U.P.B. Sci. Bull., Series C, Vol. 81, Iss. 2, 2019 ISSN 2286-3540

Stream Ciphering based on Non-Linearity of Elementary Function Compositions

Roxana Drăgănoiu[1], George Anescu[1], Florica Moldoveanu[2], Alin Moldoveanu[4]

The paper is exploring the possibility of using the non-linearity of elementary function compositions, combined with principles from number theory and combinatorics, in designing new cryptographic methods, and particularly in designing synchronous stream ciphering methods. The strength of such methods is based on the difficulty to solve with high accuracy large systems of nonlinear equations. The implementation details of an example of such stream ciphering method are presented. The security analysis and the statistical tests run on the implemented method show that the proposed approach is promising.

Keywords: Cryptography, Stream-Ciphering, Non-Linearity, Elementary Functions, Number Theory, Combinatorics, PractRand, SCNLEFC

1. Introduction

The main idea in synchronous stream ciphering is to generate, based on a mathematical algorithm, a sequence of pseudo-random bytes, called key stream, used for ciphering a stream of plain text data. Usually the plain text is xor-ed with the generated key stream, a process called one-time padding:

$$c_i = p_i \oplus r_i, \ \ i = 1, 2, \dots, \tag{1}$$

where p_i are the characters (bytes) of the plain text, r_i are the bytes of the generated key stream, and respectively c_i are the bytes of the ciphered text. The deciphering is done by using the same key stream:

$$p_i = c_i \oplus r_i, \ \ i = 1, 2, \dots. \tag{2}$$

Equally powerful is the use of the addition with modulo 256 operation:

[1] PhD student, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: `roxanadraganoiu@yahoo.com`

[2] PhD, Faculty of Power Engineering, University POLITEHNICA of Bucharest, Romania, e-mail: `george.anescu@gmail.com`

[3] Prof., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: `florica.moldoveanu@cs.pub.ro`

[4] Prof., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: `alin.moldoveanu@cs.pub.ro`

$$c_i = (p_i + r_i)\%256, \ \ i = 1, 2, \dots . \tag{3}$$

The strength of a stream ciphering method relies on the keystream generating function. In order to use a stream cipher securely one should not use the same keystream more than once. That means that a different key must be used for each invocation of the stream cipher. There are many examples of stream ciphering methods known in the cryptographic literature based on different approaches.

The traditional approach is to use algorithms based on fast bitwise operations. The eSTREAM project ([1]), ran in Europe from 2004 to 2008 by EU ECRYPT network had the purpose to identify new stream cipher primitives suitable for widespread adoption based on the traditional approach. It was set up as a result of the failure of a previous European project, the NESSIE project ([2]), which was concerned with a set of six proposed stream ciphers, but all of them were cracked. The eSTREAM portfolio was revised in September 2008 and currently it contains seven stream ciphers organized into two profiles. Profile 1 contains 4 stream ciphers more suitable for software applications with high throughput requirements: HC-128, Rabbit, Salsa20/12 and SOSEMANUK. Profile 2 contains 3 stream ciphers more suitable for hardware applications with restricted resources, such as limited storage, gate count, or power consumption: Grain v1, MICKEY 2.0 and Trivium. The eSTREAM portfolio is periodically revisited, as the algorithms mature. The first review was published in October 2009 ([3]), and the second in January 2012 ([4]). The original eSTREAM project's website ([1]) gives information on the selection process, including a timetable of the project and further technical background. The eSTREAM Book ([5]), a volume published by Springer in 2008, provides full specifications of all 16 stream ciphers that reached the final phase of the eSTREAM project.

A newer approach in designing stream ciphers is inspired from chaos theory ([6]) and is using chaotic maps (functions) in order to produce random sequences. The chaotic systems have a pseudo-random behavior, are highly sensitive to initial conditions and are able to disperse data around their working space. Unlike the traditional approach, the chaotic stream ciphers are based on various chaotic maps which employ floating point arithmetic operations, see as examples [7], [8] and [9]. A survey on the encryption algorithms based on chaos theory is given in [10]. Many proposed chaos-based cryptosystems are difficult to implement in practice with a reasonable degree of security and they are seldom accompanied by a thorough security analysis. In the study [11] the basic cryptographic requirements for such systems are analyzed and a common framework is given.

The present paper is proposing an original stream ciphering approach and is detailing an implementation example of it. The proposed approach is inspired from previous work in enhanced graphical parametric modeling methods ([12], [13], [14], [15] and [16]). Basically the graphical modeling method consists in generating expressions made by functions connected by means of specific binary operators. The functions are selected from families of functions obtained from a set of basic functions by applying some extension rules. If $f_0 : R \to [0, 1]$, $sup(f_0) = 1$ is a basic function then the family of associated functions, denoted by $F^{f_0}_{a,b,c,d}$, is formed by the functions $f \in F^{f_0}_{a,b,c,d}$ defined by ([16]):

$$f(x) = a\left[(1-b)f_0\left(\frac{x-c}{d}\right) + b\right], \tag{4}$$

where $a \in R$ is the amplitude (a scaling factor on Oy axis), $b \in [0, 1]$ is the compression (a translation factor on Oy axis), $c \in R$ is the phase (a translation factor on Ox axis) and $d \in (0, \infty)$ is the wavelength (a scaling factor on Ox axis), with the inverse of d, $e = \frac{1}{d}$, having the meaning of frequency.

The expressions are generated step by step, starting with a function and adding at each step a binary operator and a new function. Based on the proposed graphical modeling system ([16]) some powerful software tools can be implemented, which are useful for a wide range of users with various backgrounds and levels of mathematical training, providing them a deeper understanding of mathematical elementary functions and their composition through interactive visualization.

The rest of the paper is organized as follows: Section 2 presents the design principles of the proposed stream ciphering method and an implementation example. Section 3 presents aspects related to the security analysis of the proposed stream ciphering implementation, including key space estimation and periodicity estimation. Section 4 presents the results of the security tests run on sample results of the proposed stream ciphering method. Section 5 draws some conclusions and hint to further research directions. Some Appendices follow the References section and are presenting the analytical formulas of the primitive functions (Appendix A), unary operators (Appendix B) and binary operators (Appendix C).

2. SCNLEFC Method's Design

All the design details of the proposed Stream Ciphering based on Non-Linearity of Elementary Function Compositions (*SCNLEFC*) presented in this section are considered public by observing the cryptographic principle that the security of a cryptosystem should depend only on its key. Inspired by the graphical modeling system presented in [16], we designed a set of 30 basic one variable functions (we call them primitive functions or primitives here, see Appendix A.), a set of 18 unary operators (also one variable functions, see Appendix B.) and a set of 9 binary operators (two variables functions, see Appendix C.). In order to better control the computations and to avoid domain values where the primitive functions and the operators are undefined, as a general rule, we designed all of them defined on the $[0,1]$ real interval with values in the $[0,1]$ real interval. We designed the primitive functions as compositions of elementary functions with increased variability, typically higher than the variability of the unary operators. As a graphical representation example see the primitive function 17 in Fig. 1.

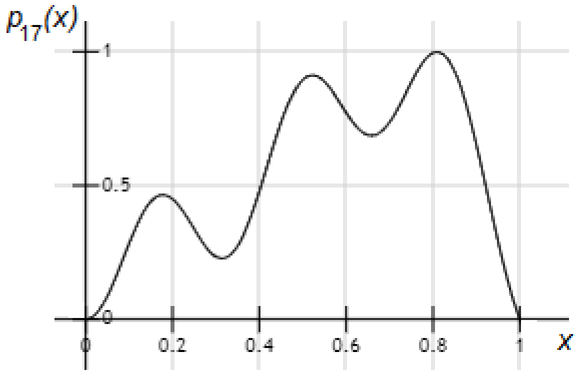

FIG. 1. Example of variability in primitive functions ($p_{17}(x)$)

Although the primitive functions are conceptually similar to the base functions used in the graphical modeling method ([16]), in order to increase the randomness of the key stream we designed them so that they do not present any symmetries. At any given computing iteration only a combinatorial arrangement of 5 primitive functions from the set of 30, $(p_{i_1}, p_{i_2}, p_{i_3}, p_{i_4}, p_{i_5})$, a combinatorial arrangement of 5 unary operators from the set of 18, $(u_{j_1}, u_{j_2}, u_{j_3}, u_{j_4}, u_{j_5})$, and a combinatorial arrangement of 4 binary operators from the set of 9 $(b_{k_1}, b_{k_2}, b_{k_3}, b_{k_4})$, are used, with the sets of indices i, j and k taking values in the corresponding ranges. Each of the currently used primitive functions can be computed only on a corresponding fixed set of $1024 = 2^{10}$ values in the $[0,1]$ real interval, generated from the expanded/contracted key data as it will be explained later. From the parameters a, b, c and d used to generate the family of associated functions (see equation (4)) only the parameters a and d are retained, and they are also generated from the expanded/contracted key data. The method can be easily redesigned using different numbers of primitive functions and unary or binary operators, and can be scaled differently, we here present only the design principles and a possible implementation example. The main phases of the *SCNLEFC* algorithm are summarized in the pseudo-code below and will be detailed in the following sub-sections:

2.1. Key Expansion/Contraction and Method Parameters Computation

All the method's parameters are generated based on the expansion of the initial key performed by employing the key expansion/contraction method

Algorithm SCNLEFC

```
Key Expansion/Contraction
Method's Parameters Computation
$i \leftarrow 1$
while $i < stream\_length$ do
    Key Stream Word Values Computation
    Key Stream Word Values Application to Ciphering
    $i \leftarrow i + 4$
```

proposed in [17]. The employed method allows any initial key length and it is able to generate expanded/contracted key data of any length. In the next subsection will be explained in detail how the generated method parameters are used in the cryptographic algorithm. The initial key data is expanded to a total size of 5180 words denoted w_i, $i = 1, \ldots, 5180$ (a word is assumed here to have 32 bits).

The first 5 words, w_i, $i = 1, \ldots, 5$, are used to generate the Oy scaling factors a_i, $i = 1, \ldots, 5$, as double precision floating point numbers in the $[0.1, 10.0]$ real interval. Based on the LSBs (less significant bits) of the words the a_i parameters are computed as:

$$a_i = \frac{100000000 + (w_i \% 900000000)}{100000000.0}, \tag{5}$$

if the LSB of word w_i is set (1), and as:

$$a_i = \frac{100000000 + (w_i \% 900000000)}{1000000000.0}, \tag{6}$$

if the LSB of word w_i is not set (0). In this way, by assuming that the bits of the generated expanded/contracted key are uniformly distributed, the Oy scaling factors a_i have equal chances to be selected in the $[0.1, 1.0]$ real interval (contraction) or in the $[1.0, 10.0]$ real interval (dilation).

The next 5 words, w_{i+5}, $i = 1, \ldots, 5$, are used to generate the Ox scaling factors d_i as double precision floating point numbers in the $[1.0, 10.0]$ real interval:

$$d_i = \frac{100000000 + (w_{i+5} \% 900000000)}{100000000.0}, \quad i = 1, \ldots, 5. \tag{7}$$

The next 19 words, w_{i+10}, $i = 1, \ldots, 19$, are providing the pseudo-random bytes used to generate the initial random combination of 7 prime numbers $prime_i$, $i = 1, \ldots, 7$, from the set of 75 prime numbers between 512 and 1024. Only the first 75 bytes (from the $19 \times 4 = 76$ bytes) are used in the known Durstenfeld's shuffling algorithm (an improved variant of the Fisher Yates shuffling algorithm, [18]). The first 5 generated prime numbers are giving the periods (lenghts) of the computing windows $pr_i = prime_i$, $i = 1, \ldots, 5$, while the next two $prime_6$ and $prime_7$ will be used in the computing of the word key stream value.

The next 5 words, w_{i+29}, $i = 1, \ldots, 5$, are used to generate the increments icr_i, $i = 1, \ldots, 5$. The increments can be odd numbers between 1 and 1023 (prime with $1024 = 2^{10}$) and therefore they are computed as:

$$icr_i = (w_{i+29}\%1024)|1, \ \ i = 1, \ldots, 5, \tag{8}$$

the LSB bits being set to 1 to ensure the oddness.

The next 8 words, w_{i+34}, $i = 1, \ldots, 8$, are providing the pseudo-random bytes used to generate the initial random combinatorial arrangement of 5 primitive functions from the set of 30 primitive functions. Only the first 30 bytes (from the $8 \times 4 = 32$ bytes) are used in the Durstenfeld's shuffling algorithm.

The next 5 words, w_{i+42}, $i = 1, \ldots, 5$, are providing the pseudo-random bytes used to generate the initial random combinatorial arrangement of 5 unary operators from the set of 18 unary operators. Only the first 18 bytes (from the $5 \times 4 = 20$ bytes) are used in the Durstenfeld's shuffling algorithm.

The next 3 words, w_{i+47}, $i = 1, \ldots, 3$, are providing the pseudo-random bytes used to generate the initial random combinatorial arrangement of 4 binary operators from the set of 9 binary operators. Only the first 9 bytes (from the $3 \times 4 = 12$ bytes) are used in the Durstenfeld's shuffling algorithm.

The next 2 words, w_{i+50}, $i = 1, \ldots, 2$, are providing the pseudo-random bytes used to generate the initial random permutation to be applied to the generated random combinatorial arrangement of 5 primitive functions. Only the first 5 bytes (from the $2 \times 4 = 8$ bytes) are used in the Durstenfeld's shuffling algorithm.

The next 5 words, w_{i+52}, $i = 1, \ldots, 5$, are used to generate the initial positions ini_i, $i = 1, \ldots, 5$. The initial positions are indices between 1 and 1024 and therefore they are computed as:

$$ini_i = (w_{i+52}\%1024) + 1, \ \ i = 1, \ldots, 5. \tag{9}$$

The next 3 words, w_{i+57}, $i = 1, \ldots, 3$, are used to generate the parameters $fact$, $sum1$ and $sum2$ used in the computing of the key stream value:

$$fact = 100000000 + (w_{58}\%900000000), \tag{10}$$

$$sum1 = w_{59}, \tag{11}$$

$$sum2 = w_{60}. \tag{12}$$

Next the arrays of fixed computing values corresponding to each primitive function, $v_{j,i}$, $j = 1, \ldots, 5$, $i = 1, \ldots, 1024$, are generated based respectively on the words:

$$w1_{j,i} = w_{1024(j-1)+i+60}, \;\; j = 1, \ldots, 5, \;\; i = 1, \ldots, 1024. \tag{13}$$

The words $w1_{j,i}$ are restricted to the domain $[1, MAXUINT - 1]$ with $MAXUINT = 2^{32} - 1 = 4294967295$, so if $w1_{j,i} = 0$ then it is changed to 1 and if $w1_{j,i} = MAXUINT$ then it is changed to $MAXUINT - 1$. Then the fixed computing values are computed as fractional parts restricted to the $(0, 1)$ real interval according to:

$$v_{j,i} = \left\{ d_j \frac{(double)w1_{j,i}}{MAXUINT} \right\}, \;\; j = 1, \ldots, 5, \;\; i = 1, \ldots, 1024, \tag{14}$$

where $\{.\}$ is the fractional part function. A flowchart of the method parameters computation process is presented in Fig. 2.

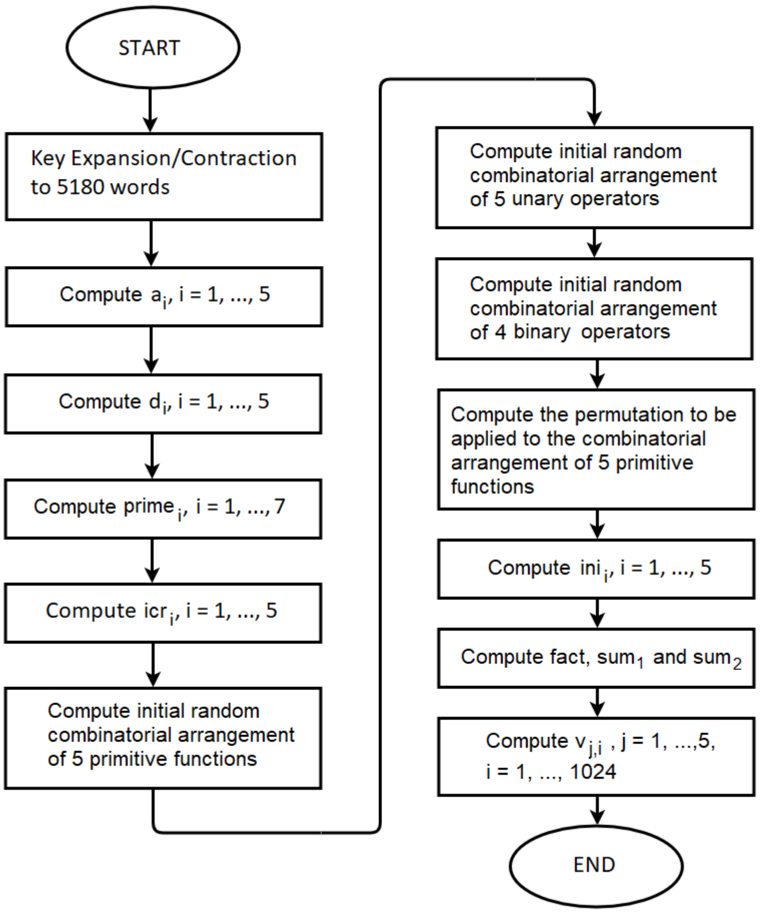
START
Key Expansion/Contraction to 5180 words
Compute a_i, i = 1, ..., 5
Compute d_i, i = 1, ..., 5
Compute $prime_i$, i = 1, ..., 7
Compute icr_i, i = 1, ..., 5
Compute initial random combinatorial arrangement of 5 primitive functions
Compute initial random combinatorial arrangement of 5 unary operators
Compute initial random combinatorial arrangement of 4 binary operators
Compute the permutation to be applied to the combinatorial arrangement of 5 primitive functions
Compute ini_i, i = 1, ..., 5
Compute fact, sum_1 and sum_2
Compute $v_{j,i}$, j = 1, ...,5, i = 1, ..., 1024
END

FIG. 2. Method Parameters Computation

2.2. Key Stream Values Computation

At a computing iteration each primitive function is positioned to an index ix_i, $i = 1, \ldots, 5$, in its corresponding array of fixed values. Initially the positioning indices of the primitive functions are given by the ini_i parameters $ix_i = ini_i$, $i = 1, \ldots, 5$, and in the next iterations they are usually incremented by 1 and wrapped around within the modulo 1024 arithmetic:

$$ix_i = (ix_i + 1)\%1024 + 1, \;\; i = 1, \ldots, 5, \tag{15}$$

with the exception of two types of discontinuities introduced by design. The first type of discontinuity appears when at least one computing window is ended. For example, if it is assumed that pr_1 is the smallest of the pr_i, $i = 1, \ldots, 5$ parameters, then the first computing window will end first after pr_1 iterations and a discontinuity will be applied by updating

$$ini_1 = (ini_1 + icr_1)\%1024 + 1, \tag{16}$$

and moving ix_1 to the new ini_1. The same first type of discontinuity logic is also applied to the remaining pr_i, $i = 2, \ldots, 5$, when their corresponding computing windows end. At each discontinuity of first type the permutation applied to the primitive functions is updated, in this way updating the method's state (the configuration of the computing binary tree is altered). The described algorithm for updating the positioning indices ensure that the initial positioning indices will repeat after a period of t_1 iterations, with:

$$t_1 = 1024 \times pr_1 \times pr_2 \times pr_3 \times pr_4 \times pr_5, \tag{17}$$

which represents the second type of discontinuity applied simultaneously for all the computing windows. This property is mathematically ensured by the primality of the pr_i, $i = 1, \ldots, 5$ periods, they being relatively prime among themselves and with the array sizes $1024 = 2^{10}$. At each discontinuity of second type the combinatorial arrangement of 5 primitive functions, the combinatorial arrangement of 5 unary operators, and the combinatorial arrangement of 4 binary operators are updated, in this way altering even more radically the method's state (the configuration of the computing binary tree is altered).

Assuming that at a computing iteration the 5 current primitive functions are computed respectively for the real values $x_1 = v_{1,ix_1}$, $x_2 = v_{2,ix_2}$, $x_3 = v_{3,ix_3}$, $x_4 = v_{4,ix_4}$ and $x_5 = v_{5,ix_5}$, then the current ciphering real value is computed according to the computing binary tree as:

$$\begin{aligned} v_1(x_1, x_2, x_3, x_4, x_5) = a_{b_4} \times ((((u_1(p_1(x_1))b_1u_2(p_2(x_2)))b_2(p_3(x_3)))b_3 \\ u_4(p_4(x_4)))b_4u_5(p_5(x_5))), \end{aligned} \tag{18}$$

where for simplification we denoted the current combinatorial arrangements with $(p_1, p_2, p_3, p_4, p_5)$ for primitive functions, $(u_1, u_2, u_3, u_4, u_5)$

for unary operators, and respectively (b_1, b_2, b_3, b_4) for binary operators. The computing tree is illustrated in Fig. 3.

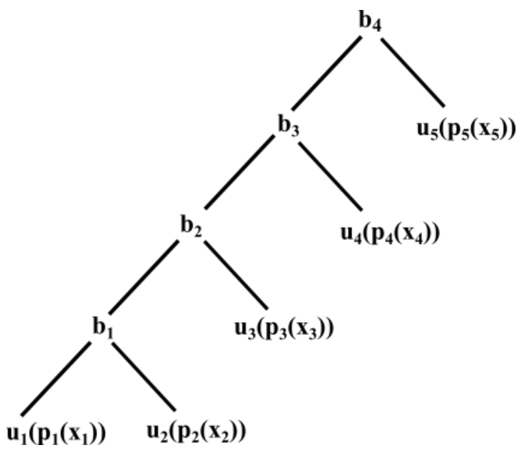

FIG. 3. Computing Tree

The word key stream value is computed according to:

$$rw = prime5 \times (prime6 \times \lfloor 100000 \times fact \times v_1 + 0.5 \rfloor + sum1) + sum2, \quad (19)$$

where $\lfloor . \rfloor$ is the integer part function. From the word key stream value 4 bytes of key stream values are generated: $r1 = (BYTE)rw$, $r2 = (BYTE)(rw >> 8)$, $r3 = (BYTE)(rw >> 16)$ and $r4 = (BYTE)(rw >> 24)$, which can be applied in the stream ciphering equation (1). A flowchart of the key stream values computation process is presented in Fig. 4.

Due to the complexity of the computing binary tree, an important concern is related to the computing speed of the proposed proposed *SCNLEFC* stream ciphering method. A speed increase solution could be the use of parallel computing for pre-calculated segments of stream for which the state of the cryptographic method (the configuration of the computing binary tree) is not changed. Both multi-core *CPU*s (Central Processing Units) and *GPU*s (Graphics Processing Units) systems can be used.

3. Security Analysis

3.1. Key Space Analysis

The strength of any cryptographic method is based on the difficulty to solve an associated mathematical problem. It is also the case for the proposed *SCNLEFC* stream ciphering method, which is based on the difficulty to solve with high numerical accuracy large systems of nonlinear equations ([19]), as it will be explained in this section. We identified three types of possible cryptanalytic attacks:

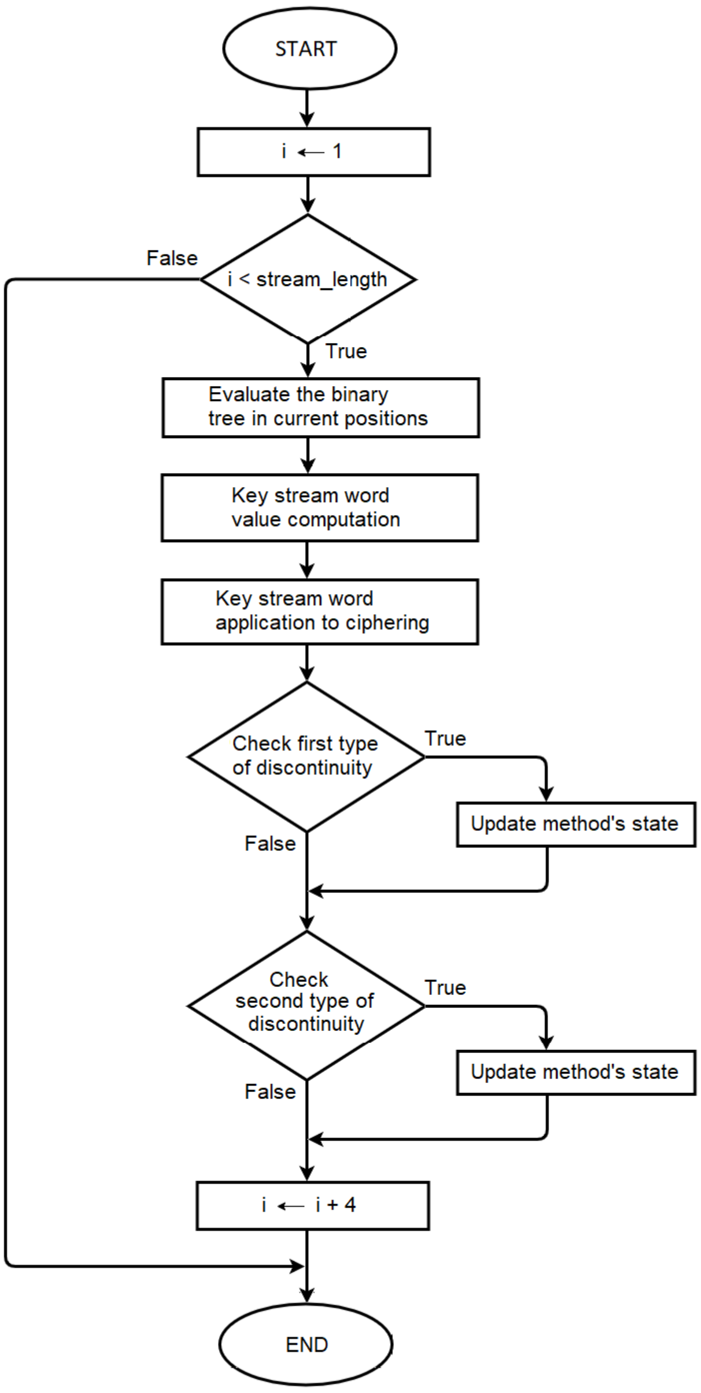
START
i ← 1
i < stream_length
False
True
Evaluate the binary tree in current positions
Key stream word value computation
Key stream word application to ciphering
Check first type of discontinuity
True
False
Update method's state
Check second type of discontinuity
True
False
Update method's state
i ← i + 4
END

FIG. 4. Key Stream Values Computation

1. A direct brute force attack to the initial key: The initial key is masked by the expansion/contraction method, and therefore only a direct brute force attack is feasible on it, but by choosing an appropriately large initial key length

($\geq$ 256 bits for the computing power of the present computers) such a brute force attack is greatly discouraged.

2. A direct brute force attack to the expanded/contracted key data: As previously explained, the initial key data is expanded to a total size of 5180 words. The first 60 words are partially used to compute the method parameters, while the remaining $5180 - 60 = 5120 = 5 \times 1024$ words are used to compute the fixed computing values $v_{j,i}$, $j = 1, \ldots, 5$, $i = 1, \ldots, 1024$, based on the words $w1_{j,i}$, $j = 1, \ldots, 5$, $i = 1, \ldots, 1024$, see the equations (14) and (13). The words $w1_{j,i}$ are restricted to $2^{32} - 2$ values. In our analysis we will divide the key space in two pieces: 2.1. the key space generated by the fixed computing values $v_{j,i}$; and, 2.2. the key space generated by method parameters (generated the first 60 expanded key data words):

2.1. We can conservatively evaluate the space dimension of the fixed computing values $v_{j,i}$, $DKS1$, as:

$$DKS1 > (2^{32} - 2)^{5120} > (2^{31})^{5120} = 2^{158720}, \tag{20}$$

which is a very large key space dimension discouraging a brute force attack.

2.2. We will evaluate the remaining space dimension generated by the first 60 words from the expanded key data. The first 60 words are partially used to compute the the scaling factors a_i, $i = 1, \ldots, 5$, the scaling factors d_i, $i = 1, \ldots, 5$, the initial random combinatorial arrangement of 7 prime numbers $prime_i$, $i = 1, \ldots, 7$, the increments icr_i, $i = 1, \ldots, 5$, the initial random combinatorial arrangement of 5 primitive functions, the initial random combinatorial arrangement of 5 unary operators, the initial random combinatorial arrangement of 4 binary operators, the initial random permutation applied to the primitive functions, the initial computing positions ini_i, $i = 1, \ldots, 5$, the initial parameters $fact$, $sum1$ and $sum2$. For example for the a_i scaling factors, each of them can take 900000000 possible values (see (5), (6)), their key space dimension being:

$$dka = 2^{\log_2(900000000^5)} = 2^{5log_2(900000000)} \approx 2^{5\times 29.745} \approx 2^{148.725}. \tag{21}$$

As another exeample, for the 7 prime numbers there are A_{75}^{7} possible arrangements, their key space dimension being:

$$dkap = 2^{\log_2(A_{75}^{7})} = 2^{43.185}. \tag{22}$$

Through similar evaluations for the remaining parameters we can finally derive a key space dimension:

$$DKS2 \approx 2^{590}, \tag{23}$$

which is large enough to discourage a brute force attack.

In all the analyzed cases (1., 2.1. and 2.2) modern cryptanalytical methods, like linear cryptanalysys and differential cryptanalysys ([20]), cannot succeed because such methods, even when applicable, have the purpose to deduce only a piece of the key space (a so called "key round") in order to reduce the key dimension, the remaining key space being found by brute force attacks. But for our *SCNLEFC* method we don't have any key rounds, and even if they were applicable, due to the large key spaces in all the analysed cases, the assumed remaining key space is still large enough and cannot be broken by brute force attacks. But even assuming that we can break the method parameters, in order to determine the fixed computing values $v_{j,i}$ we will still need to solve with a high precision a system of complex nonlinear equations with dimension 5120, which is still not practically feasible.

3.2. Periodicity of the Pseudo-Random Sequence

Here we will conservatively evaluate the periodicity of the generated key stream and prove that it is large enough to discourage attacks based on pattern repetitions. By denoting with P the period of the generated pseudo-random sequence and considering the equation (17) we have

$$t_1 > 1024 \times 512^5 = 2^{10} \times (2^9)^5 = 2^{55}. \tag{24}$$

But after a t_1 cycle the combinatorial arrangement of the 7 prime numbers is changed and the number of possible combinations is $C_{75}^{7} = 1984829850 > 2^{30}$ (in reality, combinatorial arrangements are used and their number is much larger), and also considering that each word key stream value computation generates $4 = 2^2$ bytes we have:

$$P > 2^2 \times 2^{30} \times 2^{55} = 2^{87} = 2^{47}\ Tb, \tag{25}$$

which is a relatively large number for a data stream length.

4. Randomness Tests and Results

For testing the randomness of the key stream we used the PractRand (Practically Random) test suite ([21]), version version 0.94 . PractRand is a C++ library implementing a variety of very high quality pseudo-random number generators (*PRNG*s) and statistical tests for *PRNG*s capable to find the widest variety of biases quickly if there are plenty of bits of *PRNG* output. For a testing example we applied the PractRand test suite to a *PRNG* output of 4 megabytes. From a total number of 124 statistical tests 26 reported problems with various degrees of gravity (7 with unusual, 3 with mildly suspicious, 2 with suspicious, 7 with very suspicious and 7 with fail). Considering the relatively large percentage (20.97%) of statistical tests reporting problems we can conclude that the proposed *SCNLEFC* method is not appropriate for use as a Pseudo-Random Number Generator (*PRNG*), but still due to the security considerations from the previous section, it is safe for cryptographic applications.

5. Conclusions

The paper proposed a new stream ciphering method, *SCNLEFC*, based on the non-linearity of elementary function compositions combined with principles from number theory and combinatorics, and the mathematical difficulty to solve with high accuracy large systems of nonlinear equations. Due to the failure of some of the randomness tests from the Practically Random test suite we do not recommend the *SCNLEFC* method as a *PRNG*, but due to the very large key space and large period of the pseudo-random sequence we still consider that the method is safe for stream ciphering applications. Further research will be concerned with designing, implementing and testing a stream ciphering method based on the number theory and combinatorial principles of *SCNLEFC*, but using bitwise operations (instead of floating point operations) for increased computing speed.

REFERENCES

[1] http://www.ecrypt.eu.org/stream, last time accessed in April, 2019.

[2] https://www.cosic.esat.kuleuven.be/nessie, last time accessed in April, 2019.

[3] http://www.ecrypt.eu.org/stream/D.SYM.3-v1.1.pdf, last time accessed in April, 2019.

[4] http://www.ecrypt.eu.org/ecrypt2/documents/D.SYM.10-v1.pdf, last time accessed in April, 2019.

[5] *M. Robshaw, O. Billet (Eds.)*, New Stream Cipher Designs: The eSTREAM Finalists, Springer-Verlag Berlin, Heidelberg 2008.

[6] *S.N. Elaydi*, Discrete Chaos, Chapman & Hall/CRC, 2000.

[7] *N.K. Pakeek, V. Patidar, K.K. Sud*, A Random Bit Generator Using Chaotic Maps, International Journal of Network Security, **10(1)** (2010), 32-38.

[8] *V. Patidar, K.K. Sud, N.K. Pakeek*, A Pseudo Random Bit Generator Based on Chaotic Logistic Map and its Statistical Testing, Informatica, **30** (2009), 441–452.

[9] *R.F. Martinez-Gonzalez, J.A. Diaz-Mendez*, Implementation of a Stream Cipher Based on Bernoulli's Map, International Journal of Computer Science & Information Technology (IJCSIT), **6(6)** (2014), 113-121.

[10] *J. Chauhan, A. Jain*, Survey on Encryption Algorithm Based On Chaos Theory and DNA Cryptography, International Journal of Advanced Research in Computer and Communication Engineering, **3(8)** (2014), 7801-7803.

[11] *G. Alvarez, S.J. Li*, Some Basic Cryptographic Requirements for Chaos-Based Cryptosystems, International Journal of Bifurcation and Chaos, **16(8)**, (2006), 2129-2151.

[12] *R. Drăgănoiu, A. Moldoveanu, A. Brăescu*, The Math of Programming – Interdisciplinary Approach, The International Scientific Conference eLearning and Software for Education (ELSE), Bucharest 2017, vol. 2, pp. 473-481.

[13] *R. Drăgănoiu, A. Brăescu*, DigiMathArt, A New Method of Teaching and Cognitive Evolution, Central and Eastern European Lumen Conference on New Approaches

in Social and Humanistic Sciences, Targoviște 2015, published in the volume New Approaches in Social and Humanistic Sciences (LUMEN NASHS), pp. 205-209.

[14] *R. Drăgănoiu, A. Brăescu*, DigiMathArt – Connecting Math and Art through Programming. A method of Creating new Neural Networks, Proceedings of the International Conference The Future of Education, Florence, Italy, 2015, pp. 227-231.

[15] *R. Drăgănoiu*, Programare Matematica Grafica. Modelare Matematica, Editura Atelier Didactic, Bucureşti 2017, ISBN: 978-606-8923-02-4, pp. 1-147.

[16] *R. Drăgănoiu, A. Moldoveanu, A. Brăescu, M. Mocanu, F. Moldoveanu*, Learning Math by Interactve Visual Modeling, The 14th International Scientific Conference eLearning and Software for Education (ELSE), Bucharest, April 19-20, 2018, DOI:10.12753/2066-026X-18-000.

[17] *G. Anescu*, Key Expansion Method, https://www.researchgate.net/publication/323390 610_Key_Expansion_Method (last time accessed in July, 2018), (2018), 1-7, DOI:10.13140/RG.2.2.36080.35849

[18] *R. Durstenfeld*, Algorithm 235: Random permutation, Communications of the ACM, **7(7)** (1964), 420, DOI:10.1145/364520.364540.

[19] *R.L. Burden, J.D. Faires*, Numerical Analysis, Numerical Solutions of Nonlinear Systems of Equations, 597-640, Thomson Brooks/Cole, 2005.

[20] *C. Swenson*, Modern Cryptanalysis: Techniques for Advanced Code Breaking, Wiley Publishing, 2008.

[21] http://pracrand.sourceforge.net/PractRand.txt, last time accessed in February, 2019.

Appendix A. The Illustrative Set of Primitive Functions

$$p_1(x) = 0.385\left(\frac{4\tan^{-1}(4x^2-1)}{\pi}+1\right) \tag{A.1}$$

$$p_2(x) = 0.525\left(\sin^2\left(\pi(1-x)^2\right)+\sin\left(\frac{\pi(1-x)}{2}\right)\right) \tag{A.2}$$

$$p_3(x) = 0.5\left(\sin(2\pi x)\left(\sin^2(\pi x)+1\right)+x^{0.5}\right)-0.5 \tag{A.3}$$

$$p_4(x) = 3.1(x+0.04)^{1/2}\cos^2\left(\frac{\pi(x+0.2)}{2}\right) \tag{A.4}$$

$$p_5(x) = 0.6275\left(\sin^2(0.85\pi x)+0.6\sin^2\left(1.7\pi x^2\right)\right) \tag{A.5}$$

$$p_6(x) = 0.51\left(\sin^2(0.8x)\sin\left(5x^2\right)+2x^2\right) \tag{A.6}$$

$$p_7(x) = 0.9793\left(-\sin\left(\ln\left(\frac{x}{1.15}+0.1\right)\right)+0.1\sin(9x)-0.0721144\right) \tag{A.7}$$

$$p_8(x) = 1.25545(\tan^{-1}(\ln(x^2+1)+0.3\sin(9x))+0.0710804) \tag{A.8}$$

$$p_9(x) = 0.684(2^x-1+\sin^3(3x)) \tag{A.9}$$

$$p_{10}(x) = 3.55\left(\ln\left(\frac{2}{x+1}\right) - 0.2\sin(5x-5)\right) \tag{A.10}$$

$$p_{11}(x) = -4.27(3x-1)^2\ln(x) + 0.2\sin(14x) \tag{A.11}$$

$$p_{12}(x) = \sin^2(6(x^2-0.64)^2) \tag{A.12}$$

$$p_{13}(x) = 0.541.3^{x+1}\sin^4(5.7(x-1)^2) + 0.4(x-1)^4 + 0.2\sin^2(7x-7) \tag{A.13}$$

$$p_{14}(x) = 0.523\tan^{-1}(3.2x) + 0.4|\sin(7x)| \tag{A.14}$$

$$p_{15}(x) = \sin^{-1}\left(\frac{x}{1.197}\right) + 0.2(1-|\cos(6x)|) \tag{A.15}$$

$$p_{16}(x) = 1.21\left(\tan(x) - x^3 + 0.2\sin(9x) + 0.1|\sin(13x)|\right) \tag{A.16}$$

$$p_{17}(x) = 0.7\sin(\pi x^2) + 0.4\sin(3\pi x)^2 \tag{A.17}$$

$$p_{18}(x) = 0.51\left(\sin(\sin(\pi x^2)) + \sqrt{x} + 0.3\sin(17x)\right) \tag{A.18}$$

$$p_{19}(x) = 3.41\left((x+0.5)(x-0.7)^2 + 0.05\sin^2(17(x-0.7)^2)\right) \tag{A.19}$$

$$p_{20}(x) = 1.7\left(|\tan(x^2-0.49)| + 0.1|\sin(10x-7-\sin(10x-7))|\right) \tag{A.20}$$

$$p_{21}(x) = 0.097\sin(19x - 2.85\sin(5x)) + 0.97x^2 \tag{A.21}$$

$$p_{22}(x) = 0.89\left(\sin^{-1}(0.3(x^3+x+(x)^{1/2})) + 0.1\sin(23x^3)\right) \tag{A.22}$$

$$p_{23}(x) = 0.926\left((1.2-x)(x-0.4)^2 + |\sin(8(x-0.4)(x+0.1))|\right) \tag{A.23}$$

$$p_{24}(x) = 0.0781\left(2\sin(17x^3) + \sin(\sin(19x)) + \sin(2x)\right) \tag{A.24}$$

$$p_{25}(x) = 0.772\left(\sin(\pi(2x-1)) + 0.3\sin(\pi(1-x)(x-2)^2)\right) \tag{A.25}$$

$$\begin{aligned} p_{26}(x) = 0.985\,\big(&-13(x-1)(x+0.5)(x-0.3)^2 - (x-1)x+ \\ &+ 0.1\sin(7.3(x-1)(x-3))) \end{aligned} \tag{A.26}$$

$$p_{27}(x) = \sin\left(\ln\left(\left(\frac{x+0.13}{10.0}\right)^2\right)\right)^2 \tag{A.27}$$

$$p_{28}(x) = 0.765\left(1 + 0.4\sin^4(1.5(x-0.8)(x+5)) - 2^{-9(x-0.8)^2}\right) \tag{A.28}$$

$$\begin{aligned} p_{29}(x) = 1.799\,&\big(0.2\tan^{-1}(\sin(7.3x)) + 0.4\sin^{-1}(\tan(x/2)) + \\ &+ x(x-1)(x-2)\big) \end{aligned} \tag{A.29}$$

$$p_{30}(x) = 0.5\left(|\sin(13^{x-0.5} - 1)| + |\sin((x-0.5)(x-13))|\right) \tag{A.30}$$

Appendix B. The Illustrative Set of Unary Operators

The primitive function $p(x)$ is determined at run-time.

$$\text{Identity: } u_1(x) = p(x) \tag{B.1}$$

$$\text{Negative: } u_2(x) = 1 - p(x) \tag{B.2}$$

$$\text{Inverse 1: } u_3(x) = \frac{1-p(x)}{1+p(x)} \tag{B.3}$$

$$\text{Inverse 2: } u_4(x) = \frac{2p(x)}{1+p(x)} \tag{B.4}$$

$$\text{Sinus: } u_5(x) = \sin\left(\frac{\pi p(x)}{2}\right) \tag{B.5}$$

$$\text{Cosinus: } u_6(x) = \cos(\frac{\pi p(x)}{2}) \tag{B.6}$$

$$\text{Tangent: } u_7(x) = \tan(\frac{\pi x}{4}) \tag{B.7}$$

$$\text{Cotangent: } u_8(x) = \frac{1}{\tan\left(\frac{\pi(p(x)+1)}{4}\right)} \tag{B.8}$$

$$\text{Arcsin: } u_9(x) = \frac{\sin^{-1}(2p(x)-1)}{\pi} + 0.5 \tag{B.9}$$

$$\text{Arccos: } u_{10}(x) = \frac{\cos^{-1}(2p(x)-1)}{\pi} \tag{B.10}$$

$$\text{Arctangent: } u_{11}(x) = \frac{\tan^{-1}(2p(x)-1)}{\frac{\pi}{2}} + 0.5 \tag{B.11}$$

$$\text{Arccotangent: } u_{12}(x) = 0.5 - \frac{\tan^{-1}(2p(x)-1)}{\frac{\pi}{2}} \tag{B.12}$$

$$\text{Exponential: } u_{13}(x) = \frac{e^{p(x)}-1}{e-1} \tag{B.13}$$

$$\text{Logarithm 1: } u_{14}(x) = \ln\left(1+(e-1)p(x)\right) \tag{B.14}$$

$$\text{Logarithm 2: } u_{15}(x) = \frac{\ln\left(1-p(x)\right)}{\ln(p(x))} \tag{B.15}$$

$$\text{Square: } u_{16}(x) = p^2(x) \tag{B.16}$$

$$\text{Square Root: } u_{17}(x) = (p(x))^{1/2} \tag{B.17}$$

$$\text{Power: } u_{18}(x) = p(x)^{\sqrt{a}} \tag{B.18}$$

Appendix C. The Illustrative Set of Binary Operators

For all the binary operators:

$$a = (a_l a_r)^{1/2} \tag{C.1}$$

Weighted Arithmetic Mean:

$$b_1(y) = \frac{a_l f_l(y) + a_r f_r(y)}{a_l + a_r} \tag{C.2}$$

$$y_1 b_1 y_2 = b_1(y_1, y_2) = \frac{a_l y_1 + a_r y_2}{a_l + a_r} \tag{C.3}$$

Weighted Geometric Mean:

$$b_2(y) = f_l(y)^{\frac{2a_l}{a_l+a_r}} f_r(y)^{\frac{2a_r}{a_l+a_r}} \tag{C.4}$$

$$y_1 b_2 y_2 = b_2(y_1, y_2) = y_1^{\frac{2a_l}{a_l+a_r}} y_2^{\frac{2a_r}{a_l+a_r}} \tag{C.5}$$

Weighted Harmonic Mean:

$$b_3(y) = \frac{(a_l + a_r) f_l(y) f_r(y)}{a_l f_r(y) + a_r f_l(y)} \tag{C.6}$$

$$y_1 b_3 y_2 = \frac{(a_l + a_r) y_1 y_2}{a_l y_2 + a_r y_1} \tag{C.7}$$

Weighted Composition (with Arithmetic Mean):

$$b_4(y) = \frac{a_l f_l(f_r(y)) + a_r f_r(f_l(y))}{a_l + a_r} \tag{C.8}$$

$$y_1 b_4 y_2 = \frac{a_l f_l(y_2) + a_r f_r(y_1)}{a_l + a_r} \tag{C.9}$$

Weighted Power (with Arithmetic Mean):

$$b_5(y) = \frac{a_l f_l(y)^{f_r(y)} + a_r f_r(y)^{f_l(y)}}{a_l + a_r} \tag{C.10}$$

$$y_1 b_5 y_2 = b_5(y_1, y_2) = \frac{a_l y_1^{y_2} + a_r y_2^{y_1}}{a_l + a_r} \tag{C.11}$$

Weighted Composition (with Geometric Mean):

$$b_6(y) = f_l(f_r(y))^{\frac{2a_l}{a_l+a_r}} f_r(f_l(y))^{\frac{2a_r}{a_l+a_r}} \tag{C.12}$$

$$y_1 b_6 y_2 = b_6(y_1, y_2) = f_l(y_2)^{\frac{2a_l}{a_l+a_r}} f_r(y_1)^{\frac{2a_r}{a_l+a_r}} \tag{C.13}$$

Weighted Power (with Geometric Mean):

$$b_7(y) = f_l(y)^{\frac{2a_l f_r(y)}{a_l+a_r}} f_r(y)^{\frac{2a_r f_l(y)}{a_l+a_r}} \tag{C.14}$$

$$y_1 b_7 y_2 = b_7(y_1, y_2) = y_1^{\frac{2a_l y_2}{a_l+a_r}} y_2^{\frac{2a_r y_1}{a_l+a_r}} \tag{C.15}$$

Weighted Composition (with Harmonic Mean):

$$b_8(y) = \frac{(a_l + a_r) f_l(f_r(y)) f_r(f_l(y))}{a_l f_r(f_l(y)) + a_r f_l(f_r(y))} \tag{C.16}$$

$$y_1 b_8 y_2 = b_8(y_1, y_2) = \frac{(a_l + a_r) f_l(y_2) f_r(y_1)}{a_l f_r(y_1) + a_r f_l(y_2)} \tag{C.17}$$

Weighted Power (with Harmonic Mean):

$$b_9(y) = \frac{(a_l + a_r) f_l(y)^{f_r(y)} f_r(y)^{f_l(y)}}{a_l f_l(y)^{f_r(y)} + a_r f_r(y)^{f_l(y)}} \tag{C.18}$$

$$y_1 b_9 y_2 = b_9(y_1, y_2) = \frac{(a_l + a_r) y_1^{y_2} y_2^{y_1}}{a_l y_1^{y_2} + a_r y_2^{y_1}} \tag{C.19}$$