Determining the Secrecy of Software Encryption System based on pGSSG Generator using Shannon Entropy

Antoniya Tasheva, Ognyan Nakov, Boyan Petrov
Technical University of Sofia, Bulgaria
8 Kliment Ohridski blvd., Sofia 1000, Bulgaria
atasheva@tu-sofia.bg, nakov@tu-sofia.bg, b_petrov@tu-sofia.bg

ABSTRACT

This paper focuses on the application of previously proposed *p*-ary Generalized Self-Shrinking Generator (pGSSG). Software encryption system using keystream produced by the pGSSG is developed and analyzed. For initialization of the generator Galois Field GF(257^{32}) and feedback polynomials are chosen. A mathematical model of the software encryption system is made. It is proved that it is not perfect but the empirical tests with calculation of entropy for input and encrypted files show that it has less than 0.0125% deviation from the perfect secrecy.

KEYWORDS

PRNG, pGSSG, Security, Entropy, Encryption, Stream Cipher.

1 INTRODUCTION

Recently, computer technologies have started to play a huge role in everyday life as well as at the workplace. As the Internet gains more and more popularity and becomes a major means of communication, the term information security becomes more and more important. Encryption has been studied for centuries and the need to find new and better solutions is still present to date.

When transmitting large amounts of data over communication channels such as mobile and wireless networks, and when high speed, low error propagation and resistance to attacks are needed, the use of stream ciphers is recommended. They encrypt each symbol of the transmitted message with a keystream, which is usually generated by a Pseudo Random Number Generator (PRNG), producing binary Pseudo Random Sequences (PRSs).

The elements of PRNGs that are most used in stream ciphers are Linear Feedback Shift Registers (LFSRs), because LFSR of length *n* can generate a PRS of maximum length $T = 2^n - 1$.

Since 1969, when the Berlekamp-Massey algorithm[8] was discovered, scientific researchers are looking for new methods of generating non-linear sequences. Researchers use two basic methods to generate nonlinear sequences [4]: structures based on LFSR registers, such as filter generators, combinatorial generators and clock controlled generators, and generators in finite fields such as GMW (Gordon, Mills and Welch) sequences and Bent function sequences.

Recently some clock controlled generators which use a *p*-ary PRS [3] to create nonlinear sequences have been proposed [6, 14, 15, 16]. They summarize the work of the Shrinking Generator proposed by D. Coppersmith, H. Krawczyk and Y. Mansour at Eurocrypt'93 [2], and the self-shrinking generator (SSG) proposed by W. Meier and O. Staffelbach at Eurocrypt'94 in [7].

Such generator that uses LFSR in order to produce its output PRSs is the recently proposed *p*-ary Generalized Self-Shrinking Generator [17]. It is proved that it has long period, is well balanced, has good statistical characteristics and is resistant against exhaustive search and entropy attacks.

As most of the properties of the sequences generated by pGSSG generator have been studied and proved to give good results, it was decided to build a software encryption system based on it. Its encryption properties are tested using the entropy measure [1, 5], which is the matter of this paper. An entropy measure is usually defined in terms of probability distribution. The entropy $H(X)$ of a random variable X is a measure of its average uncertainty. It is the minimum number of bits

required on the average to describe the value x of the random variable X [1]. In this study the entropy of the pGSSG PRSs is considered as well as their influence on the symbol distribution in the source and encrypted files.

The paper is organized as follows. First, the working algorithm of p-ary Generalized Self-Shrinking Generator for Galois Field GF(p^n) is given. Then the architecture of the pGSSG software encryption system is described. In Section 4 the mathematical model of the system is designed and the entropy is both evaluated and calculated.

2 ALGORITHM OVERVIEW

The proposed p-ary Generalized Self-Shrinking Generator [17], given in Figure 1, consists of a pLFSR register A, whose length will be denoted by L. It generates a sequence $(a_i)_{i\geq 0}$ with p-ary digits (i.e. $(a_i)_{i\geq 0}, 0 \leq a_i \leq p-1$) and $0 \leq i \leq L-1$. The multipliers of the feedbacks are given by coefficients $q_1, q_2, ..., q_L$, $q_L \in [0, 1, ..., p-1]$ of the primitive polynomial in GF(p^L). Every element can remember one p-ary number. The register is initialized by p-ary sequence $(a_0, a_1, \ldots, a_{L-1})$.

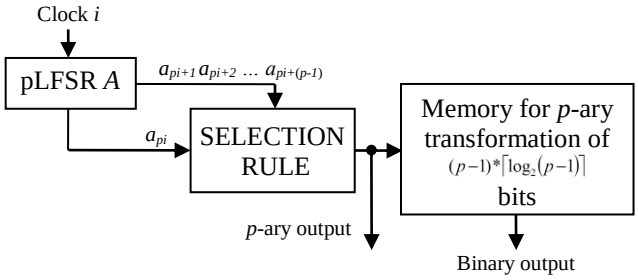


Figure 1. *p*-ary Generalized Self-Shrinking Generator

The pGSSG selects a portion of the output p-ary LFSR sequence by controlling the p-ary LFSR itself using a six-step algorithm. The p-ary LFSR is clocked with period T and its output sequence is split into p-tuples. Depending on the first element in each p-tuple, it is either discarded or the element with number equal to the value of that first element is output. The p-ary output sequence could be transformed into a binary one using a special scheme for substitution for the zero and non-zero elements.

3 SOFTWARE SYSTEM

A software encryption system has been built. Its main task is to encrypt the transmitted data in advance using a keystream, generated by the output of *pGSSG* generator. When needed the encrypted data could be put under second encryption with standard methods, such as DES in CBC mode or AES in CCM mode, which are used in the contemporary wireless networks.

The software encryption system is based on symmetric stream encryption algorithm which is initialized by the value of the secret key K. It encrypts the input data stream (the plain text) as simple XOR operation on each byte of the plain text and the keystream received from the pGSSG generator (see Fig. 1).

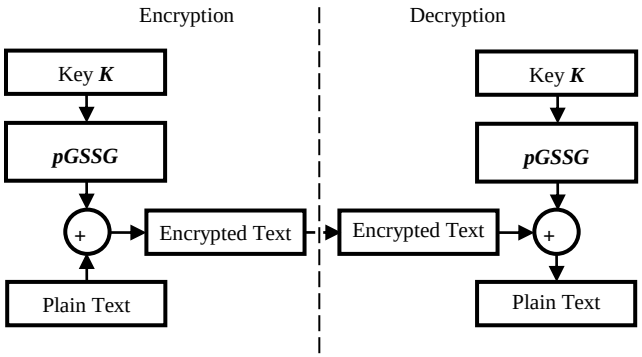


Figure 2. *Architecture of the Software Encryption System based on pGSSG Generator.*

The software encryption system uses class libraries for software representation of the LFSR register and the pGSSG generator. Although they are designed to be universal, Galois Field GF(257^{32}) was chosen for the implementation because of the ease of byte representation and therefore the possibility of faster implementation. Three of the primitive feedback polynomials used to create a pLFSR register with prime p = 257 and length L = 32 are shown in Table 1.

Table 1. Feedback polynomials used in pGSSG.

№	Feedback Polynomial
1	$x^{32} + x + 10$
2	$x^{32} + 75\,x^2 + 174\,x + 33$
3	$x^{32} + 188\,x^2 + 200\,x + 107$

The result of the encryption with that software system is that the amount of data remains the same before and after sending it into the communication channel. If the communication network provides the option for additional data encryption, it is applied as second level of encryption through a standard block cipher AES or DES. For example in WiMAX WLAN networks where data encryption is mandatory the confidentiality and the security will increase using two levels of encryption (Fig. 3).

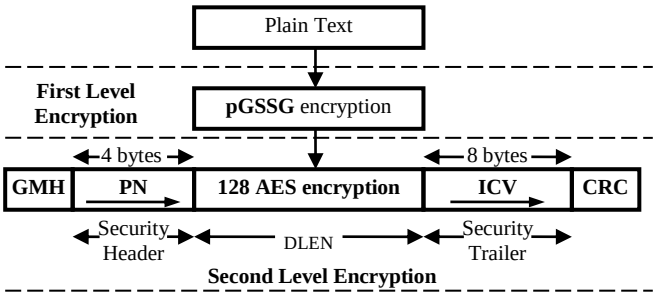


Figure 3. *Use of pGSSG Software Encryption System in WiMAX WLAN Networks.*

4 EXAMINING THE SOFTWARE SYSTEM SECRECY

Determining the theoretical secrecy of a cryptographic system is a very complex mathematical task. The use of different extended Galois Fields GF(257^L) makes the mathematical cryptanalysis of the pGSSG generator more difficult. To see if this system is usable, many questions need to be answered. They are related to its robustness and security when the attacker is not limited in time and has access to all possible means to analyze encrypted messages. Another question is could a single solution be found and what amount of data should be intercepted to get this solution.

Due to the nonlinearity of modern encryption algorithms, a comprehensive fully mathematically justified answer could not be given. However, the entropy, suggested by Claude Shannon in "A mathematical theory of communication" [12], has found wide application in analyzing these issues since 1948.

4.1 Mathematical Model

As it can be seen on figure 3, the pGSSG based software encryption system ciphers the entire plain text: both the data and the headers of the files. If we consider the byte organization of stored data, the mathematical model of the pGSSG encryption system can be presented as follows.

The system (see Fig. 4) would work with 256 (1 byte) different symbols a_0, a_1, …, a_i, …, a_{255} with corresponding probability for appearance $P(a_i)$,

i = 0, 1, ..., 255. As a result of the encryption, these symbols are mapped into cryptograms b_0, b_1, …, b_j, …, b_{255} with probability for appearance $P(b_j)$, j = 0, 1, ..., 255. The keys K_0, K_1, ..., K_n are equally possible and their maximum count N_K depends on the key length. It is possible that one input symbol is converted to a different output symbol when using different keys.

The required and sufficient condition for the system to be completely secret [12, 13] is:

$$P(b_j) = P\,(b_j / a_i),\ j = 0, 1, ..., 255. \qquad (1)$$

i.e. $P\ (b_j\ /\ a_i)$ should not depend on the input symbol a_i.

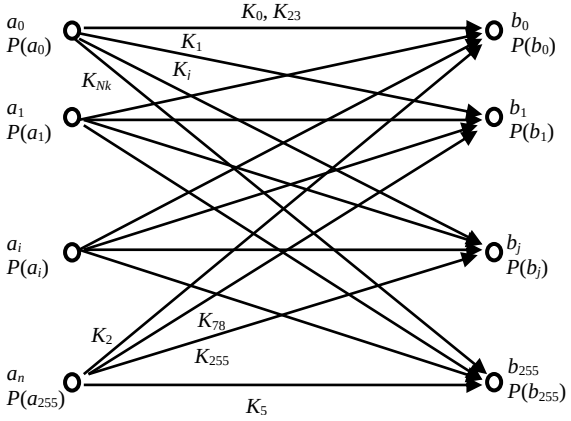


Figure 4. *Mathematical model of the pGSSG software encryption system.*

Here $P(b_j/a_i)$ is the conditional probability of the encrypted symbol b_j, provided that the input symbol is a_i. That means that it is the sum of the possibilities of all keys that transform the symbol a_i into b_j.

It is known that the perfect encryption system is achieved when the following three conditions are true [13]:

1. Each message is associated with only one cryptogram.
2. The number of keys is equal to number of messages.
3. All keys are equally possible.

Under these conditions, the entropy H of the system is

$$H(A) = -\sum_{i=0}^{n-1} P(a_i)\log_2 P(a_i) = -\sum_{i=0}^{n-1} \frac{1}{n}\log_2 \frac{1}{n} = \log_2 n \quad (2)$$

4.2 Experiments using Shannon Entropy

As evident from the mathematical model of the pGSSG software encryption system, it is not absolutely secret because it does not fulfill conditions 1 and 2. Studies are made to answer the question how close the proposed system is to the perfect case scenario in (2). For this purpose over 100 different files are tested. They are distributed equally in the main types: text documents (.doc, .docx, .txt), images (.bmp, .jpg, .png), executable files (.exe), audio files (.wav, .mp3) and archives (.zip, .rar). The frequencies of occurrence of all characters in input and encrypted files are studied and their entropy is calculated. Furthermore, for image files their three color components *R*, *G* and *B* were analyzed separately. In Table 2 the Shannon entropy of the input and output files for four files from each group are shown. Figure 5 demonstrates the distribution histograms of the plain text and encrypted text for different types of files. The results for the three color components *R*, *G* and *B* of the images are respectively shown in Table 3 and Figure 6.

More than 100 sequences of length 1 000 000 bits were generated for each password in order to check how the password transforms into keystream. They were then tested via NIST [9, 11] test suite to obtain their properties.

It is known that the crypto-analytics can use the existing dependencies in the occurrence of characters in various types of information and the model of the standard headers in different file types. That information can help them decrypt the data. Thus they have blocks of plain text and when capturing the encrypted message and they can map it to the corresponding encrypted text. The use of additional level of encryption with the software encryption system eliminates these possibilities as it uniformly changes the values of the symbols for the whole length of the file, including the header part. This conclusion is confirmed by the results shown in Tables 2 and 3.

Table 2. Entropy of input and pGSSG encrypted files.

№	File	Password	Entropy of	
			Input File	Encrypted File
1.	explorer.exe	2345explorer	5.872596706508290	7.999598134607586
2.	encryptPhD.exe	Fast2Enc%(!	7.631900643959939	7.998833667709484
3.	notepad++.exe	++нотпад++	6.091405544356415	7.999648675667926
4.	WinRAR.exe	КоМпReСиQ	6.469161170026363	7.999644709803738
5.	CompleteSet.docx	2013ВСРС	7.948642291913700	7.998595024152175
6.	DiplomnaRabota.docx	Info4диплом	7.524045264552306	7.989391747378668
7.	document.docx	ІІросто*Tekst	7.910834642297916	7.998354864955056
8.	TrainingTasks.docx	Passw0rd	7.948642291913700	7.998405935750952
9.	presentation.pptx	Imagine8№	7.937660319385949	7.999896904729666
10.	IFRToolLog.txt	лог$a@к§n&	4.714971677469938	7.999566854515114
11.	leMouse.rar	el^gAto-te	7.986186549835893	7.998523780587349
12.	Raboti.rar	lANg25cop18%	7.999329046994613	7.999986138240808
13.	Zadaniq.rar	worry35ePIc^	7.999026651474129	7.999558585297283
14.	Zadaniq.zip	rAts24mOd=	7.997750400646980	7.999661066662499
15.	ALARM.WAV	VarY39eRns&	6.871613007127728	7.997961560964973
16.	cat8.wav	NiX^pEn=Dry1	4.572460198612183	7.998654459803242
17.	TU.bmp	myUniversit1	7.735134585817764	7.999598750116132
18.	lenna.bmp	vAN83scoUR@	5.682224748742369	7.999696018375432
19.	snowman.jpg	faLL!kOR?jiltS	7.927340915095799	7.995697933926553
20.	sozopol.jpg	idS?OcHRy65	7.966401975045295	7.997800281093063

Analysis shows that the entropy of encrypted files slightly differs from the perfect encryption system entropy with less than 0,001 bit which is *0,0125 % deviation from the perfect secrecy*. The entropy of a perfect secrecy system with 256 equally possible symbols is:

$$H(A) = -\sum_{i=0}^{255} \frac{1}{256}\log_2 2^{-8} = 8 \text{ bit.} \quad (3)$$

As seen on Figure 4.5 the distribution of symbols in most file types differs radically from the uniform distribution. Exceptions are the archive files whose distribution is largely similar to the uniform and this is no coincidence, because in the

archives different algorithms for compressing data are applied. This feature of the archives determines their entropy, which depending on the compression algorithm can reach up to 7.998. However in the distribution of the symbols there are usually detectable peaks of the symbols having a value of 0 or 255 (see Figure 5.c). These anomalies in the histograms are eliminated by the use of encryption with the pGSSG generator.

5 CONCLUSIONS AND FUTURE WORK

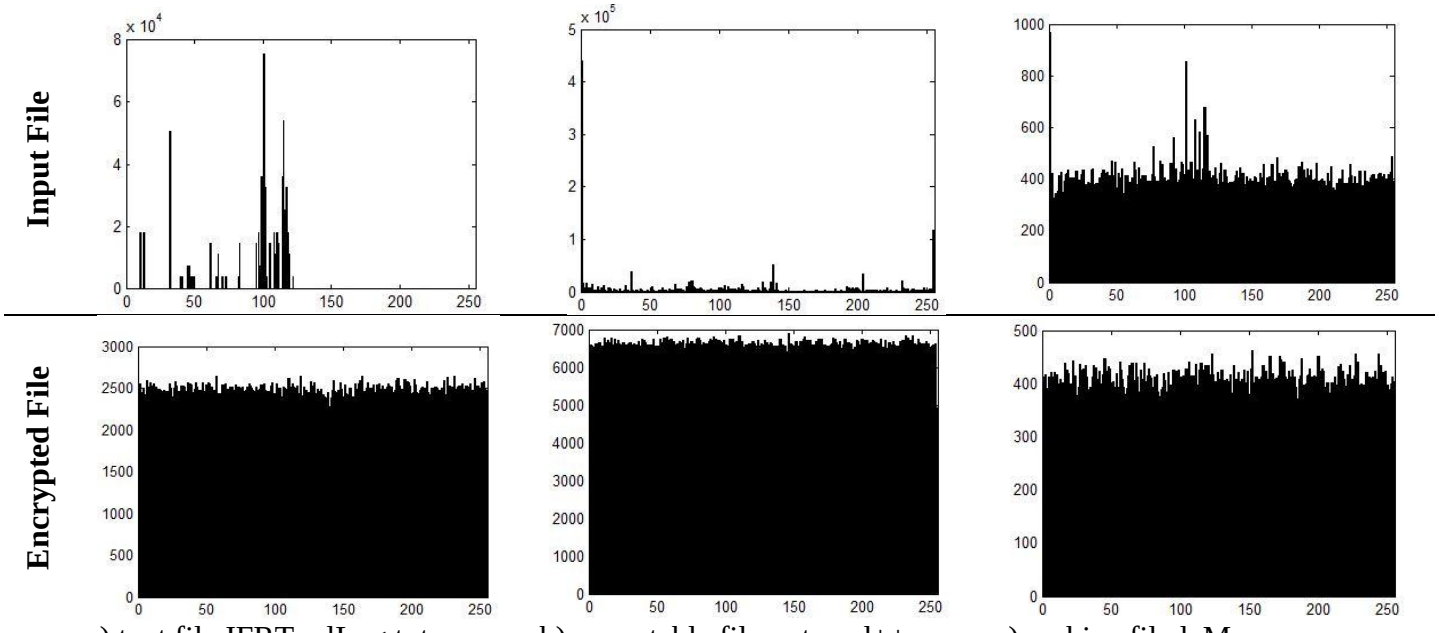
The secrecy of the software pGSSG encryption system is tested with the aim of its mathematical model using the term Entropy. It is proved that the system does not have perfect secrecy but transforms the data into such with uniform distribution of characters. The analysis shows that the entropy of encrypted files compared to the perfect encryption system differs with less than 0,001 bits, which is 0,0125 % deviation from the perfect secrecy.

The task of decrypting the received data has been made more complicated for the crypto-analytics when using known dependencies in the occurrence of characters in different types of information.

However, there are some practical issues that need to be addressed. First, to measure the degree of randomness of sequences generated by pGSSG some statistic experiments using approximate entropy [10] must be done. Second, to analyze the problem of finding the secret key in pGSSG software system min-entropy can be used [1], which determines the probability of guessing the correct value at first attempt. Moreover, it is necessary to find the average number of guesses needed to determine the key, which is given by the guessing entropy [1].

Table 3. Entropy of the colour components *R, G* and *B* of the input and pGSSG encrypted images.

№	Image	Password	Input Image			Encrypted Image		
			R	***G***	***B***	***R***	***G***	***B***
1.	medal.bmp	*&^%^%sG	6.0038	5.9726	6.0043	7.99928	7.99943	7.99937
2.	banner.bmp	*&*B”}@E	7.3162	6.9769	6.8664	7.99977	7.99973	7.99974
3.	DarkDoor.bmp	Poiuytr^B&	7.2023	7.0943	7.0060	7.99934	7.99935	7.99924
4.	pepper.bmp	M*b%V)JY	7.3388	7.4962	7.0583	7.99930	7.99924	7.99935
5.	lenna.bmp	12345678	5.0465	5.4576	4.8001	7.99914	7.99918	7.99904



a) text file IFRToolLog.txt b) executable file notepad++.exe c) archive file leMouse.rar

Figure 5. Distribution histograms for symbols in input and pGSSG encrypted files: a) text files, b) executable file, c) archive file.

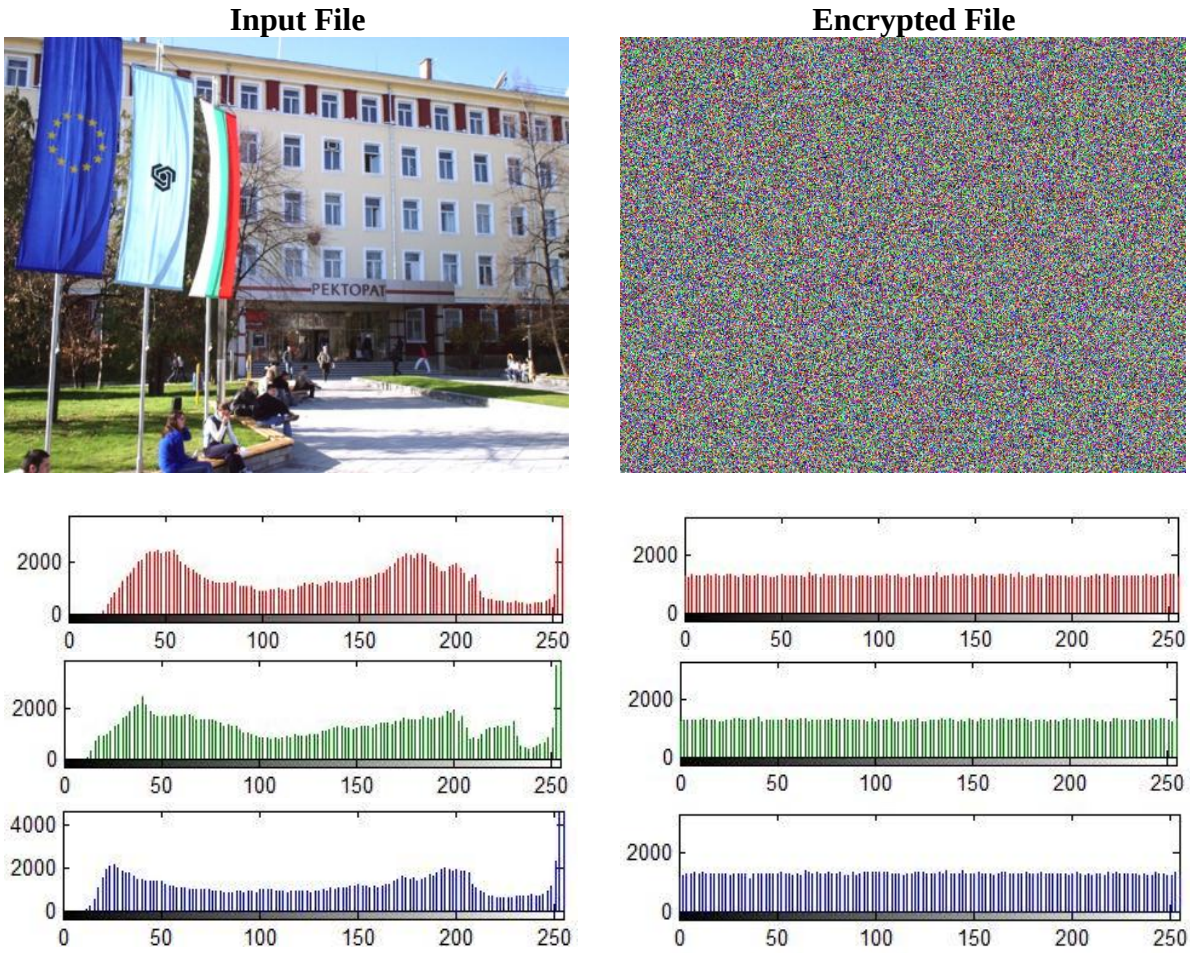


Figure 6. Input and encrypted image and the corresponding distribution histograms for the R, G and B values.

7 REFERENCES

[1] Cachin, C. *Entropy measures and unconditional security in cryptography*. Konstanz: Hartung-Gorre, 1997.

[2] Coppersmith D., H. Krawczyk, Y. Mansour, The shrinking generator, *Advances in Cryptology – EUROCRYPT'93*, vol.773 of LNCS, Berlin, Springer-Verlag, 1993, pp. 22-39.

[3] Golomb S., *Shift Register Sequences*, Aegean Park Press, Laguna Hills, Calif, USA, revised edition, 1982.

[4] Gong G., Sequence Analysis, University of Waterloo, 1999, p. 137, Avaiable from: http://calliope.uwaterloo.ca/~ggong.

[5] Gray R., *Entropy and Information Theory*, Springer, Second Edition, 2011.

[6] Kanso, A. *Clock-controlled generators*. University of London, 1999.

[7] Meier W., O. Staffelbach, The self-shrinking generator. In A.De Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, vol.950 of LNCS, Berlin, Springer-Verlag, 1995, pp. 205-214.

[8] Massey J., Shift-register synthesis and BCH decoding, *IEEE Transactions on Information Theory*, vol. 15, no. 1, 1969, pp. 122–127.

[9] NIST Statistical Test Suite, Version 2.1.1., August 11, 2010, http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html

[10] Pincus S., B. H. Singer, Randomness and degrees of irregularity, *Proc. Natl. Acad. Sci. USA*. Vol. 93, March 1996, pp. 2083-2088

[11] Rukhin A., J. Soto, et.al, NIST Special Publication 800-22rev1a: A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications, April 2010, Avaiable from: http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf.

[12] Shannon, C. E. "A mathematical theory of communication." *ACM SIGMOBILE Mobile Computing and Communications Review* 5, no. 1, 2001, pp. 3-55

[13] Shannon, C. E. "Communication theory of secrecy systems." *Bell system technical journal* 28, no. 4 (1949), pp. 656-715.

[14] Tashev, T., Bedzhev, B., Tasheva, Zh. The Generalized Shrinking-Multiplexing Generator, ACM International Conference Proceeding Series 285, Article number 48, *Proceedings of the 2007 international conference on Computer systems and technologies CompSysTech '07*, 2007.

[15] Tasheva, Z., Bedzhev, B., Stoyanov, B. P-adic shrinking-multiplexing generator, *Proceedings of the Third Workshop - 2005 IEEE Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2005*, 2005, pp. 443 – 448.

[16] Tasheva Zh. N. Design and Analysis of 3-ary Generalized Shrinking Multiplexing Generator, *International Journal of Advance in Communication Engineering 4 (2)*, 2012, pp. 129-140.

[17] Tasheva A. T., Tasheva, Zh. N., Milev, A. P., Generalization of the Self-Shrinking Generator in the Galois Field GF(p^n), *Advances in Artificial Intelligence*, vol. 2011, Article ID 464971, 10 pages, 2011. doi:10.1155/2011/464971