THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 13, Number 4/2012, pp. 368–377

GENERATION AND TESTING OF RANDOM NUMBERS FOR CRYPTOGRAPHIC APPLICATIONS*

Kinga MÁRTON [1], Alin SUCIU [1], Christian SĂCĂREA [2], Octavian CREŢ [1]

[1] Technical University of Cluj-Napoca, Computer Science Department
26-28 George Baritiu Street, Cluj-Napoca, Romania
[2] "Babeş-Bolyai" University, Mathematics Department
1 Mihail Kogalniceanu Street, Cluj-Napoca, Romania
E-mail: alin.suciu@cs.utcluj.ro

Random number sequences are of crucial importance in almost every aspect of modern digital cryptography, having a significant impact on the strength of cryptographic primitives in securing secret information by rendering it unknown, unguessable, unpredictable and irreproducible for an adversary. Although the major importance of high performance and high quality randomness generators in cryptography attracts an increasing interest from the research community, unfortunately this tendency is not always shared among security application developers. Thus many available cryptographic systems continue to be compromised due to the utilization of inadequate randomness generators. In this context, the present paper aims to accentuate the crucial importance of random numbers in cryptography and to suggest a set of efficient and practical methods for the generation and testing of random number sequences intended for cryptographic applications.

Key words: random numbers, unpredictability, randomness generators, randomness tests.

1. INTRODUCTION

The security provided by cryptographic systems is strongly related to randomness as almost every aspect of cryptography depends on the accessibility of random number generators that exhibit high statistical quality, are available, affordable, provide high throughput and at the same time are strong in the sense that can withstand analysis. Yet, in the absence of a thorough analysis of the randomness source and of the quality of produced sequences, many available cryptographic systems continue to be compromised due to the utilization of inadequate randomness generators.

The result is the compromise of the whole system that fails to provide the desired level of security. The loss and the associated costs and efforts that are necessary for recovering from the security break can be extremely high, and constitute the central motivation of the research towards choosing, designing, testing and carefully integrating high quality random number generators in cryptographic systems.

Although a standard formal definition is missing, randomness refers to the outcome of a probabilistic process that produces independent, uniformly distributed and unpredictable values that cannot be reliably reproduced [12]. The major ingredients for randomness are unpredictability (or lack of predictability), independency of values (or lack of correlation), and uniform distribution (or lack of bias).

Some of the properties of a random sequence are statistical and hence can be measured by using various statistical randomness tests. However, the most important problem in connection with randomness is the lack of certainty. By extensive analysis and thorough testing one can get a high confidence in the generator but cannot be absolutely sure. This is why there is a wide range of statistical test suites (NIST [11], TestU01 [6], Diehard [7], ENT [20], etc.) that try to rule out sequences which do not verify certain statistical properties, yet can never guarantee perfect randomness.

The statistical analysis of the random sequences is very important but alongside the application of statistical tests that assess the outcome of a randomness generator, there must be a serious analysis of the

* This material was in part presented at the international conference Romanian Cryptology Days, RCD-2011.

source the generator extracts randomness from. Consider the result of a statistical test suite that would indicate good random properties of a sequence but then it is pointed out that the sequence was in fact built from Pi's digits (no unpredictability there). Therefore random sequences used in cryptography must be unpredictable, irreproducible and should not allow the adversary to learn or predict former or subsequent values.

The rest of the paper is organized as follows. Section 2 provides a brief description of the many practical applications of random numbers in cryptography. Section 3 offers a short glance into several practical unpredictable generator designs. In Section 4 we highlight the essential process of statistical testing and provide a description of research results aimed at optimizing some of the most well-known batteries of statistical test suites. Finally in Section 5 we draw some conclusions.

2. THE MANY FACES OF RANDOMNESS IN CRYPTOGRAPHY

Cryptography comes with a wide range of techniques in order to provide solutions for different security requirements – and these techniques require random sequences for many different purposes, therefore randomness takes upon a variety of roles that we aim to identify and highlight in the following [8].

One of the most important roles randomness plays in cryptography is represented by **cryptographic keys** which determine the transformation of the plaintext into ciphertext and vice versa. Considering that both the encryption and the decryption algorithms are publicly known together with all the ciphertexts transmitted between the sender and receiver, the security of the whole cryptosystem is dependent on how the key information is managed: generated, agreed on, applied, stored and destroyed. The knowledge of the key entails the access to the secret message, thus the choice of the key space and the key derivation method is critical.

Cryptographic keys must be unpredictable for the adversary meaning a high information content and high uncertainty, and the measure of these properties is usually considered to be the **entropy**. Yet high entropy is not enough, and a very good example in this direction is a compressed file which besides its high entropy value has a highly structured content. Thus sequences chosen for cryptographic keys must also exhibit independency of values, uniform distribution and irreproducibility. As a result what cryptography needs most for its keys is randomness.

Initialization vectors (IVs) are used in both stream ciphers and block ciphers, and although the implementation differs, the goal of applying IVs is the same, namely to ensure that the ciphers produce a unique output even under the same encryption key, in order to avoid the laborious work of rekeying.

The concept of **cryptographic salt** refers to a random sequence, related to the concept of nonce, and used generally as one of the input values for key derivation functions (KDF), which generate additional cryptographic keys based on an initial keying material such as a single master key, a password or a passphrase, nonces exchanged in a key agreement protocol, etc. The most popular KDFs use hashes or block ciphers as derivation functions and can operate without a salt value, but using cryptographic salt adds significantly to the security of the output key and are of special focus when the keying material is a password or a passphrase which otherwise would be exposed to easily performable dictionary attacks.

Random **padding strings** are used in symmetric key block ciphers (in ECB and CBC modes of operation) in order to complete the last plaintext block in a way that disguises the original message length, but at the same time are of significant importance in turning deterministic public encryption schemes into randomized algorithms by appending an independently generated random padding value to the plaintext before encryption (sometimes called salting the message) in order to ensure that the same plaintext will not result each time in the same ciphertext under a fixed public key and that short messages will not be compromised by an adversary trying all the possible alternatives of the small message space. Furthermore, padding schemes associated with asymmetric encryption algorithms, like Optimal Asymmetric Encryption Padding (OAEP) or Probabilistic Signature-Encryption Padding (PSEP) also use random sequences with the major goal of ensuring that an attacker manipulating the plaintext cannot exploit the mathematical structure of the used asymmetric cipher, such as the well known distributive property of the RSA: $E(M1 \times M2) = E(M1) \times E(M2)$. There are other attacks as well that can be defeated using random padding algorithms as described in [1].

Random **nonce values** (derived from number used once) are generally used in data-origin or entity authentication (challenge-handshake authentication protocols) and authenticated key establishment protocols as challenges and have the goal of ensuring that a certain cryptographic operation is performed after the challenge was received so that the answer is not a replay and can check the freshness of the message (a session key) if the authentication mechanism also contains a proper message integrity scheme.

Whether used to ensure message freshness and principal liveness, mutual or unilateral authentication with/without the help of a trusted third party, random nonces play a very important role alongside with cryptographic keys shared secret keys (such as in Needham Schroeder symmetric key protocol and Kerberos) or public-private key pairs, providing uniqueness and timeliness assurance. Cryptographically secure random generators have to be selected for producing nonces (or challenges) so that values are not reused and are unpredictable to attackers. Usually block cipher based pseudorandom generators are applied in OFB or counter modes.

Random nonce values are very important in another type of cryptographic protocols, named **zero-knowledge proof** protocols that make possible the demonstrations of knowledge of a secret while revealing no information about the secret (other than what the verifier can deduce without the help of the prover). In satisfying this goal, zero-knowledge protocols use randomness with two different purposes, firstly as challenges similar to a certain extent to asymmetric challenge-response authentication protocols, and secondly as commitments in order to prevent cheating [9, 12].

The **blinding values** used in special classes of digital signature algorithms, called blind signature schemes (BSS), can provide additional features to authentication and nonrepudiation, like blindness the property of signing a message without exposing the message content to the signer, and untraceability – the property which ensures that the signer of the message is unable to associate signatures to messages even when signatures are revealed to the public. BSS are of great importance in many security sensitive applications especially in those focusing on the privacy of the user, like electronic voting (e-voting) or digital cash (e-cash) protocols [2, 4].

The many roles randomness plays in cryptography and the importance of each role emphasizes the high degree to which cryptography relies on randomness in providing the security requirements it is designed to deliver.

3. GENERATION OF RANDOM NUMBER SEQUENCES

Depending on the nature of the entropy source the generators rely on, RNGs can be classified in three categories: true random number generators (TRNGs), pseudo random number generators (PRNGs) and unpredictable random number generators (URNGs), a brief description of these categories is provided in the following [8].

True Random Number Generators (TRNGs). TRNGs extract randomness by sampling and digitizing natural physical phenomena (like thermal noise, jitter, radiation, etc.), the unpredictability of the generated values being guaranteed by physical laws. While TRNGs offer the highest level of nondeterminism and irreproducibility they do not necessarily present perfectly uniform distribution and independence hence their output needs to be filtered (post processed) in order to reduce possible bias (tendency towards a particular value) or correlation, and make the output more similar to a statistically perfect random sequence. Nonetheless, true random sources need specialized hardware, the generators are expensive and some of them are slow and impractical.

Pseudo Random Number Generators (PRNGs). PRNGs extract randomness from an initial value, called seed, which is expanded by means of a deterministic (usually recursive) formula, providing a modality for generating random sequences using only software methods. As a result, the effective entropy and irreproducibility level resumes to the unpredictability of the seed value (that is therefore recommended to be obtained from a true random number sequence) and the output is completely determined by the starting state of the generator – the seed. Still, the practical features of PRNGs such as high generation speed, good statistical properties and no need for additional hardware devices made these generators very attractive and are the most widely used RNGs. In cryptographic systems they are represented by the Cryptographically

Secure PRNGs-CSPRNGs based on cryptographic primitives or mathematical problems considered to be extremely difficult to solve, and hence are exposed to possible compromise if advances in technology should reveal solutions to the problems the generators rely on.

Unpredictable Random Number Generators (URNGs). URNGs are a practical approximation of TRNGs, based on the unpredictability induced by the complexity of the underlying phenomenon that the generator is based upon (e.g. volatile internal processor states [13, 14], human-computer interaction [17, 5], race conditions [3]). These generators usually extract randomness from easily available devices, like computer components, and may provide a high level of randomness, but special attention must be given to the way components built for being deterministic can be employed for generating randomness, as thorough knowledge of the underlying phenomenon may ease the prediction of internal states and hence next values. URNGs exhibit certain characteristics of both TRNGs and PRNGs being based on the behavior of hardware devices like TRNGs are, yet performing a deterministic sequence of operations like PRNGs do; nevertheless the impact of the multitude of events and parameters is so complex and any intervention in the generation process disturbs the internal state in such a way that it is practically impossible for an adversary to model and predict the produced output.

Each of the following methods for generating randomness can be classified in the category of unpredictable random number generators (URNG), which represent a suitable solution when sources of true randomness are not available, too expensive or the nature of the application employing the generator requires a higher throughput and practicality than available TRNGs might provide, but at the same time, the desired level of irreproducibility and unpredictability cannot be met by pseudorandom generators. Hence, URNGs are practical approximations of TRNGs and are generally based on the unpredictability inherent to human computer interaction and on the nondeterminism introduced by the complexity of the underlying phenomenon.

3.1. DataFlow Entropy Collector

Entropy is at the heart of all random number generators, having a significant impact on the level of randomness the generators can provide. Entropy sources must contain something fundamentally or practically unpredictable. Fundamental unpredictability represents the highest level of entropy provided by natural physical phenomena which constitute the core source of TRNGs, but accessing these sources usually implies specialized hardware devices and can often be characterized by a low level of practicality, in terms of affordability, availability and generation rate, motivating the reorientation towards largely available sources of entropy which can provide practical unpredictability and a high randomness quality.

Harvesting the entropy from different sources available within personal computers is an increasingly popular approach. The most well-known examples include the Linux random number generator accessible through /dev/random, the portable random number infrastructure EGADS – Entropy Gathering and Distribution System or the EGD – Entropy Gathering Daemon.

Based on the same principle, our data-flow entropy collector, named Entropy Studio [17] uses many different types of entropy sources from within a personal computer, and collects the entropy generated by these sources by means of a data flow mechanism expressed by the mouse movements over the areas of the system's interface assigned to different entropy sources. This way a new level of uncertainty is added: data from several unpredictable entropy sources flow inside the entropy pool, according to the user's unpredictable mouse movements (Fig. 1).

Entropy may be collected from different hardware components like the keyboard, mouse, microphone, video capture devices (webcam, TV tuner, etc.), network interfaces, screen captures, or known libraries like HAVEGE and also from a preexisting file.

Experimental results show that the collected data is highly unpredictable; the quality of data from the randomness point of view is however dependent on the ability of the user to efficiently collect entropy from as many sources as possible, to process the raw entropy bits using the available tools (filters, hash functions) and to combine the collected bits by means of the data flow mechanism provided by the system.

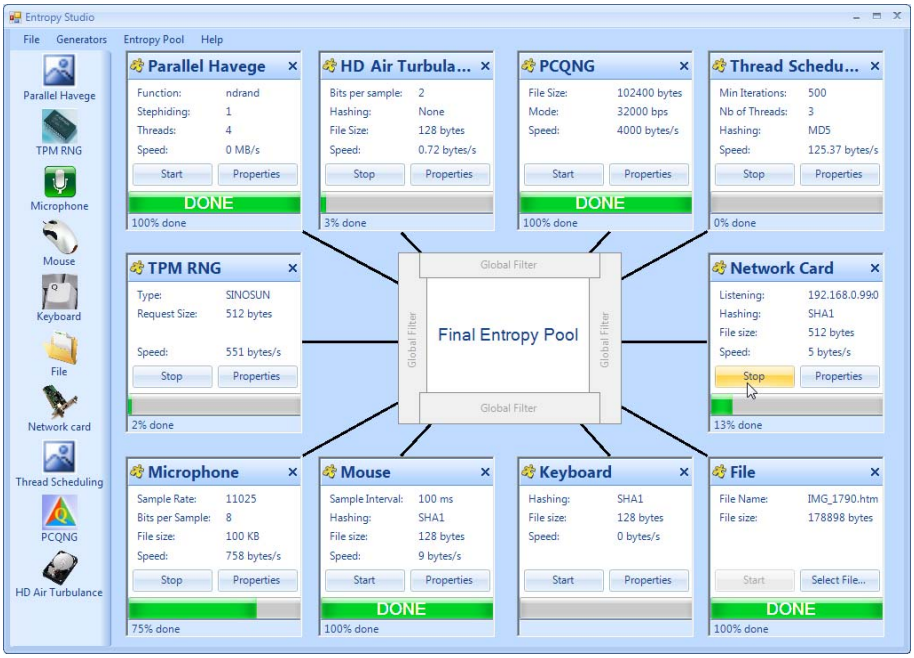

Fig. 1 – Entropy studio – graphical interface.

A major advantage of the proposed system is the possibility to pre-evaluate (using simple tests) the quality of individual sources, before final combination, and to post-process the output in order to provide high quality output sequences. Furthermore, the nondeterministic combination method and the possibility to set individual delays on each entropy source, introduce additional levels of unpredictability, which combined with independent and high quality buffers may provide cryptographically strong random number sequences.

3.2. Parallel-HAVEGE and GridHAVEGE Generators

The unpredictable generation methods Parallel-HAVEGE [16] and GridHAVEGE [18] are based on the HAVEGE generator, which gathers entropy produced by external sources in the internal volatile processor states using the memory hierarchy and the branch prediction mechanism, and expand the entropy using a PRNG (Pseudorandom Number Generator). The security of the algorithm relies on the fact that the internal state of the generator cannot be completely determined because this state is not only composed of memory mapped data but also of thousands volatile hardware states that are inaccessible even to the user running the application.

To make the algorithm more secure against an adversary trying to guess the next sequence one could skip certain results (a process called *step hiding*) by not returning from the function once the output buffer is full, but instead use this buffer (and overwrite it) to generate a new set of random numbers. This could be done several times, sacrificing speed for security.

The goal of our proposed parallel [16] and distributed [18] methods is to provide a high security level without significant loss in the generator's throughput by enabling HAVEGE to take advantage of the processing power of the latest technologies in parallel computing using multi-core architectures and distributed computing on the Grid infrastructure.

The experimental results show that the parallel and "gridified" versions of HAVEGE are useful if a large amount of unpredictable random numbers is needed. The use of any function that is safer (and therefore slower) than the *ndrand* function could also benefit from a speed boost in real life conditions where the numbers need to be consumed with a higher rate than the generator's output rate.

The more steps performed by the custom harvesting function, the more difficult will be for an adversary to predict the output and therefore the greater the security of the algorithm. However, in a serial implementation this increase in security comes at a price: as we increase the number of steps, the speed (throughput) of the generator will decrease proportionally.

The most significant advantage of having parallel and distributed implementations of HAVEGE is the ability to increase the security (number of steps) while maintaining a constant throughput by increasing the number of HAVEGE threads up to the number of available cores/nodes.

3.3. Random Number Generator Based on Hardware Performance Counters

Originally intended for design evaluation and performance analysis, hardware performance counters (HPCs) enable the monitoring of hardware events, yet are noisy by their very nature. The causes of variations in the counter values are so complex that are nearly impossible to determine. Hence, while being a major issue in the process of accurately evaluating software products, the unpredictability exhibited by HPCs offer a high potential for random number generation.

The method we proposed in [15] introduces a new unpredictable random number generator (URNG) based on HPCs and analyses the feasibility of producing cryptographic quality randomness. For cross-platform compatibility reasons only HPCs associated to *preset events* were considered and through several experiments a subset of these HPCs were identified that can be employed as sources of high unpredictability within the proposed generator.

The statistical quality of the generated randomness was thoroughly tested using the well known statistical test suites: Alphabit and Rabbit batteries of TestU01 [6] and the NIST [11] test suite, the results showing the high randomness quality of generated sequences, depicted in Fig. 2.

Furthermore, these experiments also emphasize issues that have to be considered such as: the choice of the sampled events, the decrease in randomness quality when concatenating the outputs of concurrent threads sampling the same event counter and the memory requirements for mixing the output files of concurrent threads.

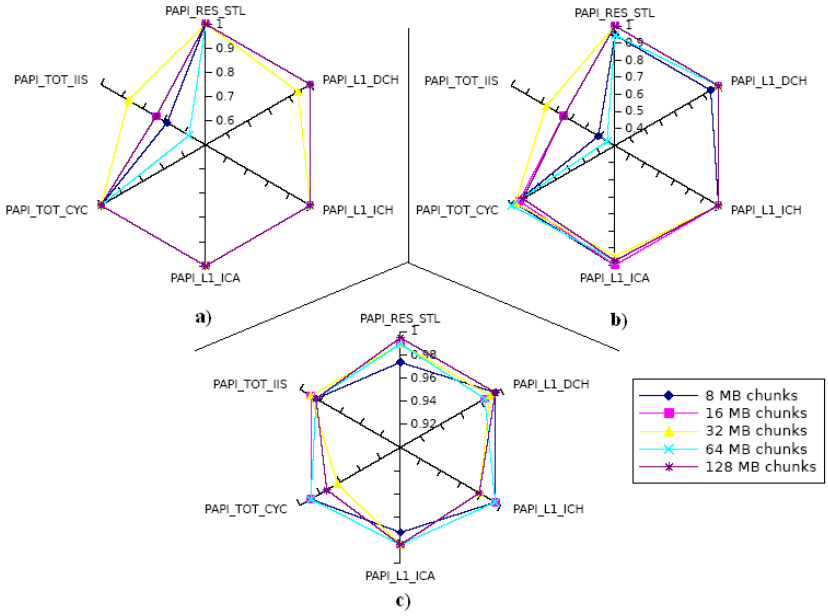

Fig. 2 – Ratio of: a) alphabit battery tests; b) rabbit battery tests; c) NIST tests passed by 1 GB output files of GPXOR.

The throughput of the proposed generator is approximately 150 KB/s, a value comparable with the throughput exhibited by the state of the art HAVEG generator. The results show that the runtime of the generator increases linearly with the size of the output file, hence the throughput remains constant at a level which makes the proposed generator comparable to the state of the art HAVEG algorithm [13] which produces between 8 and 64 KB per operating system interrupt.

Therefore all aspects regarding the unpredictability, throughput and availability of the randomness source and the quality of the generated sequences indicate that the proposed unpredictable random number generator is suitable for integration in security systems for providing cryptographic randomness.

4. TESTING OF RANDOM NUMBER GENERATORS

4.1. Statistical Testing

The most common method for testing random number generators is based on the statistical analysis of their output. Statistical test suites play a very important role in assessing the randomness quality of sequences produced by random number generators. And while these sequences constitute an essential input for applications requiring unpredictability, irreproducibility, uniform distribution or other specific properties of random number sequences, the desired quality of randomness may and do differ from one application domain to another. This is the reason why there are several statistical test suites that aim to quantify the quality of randomness provided by the tested generator and try to rule out generators that are not suitable for a given purpose.

In computing the degree of suitability, statistical tests rely on measuring certain properties of random number sequences in terms of probability, and based on the likely outcome of the tests applied on perfect random sequences can highlight possible deviations from randomness. Yet no finite set of statistical tests can be considered complete and consequently cannot ensure perfect randomness. In order to increase the confidence in the selected generator different statistical tests are applied on random sequences.

There are several well known batteries of statistical tests for random number generators such as the Diehard test suite [7] developed by Marsaglia, John Walker's ENT [20], TestU01 [6] designed by L'Ecuyer and Simard, or the well-known NIST test suite [11] developed by the National Institute of Standards and Technology as a result of a comprehensive theoretical and experimental analysis. The NIST test suite has become a standard stage in assessing the outcome of random number generators shortly after its publication.

Applying several batteries of statistical tests on large sequences of random numbers is a very time consuming operation; therefore, in order to satisfy the increasing demand for large volumes of random data, the development of high throughput randomness generators must be combined with high performance statistical test suites that take advantage of the new generation multi-core architectures.

Unfortunately the implementations of the most popular batteries of test suites are not focused on efficiency and high performance, do not benefit of the processing power offered by today's multi-core processors and tend to become bottlenecks in the processing of large volumes of data generated by various random number generators. Hence there is a stringent need for providing highly efficient statistical tests and our efforts on improving and parallelizing the NIST test suite [19] intend to fill this need.

The applied optimization steps to the original NIST statistical test suite include the paradigm shift towards byte processing mode, the possibility to process large volumes of input data (up to 1 GB) and the most significant improvement stage – the parallelization.

The result is a high performance parallel version, we called ParNIST, which was subject to a thorough benchmarking process and the results show a significant performance improvement, the execution time being reduced up to 103 times, and on average by approximately 54 times by the ParNIST system.

Table 1 presents the average speedup compared to the original version for each test improved by ParNIST.

Table 1

Benchmark results

Test No.	Test Name	Speed-up
1	Frequency (Monobit) Test	50
2	Frequency Test within a Block	54
3	Runs Test	35
4	Longest Run of Ones in a Block Test	50
7	Non-overlapping Template Matching Test	18
8	Overlapping Template Matching Test	103
10	Linear Complexity Test	30
11	Serial Test	64
12	Approximate Entropy Test	86
	Average Speedup	**54**

4.2. Visual Inspection of Random Number Sequences

The human visual system is highly trained for detecting patterns in the surrounding world, a valuable feature that can be applied in examining more abstract information as well.

The proposed FileSeer tool [10] is designed to take advantage of the enormous power and subtlety of the human visual perception and employ it for the visual inspection of the quality of random number sequences. To this end FileSeer provides a framework for representing the examined information as black and white, grayscale and color images which enable the spotting of repeated patterns and includes the visualization of certain statistical properties of the random sequence such as its balance, entropy and histogram.

In the process of testing random number sequences, the visual representations can help humans comprehend what randomness is and how it looks like by changing the process of understanding from being a cognitive task to being a perceptional task. At the same time it is important to note that by representing a sequence of numbers graphically, the human visual system is only capable of determining the degree to which the representation satisfies visual randomness but is unable to tell the difference between real randomness and visual paternlessness, hence the proposed inspection tool is designed to complete the wide range of statistical tests by providing an additional dimension to the understanding of randomness and of the usefulness of testing the quality of random sequences by assigning the abstract data a perceptual representation.

FileSeer offers the possibility to adjust the parameters of the inspection tool, in order to have a more significant view on the tested sequence or its statistical properties. These adjustments relate to the black and white, grayscale and color image aspect ratios, the maximum value that is plotted for the balance, entropy and histogram charts, and the number of samples and blocks the input sequence is divided into.

Test results for a quantum random number generator having visible repeating patterns are presented below. Figure 3 depicts the color image representation of the input sequence where the water ripple-like darker patterns deteriorate the otherwise random looking sequence.

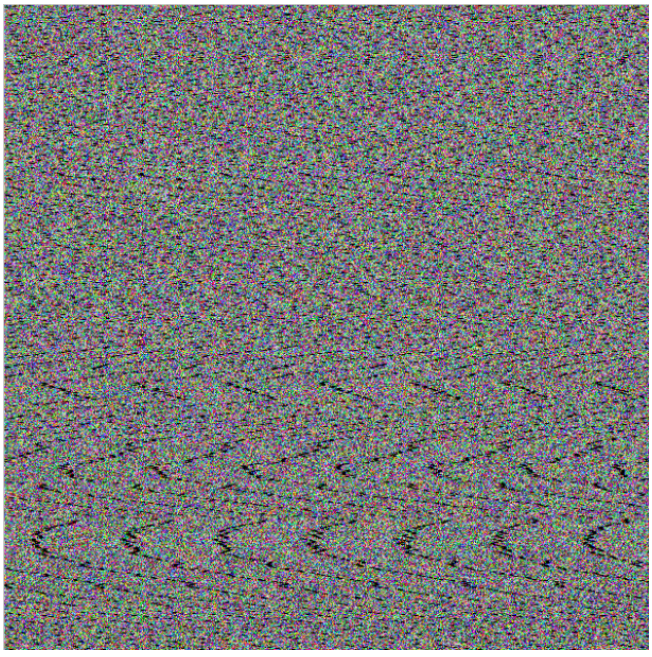

Fig. 3 – Color image representation of a sequence produced by a quantum random number generator.

The darker patterns indicate a periodically repeating bias of the generator towards 0 bits, as the 0/1 balance graphic also points this out in Fig. 4 where the balance values are computed and visualized for the above shown file considered as twenty continuous samples (above) and for each of the twenty blocks separately (below). The 0/1 ratio is constantly above 1, meaning that 0 bits appear more frequently in each of the samples and blocks.

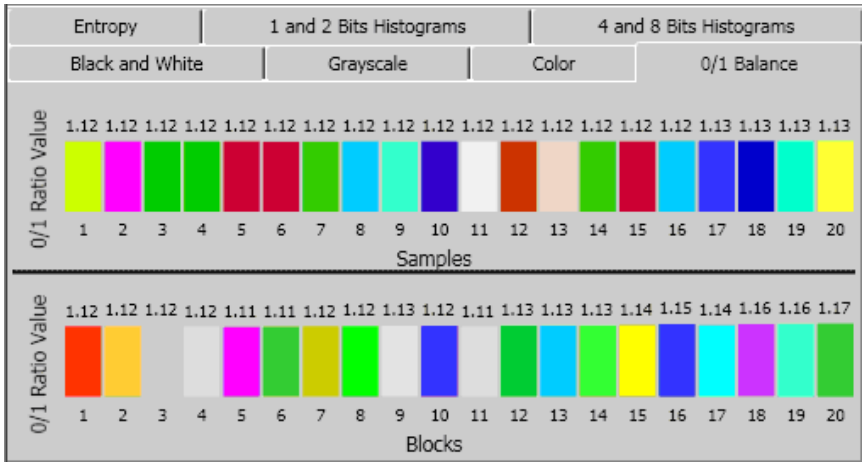

Fig. 4 – The 0/1 Balance computed for 20 continuous samples and for each of the 20 sample blocks individually.

5. CONCLUSIONS

This paper aims at emphasizing the major importance of random number sequences in digital cryptography and presents several practical approaches to random number generation and testing intended for cryptographic applications.

To this end, first a short description is provided regarding the many roles randomness plays in cryptographic techniques and protocols, each role emphasizing the high degree to which cryptography relies on randomness in providing the security requirements it is designed to deliver. The identified roles include:

- *cryptographic keys* which determine the transformation of plaintext into ciphertext and vice versa in both symmetric and asymmetric techniques,
- *initialization vectors* used in symmetric stream and block ciphers in order to ensure that the ciphers produce a unique output even under the same encryption key, thus avoiding the laborious work of rekeying,
- *nonces and challenges* used to ensure message freshness, principal liveness, mutual or unilateral authentication and demonstrations of knowledge of a secret while revealing no information about the secret,
- *cryptographic salt* used generally as one of the inputs for the key derivation functions,
- *padding strings* which in symmetric block ciphers fill the last block of plaintext in order to bring the message to a length multiple of the block size and in asymmetric techniques serves to turn deterministic public encryption schemes into randomized algorithms,
- *blinding factors* used in blind signature schemes for ensuring that the message signing process is performed without exposing the message content to the signer.

The many faces of randomness presented above also carry forth the significance of choosing and integrating suitable random number generators as security flaws in the generator can easily compromise the security of the whole system.

Cryptography requires high quality and high performance random number generators, but while true sources of randomness that rely on natural physical phenomenon can provide the highest level of entropy, other more practical sources of randomness in form of URNGs are receiving an increasing popularity due to the high availability, affordability, statistical quality and practical unpredictability and irreproducibility of the generated sequences. The presented unpredictable generation methods highlight the suitability of these approaches for cryptographic applications.

The last part of the paper focuses on the major importance of testing the outcome of random number generators in order to assess the generators suitability for specific applications.

In the context where applying several batteries of statistical tests on large sequences of random numbers is very important in order to increase the confidence in the selected generator but at the same time this process constitutes a very time consuming operation, the stringent need for providing highly efficient statistical tests is emphasized. An optimization method is presented that shows how the need for efficiently

implemented batteries of statistical tests that show high performance can be satisfied by improving the existing and well known statistical test suites to overcome their performance limitations.

REFERENCES

1. D. BONEH, *Twenty years of attacks on the RSA cryptosystems*, Notices of the American Mathematical Society (AMS), **46**, *2*, pp. 203–213, 1999.
2. D. CHAUM, *Blinding for unanticipated signatures*, Proceedings of EUROCRYPT'87, Advances in Cryptology, Springer-Verlag, pp. 227–233, 1987.
3. A. COLESA, R. TUDORAN, and S. BANESCU, *Software random number generation based on race conditions*, Proceedings of the 2008 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 439–444, 2008.
4. M. FRANKLIN, and M. YUNG, *The blinding of weak signatures*, Proceedings of EUROCRYPT'94, Advances in Cryptology, LNCS 950, Springer-Verlag, 1995, pp. 67–76.
5. Z. GUTTERMAN, B. PINKAS, and T. REINMAN, *Analysis of the Linux random number generator*, Cryptology ePrint Archive, Report 2006/086; Available online: http://eprint.iarc.org/2006/086.pdf
6. P. L'ECUYER, and R. SIMARD, *TestU01: A C library for empirical testing of random number generators*, ACM Transactions on Mathematical Software, **33**, *4*, Article 22, 2007.
7. G. MARSAGLIA. *The diehard test suite*, 2003; Available online: http://www.csis.hku.hk/ ~diehard/
8. K. MARTON, A. SUCIU, and I. IGNAT, *Randomness in digital cryptography: A survey*, Romanian Journal of Information Science and Technology – ROMJIST, **13**, 3, pp. 219–240, 2010.
9. A. J. MENEZES, P. C. VAN OORSCHOT, and S. A. VANSTONE, *Handbook of Applied Cryptography*, CRC Press, 1996
10. K. MARTON, I. NAGY, and A. SUCIU, *Visual Inspection of Random Number Sequences with FileSeer*, Automation, Computers, Applied Mathematics – ACAM, **19**, *1*, pp. 3–10, 2010.
11. A. RUKHIN, et al., *A statistical test suite for random and pseudorandom number generators for cryptographic applications*, NIST Special Publication 800–22, National Institute of Standards and Technology, revised May 2001.
12. B. SCHNEIER, *Applied cryptography: protocols, algorithms, and source code in C*, Second Edition, John Wiley & Sons, 1996.
13. A. SEZNEC, and N. SENDRIER, *Hardware Volatile Entropy Gathering and Expansion: generating unpredictable random number at user level*, INRIA Research Report, 2002; Available online: http://www.irisa.fr/caps/projects/hipsor/publications/havege-rr.pdf
14. A. SEZNEC, and N. SENDRIER, *HAVEGE: A user-level software heuristic for generating empirically strong random numbers*, ACM Trans. Model. Comput. Simul., **13**, *4*, pp.334–346, 2003.
15. A. SUCIU, S. BANESCU, and K. MARTON, *Unpredictable Random Number Generator Based on Hardware Performance Counters*, Proceedings of the International Conference on Digital Information Processing and Communications – ICDIPC 2011, Ostrava, pp. 123–137, 2011.
16. A. SUCIU, T. CAREAN, A. SEZNEC, and K. MARTON, *Parallel HAVEGE*, Proceedings of the 8th International Conference on Parallel Processing and Applied Mathematics (PPAM 2009), Wroclaw, 2009, pp. 145–154.
17. A. SUCIU, K. MARTON, AND Z. ANTAL, *Data flow entropy collector*, Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, Workshop on Generation, Testing and Applications of Random Number Sequences, 2008, pp. 445–448.
18. A. SUCIU, K. MARTON, E. CEBUC, V. T. DADARLAT, and G. SEBESTYEN, *Gathering Entropy from the Grid with GridHAVEGE*, Proceedings of the IEEE International Conference on Intelligent Computer Communication and Processing (ICCP 2010), Cluj-Napoca, 2010, pp. 459–463.
19. A. SUCIU, I. NAGY., K. MARTON, and I. PINCA, *Parallel Implementation of the NIST Statistical Test Suite*, Proceedings of the IEEE International Conference on Intelligent Computer Communication and Processing – ICCP 2010, Cluj-Napoca, pp. 363–368, 2010.
20. J. WALKER, *ENT – A pseudorandom number sequence test program*, Fourmilab, Revised in 2008; Available online: http://www.fourmilab.ch/random/.

Received March 13, 2012