A comparison of random number sequences for image encryption

Prof. Antonios Andreatos (1) and Prof. Apostolos Leros (1,2)
(1) Div. of Computer Engineering & Information Science
Hellenic Air Force Academy
Dekeleia, Attica, TGA-1010, GREECE
informatics.hafa@haf.gr, aandreatos@gmail.com

(2) Department of Automation Engineering
School of Technological Applications
Technological Educational Institute of Sterea Hellas
34400 Psachna, Evia, GREECE
lerosapostolos@gmail.com

MMCTSE

29 November 2014

OBJECTIVE

to compare a number of random and pseudorandom number sequences and examine their suitability for image encryption.

The comparison criteria used are:

distribution of random numbers, entropy, statistical tests and encrypted image autocorrelation.

All the generators examined provide satisfactory results.

1/ Introduction - Random Number Generators

A common classification of random number generators based on the source of randomness is the following:

True Random Number Generators (TRNGs),

Pseudo-Random Number Generators (PRNGs) and

Hybrid Random Number Generators (HRNGs).

CRNGs = Chaotic Random Number Generators.

TRNGs

TRNGs take advantage of unpredictable, nondeterministic sources such as natural processes or physical phenomena which can affect a sensor measuring some physical magnitude and converting the measurement into a sequence of statistically independent data.

Physical phenomena commonly exploited in the generation of random numbers are:

radioactive decay, thermal noise and cosmic microwave background.

However, the respective devices are not portable, hence unsuitable for use outside a laboratory.

PRNGs

PRNGs are algorithmic generators of numbers which have the appearance of randomness, but nevertheless, their results are predictable.

Good random number generators produce very long sequences which look random, in the sense that no efficient algorithm can guess the next number given any prefix of the sequence.

PRNGs use minimal randomness - a randomly chosen initial value called seed. For a specific seed, PRNGs produce a specific, repeatable as well as periodic pattern.

This feature is desirable in cryptographic and steganographic telecommunication systems, because the pseudorandom sequence used in the transmitter for encryption/ steganography must be faithfully reproduced in the receiver.

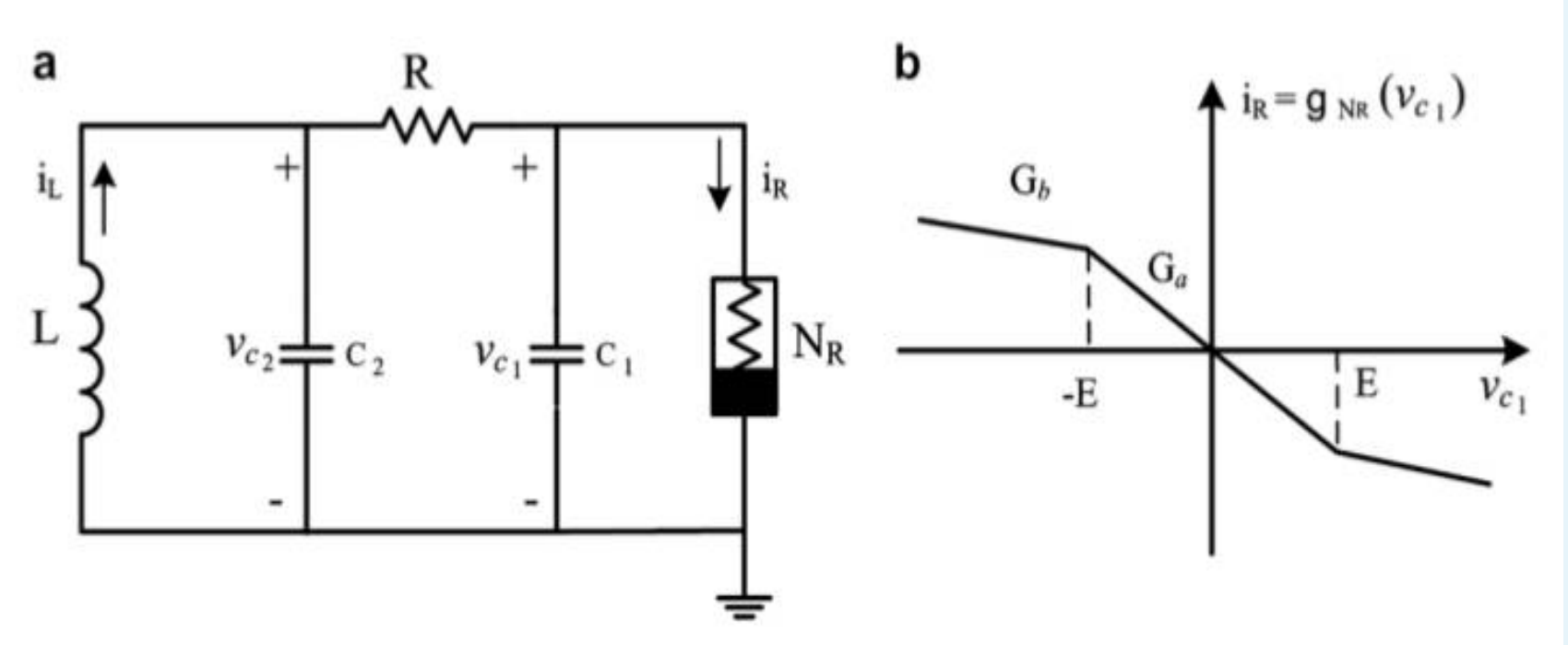
CRNGs

A special category of generators is the chaotic random number generators (CRNGs) which are based on chaotic phenomena. Physical implementations of chaotic generators approach TRNGs, because real device values have a tolerance and they are also affected by environmental reasons, aging, etc.

Software simulations of chaotic phenomena resemble PRNGs, hence they share the same advantages and can be used in cryptography and steganography.

The most famous as well as simple chaotic implementation is Chua's circuit. Several cryptographic and steganographic telecommunication systems based on Chua's circuit and its variations have been proposed.

Chua's circuit



What we examine

In this paper we compare five sources of random numbers and their suitability *for image encryption*:

1/ The PRNG of C (rand() function).

2/ The PRNG of PHP (rand() function).

3/ The PRNG of Matlab (randi function).

4/ A chaotic PRNG based on Chua's circuit, simulated in Matlab, abbreviated henceforth as CRNG.

5/ A truly random generator based on the HAVEGE algorithm.

HAVEGE

The HAVEGE (HArdware Volatile Entropy Gathering and Expansion) algorithm harvests the indirect effects of hardware events on hidden processor state (caches, branch predictors, memory translation tables, etc.) to generate a random sequence.

The effects of interrupt service on processor state are perceived as timing variations in program execution speed.

Using a branch-rich calculation that fills the processor instruction and data cache, a high resolution timer source such as the processor time stamp counter can generate a random sequence even on an idle system.

Tests carried out

The tests considered are the following:

1/ Visual tests;

2/ Entropy tests;

3/ Statistical tests;

4/ Image autoCorrelation tests.

Test image

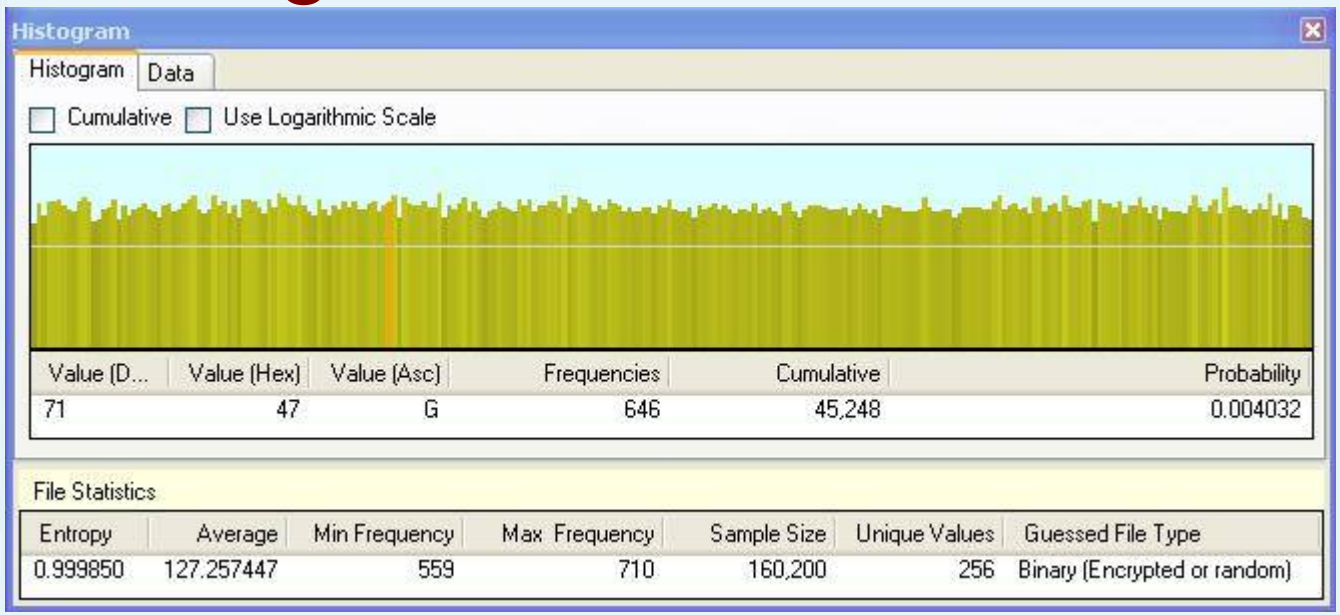
Poor encryption example

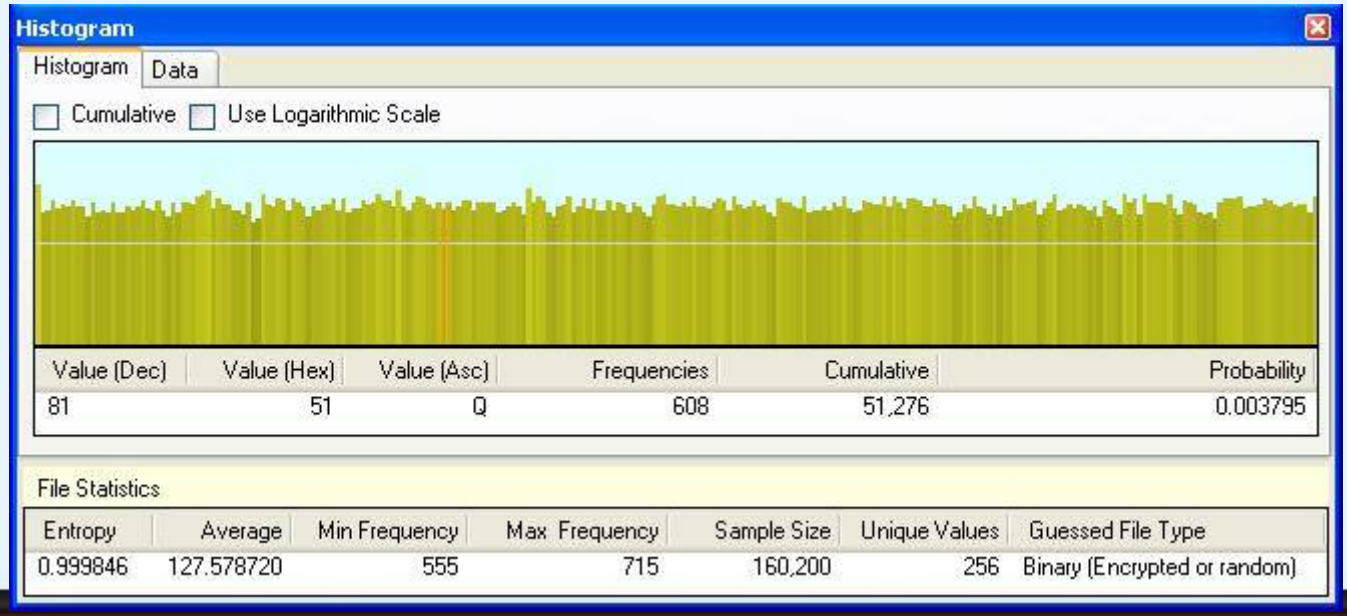
2/ Visual Tests

A. Uniformity Test **(using the Binary Viewer)**

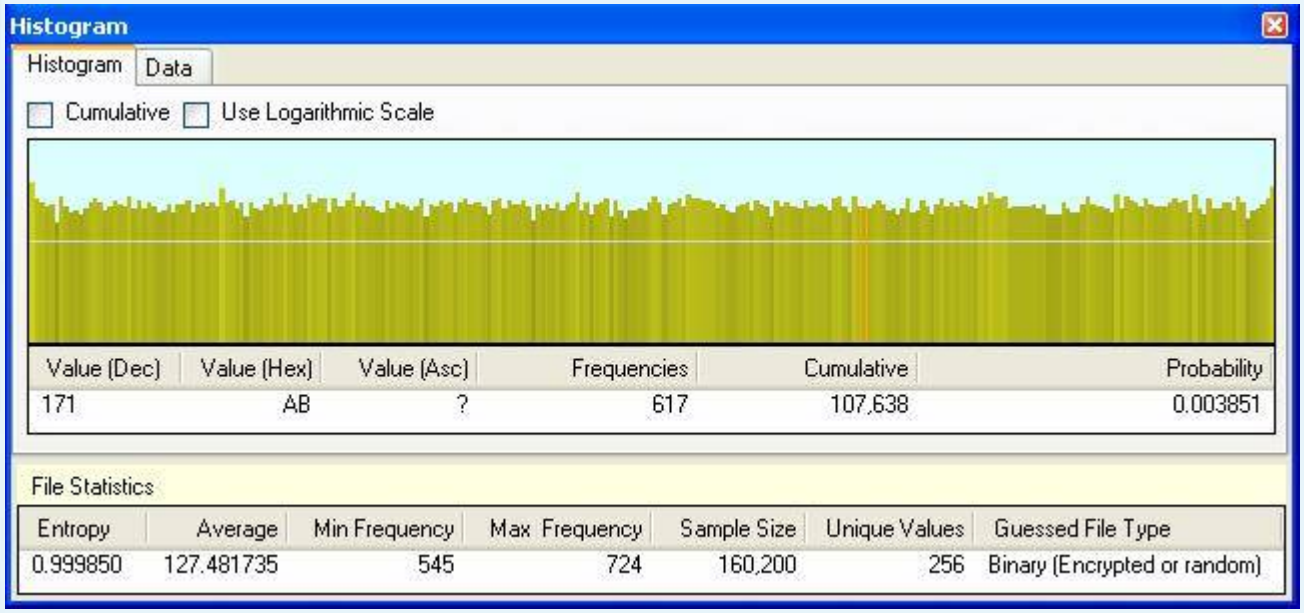
B. Cipher-image inspection

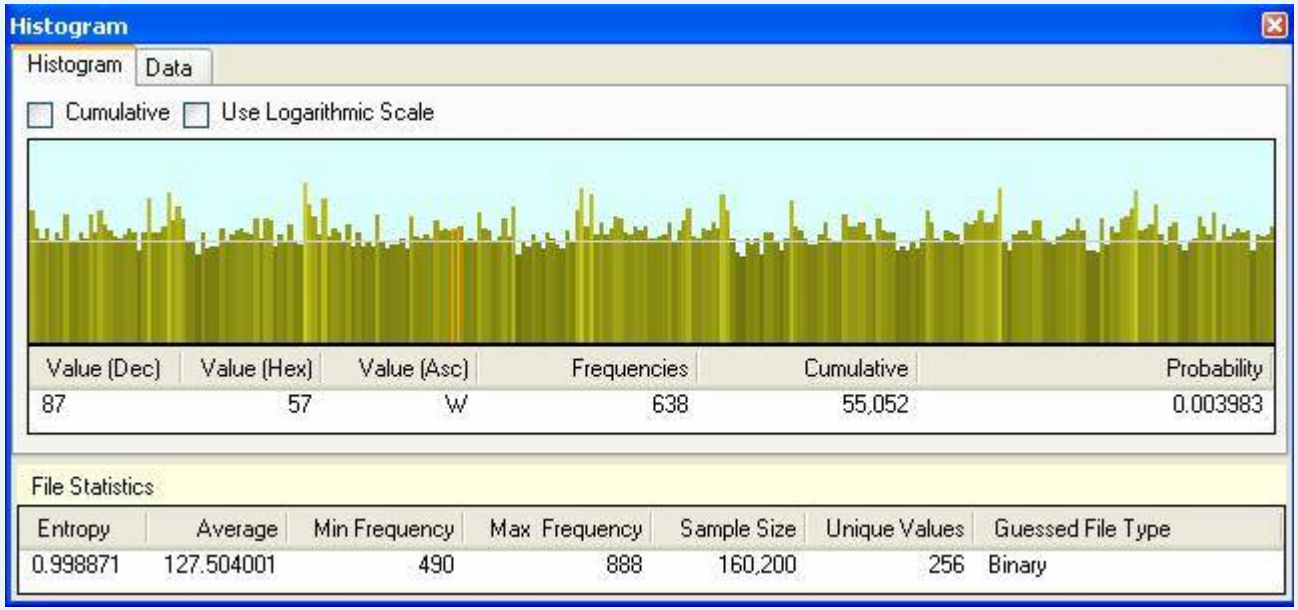
Histogram of C & PHP random numbers



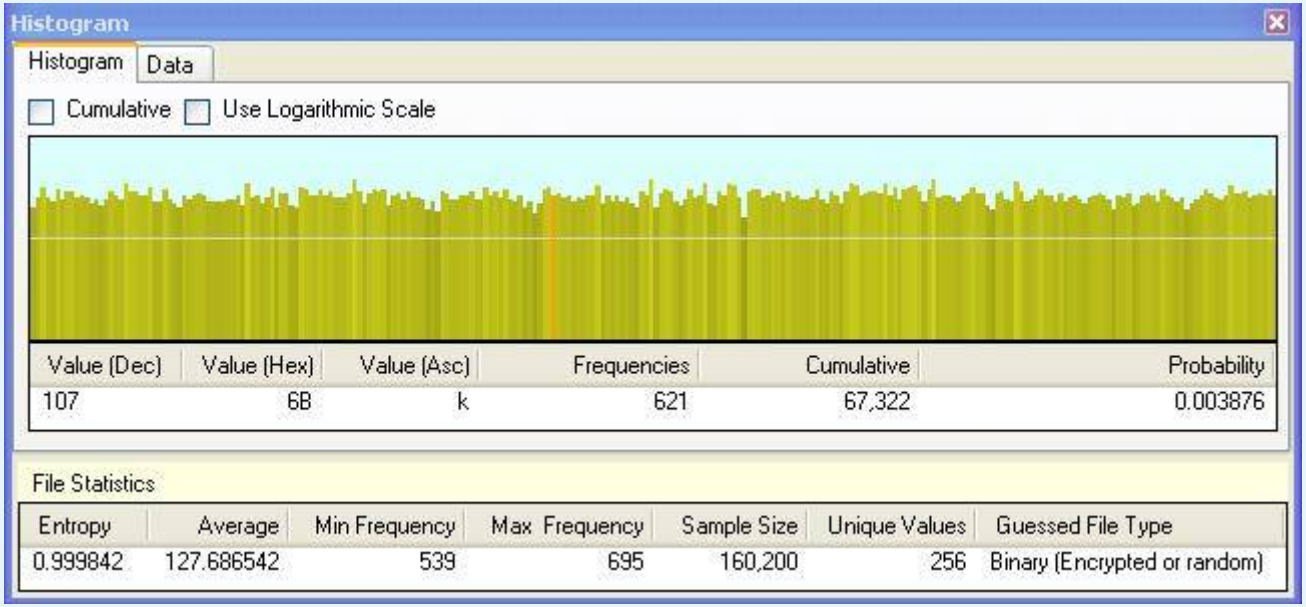


Histogram of Matlab & Chaotic random numbers

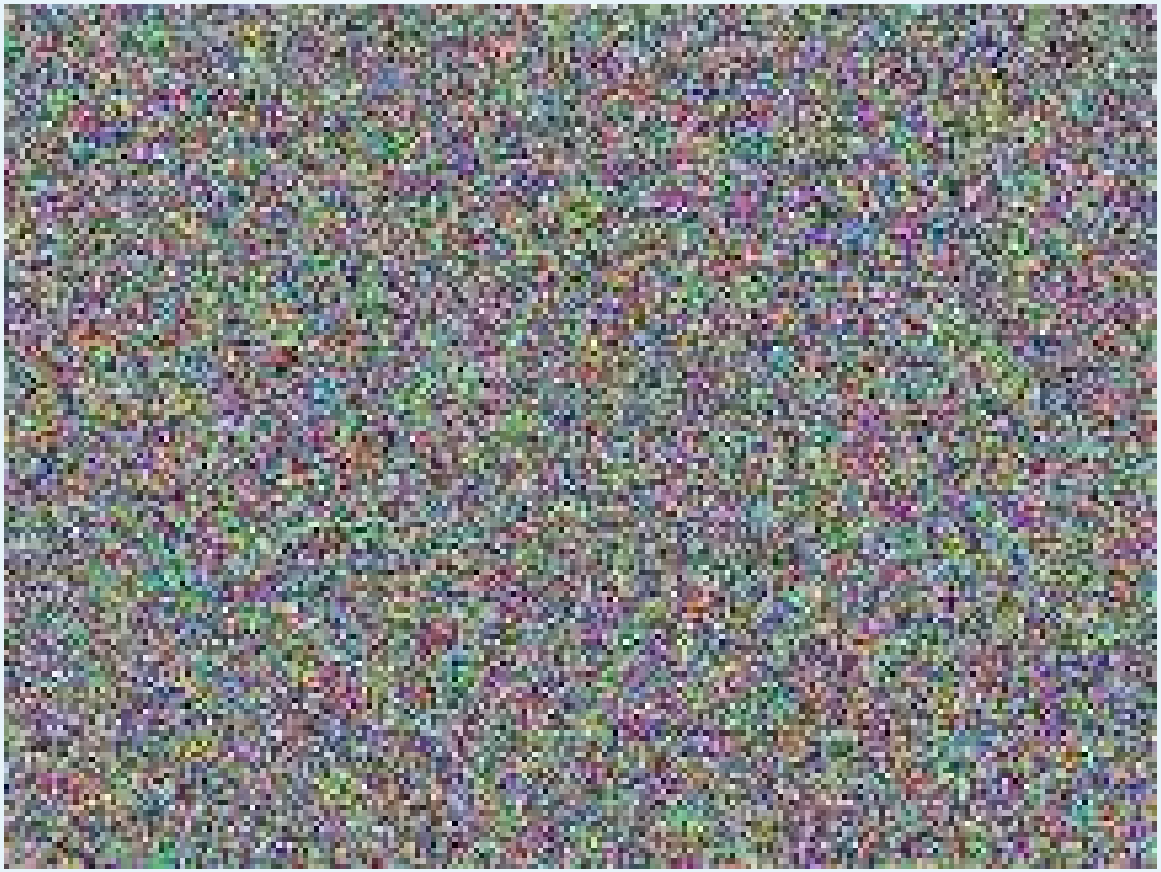




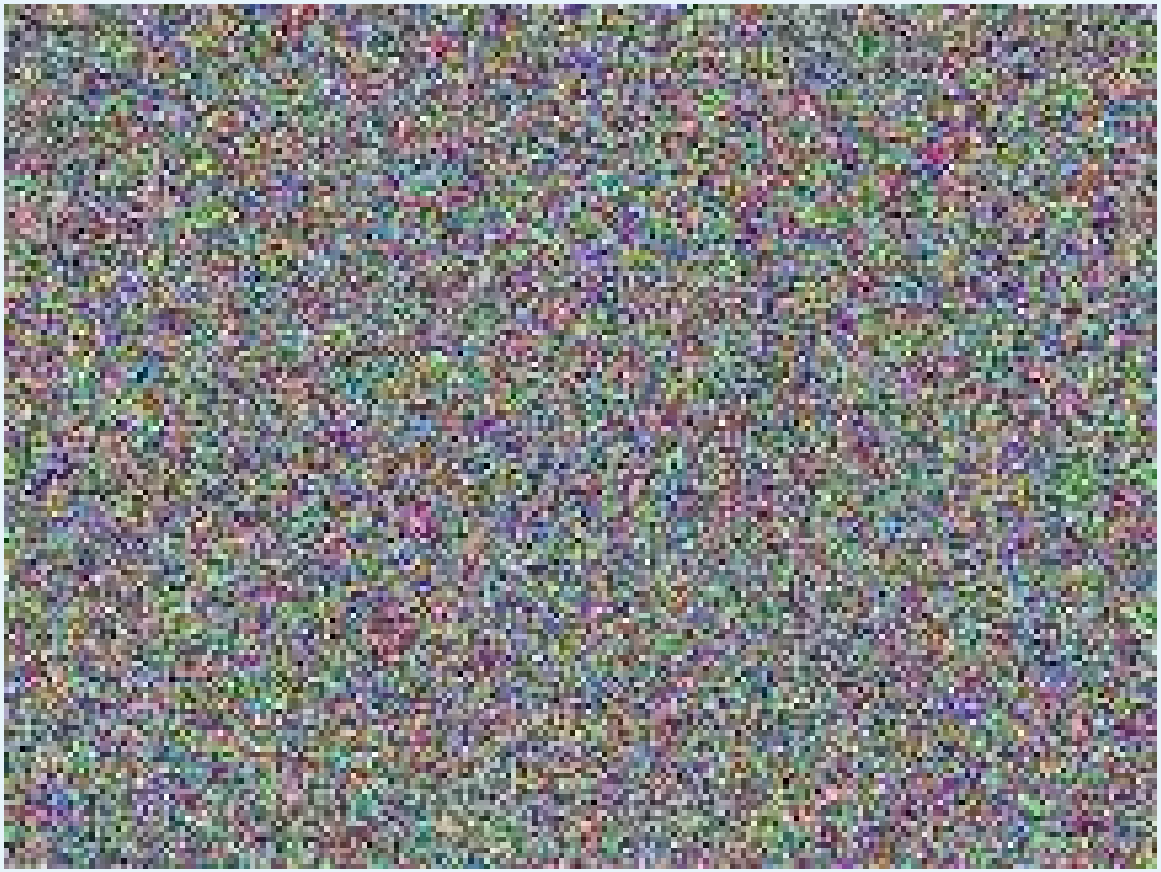
Histogram of Havege Truly random numbers



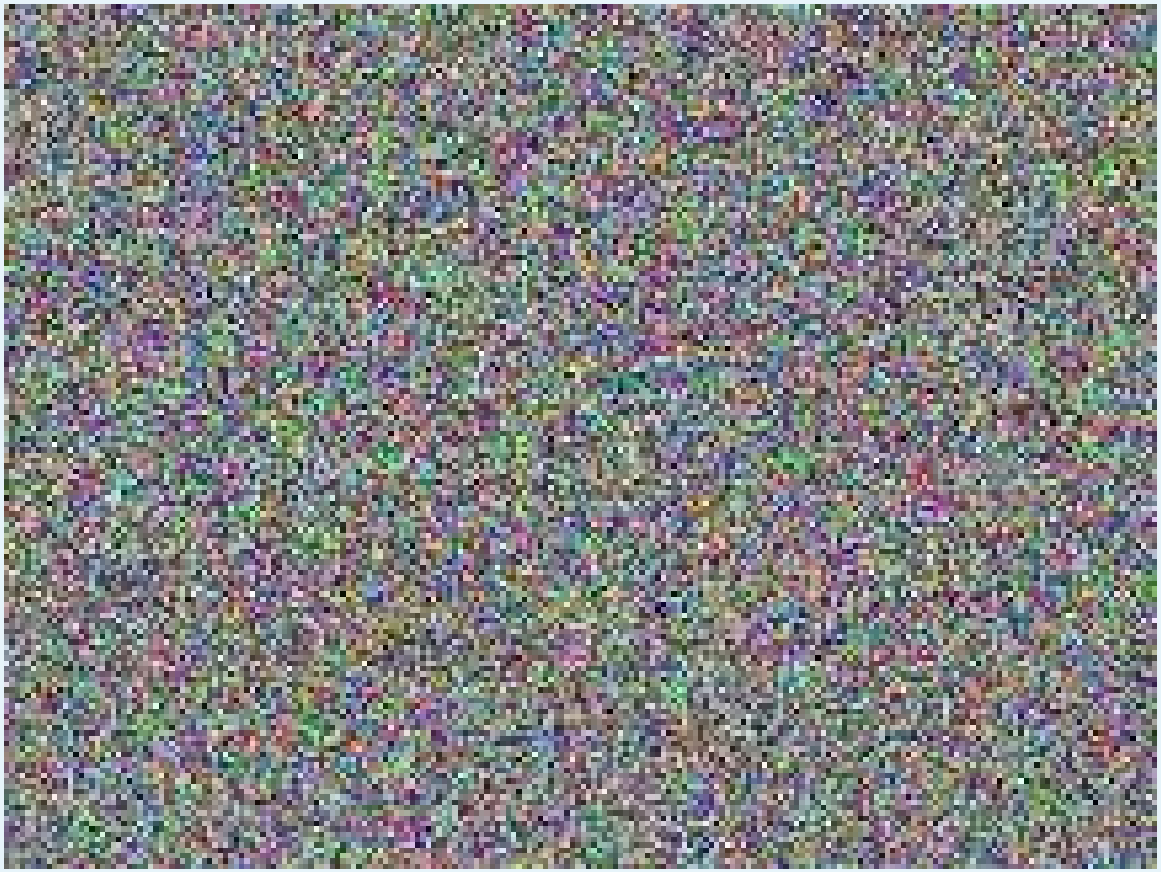
Cipher-image produced by C RNS



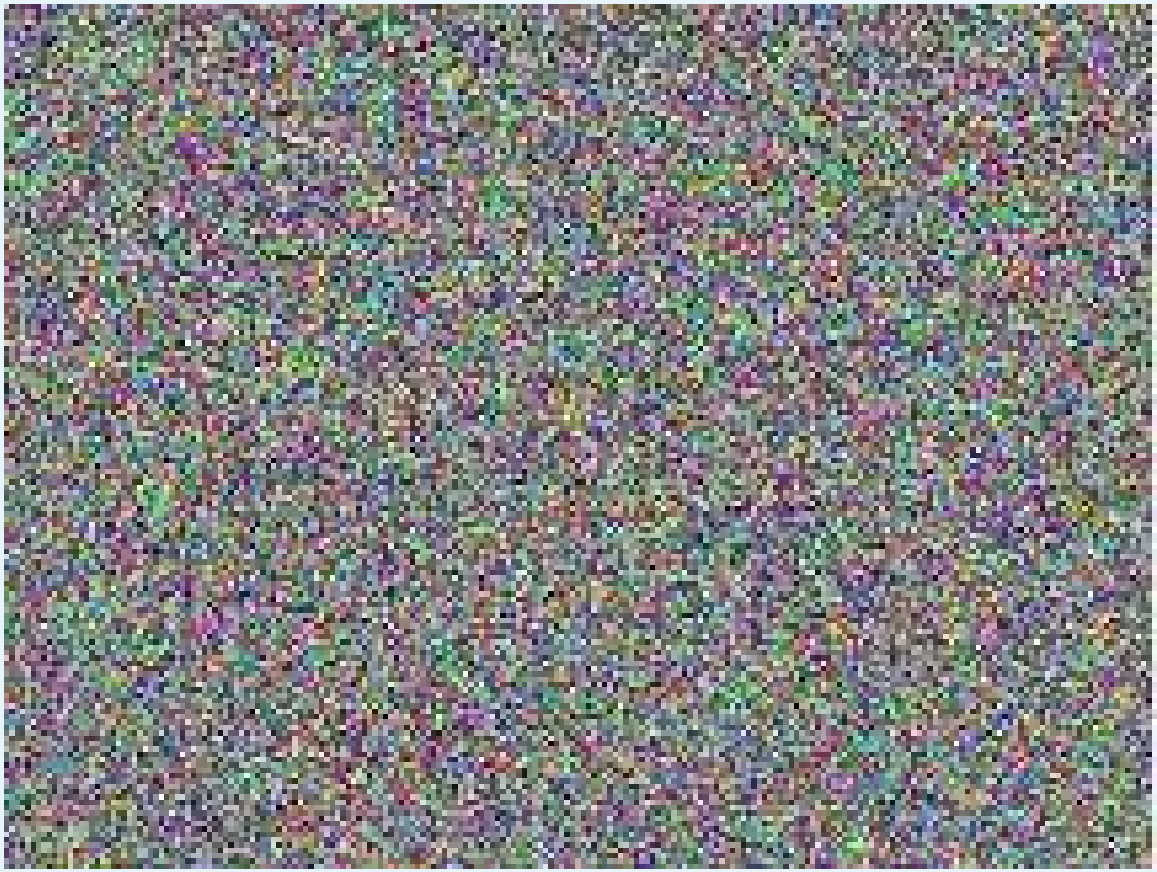
Cipher-image produced by php RNS



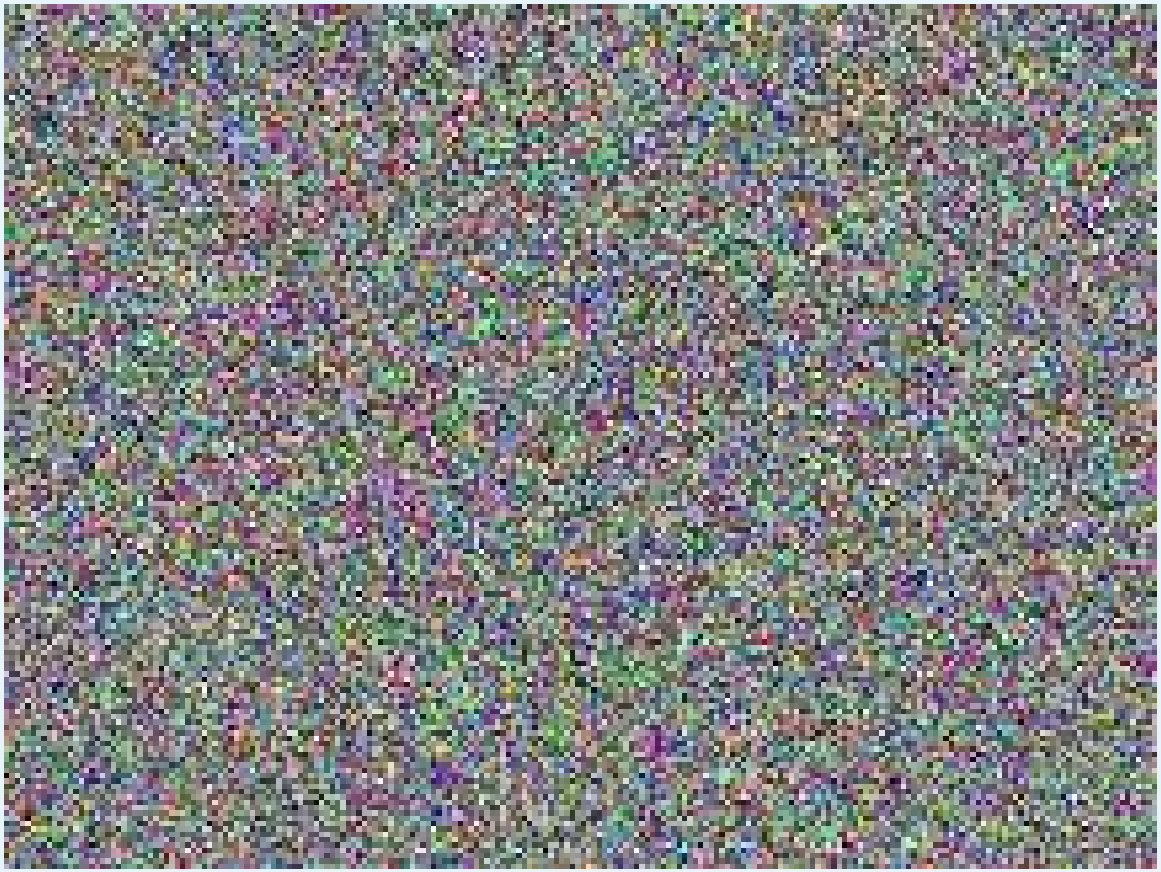
Cipher-image produced by Matlab RNS



Cipher-image produced by CRNG RNS



Cipher-image produced by Havege RNS



3/ Entropy tests

Entropy is defined as the amount of information contained in a random variable. Entropy of a truly random bit sequence equals its size in bits.

Entropy tests were performed by means of the ***ent*** bash command, as well as, the **Binary Viewer** software. The higher the Entropy, the better the quality of random numbers.

Results as shown in Table 1 below in bits per byte; a result of 8 bits per byte means perfect. In Binary Viewer (BV) the perfect entropy is represented by 1.

TABLE 1. ENTROPY TESTS

	C	PHP	Matlab	CRNG	HAVEGE
Entropy (BV)	0.999850	0.999846	0.999850	0.99871	0.999842
Entropy	7.993240	7.998768	7.998804	7.990964	7.998738
Chi square distribution	266.65	273.95	265.73	2064.94	279.93
Arithmetic mean	127.2574	127.5787	127.4817	127.5040	127.6865
Monte Carlo value for Pi	3.152359551	3.151760300	3.149213483	3.135730337	3.13378277
error %	0.34	0.32	0.24	0.19	0.25
Serial correlation coefficient	-0.003169	0.001998	-0.000521	0.000168	0.002261

4/ Statistical Tests

The FIPS 140-2 test suite was used for the Statistical Tests.

The Federal Information Processing Standard (FIPS) Publication 140-2 (FIPS PUB 140-2) is a U.S. government computer security standard used to accredit cryptographic modules. The official title is Security Requirements for Cryptographic Modules. Initial publication: May 25, 2001. Update: December 3, 2002.

RN sequences used were 1,282,600 bits long.

The FIPS 140-2 tests

For a bitstream of 20,000 bits:

- Monobit Test: The number n1 of 1's must be 9,725 <n1 < 10,275. (50%)
- Poker Test: This test determines whether the sequences of length (n = 4) appear approximately for the same number of times in the bitstream.
- Runs Test: This test determines whether the number of 0's (Gap) and 1's (Block) of various lengths are as expected for a random sequence.
- Long Run Test: no runs longer than 26 bits.

Statistical Test results

TABLE 2. STATISTICAL TESTS

	C	PHP	Matlab	CRNG	HAVEGE
Successes	64	62	59	59	63
Failures	0	2	5	5	1
Monobit	0	0	0	0	0
Poker	0	0	0	2	0
Runs	0	0	0	0	1
Long run	0	2	5	3	0
Continuous run	0	0	0	0	0

5/ Autocorrelation Tests

In this section we compute the **autocorrelation of the encrypted images**. The autocorrelation of an image is defined as the similarity of an image with itself, shifted by one pixel horizontally, vertically, diagonally and anti-diagonally. The autocorrelation was calculated in Matlab using the following formulae (1) and (2). The correlation coefficient γ for a pair of pixels is defined as described in formula (1):

$$\gamma(x,y) = \frac{\mathrm{cov}(x,y)}{\sigma_x \sigma_y} \qquad (1)$$

Where:

$$\mathrm{cov}(x,y) = \frac{1}{N}\sum_{i=1}^{N}[x_i - E(x)][y_i - E(y)] \qquad (2)$$

comment

In eq. (2) "E" is the expected value operator and σ_x^2 represents the variance of variable x. The values of $\gamma(x,y)$ lie in the range $[-1, 1]$, with 1 indicating perfect correlation, −1 indicating perfect anti-correlation and 0 indicating no correlation.

The evaluation of these formulae in MATLAB was realised by the use of built-in functions.

Cipher-Image autocorrelation

TABLE 3. AUTOCORRELATION TESTS

	Horizontal	Vertical	Diagonal	Anti-diagonal
Original	0.9758	0.9527	0.9278	0.9561
C	0.0025	-0.0028	0.0022	-0.0037
PHP	0.0002	-0.0007	-0.0003	0.0044
Matlab	0.0032	0.0010	0.0031	0.0029
CRNG	0.0035	0.0016	-0.0047	0.0003
HAVEGE	0.0037	-0.0021	-0.0008	-0.0020

6/ Discussion

One might possibly have expected that the Havege TRNG would have the best results by far;

however, all other generators produce comparable results, which means that the specific generators are of good quality.

In fact, PHP and Matlab use the **Mersenne Twister** generator which is, by far, the best and most widely used PRNG.

Mersenne twister is used by many programming languages, including PHP and Matlab; hence the good results. The commonly used version of Mersenne Twister is “MT19937”, which has a very long period of 2^19937−1.

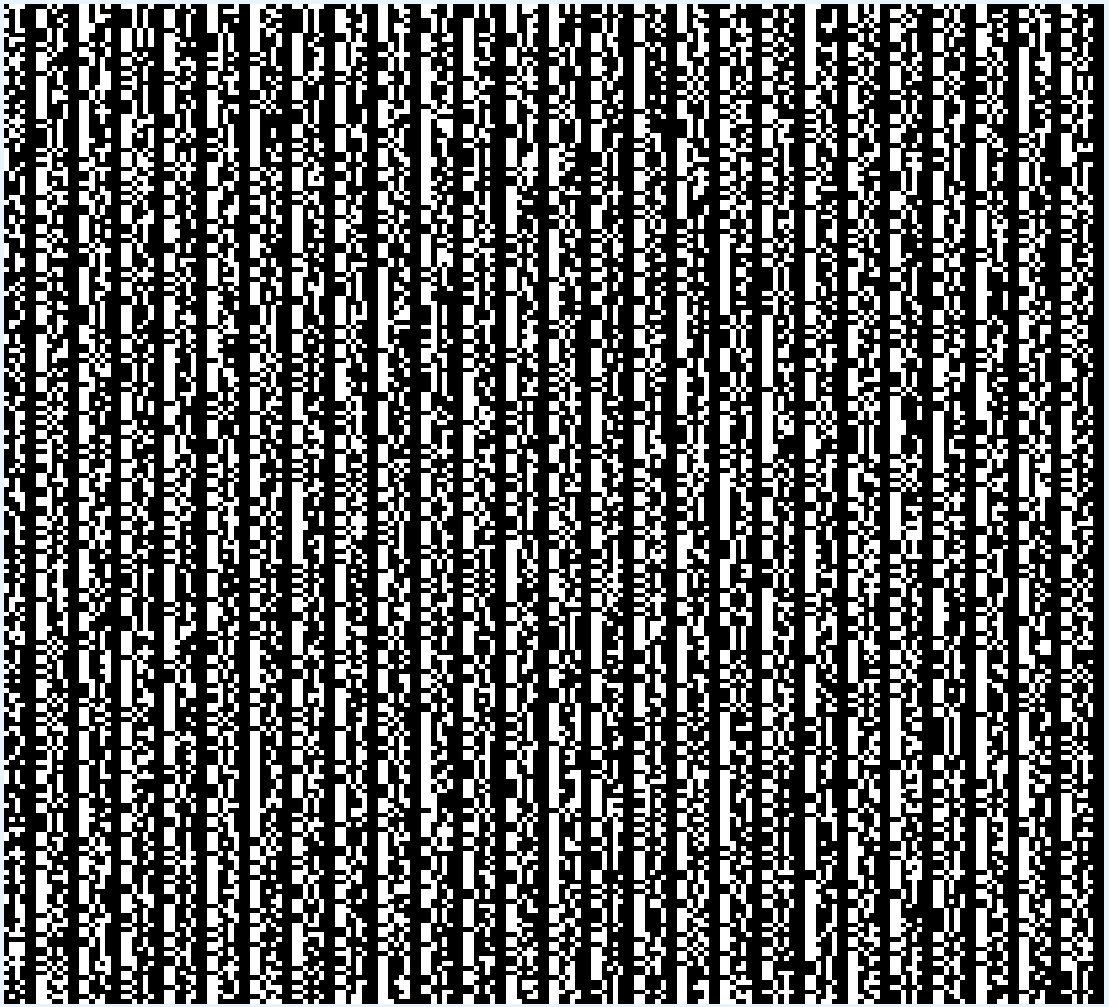
Discussion (cont.)

In order to *visually* check the periodicity of a RNS, we have to arrange its numbers (in our case, integers from 0 to 255) as pixels in a matrix. The longer the PRNG period, the longer the RNS in order to observe patterns.

This is the reason for not observing patterns in the visual tests performed in the RNS under test, since their size was only 1,281,600 bits (small compared to the PRNG period).

However, if we generate a large RNS, patterns appear.

Patterns in visualisations of PHP pseudo-random sequences indicate periodicity (40 million numbers)



7/ Conclusion

In this paper we have compared three pseudo-random number sets, a chaotic number set and a truly random number set, for use in image encryption.

From the comparison it seems that all sources produce acceptable, high quality results for the selected test image (160,200 bytes).

However, neither pseudo- nor truly random numbers are suitable for cryptography, for different reasons each.

conclusion (2)

Pseudo-random numbers are in fact periodic sequences, hence, easy to guess/ break.

The reason is that observing a sufficient number of past iterations allows us to predict all future iterations. Hence, they are not suitable for cryptography.

Truly random numbers are not reproducible; hence, it will be impossible to decode the cipher in the receiver.

conclusion (3)

The CRNG produces satisfactory results, comparable with the PRNGs and the TRNG; moreover, it has some additional advantages:

They are periodic but by applying spatiotemporal techniques, their period may become some years long, hence practically unreachable;

Under proper design of the generator, they can use multi-parametric keys rather than a single seed;

The CRNG is multi-parametric system and must be fine-tuned in order to improve its performance. A method for tuning Chua-based CRNGs has been proposed in the bibliography (future work).

END of the PRESENTATION

- Thank you for your attention

– Any questions?

More:

http://t-h.wikispaces.com/Chua-Chaos