HENKOS STREAM CIPHER

Marius Oliver Gheorghita
Regional Forecasting Center
Brestei 3A, 1100 Craiova, Romania
mariusg@meteo.craiova.pcnet.ro
Dominic Bucerzan
BB Computer
N. Balcescu 12, 2900 Arad, Romania
dominic@bbcomputer.ro

ABSTRACT

The purpose of this paper is to recommend to cryptographic community and information security specialists, for analysis and testing, a new cryptosystem based on a synchronous stream cipher and a keystream generator. The paper describes the main parts of the cryptosystem, its implementation and analysis of the statistical tests results for the keystream generator. Through its design and conception, HENKOS algorithm could be a new approach in the symmetrical encryption system evolution.

***Key words**: synchronous stream cipher, pseudorandom number generator (PRNG), master key (MK), data key (DK), HENKOS stream cipher, and statistical tests.*

INTRODUCTION

The purpose of this paper is to introduce the new cryptographic algorithm HENKOS and specially the new pseudorandom number generator on which the algorithm is based on, who appear to be secure and fast.

The algorithm was designed to satisfy these goals:

- Deducing the internal state from the result should be impossible
- There should be no short cycles
- It should be cryptographically secure
- It should be easy to implement
- The C/C++ code should be optimized for speed
- The ASM code should be optimized for speed

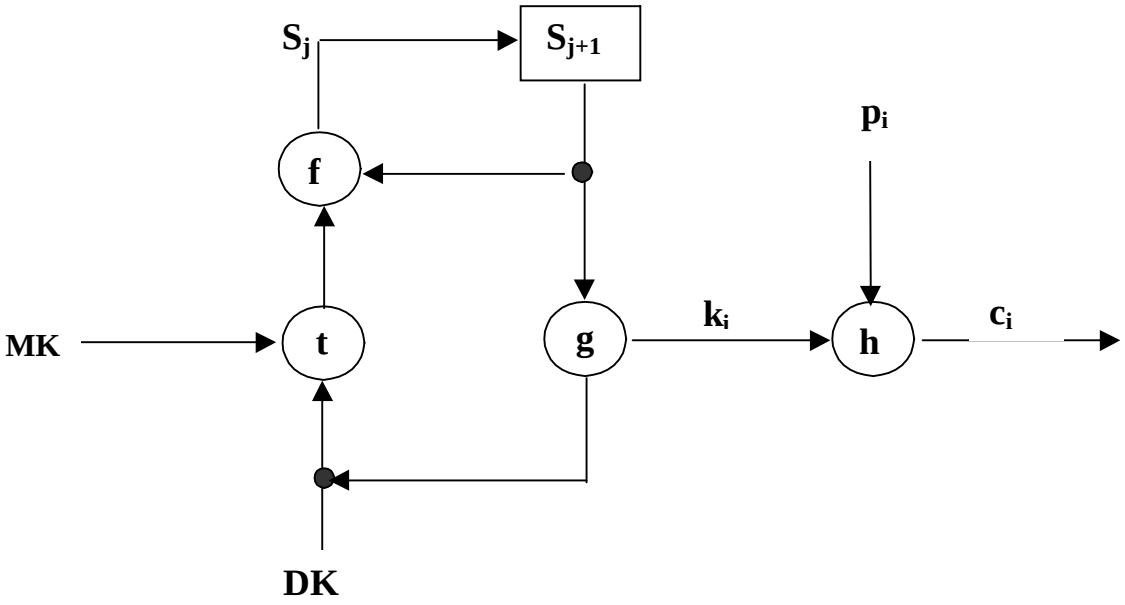
This cryptosystem is a symmetric stream cipher encryption system using two keys: a master key (MK) and a data key (DK); the master key is a secret unique key and the data key is a self generated key for each session; initially the sender and the receiver share the two keys on a trusted way. In each session the generator uses the master key and the last generated key for encryption to produce a new data key and the key-stream which is XOR-ed with the stream of plaintext.

CRYPTOSYSTEM DESCRIPTION

This cryptosystem uses a binary additive stream cipher and two types of keys:

- a short-term key named **data key** (DK) with a fixed length of 1024 bytes that is input in the keystream generator. This key can be generated with a PRNG (not necessarily a cryptographic secure PRNG) or can be an ordinary file, if is not available any PRNG. DK is used in the first communication session.
- a long-term key named **master key** (MK) with a fixed length, which contains 1024 numbers, used to mix the data key and the internal state of the keystream generator. This key must be generated with a true RNG (hardware) and shared between the parties involved in transmission only once using a secure channel.

If during the transmission an attacker intercepts the encrypted data, is not possible to decrypt correctly the ciphertext without having the master key, because there is a very large number of possible combinations of decrypted ciphertext.
Every attempt to find the master key produces a different plaintext, including the one with the same numbers but the changed order of the numbers in the key affects the decryption process.
So even if an attacker finds all the numbers from the master key, if they are not in the correct order he cannot decrypt correctly the ciphertext. If the sender encrypts and transmits the same message many times, because of the fact that every time it is used a different data key, the corresponding plaintext will be convert into a different ciphertext; in this case statistical analysis and pattern matching cannot be used to attack the cryptosystem.
The system seems to be a self synchronized stream cipher but we don't use for the next key ciphertext digits but the last key used to generate the stream.
The figure below describe the algorithm:



p_i – stream of plaintext
k_i – keystream
c_i – stream of ciphertext
S_j – internal state *j*
MKT – transformed MK

S_0 = **t** (MK, DK)
S_{j+1} = **f** (S_j, MKT)
k_i = **g** (S_i, S_{i-1})
c_i = **h** (p_i, k_i)

The algorithm can be divided in four parts: master key generation, data key generation, keystream generation, and encryption/decryption.
Notation: we will denote by MK_i the *i*-th element in the MK key.

Master key generation
In this section of the algorithm we transform the master key (MK) in the "transformed master key" (MKT) in two steps. We used two function: the first one is an additive function SUM and the second function INV produce a sort of simmetrical figures of number transformation.

$$MKSi = SUM (MKi) \qquad i = 0,1023 \qquad \text{where} \qquad (1)$$

$$SUM(MK_i) = \sum_{j=0}^{i} MK_j \qquad \text{modulo } 1024 \quad i = 0,1023$$

Finally the transformed master key (MKT) is

$$MKT_i = INV(\ SUM\ (MK_i)), \quad i = 0,1023 \qquad (2)$$

The transformation have two targets:

- Not to use the original MK key directly in the process
- To create confusion and diffusion for master key

Data key generation
In this section of the algorithm we transform the data key (DK) to obtain the real key (K) for encryption using two important functions: the first one is the essential function in this algorithm the "switch function" (SW) which will mix the bytes of the data key as follows:

- the byte *j* is switched with byte *k* in the data key, where *j* is the corespondent number from master key (MK) in the *i* position and *k* is the corespondent number from the transformed master key (MKT) in the *i* position.

$$\text{(SW):} \quad DK_j \leftrightarrow DK_k \quad \text{where } j = MK_i \text{ and } k = MKT_i \qquad \text{for } i = 0,1023 \qquad (3)$$

The next function is an additive function AD which will replace the value from each position with the sum between two near bytes.

$$\text{(AD):} \quad DK_i = DK_i + DK_{i+1} \quad \text{modulo } 256 \qquad i = 0,1023 \qquad (4)$$

These functions create a totally changed image of the data key.
After these two transformations we obtain DK1; these cycles will be repeated 64 times:

$$DK \rightarrow DK1 \rightarrow DK2 \rightarrow \ldots. \rightarrow DK63 \rightarrow DK64 \qquad (5)$$

Keystream generation
To obtain the keystream bytes (K_i) of real key (K), the last operation is the next one:

$$K_i = (DK64_i + DK64_{i+1}) \oplus DK64_i \qquad i = 0,1023;\ DK64_{1024} = DK64_0 \qquad (6)$$
$$\oplus = \text{"OR exclusive"}$$

Encryption / decryption process
The encryption function described bellow define the process:

$c_i = h(K_i, p_i)$ c_i = ciphertext, p_i = plain text, K_i = keystream

Because the algorithm is a symmetrical one the decryption process will use the same functions and the same parameters.
The encryption / decryption process will transform a stream of 1024 bytes of plain text. For the next stream of 1024 bytes of plain text we will use a new data key generated from DK64 by an iteration step from (5). This sequence will run until the plain text is finished for one session.

REMARKS

1. The data key for the next session we have generated with the same algorithm has excellent results (see the tests).
2. The master key was generated with the same algorithm starting with a chunk from a file The results of statistical tests for the file generated from this chunk were better than results for a file generated from a data key obtained from a weak PRNG (e.g. Linear Congruential Generator).

IMPLEMENTATION

The cryptosystem is implemented in the C/C++ language and ASM for Windows/32 bits. The software application and tests applied to the output file from the keystream generator were effectuated on a PC computer Pentium IV 2,4 GHz / 256 MB RAM.
For comparison were chosen another two PRNG: SHA-1 a very good PRNG fit for cryptographic purposes and Cubic Congruential Generator, an example of a bad PRNG, a completely predictable generator.
During the tests was monitoring generation sequence time for the 12.5 MB test file comparing all the time with other generators . The conclusion is that with an average time of 0,66 seconds for the C version and 0,33 seconds for the ASM version HENKOS PRNG is a very fast one. These time rates are accurate for a 1024 bytes key. For longer keys, the time for generating a sequence with the same length is much shorter.

The HENKOS PRNG has good results from analysis of the statistical tests (indicate that is a fit as a pseudo-random number generator for cryptographic purposes). Comparative with other known generators presents the advantage that is fast which it qualifies to be used as a keystream generator in the system.

STATISTICAL TEST ANALYSIS

The keystream generator is a new designed fast algorithm; it accepts large keys for seeding and passes the most important statistical tests. These tests FIPS 1401, FIPS 1402, ENT tests, Diehard battery of tests, NIST Statistical Test Suite (a statistical test suite for testing the pseudo-random number generators used in cryptographic applications) were effectuated on large ciphertext samples of 12.5 megabytes. Below are the results of the statistical tests:

1. FIPS 1401/FIPS1402 Test

FIPS statistical tests contain the Monobit Test, the Poker Test, the Runs Test, and the Long Run Test. The following tests are based on performing a pass/fail statistical test on 5000 sequences of 2500 bytes each produced by PRNG.
SHA-1, CCG and HCS pass FIPS 140-1 in proportion of 100%.
SHA-1 passes FIPS 140-2 in proportion of 99.6%, CCG in proportion of 99.4% and HENKOS in proportion of 99.64%.
For these statistical tests, even if the generators present good statistical properties this isn't a guarantee that is good for cryptographic purposes (see the results of DIEHARD tests).

2. ENT Tests

Applies various tests to sequences of bytes stored in files and reports the results of those tests. The program is useful for those evaluating pseudorandom number generators for encryption and statistical sampling applications, compression algorithms. It calculates entropy, optimum compression, chi-square distribution, arithmetic mean, Monte Carlo value for pi and serial correlation coefficient.

	SHA-1	**CCG**	**HENKOS**
Entropy	7.999988 bits per byte	7.997488 bits per byte	7.999987 bits per byte
Optimum compression would reduce the size of this 12500000 byte file by %	0	0	0
Chi square distribution for 12500000 samples is	201.61	46584.63	222.04
randomly would exceed this value % of the times	99.00	0.01	90.00
Arithmetic mean value of data bytes is	127.5123 (127.5 = random).	127.4921 (127.5 = random	127.5179 (127.5 = random).
Monte Carlo value for Pi is	3.141492983 (error 0.00 percent	3.141199828 (error 0.01 percent	3.141108983 (error 0.02 percent).
Serial correlation coefficient is	0.000406 (totally uncorrelated = 0.0)	0.000266 (totally uncorrelated = 0.0)	0.000124 (totally uncorrelated = 0.0)

3. DIEHARD Statistical Tests

The next set of tests was designed to identify weaknesses in many common non-cryptographic PRNG algorithms. These tests analyze a single large file from the output of the generator of 11 megabytes or more.
The battery of tests include: birthday spacings test, overlapping 5-permutation test, binary rank test 31x31, binary rank test 32x32, binary rank test 6x8, bitstream test, opso, oqso and DNA tests, count-the-1's test on a stream of bytes, count-the-1's test for specific bytes, parking lot test, minimum distance test, 3Dspheres test, squeeze test, overlapping sums test, runs test, craps test.

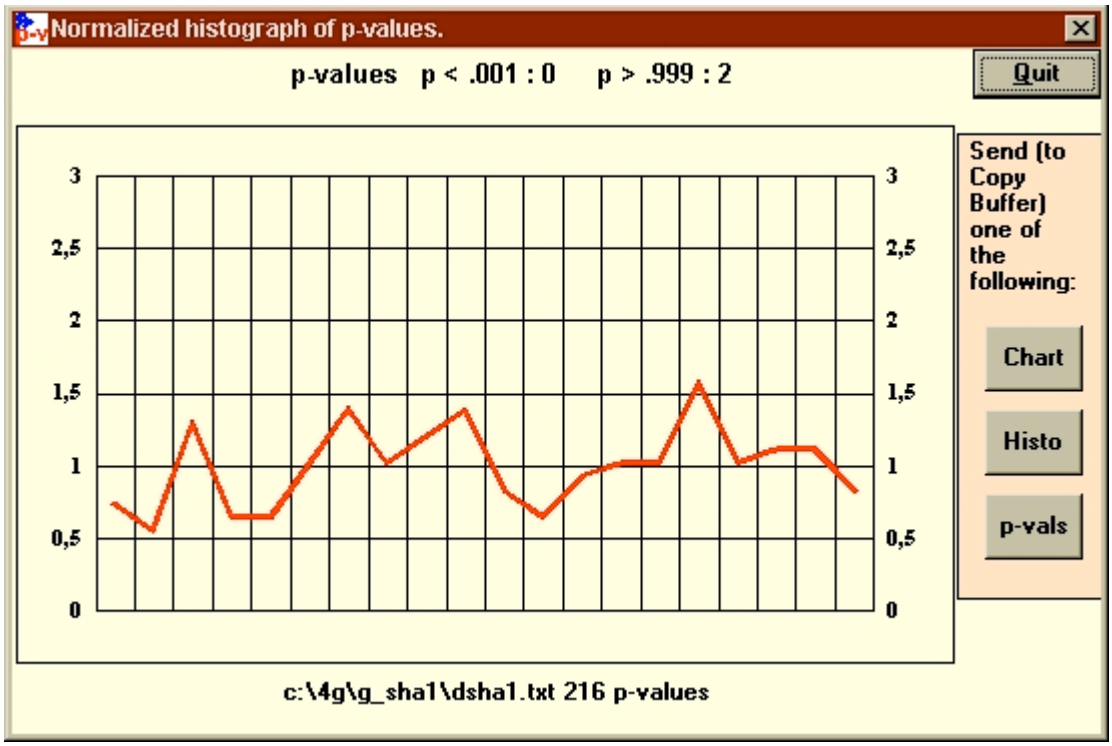


FIG. 1 Interpretation of the p-values for SHA-1 generator

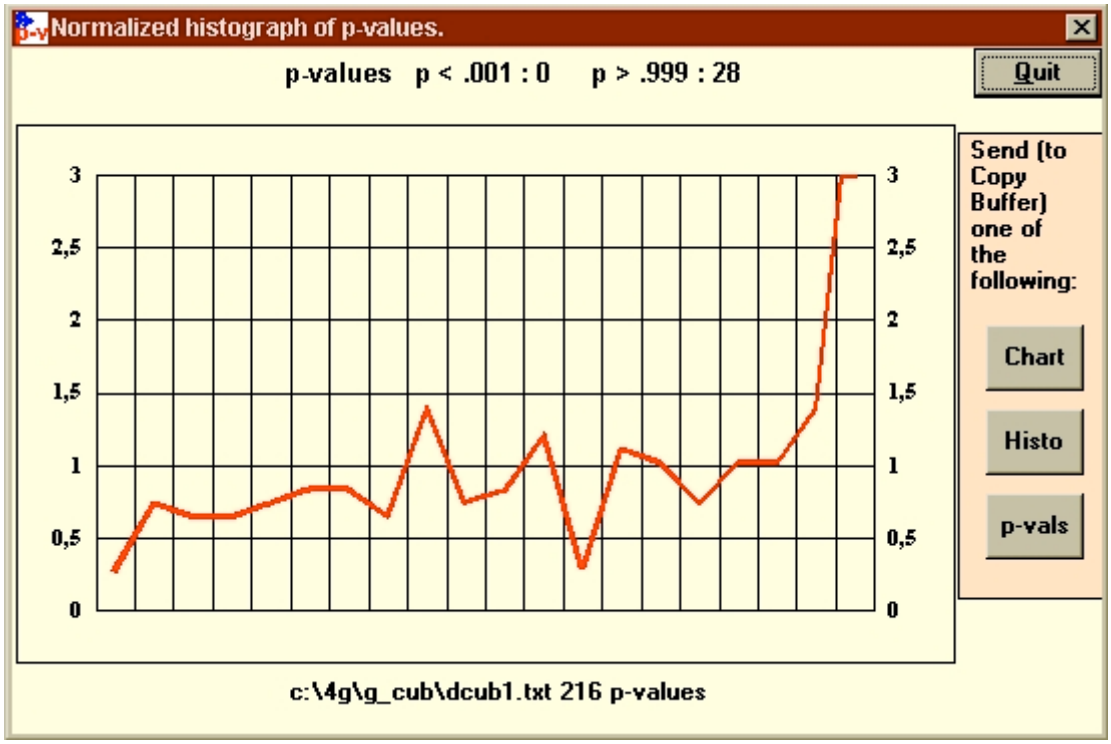


FIG. 2 Interpretation of the p-values for CC generator

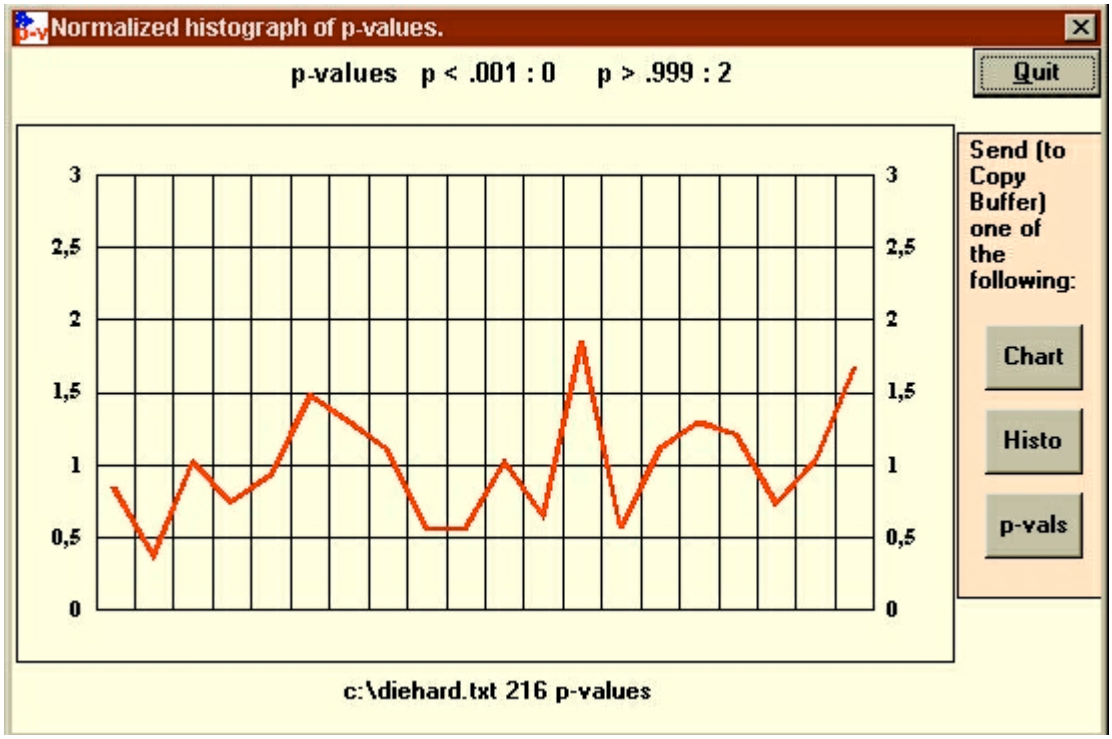


FIG. 3 Interpretation of the p-values for HENKOS generator

Majority of tests in DIEHARD return a p-value, which should be uniform on [0,1) if the input file contains truly independent random bits. Those p-values are obtained by p=F(X), where F is the assumed distribution of the sample random variable X. When a bit stream really fails, get p-values of 0 or 1 to six or more places.
For SHA-1 generator we have 2 p-value very close to 1, and for Cubic Congruential we have 28 p-value near to1. For HENKOS we have 2 p-values close to 0 or 1. The "ideal" p-values curve is one as flat as possible with hills or valleys. It is obviously that CC generator is a poor, predictable generator and HENKOS has a good curve and can be compared with SHA-1, which is known as a very good cryptographic PRNG.

We can measure the quality of the generators using the schema described in [8].
Each Diehard test produces one or more p-values (216 in total), which can be considered bad, suspect, or good:

- p-value > 0.998 is classified as bad and scores 4
- p-value between 0.95 and 0.998 is classified as suspect and and scores 2
- p-value < 0.95 is classified as good and scores 0

For each RNG we calculated the total score. The high scores indicate a poor RNG, and the low scores indicates a good RNG. The percentage is obtained raporting the total score to the total number of p-values – 216. The results for SHA-1, CCG and HENKOS (HCS) are given in the next table. For HENKOS we present 6 keystreams obtained like in remarks number 2 (self generated keys).

Keystream	Score	% versus 216 p-values
SHA-1	**31**	**0.1759**
CCG	**154**	**0.7129**
HCS-1	16	0.0740
HCS-2	10	0.0462
HCS-3	16	0.0740
HCS-4	24	0.1111
HCS-5	22	0.1018
HCS-6	30	0.1388
HCS average	**19.66**	**0.0910**

In conclusion the average score for HCS is better than the other scores (even the poor score for a HCS data is better than the other two scores).

4. NIST STATISTICAL TEST SUITE

The package includes statistical tests for: frequency, block frequency, cumulative sums, runs, long runs, *Marsaglia's* rank, spectral (based on the Discrete Fourier Transform), nonoverlapping template matchings, overlapping template matchings, *Maurer's* universal statistical, approximate entropy (based on the work of Pincus, Singer and Kalman), random excursions (due to *Baron* and *Rukhi*n), Lempel-Ziv complexity, linear complexity, and serial. In following table are the sequences who fail the tests from the suite (value under the minimum pass rate test).

The NIST framework, like many tests, is based on hypothesis testing.

1. State your null hypothesis. Assume that the binary sequence is random.
2. Compute a sequence test statistically. Testing is carried out at the bit level.
3. Compute the P-value. P-value $\in \in [0, 1]$.
4. Compare the P-value to α, where $\alpha = 0.01$. Success is declared whenever P-value $\geq \alpha$; otherwise, failure is declared.

TEST	SHA-1	CCG	3DES	BBS	MICALI	HCS
Frequency	Pass	Failure	Pass	Pass	Pass	Pass
Block frequency	Pass	Pass	Pass	Pass	Pass	Pass
Cusum	Pass	Failure	Pass	Pass	Pass	Pass
	Pass	Failure	Pass	Pass	Pass	Pass
Runs	Pass	Failure	Pass	Pass	Pass	Pass
Long-run	Pass	Pass	Pass	Pass	Pass	Pass
Rank	Pass	Pass	Pass	Pass	Pass	Pass
FFT	Pass	Failure	Pass	Pass	Pass	Pass
Aperiodic template	Pass	Failure 11	Failure 3	Pass	Failure 2	Pass
Periodic template	Pass	Pass	Pass	Pass	Pass	Pass
Universal	Pass	Pass	Pass	Pass	Pass	Pass
Apen	Pass	Failure	Pass	Pass	Pass	Pass
Random excursion	Pass	Pass	Pass	Pass	Pass	Pass
Random excursion –V	Failure 1	Pass	Failure 4	Pass	Pass	Pass
Serial	Pass	Failure1	Pass	Pass	Pass	Pass
	Pass	Pass	Pass	Pass	Pass	Pass
Lempel-Ziv	Pass	Failure	Pass	Pass	Pass	Pass
Linear-Complexity	Pass	Pass	Pass	Pass	Pass	Pass
Min. pass rate (random excursion–V)	.95239	.94598	.95353	.95146	.95406	.95178
Min.pass rate	.96015	.96015	.96015	.96015	.96015	.96015

SHA-1 generator has 1 fail at Random excursion –Variant test, the Cubic Congruential generator fails Frequency, Cumulative Sums, Runs, Aperiodic Template at 11 from 284 templates, Approximate Entropy test, serial and Lempel-Ziv test, 3DES has 3 fails at Aperiodic Template and 4 fails at Random excursion –Variant test and Micali generator has 3 fails at Aperiodic Template. HENKOS and BBS pass all tests.

SUMMARY

We developed a new cryptographic algorithm HENKOS based on a new pseudorandom number generator which is fast and pass with excellent results the tests we considered.
We consider this PRNG useful for simulation and cryptography and we wait from the cryptographic community to prove its security or find the weakness, helping us in the future development of this research.

BIBLIOGRAPHY

[1] A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography CRC Press, 1996. chapter 5,7,12,13 www.cacr.math.uwaterloo.ca/hac

[2] B. Schneier Applied Cryptography, Second Edition, J. Wiley & Sons Inc. 1996

[3] G. Marsaglia, Diehard Statistical Tests. http://stat.fsu.edu/~geo/

[4] J.Kelsey, B.Schneier, D.Wagner, C. Hall,"Cryptanalytic Attacks on Pseudorandom Number Generators" *Fast Software Encryption, Fifth International Workshop Proceedings (March 1998)*, Springer-Verlag, 1998,http://www.counterpane.com/pseudorandom_number.html

[5] Johnson B.C. Radix-b extensions to some common empirical tests for pseudorandom number generators. ACM Transactions on Modelling and Computer Simulations, S(4):261-273, 1996

[6] Mark M. Meysenburg,James A. Foster, "Random generators quality and GP performance" Proceedings of the genetic and evolutionary computation conference, vol. 2, pages 1121-1126, Orlando, Florida 13-17 July 1999

[7] Peter Gutmann, "Software Generation of Practically Strong Random Numbers" (updated June 2000), Original version presented at the 1998 Usenix Security Symposium. http://www.cryptoengines.com/~peter/06_random.pdf

[8] P. Martin, "An Analysis of Random Number Generator for a Hardware Implementation of Genetic Programming using FPGAs and Handel-C 2002 www.celoxica.com/techlib/files/CEL-W0307171J2F-23.pdf

[9] A Statistical Test Suite for Random and Pseudo-Random Number Generators for Cryptographic Application, NIST Special Publication 800-22 (with revision May 15, 2001) http://csrc.nist.gov/rng/

[10] ENT- A Pseudorandom Number Sequence Test Program www.fourmilab.ch/

[11] FIPS PUB 140-1
Security Requirements for Cryptographic Modules http://csrc.nist.gov/cryptval/140-1.htm

[12] FIPS PUB 140-2
Security Requirements for Cryptographic Modules http://csrc.nist.gov/cryptval/140-2.htm