evaluation results, the Whitenoise algorithm demonstrates good randomness capability.

3. Attack Evaluations

Based on the specification of the Whitenoise encryption algorithm, the Whitenoise algorithm uses similar idea as One-Time Pad that is a provably secure cryptosystem. Since the security of One-Time Pad has been mathematically proved, we focused on developing a specific attack on the Whitenoise algorithm implementation. Actually, in practice, most Attacks of encryption algorithms are performed on the algorithm implementations.

More specifically, the security of the Whitenoise algorithm depends on the randomness of its super keys. Two properties of the super keys must hold. (1) Super keys themselves have to exhibit no deviation from truly random binary sequences; (2) super keys must not be predicted or guessed in any encryption process. Property 1 has been verified in section 3. Property 2 is used as criteria for developing an attack in this part of evaluation process.

We developed an attack on the condition that the ciphertext and part of the plaintext (initial part) are known. More specifically, the attack aims to derive the whole super key

by reconstructing all the subkeys. The concrete attack can be divided into three steps.

Step1. Build subkey libraries

A Whitenoise subkey is constructed by two random perfect square numbers collected from the computer random number generator. Since the perfect square numbers generated by computers are finite, it is feasible to build a library that contains all the possible subkey templates. Each subkey template has the same length with the ciphertext. Suppose that there are in total *N* numbers that can be created by the computer, the number of the subkey templates will be $\mathbf{N^2}$ in the library.

Step2. Guess the constituted subkeys

Here, we assume that the Whitenoise algorithm constructs super keys using M subkeys (M is supposed to be a small integer) and the shortest subkey is of length K bits. There are totally $\mathbf{N^{2M}}$ possibilities for the set of constituted subkey templates, but we very possibly don't have to try all the possibilities at this step. More specifically, we select M subkey templates and construct a super key of K bit length, then encrypt the known initial part of the plaintext and compare the result with the corresponding bits of the known ciphertext. Once a matching happens, we can assume that the current subkey templates are the actual constituted subkey templates. (The possibility increases as K increases).

Step3. Guess the Super Key

Having obtained the possible subkey templates from step 2, the super key is ready to be reconstructed. Suppose that the Whitenoise algorithm has L prime numbers to construct an actual subkey, the total possibilities of the super key are $\mathbf{L^M}$. Each reconstructed super key can be verified using the known ciphertext and the plaintext part until a matching happens.

The attack can become infeasible if either **N or L** has a large value. Due to the software restrictions, we can't fully implement the described attacks. But, we surely can evaluate the attack feasibility by observing the values of **N** and **L**. Based on the Whitenoise key generation algorithm, **L** is known by the algorithm. Therefore, evaluating the value of N becomes critical.

In our attack evaluation, we actually implemented a program that can estimate the value of **N** by observing the result of repeated key generation process. More specifically, a random super key, whose shortest subkey is of length S, is created. And a plaintext that has the length *S* is encrypted using the random superkey. The resulted ciphertext is recorded. Then, super keys whose shortest subkeys have length *S* are repeatedly generated and encrypt the same plaintext. The resulted ciphertexts are compared with the recorded ciphertext, and the comparison times are counted (**T** is used to express the counter value). This process will not stop until two identical ciphertexts are found. Based

on Whitenoise algorithm, there is a relationship between T and N. $C^{M}_{N^2} \approx T$ is the equation that can be used to express the relationship. Therefore, we can estimate N by observing T.

Our attack evaluation program is executed on 6 Unix workstations in parallel; each workstation runs one evaluation case. Table 2 shows the details.

Table 2 Whitenoise Attack Evaluation

Workstation Name	Superkey Size	Shortest Subkey Length(S)	T values (so far)	Evaluation Status
oumou	10 subkeys	31	124 + 194 x 10E4	In Progress
mariame	10 subkeys	31	1389 + 171 x 10E4	In Progress
nae	10 subkeys	31	9357 + 170 x 10E4	In Progress
mekon	10 subkeys	197	13 + 172 x 10E4	In Progress
magout	10 subkeys	197	7241 + 171 x 10E4	In Progress
astou	10 subkeys	197	3467+ 170 x 10E4	In Progress

The evaluation program has run for a week, there is no identical ciphertexts found so far with the average of approximate 1,800,000 times of key generation.

4. Performance Evaluations

Performance of an algorithm can be measured by its computational efficiency (speed) and memory requirements, which can be affected by many factors. Some factors are associated with the algorithm structure, such as the operations involved, encryption procedures, etc. Other factors are associated with algorithm implementations (software/hardware implementation, implementation languages, etc.) and executing environment (various platforms). In this round of evaluation, we focus on evaluating the current Whitenoise encryption implementation using four criteria that include *key generation speed with different key size, encryption speed with different key size, encryption speed with different plaintext size,* and *encryption speed with different plaintext format.* The testing platform is a Pentium IV Mobile processor, 512 DDR RAM, Windows 2000 P.

4.1 Key generation speed with different key size

The super key of Whitenoise encryption algorithm is created using several subkeys. The **key size** represents the number of subkeys constituted. We evaluated the effect of the key size on the speed of super key generation. Based on the current Whitenoise software implementation, we tested 10 different key sizes (1, 2, 3, 4, … , 10), and we generated 10,000 sample keys for each key size to get the average time.

Table 3 shows the details of the evaluation results, Column 1 represents different key sizes with the subkey lengths used; Column 2 represents the corresponding average generation time of key file on 10,000 samples.

Table 3. Key Generation Time (Pentium 4 Mobile processor, 512 DDR RAM, Windows 2000 P)

Number of subkeys (Key size)	Average Time (Millisecond) (10,000 samples)
1 sub key (251)	0.4071550993
2 sub keys (251, 241)	0.6398941765
3 sub keys (251, 241, 239)	0.8770164923
4 sub keys (251, 241, 239, 233)	1.1308783658
5 sub keys (251, 241, 239, 233, 229)	1.3411844243
6 sub keys (251,241,239,233, 229, 227)	1.5673863578
7 sub keys (251, 241, 239, 233, 229, 227, 223)	1.8042712132
8 sub keys ((251, 241, 239, 233, 229, 227, 223, 211)	2.0239415903
9 sub keys (251, 241, 239, 233, 229, 227, 223, 211, 199)	2.1779052924
10 sub keys (251, 241, 239, 233, 229, 227, 223,211, 199, 197)	2.3662700148

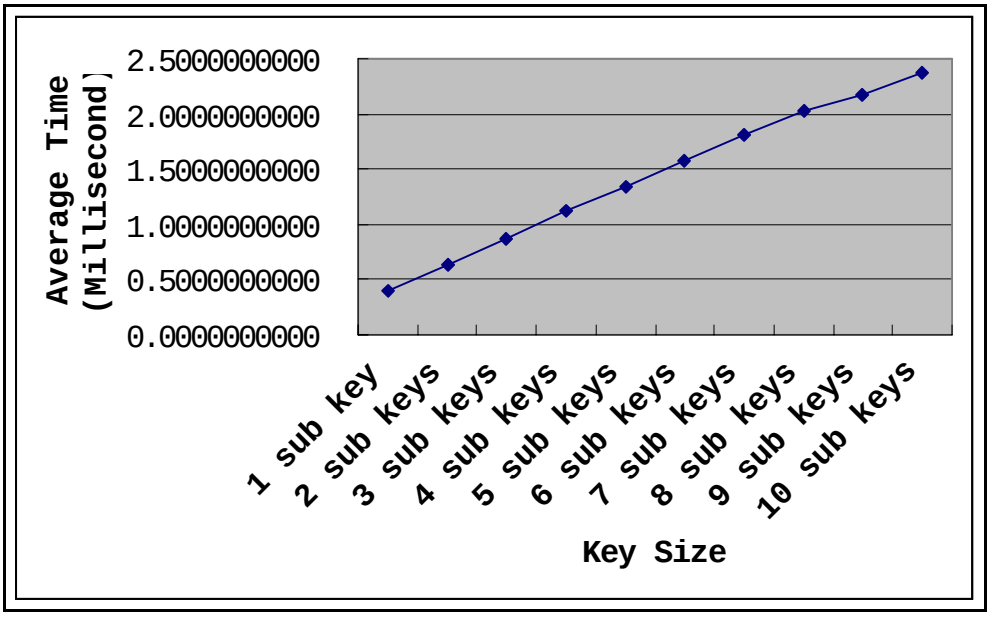

Figure 1. Key Generation Time

Figure 1 shows a chart representation of the Whitenoise key generation time corresponding to Table 3.

Comments: Key generation time increases as the key size increases; there is a linear relationship between the key generation speed and the key size. For instance, an equation, $T=\gamma N+\delta$, can be used to express the linear relationship between the key size N (the number of subkeys created) and T, the average time of key generation. In this case, based on Table 1, γ is around 0.2 ms/subkey, and δ is around 0.2 ms. The low values of γ and δ shows that the magnitude of the increase of key generation as the key size increases is not significant.

4.2 Encryption speed with different key size

The evaluation of the encryption/decryption speed with different key size is measured on small and large files using a 100 bytes file and a 10MB file. Based on the current software implementation, we tested 10 different key sizes (1, 3, 4, …, 10), and generated 10,000 keys for each key size to get the average encryption time.

Table 4 shows the details of the evaluation results, Column 1 represents different key sizes with the subkey lengths used; Column 2 represents the corresponding average encryption time on a 10 Mb file of 10,000 samples; Column 3 represents the corresponding average encryption time on a 100 bytes file of 10,000 samples.

Table 4. Encryption Time of Different Key Size (Pentium 4 Mobile processor, 512 DDR RAM, Windows 2000 P)

Number of subkeys(Key Size)	Average Encryption Time on 10 Mb File (Millisecond) (10,000samples)	Average Encryption Speed on 100 byte File (Millisecond) (10,000 samples)
1 sub key (251)	1309.3744875400	0.5654832388
2 sub keys (251, 241)	1323.8988195427	0.5877714905
3 sub keys (251, 241, 239)	1311.0750668032	0.5312407024
4 sub keys (251, 241, 239, 233)	1137.7004898667	0.5687315008
5 sub keys (251, 241, 239, 233, 229)	1222.2064536135	0.6051048387
6 sub keys (251,241,239,233, 229, 227)	1113.9326366899	0.6038756232
7 sub keys (251, 241, 239, 233, 229, 227, 223)	1151.2914630211	0.6891378653
8 sub keys ((251, 241, 239, 233, 229, 227, 223, 211)	1089.7807402896	0.6568000961
9 sub keys (251, 241, 239, 233, 229, 227, 223, 211, 199)	1178.7120988841	0.6469143844
10 sub keys (251, 241, 239, 233, 229, 227, 223,211, 199, 197)	1187.7515550161	0.6540191561

Figure 2 shows a chart representation of the Whitenoise encryption time corresponding to Table 4. In order to highlight the trend of the data flow, *log* values of the speed data are chosen instead of the original data.

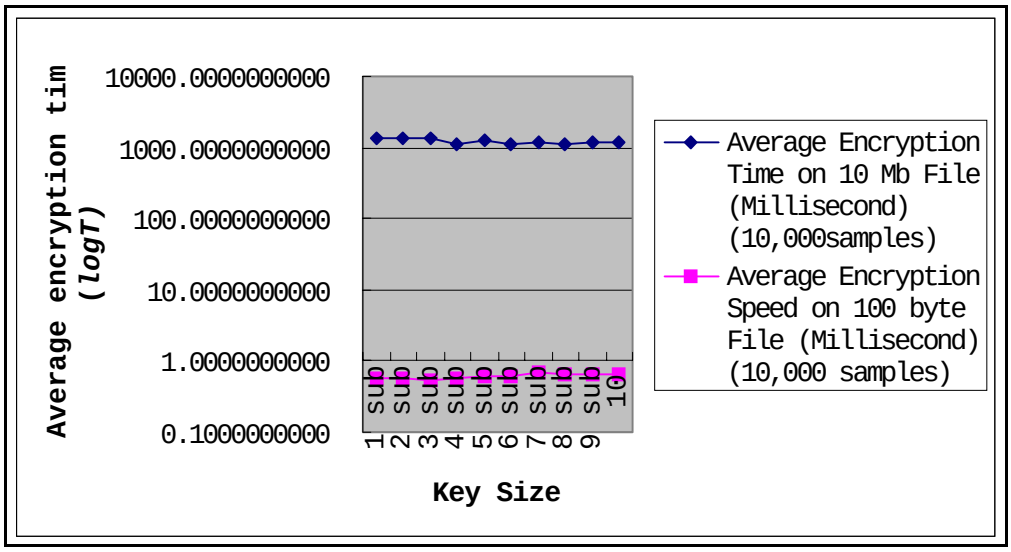

Figure 2. Encryption Time

Comments: there is no obvious variation on the time cost when encrypting using different key sizes. The Key size (i.e. the number of subkeys) of the Whitenoise encryption algorithm has no significant effect on the time cost of the encryption process.

4.3 Encryption speed with different plaintext size

This evaluation aims at measuring the relationship between Whitenoise encryption speed and the size of plaintexts. Based on the evaluation results in section 4.2, key size doesn't affect the encryption speed. A set of super key templates was randomly created for this evaluation, and a set of files whose sizes range from 10 bytes to 10Mbs was selected as the candidate plaintexts. For each super key, the encryption process was performed on each of the plaintexts using the same super key. And the process was repeated for each of the super keys to obtain the average encryption speed.

Table 5 shows the details of the evaluation results, Column 1 represents the size of plaintexts. Column 2 represents the corresponding average encryption time over the set of random super keys.

Table 5. Encryption Time with Different File Size (Pentium 4 Mobile processor, 512 DDR RAM, Windows 2000 P)

Plaintext Size (byte)	Average Encryption time (Millisecond) (1000 Sample Keys)
10	0.6509778449
100	0.6660898228
1000	0.6781047172
10k	0.7148059320
100k	7.1783348303
1M	186.1963405964
10M	1247.4841243789

Figure 3 shows a chart representation of the Whitenoise encryption time corresponding to Table 5. In order to highlight the trend of the data flow, *log* values of the original data are chosen for the coordinates.

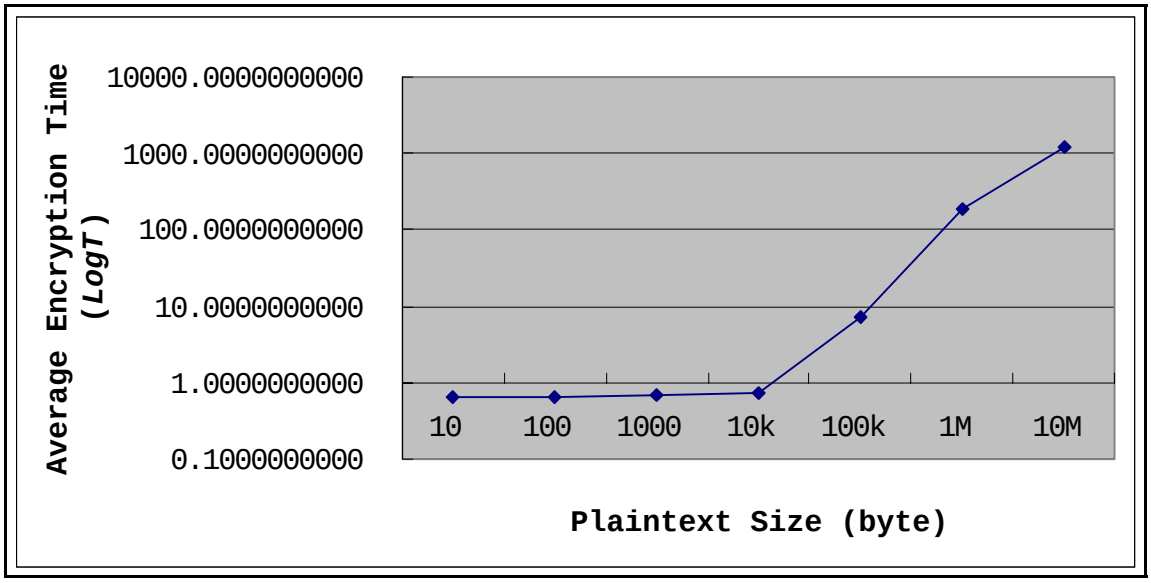

Figure 3. Encryption Speed with Different Plaintext Size

Comments: Based on the evaluation results, the encryption time remains constant and extremely low when the plaintext size is under 10k (probably because the saving time is dominant compared with the encryption time). As the size of the plaintext increases above 10k, the encryption time increases linearly at the same level as the size of the plaintext increases. But the time increase still remains negligible. For instance, an equation, $logT=\alpha logS+\beta$, can be used to express the relationship between the encryption time T and plaintext size S, when the plaintext size is over 10k. When the plaintext size is less or equl to 10k, α can be considered as 0; in this case, based on Table 5, β is around 1; therefore $logT=1$ *implying* represents T≈0s.

4.4 Encryption speed with different plaintext format

This evaluation was performed using two sets of plaintexts; each set of plaintexts consists of files that have the same size, but different format (text, sound, image, PDF). A random key is used to encrypt the data. Based on the evaluation results of section 4.3, it is better to use plaintexts whose sizes are over 10k to demonstrate the encryption speed. In this evaluation, **10M** and **40K** were chosen as the size of the plaintexts. All the plaintexts were encrypted using an identical key for 100 times (in order to obtain the average encryption speed) and the encryption speeds were recorded and compared.

Table 6 shows the details of the evaluation results, column 1 represents the format of the plaintexts; column 2 represents the corresponding average encryption time on the 40k

files using the same random key; column 3 represents the corresponding average encryption time on the 10M files using the same random key.

Table 6. Encryption Time with Different Plaintext Format (Pentium 4 Mobile processor, 512 DDR RAM, Windows 2000 P)

Plaintext Format	Average Encryption Speed (100 times repeated encryption)
text	4.6503344005E-03
image	4.2486967599E-03
sound	4.0115154461E-03
PDF	4.6230241875E-03

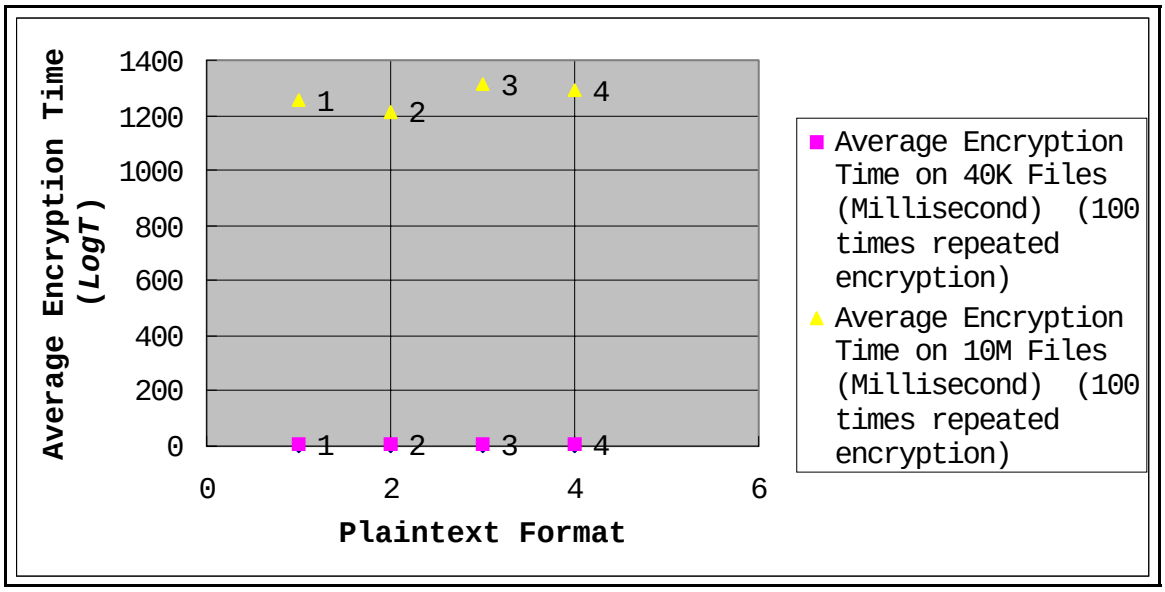

Figure 4. Encryption Speed with Different Plaintext Format

Figure 4 shows a chart representation of the Whitenoise encryption time corresponding to Table 6.

Comments: according to Table 3, the magnitude of the variation of the encryption time using different file format is negligible. We can make a conclusion that the plaintext format has no significant effect on the Whitenoise encryption speed.

5. Summary

In this mid-term evaluation report, we provided the evaluation results of the Whitenoise encryption algorithm for the performance testing, the output randomness testing, and the algorithm attack analysis. Our evaluation investigated the Whitenoise algorithm under several perspectives that are essential for cryptographic systems. The Whitenoise super key generation speed is associated with the key size. The Whitenoise encryption speed has a linear relationship with the plaintext size, but remains the same for different plaintext formats. The Whitenoise algorithm has a provable mathematical foundation, and the output of the Whitenoise algorithm exhibit good randomness based on the NIST statistical testing approaches. Also, a possible attack on the Whitenoise implementation is discussed.

Reference

[1] S. Boren, A. Brisson, *Software Specifications for Whitenoise Encryption*, Confidential Document, BSB Utilities Inc.

[2] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, E. Roback, *Report on the Development of the Advanced Encryption Standard (AES)*, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, October 2, 2000.

[3] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, *A Statistical Test Suite For Random And Pseudorandom Number Generators For Cryptographic Applications*, NIST Special Publication 800-22, December 2000.

[4] J. Soto, Jr. *Randomness Testing of the AES Candidate Algorithms*, National Institute of Standards and Technology,

Appendix

Tables 7 ~10 in the Appendix give the statistical analysis reports of the four Whitenoise data categories considered with a=0.001 and 1000 data sequences. The statistical results are derived from the empirical results obtained from the NIST Statistical Test Suite. Column 1 represents the randomness testing method; column 2 represents the proportion of binary sequences that passed the corresponding test; column 3 represents the uniformity checking result for the corresponding test, *P-valueT* is calculated to express the uniformity.

Table 7. Statistical Analysis Report on *Plaintext/Ciphertext Correlation* Data Category (a=0.001, 1000 data sequences)

Test Method	Proportion (>=0.96371)	P-valueT (>=0.0001)
Frequency	0.9847	0.286306
Block-Frequency	1	0.100508
Cusum	0.9944	0.160294
Cusum	0.9766	0.134686
Runs	0.9766	0.407091
Long-Run	0.9922	0.275709
Rank	1	0.407091
FFT	0.9766	0.287306
Aperiodic-Template	1	0.407091
Aperiodic-Template	1	0.875539
Aperiodic-Template	0.9922	0.105618
Aperiodic-Template	1	0.222869
Aperiodic-Template	1	0.619772
Aperiodic-Template	1	0.77276
Aperiodic-Template	0.9922	0.023812
Aperiodic-Template	0.9922	0.888137
Aperiodic-Template	0.9844	0.484646
Aperiodic-Template	0.9766	0.074177
Aperiodic-Template	1	0.551026
Aperiodic-Template	0.9922	0.991468
Aperiodic-Template	0.9844	0.110952
Aperiodic-Template	0.9922	0.500934
Aperiodic-Template	1	0.031497
Aperiodic-Template	0.9922	0.299251

Aperiodic-Template	0.9688	0.013411
Aperiodic-Template	1	0.422034
Aperiodic-Template	0.9766	0.32418
Aperiodic-Template	0.9922	0.804337
Aperiodic-Template	1	0.170294
Aperiodic-Template	1	0.350485
Aperiodic-Template	0.9922	0.484646
Aperiodic-Template	0.9922	0.689019
Aperiodic-Template	0.9766	0.706149
Aperiodic-Template	0.9922	0.706149
Aperiodic-Template	0.9844	0.253551
Aperiodic-Template	0.9922	0.311542
Aperiodic-Template	1	0.222869
Aperiodic-Template	0.9844	0.364146
Aperiodic-Template	0.9922	0.468595
Aperiodic-Template	0.9922	0.037157
Aperiodic-Template	1	0.468595
Aperiodic-Template	0.9844	0.253551
Aperiodic-Template	0.9766	0.788728
Aperiodic-Template	0.9844	0.619772
Aperiodic-Template	0.9844	0.204076
Aperiodic-Template	0.9922	0.941144
Aperiodic-Template	0.9922	0.517442
Aperiodic-Template	0.9766	0.568055
Aperiodic-Template	1	0.484646
Aperiodic-Template	0.9844	0.689019
Aperiodic-Template	0.9844	0.534146
Aperiodic-Template	0.9766	0.082177
Aperiodic-Template	1	0.862344
Aperiodic-Template	0.9922	0.654467
Aperiodic-Template	0.9922	0.23276
Aperiodic-Template	0.9922	0.97606
Aperiodic-Template	1	0.110952
Aperiodic-Template	1	0.009422
Aperiodic-Template	1	0.392456
Aperiodic-Template	0.9922	0.888137
Aperiodic-Template	1	0.468595

Aperiodic-Template	0.9766	0.066882
Aperiodic-Template	0.9844	0.186566
Aperiodic-Template	0.9844	0.134686
Aperiodic-Template	0.9922	0.350485
Aperiodic-Template	0.9922	0.970538
Aperiodic-Template	0.9922	0.671779
Aperiodic-Template	0.9922	0.637119
Aperiodic-Template	0.9766	0.392456
Aperiodic-Template	0.9922	0.134686
Aperiodic-Template	0.9844	0.074177
Aperiodic-Template	0.9766	0.078086
Aperiodic-Template	0.9766	0.32418
Aperiodic-Template	0.9844	0.043745
Aperiodic-Template	0.9922	0.619772
Aperiodic-Template	0.9922	0.804337
Aperiodic-Template	0.9922	0.275709
Aperiodic-Template	0.9922	0.082177
Aperiodic-Template	0.9922	0.551026
Aperiodic-Template	1	0.051391
Aperiodic-Template	0.9922	0.275709
Aperiodic-Template	1	0.706149
Aperiodic-Template	1	0.407091
Aperiodic-Template	1	0.452799
Aperiodic-Template	0.9922	0.900104
Aperiodic-Template	1	0.568055
Aperiodic-Template	0.9766	0.122325
Aperiodic-Template	0.9922	0.337162
Aperiodic-Template	0.9844	0.026648
Aperiodic-Template	0.9922	0.155209
Aperiodic-Template	0.9766	0.337162
Aperiodic-Template	1	0.128379
Aperiodic-Template	0.9922	0.468595
Aperiodic-Template	0.9844	0.819544
Aperiodic-Template	1	0.484646
Aperiodic-Template	0.9844	0.253551
Aperiodic-Template	0.9844	0.452799
Aperiodic-Template	0.9922	0.028181

Aperiodic-Template	1	0.888137
Aperiodic-Template	0.9766	0.32418
Aperiodic-Template	0.9922	0.848588
Aperiodic-Template	1	0.23276
Aperiodic-Template	0.9922	0.888137
Aperiodic-Template	0.9766	0.264458
Aperiodic-Template	0.9922	0.186566
Aperiodic-Template	0.9922	0.756476
Aperiodic-Template	0.9766	0.204076
Aperiodic-Template	0.9766	0.311542
Aperiodic-Template	0.9922	0.468595
Aperiodic-Template	0.9922	0.517442
Aperiodic-Template	0.9922	0.756476
Aperiodic-Template	0.9766	0.264458
Aperiodic-Template	1	0.949602
Aperiodic-Template	1	0.086458
Aperiodic-Template	0.9844	0.015963
Aperiodic-Template	1	0.242986
Aperiodic-Template	0.9844	0.392456
Aperiodic-Template	0.9844	0.063482
Aperiodic-Template	0.9922	0.756476
Aperiodic-Template	1	0.031497
Aperiodic-Template	1	0.253551
Aperiodic-Template	0.9844	0.110952
Aperiodic-Template	0.9688	0.170294
Aperiodic-Template	0.9922	0.32418
Aperiodic-Template	0.9922	0.082177
Aperiodic-Template	0.9922	0.517442
Aperiodic-Template	1	0.299251
Aperiodic-Template	0.9922	0.568055
Aperiodic-Template	1	0.090936
Aperiodic-Template	0.9766	0.875539
Aperiodic-Template	0.9922	0.204076
Aperiodic-Template	1	0.128379
Aperiodic-Template	1	0.407091
Aperiodic-Template	0.9844	0.756476
Aperiodic-Template	0.9922	0.213309

Aperiodic-Template	0.9844	0.311542
Aperiodic-Template	0.9766	0.017912
Aperiodic-Template	0.9844	0.063482
Aperiodic-Template	0.9844	0.517442
Aperiodic-Template	1	0.437274
Aperiodic-Template	0.9922	0.484646
Aperiodic-Template	0.9766	0.116519
Aperiodic-Template	0.9844	0.848588
Aperiodic-Template	0.9922	0.637119
Aperiodic-Template	0.9766	0.178278
Aperiodic-Template	0.9922	0.007422
Aperiodic-Template	1	0.134686
Aperiodic-Template	1	0.931952
Aperiodic-Template	0.9766	0.066882
Aperiodic-Template	0.9922	0.468595
Aperiodic-Template	1	0.204076
Aperiodic-Template	0.9922	0.422034
Aperiodic-Template	0.9922	0.029796
Aperiodic-Template	0.9922	0.350485
Aperiodic-Template	1	0.468595
Aperiodic-Template	1	0.437274
Periodic-Template	0.9766	0.195163
Universal	1	0.253551
Apen	0.9922	0.00788
Random-Excursion	0.988	0.546791
Random-Excursion	0.988	0.919445
Random-Excursion	0.9639	0.472584
Random-Excursion	1	0.425817
Random-Excursion	1	0.496841
Random-Excursion	0.988	0.823278
Random-Excursion	0.9759	0.340461
Random-Excursion	0.988	0.109597
Random-Excursion-V	0.9718	0.109597
Random-Excursion-V	0.9718	0.676097
Random-Excursion-V	0.9798	0.340461
Random-Excursion-V	0.9718	0.026277
Random-Excursion-V	0.9718	0.117948

Random-Excursion-V	0.9768	0.075138
Random-Excursion-V	0.9639	0.026277
Random-Excursion-V	0.9639	0.823278
Random-Excursion-V	0.988	0.752361
Random-Excursion-V	0.988	0.845066
Random-Excursion-V	0.988	0.845066
Random-Excursion-V	0.988	0.598138
Random-Excursion-V	1	0.624107
Random-Excursion-V	1	0.302291
Random-Excursion-V	1	0.959132
Random-Excursion-V	1	0.448892
Random-Excursion-V	0.9639	0.5216
Random-Excursion-V	0.9639	0.220448
Serial	1	0.619772
Serial	1	0.756476
Lempel-Ziv	1	0.576478
Linear-Complexity	0.9922	0.602458

Comments: The data set of *Plaintext/Ciphertext Correlation* didn't demonstrate deviation from randomness.

Table 8. Statistical Analysis Report on *Random Plaintext/ Random Keys* Data Category (a=0.001, 1000 data sequences)

Test Method	Proportion (>=0.96371)	P-valueT (>=0.0001)
Frequency	0.9922	0.264458
Block-Frequency	0.9922	0.054199
Cusum	0.9922	0.739918
Cusum	0.9922	0.788728
Runs	0.9766	0.035174
Long-Run	0.9766	0.155209
Rank	1	0.500934
FFT	1	0.585209
Aperiodic-Template	1	0.452799
Aperiodic-Template	0.9922	0.654467
Aperiodic-Template	0.9631	0.654467
Aperiodic-Template	0.9844	0.804337

Aperiodic-Template	0.9688	0.452799
Aperiodic-Template	0.9844	0.337162
Aperiodic-Template	0.9766	0.671779
Aperiodic-Template	0.9922	0.110952
Aperiodic-Template	0.9922	0.77276
Aperiodic-Template	0.9922	0.128379
Aperiodic-Template	0.9766	0.242986
Aperiodic-Template	1	0.195163
Aperiodic-Template	0.9844	0.585209
Aperiodic-Template	1	0.484646
Aperiodic-Template	0.9844	0.756476
Aperiodic-Template	0.9922	0.689019
Aperiodic-Template	0.9922	0.066882
Aperiodic-Template	0.9922	0.063482
Aperiodic-Template	0.9922	0.32418
Aperiodic-Template	0.9922	0.848588
Aperiodic-Template	0.9922	0.706149
Aperiodic-Template	1	0.009998
Aperiodic-Template	0.9844	0.253551
Aperiodic-Template	1	0.222869
Aperiodic-Template	1	0.788728
Aperiodic-Template	0.9766	0.001399
Aperiodic-Template	0.9766	0.350485
Aperiodic-Template	0.9844	0.988549
Aperiodic-Template	0.9922	0.875539
Aperiodic-Template	0.9922	0.484646
Aperiodic-Template	1	0.074177
Aperiodic-Template	0.9922	0.517442
Aperiodic-Template	0.9922	0.862344
Aperiodic-Template	0.9766	0.689019
Aperiodic-Template	1	0.162606
Aperiodic-Template	1	0.070445
Aperiodic-Template	0.9922	0.162606
Aperiodic-Template	1	0.021262
Aperiodic-Template	0.9922	0.392456
Aperiodic-Template	0.9922	0.689019
Aperiodic-Template	0.9844	0.931952

Aperiodic-Template	1	0.350485
Aperiodic-Template	1	0.170294
Aperiodic-Template	0.9922	0.723129
Aperiodic-Template	0.9844	0.468595
Aperiodic-Template	0.9844	0.637119
Aperiodic-Template	0.9688	0.77276
Aperiodic-Template	0.9922	0.195163
Aperiodic-Template	0.9844	0.484646
Aperiodic-Template	0.9766	0.275709
Aperiodic-Template	0.9844	0.900104
Aperiodic-Template	0.9922	0.082177
Aperiodic-Template	0.9844	0.275709
Aperiodic-Template	0.9844	0.186566
Aperiodic-Template	0.9688	0.32418
Aperiodic-Template	0.9922	0.619772
Aperiodic-Template	1	0.437274
Aperiodic-Template	0.9922	0.671779
Aperiodic-Template	1	0.848588
Aperiodic-Template	0.9922	0.500934
Aperiodic-Template	0.9922	0.090936
Aperiodic-Template	0.9844	0.195163
Aperiodic-Template	1	0.437274
Aperiodic-Template	0.9844	0.186566
Aperiodic-Template	1	0.32418
Aperiodic-Template	0.9922	0.468595
Aperiodic-Template	1	0.654467
Aperiodic-Template	0.9922	0.723129
Aperiodic-Template	0.9922	0.500934
Aperiodic-Template	0.9844	0.364146
Aperiodic-Template	0.9844	0.311542
Aperiodic-Template	0.9922	0.407091
Aperiodic-Template	0.9766	0.862344
Aperiodic-Template	0.9922	0.016911
Aperiodic-Template	1	0.337162
Aperiodic-Template	0.9844	0.186566
Aperiodic-Template	0.9844	0.204076
Aperiodic-Template	0.9922	0.900104

Aperiodic-Template	0.9844	0.756476
Aperiodic-Template	0.9922	0.723129
Aperiodic-Template	0.9844	0.77276
Aperiodic-Template	0.9766	0.619772
Aperiodic-Template	0.9844	0.021262
Aperiodic-Template	0.9922	0.337162
Aperiodic-Template	0.9922	0.350485
Aperiodic-Template	1	0.170294
Aperiodic-Template	1	0.031497
Aperiodic-Template	1	0.350485
Aperiodic-Template	0.9922	0.484646
Aperiodic-Template	0.9922	0.862344
Aperiodic-Template	1	0.585209
Aperiodic-Template	0.9922	0.77276
Aperiodic-Template	0.9922	0.911413
Aperiodic-Template	1	0.484646
Aperiodic-Template	0.9844	0.900104
Aperiodic-Template	0.9922	0.654467
Aperiodic-Template	0.9844	0.602458
Aperiodic-Template	0.9688	0.051391
Aperiodic-Template	0.9922	0.911413
Aperiodic-Template	0.9922	0.568055
Aperiodic-Template	0.9766	0.029796
Aperiodic-Template	0.9922	0.706149
Aperiodic-Template	0.9688	0.134686
Aperiodic-Template	0.9922	0.957319
Aperiodic-Template	0.9844	0.078086
Aperiodic-Template	0.9922	0.155209
Aperiodic-Template	0.9922	0.654467
Aperiodic-Template	1	0.452799
Aperiodic-Template	0.9844	0.016911
Aperiodic-Template	0.9922	0.452799
Aperiodic-Template	0.9922	0.911413
Aperiodic-Template	0.9766	0.756476
Aperiodic-Template	1	0.122325
Aperiodic-Template	0.9922	0.422034
Aperiodic-Template	0.9922	0.422034

Aperiodic-Template	0.9922	0.964295
Aperiodic-Template	0.9922	0.017912
Aperiodic-Template	1	0.900104
Aperiodic-Template	0.9922	0.134686
Aperiodic-Template	0.9766	0.148094
Aperiodic-Template	0.9844	0.739918
Aperiodic-Template	1	0.534146
Aperiodic-Template	0.9922	0.022503
Aperiodic-Template	0.9922	0.392456
Aperiodic-Template	0.9688	0.213309
Aperiodic-Template	1	0.585209
Aperiodic-Template	0.9844	0.997147
Aperiodic-Template	0.9922	0.299251
Aperiodic-Template	0.9844	0.122325
Aperiodic-Template	0.9922	0.054199
Aperiodic-Template	0.9609	0.350485
Aperiodic-Template	1	0.337162
Aperiodic-Template	0.9922	0.337162
Aperiodic-Template	0.9844	0.689019
Aperiodic-Template	1	0.299251
Aperiodic-Template	0.9922	0.834308
Aperiodic-Template	0.9922	0.028181
Aperiodic-Template	0.9766	0.222869
Aperiodic-Template	0.9922	0.568055
Aperiodic-Template	0.9922	0.911413
Aperiodic-Template	0.9922	0.178278
Aperiodic-Template	0.9922	0.128379
Aperiodic-Template	0.9922	0.980883
Aperiodic-Template	0.9766	0.585209
Aperiodic-Template	0.9844	0.654467
Aperiodic-Template	0.9922	0.350485
Aperiodic-Template	0.9922	0.195163
Aperiodic-Template	0.9922	0.021262
Periodic-Template	0.9922	0.337162
Universal	0.9688	0.100508
Apen	0.9922	0.689019
Random-Excursion	0.9875	0.414525

Random-Excursion	0.9875	0.414525
Random-Excursion	0.9875	0.855534
Random-Excursion	0.9875	0.611108
Random-Excursion	1	0.371101
Random-Excursion	1	0.509162
Random-Excursion	0.9625	0.027405
Random-Excursion	0.9875	0.414525
Random-Excursion-V	1	0.894201
Random-Excursion-V	0.9875	0.460664
Random-Excursion-V	0.975	0.392456
Random-Excursion-V	0.975	0.764655
Random-Excursion-V	0.975	0.689019
Random-Excursion-V	0.9875	0.875539
Random-Excursion-V	1	0.484646
Random-Excursion-V	0.9875	0.186566
Random-Excursion-V	0.9875	0.559523
Random-Excursion-V	1	0.151616
Random-Excursion-V	0.975	0.078086
Random-Excursion-V	0.975	0.66313
Random-Excursion-V	1	0.534146
Random-Excursion-V	1	0.122325
Random-Excursion-V	1	0.927083
Random-Excursion-V	0.9875	0.392456
Random-Excursion-V	1	0.162606
Random-Excursion-V	0.9875	0.66313
Serial	0.9922	0.32418
Serial	1	0.551026
Lempel-Ziv	1	0.4567888
Linear-Complexity	0.9922	0.819544

Comments: The data set of *Random Plaintext/ Random Keys* didn't demonstrate deviation from randomness.

Table 9. Statistical Analysis Report on *Low Density Plaintext* Data Category (a=0.001, 1000 data sequences)

Test Method	Proportion (>=0.96371)	P-valueT (0.0001)
Frequency	1	0.337162

Block-Frequency	0.9844	0.834308
Cusum	1	0.242986
Cusum	1	0.275709
Runs	0.9844	0.128379
Long-Run	0.9844	0.452799
Rank	0.9922	0.078086
FFT	1	0.422034
Aperiodic-Template	0.9844	0.253551
Aperiodic-Template	0.9844	0.148094
Aperiodic-Template	0.9922	0.888137
Aperiodic-Template	0.9922	0.619772
Aperiodic-Template	0.9609	0.602458
Aperiodic-Template	0.9922	0.32418
Aperiodic-Template	1	0.148094
Aperiodic-Template	0.9922	0.063482
Aperiodic-Template	0.9675	0.311542
Aperiodic-Template	0.9922	0.100508
Aperiodic-Template	1	0.100508
Aperiodic-Template	1	0.364146
Aperiodic-Template	0.9844	0.001588
Aperiodic-Template	0.9688	0.242986
Aperiodic-Template	0.9766	0.900104
Aperiodic-Template	0.9844	0.422034
Aperiodic-Template	0.9766	0.043745
Aperiodic-Template	0.9922	0.77276
Aperiodic-Template	0.9922	0.01125
Aperiodic-Template	0.9922	0.834308
Aperiodic-Template	0.9844	0.534146
Aperiodic-Template	0.9922	0.002316
Aperiodic-Template	0.9766	0.941144
Aperiodic-Template	0.9766	0.922036
Aperiodic-Template	0.9922	0.706149
Aperiodic-Template	0.9766	0.178278
Aperiodic-Template	0.9844	0.32418
Aperiodic-Template	1	0.637119
Aperiodic-Template	1	0.723129
Aperiodic-Template	0.9922	0.222869

Aperiodic-Template	0.9688	0.026648
Aperiodic-Template	0.9922	0.141256
Aperiodic-Template	1	0.484646
Aperiodic-Template	0.9766	0.689019
Aperiodic-Template	1	0.392456
Aperiodic-Template	0.9922	0.116519
Aperiodic-Template	0.9922	0.900104
Aperiodic-Template	0.9922	0.452799
Aperiodic-Template	0.9922	0.911413
Aperiodic-Template	0.9922	0.287306
Aperiodic-Template	0.9766	0.186566
Aperiodic-Template	1	0.155209
Aperiodic-Template	0.9844	0.819544
Aperiodic-Template	0.9766	0.407091
Aperiodic-Template	1	0.23276
Aperiodic-Template	0.9844	0.637119
Aperiodic-Template	0.9844	0.957319
Aperiodic-Template	0.9844	0.468595
Aperiodic-Template	1	0.788728
Aperiodic-Template	0.9688	0.77276
Aperiodic-Template	1	0.186566
Aperiodic-Template	0.9922	0.739918
Aperiodic-Template	0.9844	0.602458
Aperiodic-Template	0.9844	0.060239
Aperiodic-Template	0.9844	0.213309
Aperiodic-Template	0.9922	0.819544
Aperiodic-Template	1	0.100508
Aperiodic-Template	0.9922	0.452799
Aperiodic-Template	0.9844	0.834308
Aperiodic-Template	0.9766	0.299251
Aperiodic-Template	0.9922	0.922036
Aperiodic-Template	0.9609	0.991468
Aperiodic-Template	0.9844	0.848588
Aperiodic-Template	0.9922	0.275709
Aperiodic-Template	0.9922	0.134686
Aperiodic-Template	0.9922	0.275709
Aperiodic-Template	0.9844	0.585209

Aperiodic-Template	0.9844	0.888137
Aperiodic-Template	1	0.090936
Aperiodic-Template	0.9844	0.141256
Aperiodic-Template	1	0.046169
Aperiodic-Template	0.9922	0.689019
Aperiodic-Template	0.9922	0.337162
Aperiodic-Template	0.9766	0.875539
Aperiodic-Template	0.9844	0.253551
Aperiodic-Template	0.9766	0.020085
Aperiodic-Template	0.9609	0.534146
Aperiodic-Template	0.9844	0.186566
Aperiodic-Template	1	0.275709
Aperiodic-Template	0.9922	0.095617
Aperiodic-Template	0.9922	0.484646
Aperiodic-Template	0.9922	0.551026
Aperiodic-Template	0.9922	0.155209
Aperiodic-Template	0.9922	0.392456
Aperiodic-Template	0.9922	0.204076
Aperiodic-Template	0.9609	0.788728
Aperiodic-Template	0.9766	0.452799
Aperiodic-Template	0.9844	0.985035
Aperiodic-Template	1	0.407091
Aperiodic-Template	0.9844	0.299251
Aperiodic-Template	1	0.834308
Aperiodic-Template	0.9766	0.804337
Aperiodic-Template	1	0.264458
Aperiodic-Template	0.9922	0.602458
Aperiodic-Template	1	0.186566
Aperiodic-Template	0.9922	0.23276
Aperiodic-Template	0.9766	0.264458
Aperiodic-Template	0.9922	0.706149
Aperiodic-Template	0.9922	0.922036
Aperiodic-Template	0.9766	0.001232
Aperiodic-Template	0.9844	0.957319
Aperiodic-Template	0.9844	0.706149
Aperiodic-Template	0.9922	0.222869
Aperiodic-Template	0.9922	0.848588

Aperiodic-Template	0.9922	0.32418
Aperiodic-Template	0.9766	0.788728
Aperiodic-Template	0.9922	0.949602
Aperiodic-Template	1	0.585209
Aperiodic-Template	1	0.016911
Aperiodic-Template	0.9922	0.619772
Aperiodic-Template	0.9844	0.392456
Aperiodic-Template	0.9766	0.437274
Aperiodic-Template	0.9922	0.035174
Aperiodic-Template	0.9766	0.264458
Aperiodic-Template	0.9844	0.888137
Aperiodic-Template	1	0.048716
Aperiodic-Template	1	0.988549
Aperiodic-Template	0.9922	0.654467
Aperiodic-Template	0.9766	0.057146
Aperiodic-Template	0.9766	0.275709
Aperiodic-Template	0.9922	0.970538
Aperiodic-Template	0.9922	0.834308
Aperiodic-Template	0.9766	0.739918
Aperiodic-Template	0.9844	0.602458
Aperiodic-Template	0.9922	0.551026
Aperiodic-Template	0.9922	0.178278
Aperiodic-Template	1	0.848588
Aperiodic-Template	0.9922	0.689019
Aperiodic-Template	0.9922	0.337162
Aperiodic-Template	0.9922	0.162606
Aperiodic-Template	0.9688	0.484646
Aperiodic-Template	0.9922	0.931952
Aperiodic-Template	0.9922	0.654467
Aperiodic-Template	1	0.021262
Aperiodic-Template	0.9922	0.170294
Aperiodic-Template	0.9844	0.970538
Aperiodic-Template	0.9844	0.911413
Aperiodic-Template	0.9844	0.862344
Aperiodic-Template	0.9922	0.275709
Aperiodic-Template	0.9844	0.186566
Aperiodic-Template	0.9922	0.422034

Aperiodic-Template	0.9922	0.422034
Aperiodic-Template	0.9922	0.090936
Aperiodic-Template	0.9844	0.155209
Aperiodic-Template	1	0.671779
Aperiodic-Template	0.9844	0.819544
Aperiodic-Template	0.9844	0.988549
Aperiodic-Template	0.9766	0.875539
Periodic-Template	0.9844	0.452799
Universal	0.9766	0.213309
Apen	0.9844	0.186566
Random-Excursion	0.9885	0.017156
Random-Excursion	0.9885	0.624107
Random-Excursion	1	0.381687
Random-Excursion	0.9640	0.180322
Random-Excursion	1	0.800471
Random-Excursion	0.9885	0.624107
Random-Excursion	1	0.0187
Random-Excursion	1	0.885045
Random-Excursion-V	1	0.496841
Random-Excursion-V	1	0.136304
Random-Excursion-V	1	0.101765
Random-Excursion-V	1	0.014424
Random-Excursion-V	1	0.676097
Random-Excursion-V	1	0.028577
Random-Excursion-V	1	0.5216
Random-Excursion-V	1	0.676097
Random-Excursion-V	1	0.001334
Random-Excursion-V	0.9885	0.5216
Random-Excursion-V	0.9885	0.752361
Random-Excursion-V	0.9655	0.075138
Random-Excursion-V	0.974	0.101765
Random-Excursion-V	0.9655	0.320988
Random-Excursion-V	0.974	0.865697
Random-Excursion-V	0.9655	0.845066
Random-Excursion-V	0.974	0.546791
Random-Excursion-V	0.9885	0.5216
Serial	0.9766	0.900104

Serial	0.9844	0.222869
Lempel-Ziv	1	0.5757564
Linear-Complexity	0.9922	0.848588

Comments: The data set of *Low Density Plaintext* didn't demonstrate deviation from randomness.

Table 10. Statistical Analysis Report on *High Density Plaintext* Data Category (a=0.001, 1000 data sequences)

Test Method	Proportion (>=0.96371)	P-valueT (0.0001)
Frequency	0.9844	0.980883
Block-Frequency	0.9844	0.095617
Cusum	0.9922	0.500934
Cusum	0.9922	0.739918
Runs	1	0.046169
Long-Run	0.9922	0.378138
Rank	0.9766	0.97606
FFT	0.9922	0.568055
Aperiodic-Template	1	0.204076
Aperiodic-Template	0.9844	0.654467
Aperiodic-Template	1	0.178278
Aperiodic-Template	0.9922	0.654467
Aperiodic-Template	1	0.407091
Aperiodic-Template	1	0.070445
Aperiodic-Template	1	0.043745
Aperiodic-Template	0.9844	0.155209
Aperiodic-Template	0.9922	0.468595
Aperiodic-Template	1	0.242986
Aperiodic-Template	1	0.922036
Aperiodic-Template	0.9844	0.602458
Aperiodic-Template	1	0.090936
Aperiodic-Template	1	0.122325
Aperiodic-Template	1	0.602458
Aperiodic-Template	0.9922	0.654467
Aperiodic-Template	0.9766	0.484646
Aperiodic-Template	1	0.788728
Aperiodic-Template	1	0.407091

Aperiodic-Template	0.9922	0.452799
Aperiodic-Template	0.9688	0.637119
Aperiodic-Template	0.9844	0.788728
Aperiodic-Template	0.9688	0.242986
Aperiodic-Template	0.9922	0.350485
Aperiodic-Template	0.9609	0.039244
Aperiodic-Template	0.9844	0.619772
Aperiodic-Template	1	0.242986
Aperiodic-Template	1	0.134686
Aperiodic-Template	0.9922	0.086458
Aperiodic-Template	0.9766	0.275709
Aperiodic-Template	0.9844	0.141256
Aperiodic-Template	0.9922	0.392456
Aperiodic-Template	0.9766	0.739918
Aperiodic-Template	0.9922	0.671779
Aperiodic-Template	1	0.949602
Aperiodic-Template	0.9844	0.039244
Aperiodic-Template	0.9922	0.311542
Aperiodic-Template	1	0.016911
Aperiodic-Template	0.9922	0.619772
Aperiodic-Template	0.9922	0.116519
Aperiodic-Template	0.9688	0.585209
Aperiodic-Template	0.9844	0.834308
Aperiodic-Template	0.9844	0.392456
Aperiodic-Template	0.9766	0.392456
Aperiodic-Template	0.9766	0.186566
Aperiodic-Template	0.9922	0.134686
Aperiodic-Template	0.9844	0.654467
Aperiodic-Template	0.9844	0.723129
Aperiodic-Template	1	0.128379
Aperiodic-Template	0.9922	0.195163
Aperiodic-Template	1	0.422034
Aperiodic-Template	1	0.637119
Aperiodic-Template	1	0.957319
Aperiodic-Template	0.9922	0.985035
Aperiodic-Template	1	0.788728
Aperiodic-Template	0.9766	0.148094

Aperiodic-Template	0.9922	0.195163
Aperiodic-Template	0.9844	0.654467
Aperiodic-Template	0.9922	0.788728
Aperiodic-Template	1	0.253551
Aperiodic-Template	0.9844	0.299251
Aperiodic-Template	0.9922	0.437274
Aperiodic-Template	0.9922	0.671779
Aperiodic-Template	0.9922	0.888137
Aperiodic-Template	0.9688	0.090936
Aperiodic-Template	1	0.003577
Aperiodic-Template	0.9922	0.264458
Aperiodic-Template	0.9844	0.178278
Aperiodic-Template	1	0.155209
Aperiodic-Template	0.9766	0.689019
Aperiodic-Template	0.9922	0.819544
Aperiodic-Template	0.9922	0.911413
Aperiodic-Template	0.9766	0.242986
Aperiodic-Template	1	0.033288
Aperiodic-Template	1	0.204076
Aperiodic-Template	1	0.070445
Aperiodic-Template	1	0.311542
Aperiodic-Template	0.9844	0.350485
Aperiodic-Template	1	0.043745
Aperiodic-Template	0.9844	0.364146
Aperiodic-Template	0.9766	0.046169
Aperiodic-Template	0.9922	0.77276
Aperiodic-Template	0.9766	0.222869
Aperiodic-Template	0.9844	0.378138
Aperiodic-Template	1	0.020085
Aperiodic-Template	0.9922	0.654467
Aperiodic-Template	0.9609	0.484646
Aperiodic-Template	1	0.337162
Aperiodic-Template	1	0.392456
Aperiodic-Template	0.9844	0.723129
Aperiodic-Template	0.9922	0.689019
Aperiodic-Template	0.9922	0.739918
Aperiodic-Template	0.9844	0.500934

Aperiodic-Template	0.9844	0.77276
Aperiodic-Template	1	0.834308
Aperiodic-Template	0.9844	0.128379
Aperiodic-Template	0.9922	0.287306
Aperiodic-Template	0.9922	0.116519
Aperiodic-Template	1	0.162606
Aperiodic-Template	1	0.337162
Aperiodic-Template	0.9922	0.517442
Aperiodic-Template	0.9922	0.213309
Aperiodic-Template	0.9922	0.568055
Aperiodic-Template	0.9922	0.637119
Aperiodic-Template	0.9922	0.941144
Aperiodic-Template	0.9922	0.074177
Aperiodic-Template	1	0.723129
Aperiodic-Template	0.9922	0.534146
Aperiodic-Template	0.9922	0.804337
Aperiodic-Template	1	0.054199
Aperiodic-Template	1	0.041438
Aperiodic-Template	0.9922	0.534146
Aperiodic-Template	0.9688	0.350485
Aperiodic-Template	1	0.23276
Aperiodic-Template	1	0.213309
Aperiodic-Template	0.9844	0.468595
Aperiodic-Template	1	0.911413
Aperiodic-Template	0.9922	0.134686
Aperiodic-Template	0.9844	0.739918
Aperiodic-Template	0.9922	0.957319
Aperiodic-Template	1	0.275709
Aperiodic-Template	1	0.337162
Aperiodic-Template	0.9922	0.931952
Aperiodic-Template	0.9922	0.014216
Aperiodic-Template	1	0.534146
Aperiodic-Template	1	0.162606
Aperiodic-Template	0.9922	0.378138
Aperiodic-Template	1	0.756476
Aperiodic-Template	0.9922	0.378138
Aperiodic-Template	0.9766	0.204076

Aperiodic-Template	0.9922	0.452799
Aperiodic-Template	0.9844	0.100508
Aperiodic-Template	0.9766	0.484646
Aperiodic-Template	0.9922	0.350485
Aperiodic-Template	0.9922	0.985035
Aperiodic-Template	0.9922	0.551026
Aperiodic-Template	0.9766	0.009998
Aperiodic-Template	0.9844	0.452799
Aperiodic-Template	0.9766	0.155209
Aperiodic-Template	0.9922	0.875539
Aperiodic-Template	1	0.484646
Aperiodic-Template	1	0.378138
Aperiodic-Template	0.9766	0.834308
Aperiodic-Template	0.9922	0.048716
Aperiodic-Template	0.9844	0.155209
Aperiodic-Template	0.9922	0.043745
Aperiodic-Template	0.9844	0.000199
Aperiodic-Template	1	0.037157
Periodic-Template	1	0.264458
Universal	0.9922	0.602458
Apen	1	0.900104
Random-Excursion	0.9651	0.484646
Random-Excursion	0.9884	0.739918
Random-Excursion	1	0.052778
Random-Excursion	0.9884	0.788728
Random-Excursion	0.9651	0.585209
Random-Excursion	1	0.484646
Random-Excursion	1	0.437274
Random-Excursion	0.9884	0.637119
Random-Excursion-V	0.9767	0.927083
Random-Excursion-V	0.9884	0.559523
Random-Excursion-V	0.9884	0.041438
Random-Excursion-V	0.9884	0.534146
Random-Excursion-V	0.9884	0.585209
Random-Excursion-V	0.9884	0.811993
Random-Excursion-V	0.9884	0.991468
Random-Excursion-V	1	0.392456

Random-Excursion-V	1	0.371101
Random-Excursion-V	1	0.227773
Random-Excursion-V	0.9884	0.090936
Random-Excursion-V	0.9884	0.105618
Random-Excursion-V	0.9767	0.953553
Random-Excursion-V	0.9651	0.534146
Random-Excursion-V	0.9767	0.275709
Random-Excursion-V	0.9767	0.585209
Random-Excursion-V	0.9767	0.242986
Random-Excursion-V	0.9651	0.052778
Serial	1	0.311542
Serial	0.9844	0.364146
Lempel-Ziv	1	0.78587
Linear-Complexity	0.9922	0.77276

Comments: The data set of *High Density Plaintext* didn't demonstrate deviation from randomness.