Information & Security

Published since 1998

I. Blagoev
vol. 47, no. 1 (2020): 62-76
https://doi.org/10.11610/isij.4704

ISSN 0861-5160 (print), ISSN 1314-2119 (online)

Research Article

Neglected Cybersecurity Risks in the Public Internet Hosting Service Providers

Ivan Blagoev (✉)

Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Sofia, Bulgaria, http://iict.bas.bg/en

ABSTRACT:

The provision of cybersecurity is of basic importance for every effective information system. It is possible to provide most rich information services, but only one neglected cybersecurity risk may compromise the system and all services it provides. Therefore, meeting the cybersecurity requirements is a prerequisite for the safety and security of IT infrastructures, digital resources, and the protection of private data. In that respect, the themes of cryptography and sufficiently robust random number generation are of particular interest. This article looks for the "golden ratio" between the provision of mass services and the efforts to meet cybersecurity requirements. It suggests a method and discusses the possibilities to increase the cryptographic protection in information systems.

ARTICLE INFO:

RECEIVED: 03 JUN 2020
REVISED: 31 AUG 2020
ONLINE: 21 SEP 2020

KEYWORDS:

cybersecurity, cryptography, RNG, digitization, cryptography protocols, web services

At the heart of any encryption system is an algorithm and a random number generator. Therefore, it can be considered that no matter how complex cryptographic algorithms we are using, they are as strong as strong is the random number generator, which is the basis of the system

In the cryptography systems have a two basis types of random numbers generators:

- Random Number Generator (RNG): Apply at any given time to the RNG to generate values that must be unique and should not be repeated on

✉ E-mail: i.blagoev@iit.bas.bg

subsequent calls to the RNG.[1] The numbers obtained with this type of RNG are applied to operations that require unique / unobservable numerical values generated over time. An example of such a situation is generating a cryptographic key for encoding / decoding data, initializing vectors, initial numerical values for controlled RNG, etc.

- Pseudo Random Number Generator (PRNG): this generator uses the initial SEED number.[2] All randomly generated consecutive numbers come from the algorithm and this initial SEED value. All produced values are in order of their sequence and it's are re-reproducible if initial SEED is the same. The only really unexpected and secret value that should be as unpredictable as possible is the SEED number, which is the root of the base of this numerical sequence and the basis for generating the entire numerical array. This technology is most commonly used for often for One Time Password (OTP) authentication and the generation of cryptographic keys derived from the Master Root Key, which is used to compile wallets in Block Chain - distributed ledger technology, HMAC authentication, etc.

But how predictable and vulnerable is a given random number generator?

This is a question that has excited the community since time immemorial. Even if we use a good mix of cryptographic algorithms, which is designed to protect the operation of an information system. If our RNG is not good, the whole cryptographic solution may be vulnerable.[3] The encryption algorithms themselves are reproducible, and the simplest way is for the criminal person to try predicted values from random numbers coming from our RNG. He can make a RNG values are fed as input to the cryptographic algorithms and compare the output to the intercepted encrypted stream. If this attacker can predict correctly the RNG value, the security of the entire cryptographic solution will fail.[5] Conventional brute force attacks over cryptographic keys requires a lot of time and calculation resources. Therefore, brute force attacks directed against reliable modern cryptographic algorithms are considered ineffective. Entropy and predictability detection attacks in a bad random number generator require much less time and resources to make these attacks possible. We, as a victim, can feel deceptively secure because we believe that the system relies on a good combination of modern cryptographic algorithms without suspecting our RNG weakness.[3]

The effectiveness of RNG is measured by the degree of entropy for the generation of random numbers. For example, we take a binary bit and it can have a value of 0 or 1. If we have no idea what the value is, we have entropy 1 bit (i.e. coin throwing with 2 possible outcomes). If the generated value is always 1 and we know this, we have entropy 0 bits. The predictability is opposed to unpredictability. If the binary bit 1 is falling in 99 % of all cases, entropy may only be a fraction over 0 bits. In the area of cryptography, the more unpredictable bits we would obtain so much the better.

In other areas, such as statistics, signal processing, econometrics, and mathematical finance, to find cycle times and predict future values, time series are used.[5] Since a time series is a sequence of data points typically measured at successive time points located at unified time intervals, it is also possible to apply this approach to the quality analysis of a random number generation system. Regardless of which random number generator is more often used (uncontrolled or controlled), the overall success of the system is based on the quality of the random numbers produced.

The complexity of analysing a given RNG is a function of the quality of its entropy, such as seasonality and collision tendencies, or the creation of repetitive patterns. These are the moments when the RNG will generate a value that is cyclical or a range of values that lead to a repeat of an already output result or the generation of a new but expected value.

In the end, the normal development of humanity leads us to more and more mass digitalization. More and more activities and processes are much more productive and effectively managed through technology. These processes of digitalization even accelerated and proved their importance when the world was hit by the global COVID-19 pandemic. Processes that would take years had to happen in months and society had look for a new way of life that is much more related to technology. At first glance looks like the world was prepared for such a technological challenge. To some extent, this is the case, but the number of Cybercrime has increased and the encroachment of personal data, money and information loss, extortion due to information loss has also escalated to unprecedented levels. All of this is a strong indicator that while technology and computing infrastructure have met the challenge, from the side of cybersecurity we are not ready. The aim of this study is to focus on cybersecurity issues in public services, which are easily accessible and account for a large share of mass consumption.

For the needs of our current research we used web hosting services of one established provider of this sector. Our web application service has been installed over the rented hosting. The Web certificate has been added and SSL access has been activated on standard port 443. The control over the service does not allow to change the cryptography ciphers configuration from which is depends all encryption which is connected with connectivity protocol TLS.[6] But let's check active TLS versions and cryptography ciphers which is configured from our service provider:

1. Protocol version: TLSv1.0:

- cryptography algorithm ciphers:

  TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, (*) – A->B (till February 2020)
  TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (*) – A security class → B security class (till February 2020)

- compression: not supported

2. Protocol version: TLSv1.1:

- cryptography algorithm ciphers:

  TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (*) – A security class → B security class (till February 2020)

TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (*) – A security class → B security class (till February 2020)

- compression: not supported

3. Protocol version: TLSv1.2:

- cryptography algorithm ciphers:

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (*) – A security class
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (*) – A security class
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (*) – A security class
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (*) – A security class
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (*) – A security class
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (*) - A security class

- compression: not supported

Elliptic curves for Diffie Hellman maintained by the server in a preferential order - secp256r1, secp521r1, brainpoolP512r1, brainpoolP384r1, secp384r1, brainpoolP256r1, secp256k1, sect571r1, sect571k1, sect409k1, sect409r1, sect283k1, sect283r1

From what has been obtained so far, it can be said that the cryptographic protocols that the server offers: TLSv1.0 and TLSv1.1 should not be supported and offered at all, because they have long-known weaknesses and are obsolete. From the point of view of cyber security, the ability to establish a connection between a client server through them is a significant weakness.

The protocol: TLSv1.2 is still in use, but in cryptographic algorithms order have weak and reliable algorithms.[6] For establish tunnelling connectivity, the algorithm list provided by this protocol should be reduced. Therefore, the list of offered algorithms ciphers should be reduced to the following type, which is currently still up to date:

- TLSv1.2 and identifiers of supported current cryptographic ciphers algorithms:
  TLS_ECDHE_RSA(ECDSA)_WITH_AES_256_GCM_SHA384 (secp521r1, secp384r1) – A security class
  TLS_ECDHE_RSA(ECDSA)_WITH_AES_128_GCM_SHA256 (secp521r1, secp384r1) – A security class
  TLS_ECDHE_RSA(ECDSA)_WITH_AES_256_CBC_SHA384 (secp521r1, secp384r1) – A security class
  TLS_ECDHE_RSA(ECDSA)_WITH_AES_128_CBC_SHA256 (secp521r1, secp384r1) – A security class
- for asymmetric cryptography with RSA, the key must not be smaller than RSA4096 (ECDSA384+)
- compression: not supported (enabling compression also opens up vulnerabilities in cryptography)

Another significant drawback is that TLSv1.3 is not supported, this is the most secure up-to-date protocol for the SSL tunnelling connectivity at the moment. In this security communication protocol, all vulnerable cryptographic ciphers and compromised cryptographic algorithms is not available. Also have some very fundamentally changes in the way of establishing connectivity, which significantly increase the cyber protection of clients and the server. Furthermore, when users using client certificate authentication, the private information about users is hidden from the network spies. In previous versions of the TLS protocol was possible for the eavesdropper to collect information about the users which authenticated self with certificates front the server.

As already mentioned, all cryptography protection is strong how strong is a random number generator in it base. But let's check what RNG entropy which provides us this service to covering all cryptography protections considered so far. On the first our service is shared hosting. The here specific is that platforms of this type shares all hardware resource between large number of users and their web service applications. We do not know if this shared hardware has a true RNG. But if the True RNG exists on the server and a lot of users trying to use this shared RNG through their web applications at the same time, then the RNG entropy also go to collapse.[4] Then all RNG entropy will be compromised but it will stay hidden about the clients of shared hosting services, which will continue to work as if everything is normal. This will be a perfect moment for the any hacker attacks over cryptography protection. They are also can be used True RNG's that are very powerful and is designed to be a very fast and with rich entropy but it's a extra HWRNG security modules and not exists in the standard server configuration.[2] But let's actually check what is of our service status:

- IP address;
- TCP ports;
- Web service;
- cPanel service;
- DNS administration panel.

In order to protect when transmitting data over HTTP, the channel between the end client (usually the web browser) and the server is secured by a TLS tunnel, which at the packet level envelops the data transmitted in its clear format. The TLS protocol consists of 2 phases – handshakes and establishing a channel with a session key. The level of information security is determined by the agreed set of cryptographic algorithms, key lengths and generated random numbers, which are exchanged between the server and the client in the handshake phase (Diagram 1):

The unpredictability in obtaining random numbers from the system significantly affects the initial results of cryptographic operations. The TLS session key is formed after transformations with random numbers generated by the server and the client. Because the client does not always have a reliable method for obtaining real random numbers, then this task is implicitly transferred from the server side. To study the degree of unpredictability in the generation of random numbers by the server responsible for the site. On the follows shows Server Hello handshake and server random number:

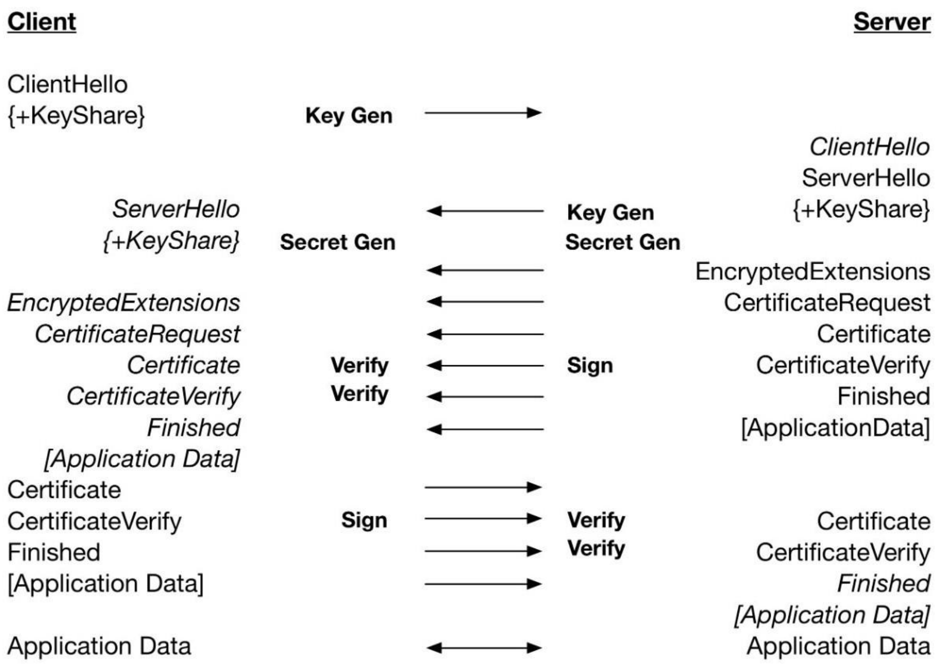

Diagram 1: TLSv1.2 handshake diagram.

Version: 3.3 (TLS/1.2)
SessionID: 23 6C F0 EA 52 FA 7A E9 40 35 AA 23 17 55 1E 24 6C 9D C8 81 59 F5 CF 92 30 D2 11 1D 12 F9 2A 33
Random: 95 C6 63 18 AA 32 44 47 28 00 4B 94 2D AA F9 3B 12 9D 69 54 4B 45 1A B1 1E CA 4D DE B0 A3 86 5F
Cipher: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 [0xC02F]
CompressionSuite: NO_COMPRESSION [0x00]
Extensions:
server_name empty
renegotiation_info 00
ec_point_formats uncompressed [0x0], ansiX962_compressed_prime [0x1], ansiX962_compressed_char2 [0x2]
ALPN http/1.1

For this purpose, it was necessary to extract from the server an array of random numbers (which is showed above), which comes from a source responsible for the operation of cryptography. The ways to do that are the following:

1. writing a program code to be installed on the provided web space and executed by the web client. The received random numbers can be saved in a file or sent directly via stream to the client.

2. Even if you have no control over the hosting, it can be done as client when making web requests to the hosting service. In our case using a Python computer program that establishes TLS connections like a regular web client. In the Server Hello phase and the TLS handshake phase, data from the random number generator is retrieved. The received data is saved in a file. When this operation is repeated in the loop, the larger random number generator data will be collected. However, a certain delay must be provided here so as not to overload the server with an excessive number of connections and to interfere with normal operation. Therefore, this approach for the same amount of data would take longer.

Diagram 2 visually describes this process.

Following the above methods is collected a binary array that is written in a file in the form of data. These random data were subjected to high-intensity computational analysis using specialized open source software called Dieharder and authored by Robert G. Brown of the Duke University Physics Department.[7] The results of the analysis are in Appendix 1.

The following is a description of the mathematical analysis applied to determine the quality of the random numbers array:

Birthday spacings: Choose m birthdays in a year of n days. List the spacings between the birthdays. If j is the number of values that occur more than once in that list, then j is asymptotically Poisson distributed with mean m3 ÷ (4n). Experience shows n must be quite large, say n ≥ 218, for comparing the results to the Poisson distribution with that mean. This test uses n = 224 and m = 29, so that the underlying distribution for j is taken to be Poisson with λ = 227 ÷ 226 = 2. A sample of 500 js is taken, and a chi-square goodness of fit test provides a p value. The first test uses bits 1–24 (counting from the left) from integers in the specified file. Then the file is closed and reopened. Next, bits 2–25 are used to provide birthdays, then 3–26 and so on to bits 9–32. Each set of bits provides a p-value, and the nine p-values provide a sample for a KSTEST;[7]

The overlapping 5-permutation test: This is the OPERM5 test. It looks at a sequence of one million 32-bit random integers. Each set of five consecutive integers can be in one of 120 states, for the 5! possible orderings of five numbers. Thus the 5th, 6th, 7th, ... numbers each provide a state. As many thousands of state transitions are observed, cumulative counts are made of the number of occurrences of each state. Then the quadratic form in the weak inverse of the 120×120 covariance matrix yields a test equivalent to the likelihood ratio test that the 120 cell counts came from the specified (asymptotically) normal distribution with the specified 120×120 covariance matrix (with rank 99). This version uses 1000000 integers, twice. This test may have unresolved bugs resulting in consistently poor p-values;[7]

Ranks of matrices: This is the binary rank test for 32x32 matrices. A random 32x32 binary matrix is formed, each row a 32-bit random integer. The rank is determined. That rank can be from 0 to 32, ranks less than 29 are rare, and their counts are pooled with those for rank 29. Ranks are found for 40,000 such

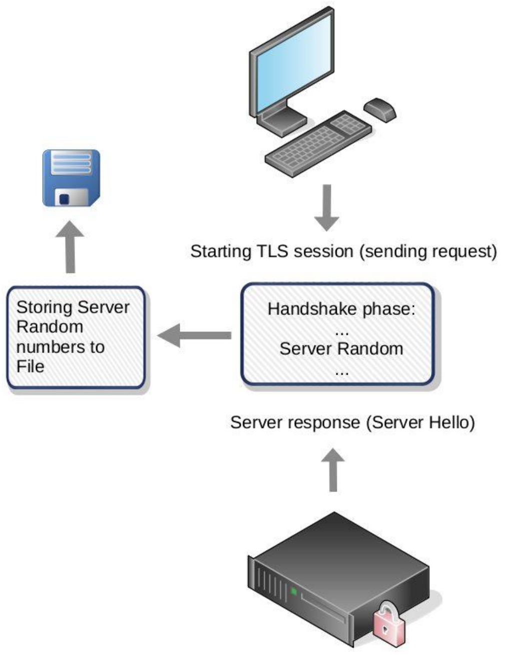
Starting TLS session (sending request)
Storing Server Random numbers to File
Handshake phase: ... Server Random ...
Server response (Server Hello)

Diagram 2. Random number generation.

random matrices and a chi-square test is performed on counts for ranks 32, 31, 30 and <=29.

As always, the test is repeated and a KS test applied to the resulting p-values to verify that they are approximately uniform;[7]

Monkey tests: Treat sequences of some number of bits as "words". Count the overlapping words in a stream. The number of "words" that do not appear should follow a known distribution. The name is based on the infinite monkey theorem;[7]

Count the 1s: Count the 1 bits in each of either successive or chosen bytes. Convert the counts to "letters", and count the occurrences of five-letter "words;[7]

Minimum distance test: Randomly place 8000 points in a 10000×10000 square, then find the minimum distance between the pairs. The square of this distance should be exponentially distributed with a certain mean;[7]

Random spheres test: Randomly choose 4000 points in a cube of edge 1000. Center a sphere on each point, whose radius is the minimum distance to another point. The smallest sphere's volume should be exponentially distributed with a certain mean;[7]

The squeeze test: Multiply 2^{31} by random floats on (0,1) until you reach 1. Repeat this 100000 times. The number of floats needed to reach 1 should follow a certain distribution;[7]

Overlapping sums test: Generate a long sequence of random floats on (0,1). Add sequences of 100 consecutive floats. The sums should be normally distributed with characteristic mean and variance;[7]

Runs test: Generate a long sequence of random floats on (0,1). Count ascending and descending runs. The counts should follow a certain distribution;[7]

The craps test: Play 200000 games of craps, counting the wins and the number of throws per game. Each count should follow a certain distribution;[7]

In the above (and attached description) final result, the simulation of 114 tests and various cryptographic operations. The quality of the values applied in the cryptography by the hosting server is below the levels of reliable cryptographic and Cyber security of FIPS-140 and other world laboratories:

Summary of the random numbers simulation test data:

- Only 25 have passed successfully;
- Failed, which have compromised/predictable value and therefore detectable cryptography 76;
- Vulnerable, where cryptography can be revealed with relatively good computer hardware are 13;

In the case of a breaking of cryptographic security, different types of hacker attacks of can be successfully implemented. Some of the more well-known of this type are, such as substituting content to mislead a user to mislead the user into certain actions, fake news, leaking personal data, server resources encroachment and etc.[4]

Technologically, public hosting services have major fundamental shortcomings. But they are extremely affordable, easy to configure, and very cheap. Not should be neglected the benefits of them. However, from the analysis of the research results, it can be concluded that it is not desirable to use them for critical systems that's need from reliable cyber security protection. Web services and portals which is critical and processing critical data should be hosted on a private hosting or virtual private server (VPS).

In this case we can control available server resources and the corresponding level of cyber protection can be configured. In addition, the same server resources will not be shared with other Web applications that can consume hardware resources and the entropy of random numbers. This can lead to the loss of critical resources and major vulnerabilities in the cyber security to be open.

Conclusions

The research of the represented services and their level of cybersecurity and also growing technological needs for more digitalization in all social and economic activities, it can be concluded that the existing technological infrastructure can meet today's challenges of more and more computer power. In terms of Cyber security, current IT services are still lagging behind. Increasing the success rate of Cybercrime would lead to a loss of consumer confidence and a halt to scientific and technological progress. This will lead to a significant slowdown and even hibernation in the development of many other areas such as economy, security, technology and others. Therefore, it can be said that it is absolutely necessary to make efforts to increasing the Cyber resilience in all directions. Improving the quality of Cyber protection of all technological activities and services, as well as increasing the level of Cyber hygiene of users.

Acknowledgment

This paper is supported by the National Research Program ICT in Science, Education and Security and the budget project of the IICT in digital transformation.

Appendix 1. Neglected Cybersecurity risks in the public Internet hosting service providers

Dieharder RNG analysis results:

```
rng_name    |           filename             |rands/second|
 file_input_raw|                  sh-random.bin|  6.02e+07  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
# The file file_input_raw was rewound 52 times
   diehard_birthdays|   0|       100|     100|0.38951753|  PASSED
# The file file_input_raw was rewound 434 times
      diehard_operm5|   0|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 922 times
  diehard_rank_32x32|   0|     40000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 1151 times
    diehard_rank_6x8|   0|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 1251 times
   diehard_bitstream|   0|   2097152|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 2051 times
        diehard_opso|   0|   2097152|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 2584 times
        diehard_oqso|   0|   2097152|     100|0.00000000|  FAILED
```

```
# The file file_input_raw was rewound 2834 times
        diehard_dna|   0|   2097152|     100|0.00020081|   WEAK
# The file file_input_raw was rewound 2859 times
diehard_count_1s_str|   0|    256000|     100|0.01953386|  PASSED
# The file file_input_raw was rewound 3347 times
diehard_count_1s_byt|   0|    256000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 3356 times
 diehard_parking_lot|   0|     12000|     100|0.33358085|  PASSED
# The file file_input_raw was rewound 3362 times
    diehard_2dsphere|   2|      8000|     100|0.03334174|  PASSED
# The file file_input_raw was rewound 3367 times
    diehard_3dsphere|   3|      4000|     100|0.92330162|  PASSED
# The file file_input_raw was rewound 4246 times
     diehard_squeeze|   0|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 4246 times
        diehard_sums|   0|       100|     100|0.14880589|  PASSED
# The file file_input_raw was rewound 4284 times
        diehard_runs|   0|    100000|     100|0.00000004|  FAILED
        diehard_runs|   0|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 4797 times
       diehard_craps|   0|    200000|     100|0.00000000|  FAILED
       diehard_craps|   0|    200000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 12427 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00000000|  FAILED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 12465 times
         sts_monobit|   1|    100000|     100|0.00000222|   WEAK
# The file file_input_raw was rewound 12503 times
            sts_runs|   2|    100000|     100|0.03544040|  PASSED
# The file file_input_raw was rewound 12541 times
          sts_serial|   1|    100000|     100|0.00000278|   WEAK
          sts_serial|   2|    100000|     100|0.00000062|  FAILED
          sts_serial|   3|    100000|     100|0.00198630|   WEAK
          sts_serial|   3|    100000|     100|0.24256527|  PASSED
          sts_serial|   4|    100000|     100|0.00001279|   WEAK
          sts_serial|   4|    100000|     100|0.00000000|  FAILED
          sts_serial|   5|    100000|     100|0.14830043|  PASSED
          sts_serial|   5|    100000|     100|0.00006248|   WEAK
          sts_serial|   6|    100000|     100|0.01594394|  PASSED
          sts_serial|   6|    100000|     100|0.58120558|  PASSED
          sts_serial|   7|    100000|     100|0.00001243|   WEAK
          sts_serial|   7|    100000|     100|0.00650289|  PASSED
          sts_serial|   8|    100000|     100|0.00000000|  FAILED
          sts_serial|   8|    100000|     100|0.00000000|  FAILED
          sts_serial|   9|    100000|     100|0.00000000|  FAILED
          sts_serial|   9|    100000|     100|0.14200950|  PASSED
          sts_serial|  10|    100000|     100|0.00000000|  FAILED
          sts_serial|  10|    100000|     100|0.00003391|   WEAK
          sts_serial|  11|    100000|     100|0.29281609|  PASSED
          sts_serial|  11|    100000|     100|0.00000000|  FAILED
          sts_serial|  12|    100000|     100|0.10890305|  PASSED
          sts_serial|  12|    100000|     100|0.04145417|  PASSED
          sts_serial|  13|    100000|     100|0.00000000|  FAILED
          sts_serial|  13|    100000|     100|0.00000000|  FAILED
          sts_serial|  14|    100000|     100|0.00000037|  FAILED
          sts_serial|  14|    100000|     100|0.51404682|  PASSED
          sts_serial|  15|    100000|     100|0.32460847|  PASSED
          sts_serial|  15|    100000|     100|0.00000000|  FAILED
          sts_serial|  16|    100000|     100|0.00651735|  PASSED
          sts_serial|  16|    100000|     100|0.00115580|   WEAK
# The file file_input_raw was rewound 12618 times
         rgb_bitdist|   1|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 12770 times
         rgb_bitdist|   2|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 12999 times
         rgb_bitdist|   3|    100000|     100|0.03240048|  PASSED
# The file file_input_raw was rewound 13304 times
         rgb_bitdist|   4|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 13686 times
         rgb_bitdist|   5|    100000|     100|0.88959066|  PASSED
# The file file_input_raw was rewound 14143 times
         rgb_bitdist|   6|    100000|     100|0.00000006|  FAILED
# The file file_input_raw was rewound 14677 times
         rgb_bitdist|   7|    100000|     100|0.07126523|  PASSED
# The file file_input_raw was rewound 15288 times
         rgb_bitdist|   8|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 15974 times
         rgb_bitdist|   9|    100000|     100|0.32917367|  PASSED
# The file file_input_raw was rewound 16737 times
         rgb_bitdist|  10|    100000|     100|0.00050227|   WEAK
# The file file_input_raw was rewound 17577 times
         rgb_bitdist|  11|    100000|     100|0.15629093|  PASSED
# The file file_input_raw was rewound 18492 times
         rgb_bitdist|  12|    100000|     100|0.00001785|   WEAK
# The file file_input_raw was rewound 18568 times
rgb_minimum_distance|   2|     10000|    1000|0.00000012|  FAILED
# The file file_input_raw was rewound 18683 times
rgb_minimum_distance|   3|     10000|    1000|0.00000022|  FAILED
# The file file_input_raw was rewound 18836 times
rgb_minimum_distance|   4|     10000|    1000|0.00000000|  FAILED
```

```
# The file file_input_raw was rewound 19026 times
rgb_minimum_distance|   5|     10000|    1000|0.00206076|   WEAK
# The file file_input_raw was rewound 19103 times
   rgb_permutations|   2|    100000|     100|0.00012861|   WEAK
# The file file_input_raw was rewound 19217 times
   rgb_permutations|   3|    100000|     100|0.00000003|  FAILED
# The file file_input_raw was rewound 19370 times
   rgb_permutations|   4|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 19560 times
   rgb_permutations|   5|    100000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 19942 times
     rgb_lagged_sum|   0|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 20705 times
     rgb_lagged_sum|   1|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 21849 times
     rgb_lagged_sum|   2|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 23375 times
     rgb_lagged_sum|   3|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 25282 times
     rgb_lagged_sum|   4|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 27571 times
     rgb_lagged_sum|   5|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 30241 times
     rgb_lagged_sum|   6|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 33293 times
     rgb_lagged_sum|   7|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 36726 times
     rgb_lagged_sum|   8|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 40541 times
     rgb_lagged_sum|   9|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 44737 times
     rgb_lagged_sum|  10|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 49315 times
     rgb_lagged_sum|  11|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 54274 times
     rgb_lagged_sum|  12|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 59615 times
     rgb_lagged_sum|  13|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 65337 times
     rgb_lagged_sum|  14|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 71440 times
     rgb_lagged_sum|  15|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 77925 times
     rgb_lagged_sum|  16|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 84792 times
```

```
      rgb_lagged_sum|  17|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 92040 times
      rgb_lagged_sum|  18|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 99669 times
      rgb_lagged_sum|  19|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 107680 times
      rgb_lagged_sum|  20|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 116072 times
      rgb_lagged_sum|  21|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 124846 times
      rgb_lagged_sum|  22|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 134001 times
      rgb_lagged_sum|  23|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 143538 times
      rgb_lagged_sum|  24|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 153456 times
      rgb_lagged_sum|  25|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 163756 times
      rgb_lagged_sum|  26|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 174437 times
      rgb_lagged_sum|  27|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 185500 times
      rgb_lagged_sum|  28|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 196944 times
      rgb_lagged_sum|  29|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 208769 times
      rgb_lagged_sum|  30|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 220976 times
      rgb_lagged_sum|  31|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 233565 times
      rgb_lagged_sum|  32|   1000000|     100|0.00000000|  FAILED
# The file file_input_raw was rewound 233603 times
     rgb_kstest_test|   0|     10000|    1000|0.02580373|  PASSED
# The file file_input_raw was rewound 234189 times
     dab_bytedistrib|   0|  51200000|       1|0.00000000|  FAILED
# The file file_input_raw was rewound 234238 times
             dab_dct| 256|     50000|       1|0.00000000|  FAILED
Preparing to run test 207.  ntuple = 0
# The file file_input_raw was rewound 234669 times
        dab_filltree|  32|  15000000|       1|0.00000000|  FAILED
        dab_filltree|  32|  15000000|       1|0.00000000|  FAILED
Preparing to run test 208.  ntuple = 0
# The file file_input_raw was rewound 234781 times
       dab_filltree2|   0|   5000000|       1|0.00000000|  FAILED
       dab_filltree2|   1|   5000000|       1|0.00000000|  FAILED
```

```
Preparing to run test 209.  ntuple = 0
# The file file_input_raw was rewound 235029 times
      dab_monobit2|  12|  65000000|       1|1.00000000|  FAILED
```

References

1. Todor Balabanov, Iliyan Zankinski, and Bozhidar Shumanov, "Slot Machines RTP Optimization with Generic Algorithms," in: *Numerical Methods and Applications*, LNCS, vol. 9374 (Switzerland: Springer, 2015), 210-217.
2. Random Number Service, https://www.random.org.
3. Tatiana Atanasova and M. Barova, "Exploratory analysis of Time Series for hypothesizes feature values," *International Scientific Conference UniTech 2017*, vol. II (Gabrovo: V. Aprilov University Publishing House, 2017), 399-403.
4. Tasho Tashev and Vladimir Monov, "Large-Scale Simulation of Uniform Load Traffic for Modeling of Throughput on a Crossbar Switch Node," *8-th Int. Conf. "Large-Scale Scientific Computations,"* Sozopol, Bulgaria, 6-10 June 2011, LNCS, vol. 7116 (Springer, 2012), 630-637.
5. Pseudo-Random Number Generators, https://crypto.stanford.edu/pbc/notes/crypto/prng.html.
6. Benoit Badrignans, Jean Luc Danger, Viktor Fischer, Guy Gogniat, and Lionel Torres (eds.) *Security Trends for FPGAS - From Secured to Secure Reconfigurable Systems* (Netherlands: Springer, 2011).
7. Robert G. Brown's General Tools Page, https://phy.duke.edu/