Journal of Engineering Science and Technology
Vol. 14, No. 6 (2019) 3372 - 3386
© School of Engineering, Taylor's University

DESIGN AND IMPLEMENTATION OF A KEY GENERATOR-BASED STREAM CIPHER FOR SECURING TEXT DATA

BASHEER H. ALI[1,*], MOHAMMED J. ZAITE[2], ABDULLAH S. AL-HASHIMI[3]

[1]College of Engineering, Department of Computer Engineering, AL-Iraqia University, Baghdad- Iraq
[2,3]Engineering Technical College, Department of Computer Technique Engineering, Middle Technical University, Baghdad- Iraq
*Corresponding Author: basheer.husham@aliraqi.edu.iq

Abstract

With the evolution of communication systems and information technology, information that is transmitted over networks have become more valuable. This leads to catch up the attention of attackers and researchers to focus in this area. In this paper, a strong key generator method for securing data based on stream cipher were designed and constructed. A stream cipher technique, which considered the best cryptography modern methods, was used in the implementation. A stream cipher is better than other methods such as block cipher and very suitable for securing data. Its' hardware is not complex, and its' encryption speed is very high. However, this technique is vulnerable for attacks especially when its' key is used twice. A new form of several types of generators was used and combined together to generate a random key that is hard to be predicted by attackers. This was our contribution to this paper. These types are Linear Feedback Shift Registers (LFSRs), Non-Linear Feedback Shift Registers (NLFSRs), and Feedback with Carry Shift Registers (FCSRs). Then, they are combined by using a nonlinear Boolean function, which makes the generator more secure. In addition, the performance and behaviour of our design were evaluated using the National Institute of Standards and Technology (NIST). Finally, the p-value results of the majority of tests used in our experiment were over than 0.01, which is the limited point that considers a generated key as random.

Keywords: Key generators, NIST tests, PRNG, Stream cipher, Vulnerabilities.

1. Introduction

With the development of communications systems and the transition of information among people and/or companies, this leads to increase the risks of falling these data in hand of attackers. In the field of telecommunications devices, smartphones and tablets have become more affected by these problems. For example, Man-in-the-Middle (MITM) is a type of attack that threatens the sensitive data that are transmitted among legitimate users. In this type of attack, attackers can control the whole communication between users. They can deceive victims by intercepting or eavesdropping their messages while in fact they are sending and receiving their messages through attackers [1]. Attackers can also intercept users' texts, files, images, or videos that are sent from their devices. Therefore, data security has taken seriously in order to maintain confidentiality, integrity, and availability.

To avoid these problems, researchers proposed several solutions based on cryptography methods. Symmetric cryptography considers a good method that used for encrypting the users' data. In this method, the same key is used for encryption and decryption process [2, 3]. It consists of two types, which are block cipher and stream cipher. Although block cipher is used a lot in the field of computer communication encryption, a stream cipher is considered more suitable for data encryption in smartphones for many reasons. First, the process of encryption in stream cipher is very fast. Secondly, stream cipher does not need a large amount of hardware [4-6]. These two reasons are suitable for the environment of a mobile phone. Therefore, a stream cipher is considered as a good option.

However, stream cipher has several drawbacks, which make it vulnerable to several kinds of attacks. There are many kinds of attacks, which can exploit these vulnerabilities. Exhaustive search attack, algebraic attack, correlation attack, fault attack, distinguishing attack, chosen IV-attack, slide attack, cube attack, time-memory trade-off attack, and guess and determine attack are all types of attacks that are targeted stream cipher. Discovering time, memory space, and data for stream cipher by attackers is one reason. Another reason is low degree polynomial representation function for the algorithm. The most critical one is that the key to a stream cipher is used twice [7]. It must be random to prevent attackers from identifying it easily. There are several methods to generate a random key. The most famous way is a pseudo-random number generator (PRNG). The specification of PRNG gives a high-performance speed, which generates random sequences and unpredictable [3]. There are multi algorithms of PRNG are used in the stream cipher like [3, 8-10]. Some algorithms have proved to be very effective and the others were insecure and weak against attacks [10]. In this paper, strong generators were chosen to implement a new form of a generator. These types are Linear Feedback Shift Registers (LFSRs), Feedback with Carry Shift Registers (FCSRs), and Non-Linear Feedback Shift Registers (NLFSRs). Finally, these types are combined into the non-linear combination generator to create a new key generator.

The remainder of the paper is organized as the following. Section 2 was specified for related work. Section 3 describes briefly the background theoretical about the types of generators that are used in our design. The proposed design was explained in Section 4. Then, the performance was evaluated by using NIST

tests in the same section. Then, Section 5 summarized this paper. Finally, we ended up with future work and references.

2. Related work

Key generators based-stream cipher is used in several different platforms for securing data for organizations, companies, and/or governments. First, Ashouri [11] designed a new algorithm that is based on the clock-controlled combination generator with memory. It was evaluated to resist most known attacks. It also gave a random keystream, enormous period, and huge linear complexity. Due to these excellent results, the designer suggested using it in many applications and chiefly for encryption the financial data.

In addition, Tasheva [12] suggested GSMG generator that increases the linear complexity (LC), which considers one of the main important criteria for designing a generator. The advantages of the GSMG are that produce a large period, huge LC, good randomness properties. However, the nonlinearity of this method cannot be detected. This drawback led to decrease the speed of generation, but it was overcome by using a buffer [12]. Another paper proposed a new generator called (5-MGG). The researcher of this paper suggested developing the Geffe generator by increasing the number of LFSR from three to five generators. Moreover, a new combination nonlinear function to produce a perfect statistical test was used in the implementation. However, it is not recommended for using 5-MGG because of its failure of the randomness tests, which make the generator insecure and inappropriate in cryptography [13].

Moreover, Marghescu et al. [14] designed a key generator based on the smartphones sensors devices for Android systems. They constructed it by using various sensors, which perform to generate a key as a form of True Random Number Generator (TRNG). Thereafter, they evaluated it by using statistical tests that called a priory. The results of this design showed a great random number generator so they improve the nature of the generator. However, we saw in this method was not suitable because the process to get the same key for both parties (sender and receiver) not possible by using TRNG method.

Furthermore, Garcia-Bosque et al. [15] in 2016 proposed two different generators for the stream cipher, which they used in the field of communication systems for securing data. The first was named Skew Tent Map (STM), and the other was called Modified Logistic Map (MLM). The two-design were based on a chaotic map, which has many advantages such as simplicity in implementing, High productivity, good properties. However, when they tested the randomness by using NIST tests, the results were very bad. Thus, the researchers were adding LFSR on the chaotic map to increase the period to get rid of the poor in randomness, which was existed in STM and MLM. After that, the results were better than using single STM and MLM (p-values were more than 0.01). Finally, the last two generators were recommended in the applications, which require high-speed encryption in addition to a high level of security because it was very suitable for them.

In addition, Wang, et al. [16] in 2016 presented a novel pseudo-random number generator (PRNG) based on the piecewise logistic map (PLM). They used PLM for overcoming the problems were found in the logistic map. Where the

properties showed the PLM was much better than the logistic map. Therefore, the PLM was very important and helpful for using with the PRNG. The advantages of this design, the randomness of the keystream were passed the entire statistical test (NIST tests). In addition, the correlations between the sequences were very high, which considered good. All the improvements shown in this paper proved that the generator is efficient, safe and easy to construct, so they suggested that the generator is suitable for utilizing in the stream cipher method.

Moreover, Mandal et al. [17] in 2016 presented a new lightweight design (Warbler Family), which is used for the provision of security services for different intelligent devices such as smartphones. The design was based on a new PRNG where was generated by using NLFSR, Boolean function, and Non-Linear Feedback Welch-Gong Generator. Al-Khatib and Lone [18] in 2018 proposed a key-generator by using cryptographically secure PRNG in another paper, Acoustic Lightweight Pseudo-Random Number Generator (ALPRNG). They based on LFSR to construct their generator because it has many good properties such as lightness and fast in implementing, generate long sequences, consume less RAM, and take less size in CPU.

Finally, in 2016, Jallouli et al. [2] designed a novel key generator that is based on the chaotic system. They employed this method to encrypt images by using pseudo chaotic number generator (PCNG) that generates a strong keystream. The proposed generator evaluated by NIST tests and it passed most of them. Thus, PCNG considered being an excellent random sequence generator. Finally, the generator proved to be resisted against most attacks.

3. Background Theoretical

As mentioned, stream cipher depended on PRNG to generate a strong key. There are many types of PRNG that are used in a stream cipher. One of them called LFSR. It is considered the most common type in PRNG. It consists of two parts, which are shift register and feedback function as shown in Fig. 1. A shift register is a group of stages (sometimes called states) that used for storing a one-bit (1 or 0). The initial state values for the register are called a seed. These stages are shifted together from left to right by the outer clock. The new bit of the register is computed by the feedback function where it uses XOR function through connected some of the bits by this function (these bits are known the tap sequence) [10].

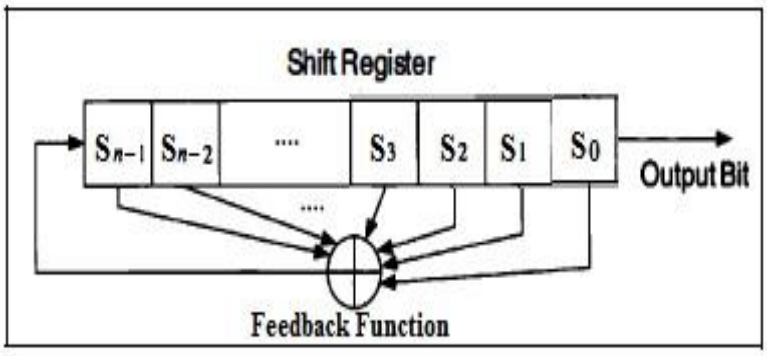


Fig. 1. Configuration of LFSR [10].

LFSR has a speed of generation, and it is suitable for implementation in software as well as in hardware. It also gives a sequence with good distribution properties [8].

The second type of PRNG is NLFSR. It is used instead of LFSR because of the weakness of the linearity in LFSR [2]. It has not found any mathematical theory to analyse them until now, which makes NLFSR the most secure, powerful, and reliable way to design the keystream [3]. It differs from the LFSR by containing an additional function. It names the non-linear function as shown in Fig. 2. However, NLFSR has two disadvantages. Firstly, it does not always generate a sequence of bits that it equals the maximal period. Second, The period for the NLFSR (n, k) does not achieve the first and second postulates of Golomb. To address this problem, the Fibonacci configuration type can be used [19, 20].

The third type is the FCSR. It is like LFSR, but it contains a carry register (additional memory) as shown in Fig. 3. Moreover, FCSR uses a new function, which is named a summation of integers where it performs to compute the values of active taps and memory. The result will be an integer number. It denotes a parity bit (σ). Then, it can calculate the new bit by taking mod two of the parity bit (σ mod 2). The new memory also is calculated by divide the σ over two with taking the integer part of the result ($\lfloor \sigma/2 \rfloor$), where the $\lfloor\ \rfloor$ is the greatest integer [10, 20].

FCSR still a new modern generator [10]. It characterized by the high-speed generation, effectively implemented in software and hardware, and producing huge periods with perfection statistical properties [20].

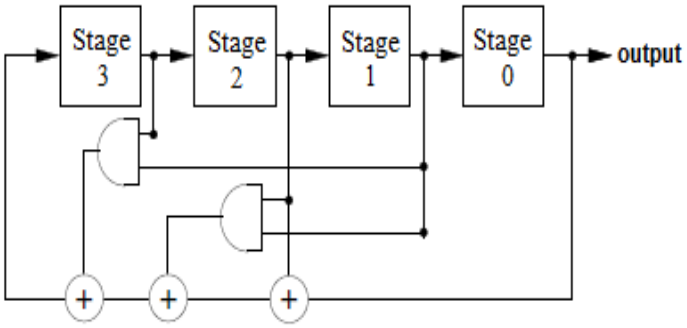


Fig. 2. Example of Fibonacci NLFSR [21].

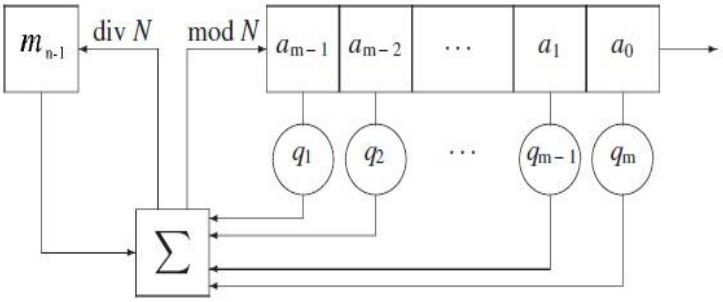


Fig. 3. Configuration of FCSR [20].

4. Proposed Generator

In this section, the flow charts of stages of encryption/ decryption by using a stream cipher technique were presented. The design of the proposed key generator was also described in detail. Then, the experimental results for the proposed generator were displayed by using the NIST statistical tests.

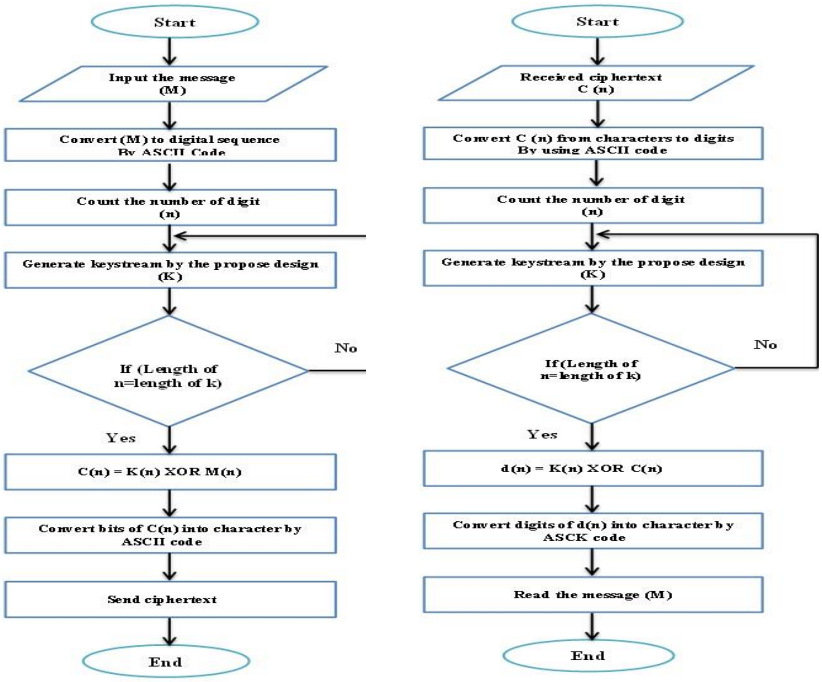
4.1. Flow charts and structure of proposed generator

Flow charts of stages of encryption/ decryption are depicted in Fig. 4. The stages of encryption/ decryption by using the stream cipher technique were implemented through two flow charts; Figs. 4(a) and (b). Figure 4(a) represents the process of encryption, which will be implemented in our project. Many steps were illustrated to complete this operation. Firstly, the message/or plaintext was entered. Then, the original message was converted into digital form (bits o or 1) by using the ASCII code. After that, the numbers of bits were counted in order to prepare the initialization of encryption. Secondly, the key was generated by using our design, which will use with the plaintext to produce ciphertext. The key generator must be equal to the length of the message. Therefore, if a statement was put to avoid this problem. Thirdly, if the number of bits for k (n) was equal the number of bits for M (n) then, the method of stream cipher was completed by using XOR process between key and message to produce ciphertext. After that, the number of bits for ciphertext was converted from bits form to a character (by using ASCII code) to become the message text incomprehensible. Finally, the message was encrypted and it will be sent to the receiving party.

Figure 4(b) illustrates the process of decryption, which will be implemented in our project. In addition, many steps were demonstrated to complete this process. Firstly, ciphertext was received from the sender part as a form of character, which will be converted to a number of bits by using ASCII code. Then, the number of bits was counted in order to generate a key, which equals the length of the ciphertext. Secondly, keystream was generated by using our design, which will use with the ciphertext to produce the plaintext. If the statement was put to make the length of key equal the length of the ciphertext. Finally, the process of decryption was completed by using the XOR process between $K(n)$ and $C(n)$ to produce $d(n)$. After that, the number of bits for decryption was converted to the character (by using ASCII code) to become the text readable.

The proposed generator is presented in Fig. 5. It consists of three main parts: Inputs, combination function and comparator.

Input part (as shown in Fig. 4): It consists of several generators, which are LFSR, NLFSR and FCSR. In the LFSR generator, a keystream with the maximal period was generated by using a type of equation that is called primitive polynomials. The maximum sequence is equal to $2^n - 1$ [8, 10]. It was a type of non-singular, which it makes the sequence periodically (non-singular means that the high order degree of the primitive polynomial equation equal to the length of the LFSR) [8]. A dense (not-sparse) primitive polynomial was used in order to enhance the level of security for our design, which increases the protection against most attacks [8]. The primitive polynomial equations for the four LFSR are shown in the Table 1.



(a) Encryption stages. **(b) Decryption stages.**

Fig. 4. Encryption/decryption stages by using stream cipher.

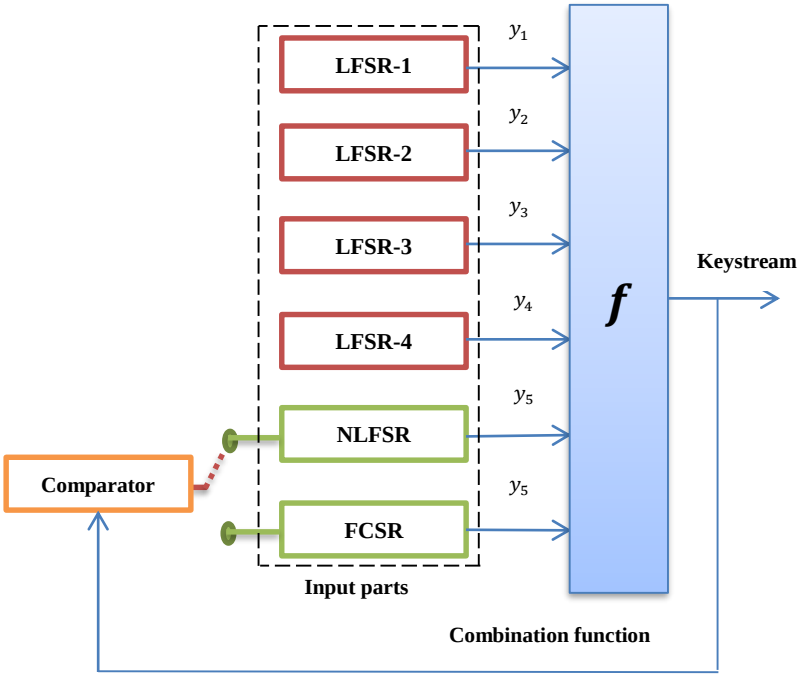


Fig. 5. Structure of proposed generator.

The equations for LFSR, NLFSR, and FCSR, were illustrated in Table 1, where the specifications for all equations were explained in the input part and combination function.

In the last two generators, which are NLFSR and FCSR, a Fibonacci type was used because it considers more complex in the design than the other types. It can enhance the security in the proposed generator, and it produces a maximum period that is equal $2^n - 1$. Thus, the equation that based on Fibonacci NLFSR by Dubrova [21] was used as shown in Table 1.

Table 1. Equation form choices for generators.

Type of generators	Equation form
LFSR-1	$1 + x^2 + x^5 + x^8 + x^9 + x^{10} + x^{12} + x^{13} + x^{16} + x^{17} + x^{18} + x^{19} + x^{21} + x^{22} + x^{23} + x^{24} + x^{29}$
LFSR-2	$1 + x + x^2 + x^4 + x^5 + x^7 + x^{10} + x^{11} + x^{12} + x^{13} + x^{15} + x^{17} + x^{24} + x^{25} + x^{27}\, x^{28} + x^{31}$
LFSR-3	$1 + x + x^3 + x^6 + x^8 + x^{11} + x^{13} + x^{14} + x^{15} + x^{17} + x^{20} + x^{22} + x^{23} + x^{25} + x^{26} + x^{28} + x^{29} + x^{30} + x^{33}$
LFSR-4	$1 + \mathrm{x} + \mathrm{x}^2 + \mathrm{x}^4 + \mathrm{x}^8 + \mathrm{x}^9 + \mathrm{x}^{10} + \mathrm{x}^{12} + \mathrm{x}^{13} + \mathrm{x}^{14} + \mathrm{x}^{15} + \mathrm{x}^{16} + \mathrm{x}^{20} + \mathrm{x}^{22} + \mathrm{x}^{23} + \mathrm{x}^{25} + \mathrm{x}^{27} + \mathrm{x}^{28} + \mathrm{x}^{29} + \mathrm{x}^{30} + \mathrm{x}^{37}$
NLFSR	$1 + x * x^5 + x^3 + x^6 + x^7 + x^{11} + x^{13}$
FCSR	q = 8221
Combination function (*f*)	$y_2y_4y_5 + y_1y_2 + y_1y_5 + y_2y_5 + y_3y_5 + y_3$

For the FCSR, the initial state was selected carefully in order to guarantee the sequence periodically. It can determine the initial state by following equation [20].

$$a_n = \left(y^n (mod\ q)\right)(mod 2) \tag{1}$$

The number of stages for the register and the carry by the following equations were determined by the following equations [20]:

$$\lfloor log_2\, q + 1 \rfloor \tag{2}$$

$$\lfloor log_2\, r \rfloor \tag{3}$$

The maximum period for the FCSR equal q-1 (where q is the connection integer and it should be a prime number).The connection integer was chosen by [10] as shown in Table 1.

Combination function (*f*): Combination function is one type of a nonlinear generator technique, which is used to increase the complexity and the efficiency in the design. The purpose of using this technique is to destroy the linearity that exists in the LFSR [8]. Also, it was used for the purpose of adding strength and protection to the LFSR, because the LFSRs are very vulnerable to attacks by the Berlekamp-Massey algorithm [22]. However, a combination function is a nonlinear equation, and it is express by using the algebraic normal form (ANF) (or it is called resilient Boolean function). The goal of using a combination function also is to increase the linear-complexity (LC) and the period. Finally, it satisfies several important criteria in order to make the generator more secure [8].

The resilient Boolean function for the nonlinear combination function (f) achieves four criteria. The first criterion is the balance. In other words, the number of one's equals to the number of zeros in results of the generator [23, 24]. This feature is very important because it makes the keystream random. It also leads to increase the correlation immunity and the nonlinearity [8]. The second and third criterion is the nonlinearity and algebraic degree. These features should be high in order to increase the linear complexity that makes the process of computation the LC by using the Berlekamp-Massey algorithm infeasible [23]. The last criterion is the correlation immunity (CI). It means the output of combination function (f) cannot give any information about the output for each register (inputs part). It must be high to resist many kinds of attacks such as correlation attacks and fast correlation attacks [23, 24].

Finally, five variables for the combination function, which was (5,1,3,12) was used in our design. The ANF represented by the function f (y_1, y_2, y_3, y_4, y_5) = $y_2y_4y_5 + y_1y_2 + y_1y_5 + y_2y_5 + y_3y_5 + y_3$.

Comparator part: A new idea in our design, which is a comparator was added. It leads to increase in the efficiency level of the security for the generator key. The principle work of the comparator depended on the contents of the two generators (NLFSR and FCSR). It selects either NLFSR or FCSR for every time of period (let say either 100 or 1000 period). When the output of (f) is one, the comparator will choose the greatest value for the contents of registers. Otherwise, it will choose the smallest value. The utility of using comparator is to prevent detection of the initial states for the NLFSR and FCSR by the attackers. Hence, it cannot determine any register switches on or off which, makes discovering the initial states for the two registers is very difficult.

4.2. Description of NIST statistical test

This section describes NIST tests used in this paper. The first test (frequency test), the focal point of this test is to measure the proportion of ones and zeros from the output-sequence for the generator key. The motivation from this test is to decide if the number of ones and zeros in the sequence are the same. This test is very important because all other tests are depended on it. If this test fails then, the probability of failure of other tests are very high. The second test (Frequency Test within a Block), it performs to measure the proportion of ones within each M-bit blocks. The motivation behind this test is to determine whether the frequency of ones of every M-bit block is approximately M/2. The third test is the run test. This test calculates the number of runs in the output-sequence and it considers the central point of this test. The objective of this test to specify the number of run times for the expected ones and zeros of the random sequence. In addition, the speed of transitions that occur between ones and zeros can be recognized very fast or very slow. The fourth test called the longest run of ones in a block. This test calculates the longest run of ones for each block, which it considers the focal point of this test. It is very necessary to identify the length of the longest run of ones that is regular or irregular in the expected random sequence.

The non-overlapping template is the five test. This test calculates the number of occurrences of pre-specified target strings. The objective of this test is to check the sequence has a predictable that it will appear diminution in randomness. While sixth test is the overlapping template. It is the same as non-overlapping

test, but the difference is in working. Maurer's "Universal statistical" is the seven test. It performs to measure the number of bits between matching patterns and it considers the core point for this test. The purpose of this test measures the sequence, which can succeed to compress without losing any information. According to the NIST manual, when the sequence is extremely compressible, then the p-value is low and it considers a not-random. Eighth test is the serial test. This test cares about the properties uniformity in the random sequence. It is meaning every n-bit sequence has a chance of appearing like each other of the n-bit sequence. Nineth test is the approximate entropy test. It is similar to the serial test. Both of them look at nested patterns in the sequences s-bit and it considers the central point of this test. The last test is the random excursions test. This test calculates the number of cycles having n visits in the path of the cumulative sum random and it considers the focal point of this test. A cycle of a random walk consists of a sequence of steps of unit length capturing randomly that start at and back to the origin. The purpose of this test is to specify if the number of visits for a certain state inside a cycle deviates from what someone will predict for the random sequence.

4.3. Results of the NIST statistical test

The robustness of the proposed generator was evaluated by measuring the randomness key by using the National Institute of Standards and Technology (NIST). NIST is based on the mathematical theory to test the randomness of the keystream that produced by the pseudo-random number generators [25]. Eleven tests of NIST were chosen to evaluate our design as shown in Table 2. A sequence of bits was generated (approximately $n = 10^6$ bits) according to the size of the recommendation of the NIST. The P-value that should be generated for each test must be equal or larger than 0.01 in order to ensure the randomness of the generated key. The randomness leads to strong key against known attacks.

In the random excursions test (test No. 11), we generated n = 5000 bits. The results were shown in Table 3 below. All states were successful in this test but exception the state no. -4 where p-value was lee than 0.01 (where this test has eight states, each one consist of p-value to measure the level of randomness [25]).

Table 2. P-value for proposed key generator.

No	Tests of NIST	p-value	Parameters
1	Frequency (Monobit) test	0.693	-
2	Frequency test within a block	0.92	-
3	Runs test	0.681	$M \geqslant 0.1$*n, $N <$ 100
4	Longest run of ones in a block	0.444	k =6, M =10^4
5	Non-overlapping template	0.999	m =9, N =8, B = 000000001
6	Overlapping template	1	m = 9, M = 1032, N = 968 B = 111111111
7	Maurer's "universal statistical"	0.95	L = 7, Q =1280
8	Serial test	p-value 1 = 0.480512 p-value 2 = 0.62555	m = 2
9	Approximate entropy test	0.63096	m =3
10	Linear Complexity	0	$n = 10^6$

Table 3. P-value for random excursions test.

State = *x*	p-value	Conclusion
-4	0.009251756	Non-random
-3	0.013040615	Random
-2	0.026396547	Random
-1	0.754563032	Random
1	0.860721006	Random
2	0.129809256	Random
3	0.106760575	Random
4	0.207664018	Random

4.4. Comparisons results for our design with others

The results of our design were compared with others such as STM, MLM, ALPRNG, PLM, and Warbler Family design that are explained in section 2. First, results of P-values for our design were better than results of SLM and MLM with some exceptions as shown in Table 4. For example, the result of linear complexity for STM and MLM was a little bit larger than ours, but it also considered as a fail test because the minimum amount of p-value should be (0.01) to succeed. Another example, the random excursion for MLM was better than our design. However, our design is better than SLM and MLM as shown below. In addition, when we compare our design with the last column that is shown in Table 4, we found that all tests' p-value results except run test and linear-complexity test were better than ALPRNG p-value tests.

In addition, the last two designs (PLM and Warbler Family) were compared with our design as shown in Table 5. Most of p-values were ≥ 0.01 that it considered good results except for some of the following tests. Firstly, the non-overlapping test in the last column, the result was not calculated [13]. Secondly, the linear complexity test in ours was 0 although others were better than our design. Finally, one of the state values (x = -4) in the excursions test did not succeed in our design. While the other designs, p-values were more than 0.01.

Table 4. Results of NIST tests for our design with others.

No.	Tests	Max P-valu1e for our design	Max P-valu1e for STM	Max P-valu1e for MLM	Max P-valu1e for ALPRNG
1	Monobit test	0.82	0.001	0.001	0.780
2	Freq test within a block	0.981	0.001	0.001	0.467
3	Runs	0.817	0.001	0.001	0.957
4	Longest ones	0.944	0.001	0.001	0.707
5	Non-overlapping	0.999	0.001	0.9	0.466
6	Overlapping	1	0.001	0.001	0.264
7	Universal test	0.995	0.04	0.001	0.844
8	Linear complexity	0	0.001	0.001	0.719
9	Serial test	1	0.001	0.001	0.590
10	Entropy test	0.764	0.001	0.001	0.398
11	Excursions test	0.86	0.5	0.9	0.743

Table 5. Results of NIST tests for our design with others.

No.	Tests	P-valu1e for our design	P-valu1e for PLM	P-valu1e for WF in TS1
1	Monobit test	0.82	0.51	1
2	Freq test within a block	0.981	0.203	1
3	Runs	0.817	0.967	0.99
4	Longest ones	0.944	0.492	0.99
5	Non-overlapping	0.999	0.473	-
6	Overlapping	1	0.245	0.99
7	Universal test	0.995	0.307	0.99
8	Linear complexity	0	0.575	0.97
9	Serial test-1	1	0.522	0.99
	Serial test-2	1	0.487	0.99
10	Entropy test	0.764	0.146	1
11	Excursions test:	0.009251756	0.783019	0.98
	X = -4			
	X = -3	0.013040615	0.063137	0.99
	X = -2	0.026396547	0.605501	0.99
	X = -1	0.754563032	0.037566	1
	X = 1	0.860721006	0.808515	0.99
	X = 2	0.129809256	0.247255	1
	X = 3	0.106760575	0.937919	0.99
	X = 4	0.207664018	0.216485	1

4.5. Execute our design for mobile phones

After implementing our design and proving the strength of generating randomness sequences by using statistical tests for NIST, an application has implemented on the Android smartphones devices as shown in Fig. 6. In other words, the stream cipher method was used to encrypt the plaintext by using our generator "keystream" to produce ciphertext in the sender side. Then, the same generator was used with the ciphertext to generate the original plaintext in the receiver side. Finally, the speed time of encryption and decryption was also calculated by using this application, which demonstrates the efficiency of our design in the smartphones systems.

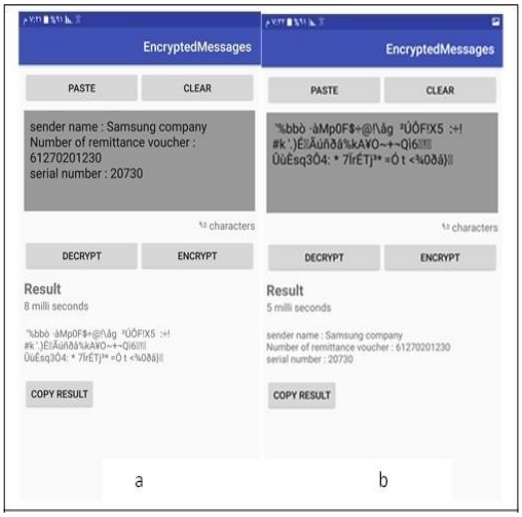


Fig. 6. Our application for encryption and decryption text data.

5. Conclusion

In this paper, data that are transmitted through device communication face some security challenges. Stream cipher considers as a good method to protect these data. However, this method has several drawbacks. The most critical one is using the key twice. In order to overcome these challenges, a new form of key generator based on stream cipher is presented in this paper.

Three types of key generators were used, which are LFSR, FCSR, NLFSR to implement our method. Then, they combined together into a nonlinear Boolean function (combining function). The main goal for this design is to increase the level of efficiency and enhance the ability of the generator to face the known attacks. The generator was tested by measuring the randomness of the key by using the NIST statistical test. All the tests that are used in the evaluation were passed.

We will make a comparison between this method with other classical methods to discover the effectiveness of our design. We will try to use this method to encrypt and decrypt video, voice, and photo data.

Nomenclatures

f	Combination function
L	Length of bits for each block (Table 2)
M	Number of bit for each block (Table 2)
m	Length of bits was used in B (Table 2)
N	Number of blocks (Table 2)
n	Number of bits
p-value	Probabilistic value
Q	Number of blocks in part of initialization segment (Table 2)
q	Connection integer

Greek Symbols

σ	Parity bit
⌊ ⌋	Integer part, or greatest integer

Abbreviations

ANF	Algebraic Normal Form
FCSR	Feedback with Carry Shift Register
LFSR	Linear Feedback Shift Register
MITM	Man in the Middle
MLM	Modified Logistic Map
NIST	National Institute of Standard and Technology
NLFSR	Non-Linear Feedback Shift Register
PRNG	Pseudo-Random Number Generator
STM	Skew Tent Map

References

1. Kaka, S.; Sastry, V.N.; and Maiti, R.R. (2016). On the MitM vulnerability in mobile banking applications for Android devices. *Proceedings of the IEEE*

International Conference on Advanced Networks and Telecommunications Systems (ANTS). Bangalore, India, 1-6.

2. Jallouli, O.; El Assad, S.; and Chetto, M. (2016). Robust chaos-based stream-cipher for secure public communication channels. *Proceedings of the IEEE 11th International Conference for Internet Technology and Secured Transactions (ICITST).* Barcelona, Spain, 23-26.
3. Salhab, O.; Jweihan, N.; Jodeh, M.A.; Taha, M.A.; and Farajallah, M. (2018). Survey paper pseudo random number generator and security tests. *Journal of Theoretical and Applied Information Technology*, 96(7), 1951-1970.
4. Paar, C.; and Pelzl, J. (2010). *Understanding cryptography*. Berlin, Heidelberg: Springer-Verlag.
5. Vidal, G.; Baptista, M.S.; and Mancini, H. (2014). A fast and light stream cipher for smartphones. *The European Physical Journal Special Topics*, 223(8), 1601-1610.
6. Manifavas, C.; Hatzivasilis, G.; Fysarakis, K.; and Papaefstathiou, Y. (2016). A survey of lightweight stream ciphers for embedded systems. *Security and Communication Networks*, 9(10), 1226-1246.
7. Banegas, G. (2014). Attacks in stream ciphers: A survey. *IACR Cryptology ePrint Archive* , 1-16.
8. Menezes, A.J.; Vanstone, S.A.; and Van Oorschot, P.C. (1996). *Handbook of applied cryptography* (1st ed.). Boca Raton, Florida: CRC Press, Inc.
9. Cusick, T.W.; and Stanica, P. (2017). *Cryptographic Boolean function and application* (2nd ed.). Oxford, United Kingdom: Academic Press.
10. Schneier, B. (2015). *Applied cryptography: Protocols, algorithms and source code in C* (20th ed.). Indianapolis, United States of America: John Wiley & Sons.
11. Ashouri, M. (2018). Design of a new stream cipher: PALS. *arXiv.org*, 8 pages.
12. Tasheva, Z.N. (2015). On linear complexity of generalized shrinking-multiplexing generator. *Journal of Basic and Applied Research International*, 4(1), 8-17.
13. Al_Sharifi, H.A.M. (2014). Frequency postulate's theoretical calculation for the sequences produced by modified geff generator. *Journal of Kerbala University*, 12(2), 199-208.
14. Marghescu, A.; Teseleanu, G.; and Svasta, P. (2014). Cryptographic key generator candidates based on smartphone built-in sensors. *Proceedings of the IEEE 20th International Symposium for Design and Technology in Electronic Packaging (SIITME).* Bucharest, Romania, 239-243.
15. Garcia-Bosque, M.; Sanchez-Azqueta, C.; Royo, G.; and Celma, S. (2016) Lightweight ciphers based on chaotic map-LFSR architectures. *Proceedings of the IEEE 12th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME).* Lisbon, Portugal, 1-4.
16. Wang, Y.; Liu, Z.; Ma, J.; and He, H. (2016). A pseudorandom number generator based on piecewise logistic map. *Nonlinear Dynamic*, 83(4), 2373-2391.
17. Mandal, K.; Fan, X.; and Gong, G. (2016). Design and implementation of warbler family of lightweight pseudorandom number generators for smart devices. *ACM Transactions on Embedded Computing Systems*, 15(1), 1-28.

18. Al-Khatib, M.A.S.; and Lone, A.H. (2018). Acoustic lightweight pseudo random number generator based on cryptographically secure LFSR. *International Journal of Computer Network and Information Security*, 10(2), 38-45.
19. Zhiqiang, L. (2013). The transformation from the galois NLFSR to the fibonacci configuration. *Proceedings of the IEEE Fourth International Conference on Emerging Intelligent Data and Web Technologies (EIDWT)*. Xi'an, China, 335-339
20. Goresky, M.; and Klapper, A. (2012). *Algebraic shift register sequences* (1st ed.). Cambridge, United Kingdom: Cambridge University Press.
21. Dubrova, E. (2012). A list of maximum period NLFSRs. *IACR Cryptology ePrint Archive*, 9 pages.
22. Brock, T.B. (2006). Linear feedback shift register sequences and cyclic codes in sage. *Rose-Hulman Undergraduate Mathematics Journal*, 7(2), 1-18:
23. Fu, S.; Sun, B.; Li, C.; and Qu, L. (2011). Construction of odd-variable resilient Boolean functions with optimal degree. *Journal of Information Science and Engineering*, 27(6), 1931-1942.
24. Burnett, L.; Millan, W.; Dawson, E.; and Clark, A. (2004). Simpler methods for generating better Boolean functions with good cryptographic properties. *Australasian Journal of Combinatorics*, 29, 231-247.
25. Bassham, L.E.; Rukhin, A.L.; Soto, J.; Nechvatal , J.R.; Smid, M.E.; Barker, E.B.; Leigh, S.D.; Levenson, M.; Vangel, M.; Banks, D.L.; Heckert, N.A.; Dray, J.F.; and Vo, S. (2010). A statistical test suite for random and pseudorandom number generators for cryptographic applications. *NIST SP, Special Publication*, 800-22 Rev. 1a.