Parallel Data Encryption Based on Tinkerbell and Ikeda Chaotic Functions

Borislav Stoyanov [1] and Dimitar Dobrev [1]

[1] *Konstantin Preslavsky University of Shumen, 115 Universitetska Str., Shumen, 9700, Bulgaria*

Abstract

In this paper, we present a parallel encryption algorithm based on the Tinkerbell and Ikeda chaotic functions in order to meet the multi-core processor. The parallel algorithm is constructed with a data parallel approach. The experimental outputs show that the parallel encryption proposes more efficient performance against the serial execution of the scheme.

Keywords

Parallel encryption, Tinkerbell chaotic function, Ikeda chaotic function, NIST statistical tests, PractRand statistical tests, ENT statistical tests

1. Introduction

The enchanting knowledge in data processing and communication interchange have created a high demand for the secure of multimedia communication over the Internet. A major challenge is to protect confidentiality for the information in the networks. Modern ciphers like RC4 [1], Salsa20 [2], and SEAL [3] are constructed with confusion and diffusion characteristics. These two characteristics can likewise be found in chaotic functions, which are normally recognized by sensitive dependence on initial parameters and by having evolution through phase space that appears to be perfectly random. As of late, various chaos-based cryptographic encryption scheme have been presented [4, 5, 6]. In [7], a novel pseudo-random number generator (PRNG), based on a novel logarithmic chaotic map is presented. Two Tinkerbell functions are used to novel image encryption scheme designed in [8].

Parallel computations are used to make encryption quicker and less expensive. Parallel encryption algorithm for multi-core processor based on logistic map are presented in [9, 10, 11, 12, 13]. In this study, parallel calculations with multi-core CPU is used to gain both time execution and security level of the designed

Information Systems & Grid Technologies: Fifteenth International Conference ISGT'2022, May 27–28, 2022, Sofia, Bulgaria
EMAIL: borislav.stoyanov@shu.bg (B. Stoyanov); d.d.dobrev@shu.bg (D. Dobrev)
ORCID: 0000-0002-7307-5914 (B. Stoyanov); 0000-0001-5789-9115 (D. Dobrev)

encryption scheme. Inspired from [14], the main contributions of our study can be summarized as follows:

- we present novel pseudo-random byte generation scheme based on Tinkerbell and Ikeda chaotic functions, which has good statistical characteristics;
- we design novel parallel encryption algorithm with high level of security.

2. New pseudo-random byte generator based on Tinkerbell and Ikeda chaotic functions

2.1. Description of the chaotic functions

The Tinkerbell map [15] is a two-dimensional discrete-time dynamical system given by:

$$x_{n+1} = x_n^2 - y_n^2 + ax_n + by_n \tag{1}$$

$$y_{n+1} = 2x_n y_n - cx_n + dy_n, \tag{2}$$

where $a = 0.9$, $b = 0.6013$, $c = 2.0$ and $d = 0.50$.

$$z_{n+1} = 1 + u(\cos t_n + w_n \sin t_n) \tag{3}$$

$$w_{n+1} = u(z_n \sin t_n + w_n \cos t_n) \tag{4}$$

The two-dimensional Ikeda chaotic function is given by [16, 17]:

where u is a parameter $u = 0.7941$ and

$$t_n = 0.4 - \frac{6}{1 + z_n^2 + w_n^2} \tag{5}$$

2.2. New pseudo-random byte generation algorithm

The proposed generator consists of the next steps:

1. The initial values x_0, y_0, z_0, and w_0 from Tinkerbell and Ikeda chaotic maps are determined.
2. The two chaotic maps are iterated for J and K times, respectively.
3. The iteration continues, and as a result, four real fractions are generated and postprocessed as positive bytes.
4. Perform XOR operation between x_i, y_i, z_i, and w_i to get a single output byte.
5. Return to Step 2 until the byte output is reached.

2.3. Performance evaluation

In case of serial encryption, the initial key is a set of all initial values of the proposed pseudo-random generator. There are four 64 bits floating-point values by 51 significant bits each [18]. The initial seed of the proposed byte generation algorithm is 204 bits. This is sufficiently high against exhaustive search [19].

In order to estimate pseudo-randomness of the output bytes, we used the statistical programs NIST [20], PractRand [21], and ENT [22].

The NIST application consists 15 statistical tests: frequency, block frequency, cumulative sums forward and reverse, runs, longest run of ones, rank, spectral, non-overlapping templates, overlapping templates, universal, approximate entropy, serial first and second, linear complexity, random excursion, and random excursion variant. For the NIST tests, we generated 1000 different byte streams of length 125,000 bytes each. The outputs from the tests are given in Table 1.

Table 1
NIST test application results

Test	*P*-value	Pass rate
Frequency (monobit)	0.643366	993/1000
Block-frequency	0.193767	990/1000
Cumulative sums (Forward)	0.814724	993/1000
Cumulative sums (Reverse)	0.870856	995/1000
Runs	0.310049	984/1000
Longest run of Ones	0.332970	995/1000
Rank	0.009810	996/1000
FFT	0.104371	990/1000
Non-overlapping templates	0.506391	990/1000
Overlapping templates	0.977480	982/1000
Universal	0.867692	990/1000
Approximate entropy	0.637119	983/1000
Random-excursions	0.654500	619/627
Random-excursions Variant	0.465608	620/627
Serial 1	0.994250	994/1000
Serial 2	0.504219	991/1000
Linear complexity	0.433590	996/1000

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 980 for a sample size = 1000 byte streams. The minimum pass rate for the random excursion (variant) test is approximately = 613 for a sample size = 627 byte streams. The designed pseudo-

random byte generation algorithm passed successfully all the NIST statistical tests.

The PractRand application [21] outputs 34 *p*-values and they are in the appropriate (0,1] interval, Table 2 and Table 3. The proposed new pseudo-random byte generator passed successfully all of PractRand tests.

Table 2
PractRand test application results-I

Test	*P*-value	Pass rate
BCFN(2,13):!	R= +0.0	"pass"
BCFN(2+0,13-2)	R= -4.4	$p = 0.970$
BCFN(2+1,13-3)	R= -1.4	$p = 0.715$
BCFN(2+2,13-3)	R= -2.8	$p = 0.873$
BCFN(2+3,13-4)	R= +2.1	$p = 0.190$
BCFN(2+4,13-5)	R= -3.3	$p = 0.923$
BCFN(2+5,13-5)	R= -3.3	$p = 0.925$
BCFN(2+6,13-6)	R= -1.8	$p = 0.770$
BCFN(2+7,13-6)	R= +0.8	$p = 0.335$
BCFN(2+8,13-7)	R= +1.4	$p = 0.252$
BCFN(2+9,13-8)	R= -0.4	$p = 0.499$
DC6-9x1Bytes-1	R= -2.0	$p = 0.933$
Gap-16:!	R= +0.0	"pass"
Gap-16:A	R= +1.9	$p = 0.171$
Gap-16:B	R= +0.9	$p = 0.270$

Table 3
PractRand test application results-II

Test	*P*-value	Pass rate
Low1/8 BCFN(2,13):!	R= +0.0	"pass"
Low1/8 BCFN(2+0,13-4)	R= +0.0	p = 0.474
Low1/8 BCFN(2+1,13-5)	R= +0.4	p = 0.401
Low1/8 BCFN(2+2,13-5)	R= +0.7	p = 0.363
Low1/8 BCFN(2+3,13-6)	R= -2.6	p = 0.868
Low1/8 BCFN(2+4,13-6)	R= -1.4	p = 0.696
Low1/8 BCFN(2+5,13-7)	R= -3.0	p = 0.925
Low1/8 BCFN(2+6,13-8)	R= -1.2	p = 0.664
Low1/8 DC6-9x1Bytes-1	R= -1.8	p = 0.906
Low1/8 Gap-16:!	R= +0.0	"pass"
Low1/8 Gap-16:A	R= -1.5	p = 0.940
Low1/8 Gap-16:B	R= -0.0	p = 0.493
Low4/32 BCFN(2,13):!	R= +0.0	"pass"
Low4/32 BCFN(2+0,13-4)	R= -5.2	p =1-9.6e-3

Low4/32 BCFN(2+1,13-5)	R= +1.4	p = 0.269
Low4/32 BCFN(2+2,13-5)	R= -2.7	p = 0.873
Low4/32 BCFN(2+3,13-6)	R= -2.0	p = 0.791
Low4/32 BCFN(2+4,13-6)	R= +2.2	p = 0.169
Low4/32 BCFN(2+5,13-7)	R= +0.6	p = 0.349
Low4/32 BCFN(2+6,13-8)	R= +2.2	p = 0.160
Low4/32 DC6-9x1Bytes-1	R= +3.1	p = 0.116
Low4/32 Gap-16:!	R= +0.0	"pass"
Low4/32 Gap-16:A	R= +2.7	p = 0.080
Low4/32 Gap-16:B	R= -1.5	p = 0.846

The ENT software [22] is a set of 6 tests. The proposed pseudo-random byte algorithm passed successfully ENT tests, Table 4.

Table 4
ENT test application results

Test	Result
Entropy	7.999998 bits per byte
Optimum compression	OC would reduce the size of this 125000000-byte file by 0 %
χ^2 distribution	For 125000000 samples is 262.30, and randomly would exceed this value 36.33 % of the time
Arithmetic mean value	127.5048 (127.5 = random)
Monte Carlo π estim.	3.3.141320114 (error 0.01 %)
Serial correl. coeff.	-0.0000091 (totally uncorrelated = 0.0)

3. New algorithm of parallel encryption

3.1. New parallel data encryption

The new pseudo-random byte generator based on Tinkerbell and Ikeda chaotic functions is used in the construction of the parallel encryption algorithm. The proposed parallel encryption algorithm consists of the next steps:

1. The input data file is divided into p fragments of the available cores (threads).
2. The initial values of p pseudo-random generators (from Section 2.2) based on Tinkerbell and Ikeda chaotic functions, $1 \le j \le p$, x_{0j}, y_{0j}, z_{0j}, and w_{0j} are determined.
3. The p different chaotic pseudo-random generators are iterated each one on a thread, and as a result p different pseudo-random bytes are obtained simultaneously.
4. Perform XOR operation between the p pseudo-random bytes and p bytes from each part of the fragmented input data file.

5. Return to Step 3 until the input byte parts are encrypted.
6. The encrypted parts from all threads are merged in output file.

3.2. Performance evaluation

In case of parallel encryption, the initial key space is a set of p different initial keys of p byte generation algorithms, Section 2.2. Consequently, the overall key space of the parallel scheme is $p \times 204$ bits. For example, when 8 threads are used, the initial seed will be equal to 1632 bits.

For the performance evaluation in the parallel case, tests with different thread numbers are performed. The laptop CPU in the experiment is Intel(R) Core(TM) i7-8550U CPU@1.80 GHz 2.00 GHz, 4 cores, 8 threads, hyper-threading activated; RAM: 8 GB; cache: 8 MB. The speed-up $S(p)$ on p threads of the presented parallel algorithm is given by the equation [23]:

$$E = \frac{T(1)}{T(p)'} \tag{6}$$

Table 5 presents the encryption results and speed-up as the number of threads increase. The encryption is conducted with p initial keys, as p separate parts are encrypted with a different key. Apparently, that with 8 cores the proposed algorithm achieves its maximum speed-up. The experiments were performed with 9.92 MB data file.

The results of the security and execution outputs are summarized in Table 6. Using the given experimental findings, we can complete that the novel parallel scheme, based on the Tinkerbell and Ikeda maps, has acceptable statistical properties and provide sufficient data security. One such use of our novel algorithm in the image and audio encryption techniques. More blocks of the encryption image can be processed at the same time. Another possible application is the real-time selective video encryption. More sensitive layers can be encrypted in parallel with our proposed scheme.

Table 5
Experimental outputs of encryption time of a file with varying number of threads, the corresponding speed-up, and efficiency.

Threads	Encryption time (s)	Throughput (MB/s)	Speed-up	Efficiency
1	3.61	2.75	–	–
2	2.12	4.68	1.70	0.85
4	1.53	6.48	2.36	0.59
8	1.24	8.00	2.90	0.36

Table 6
Comparison of our algorithm with other closely related techniques.

Algorithm	Entropy	Throughput (MB/s)	Key size (b)	Correlation coefficient
Proposed	7.9999	8.00	1632	-0.0000091
Ref. [13]	7.9979	6.48	505	0.00048
Ref. [14]	7.9972	-	888	0.0052

4. Summary

We have proposed a novel pseudo-random byte generation scheme based on the Tinkerbell and Ikeda functions. The parallel algorithm is created with a data parallel method. The experimental outputs show that the parallel encryption presents more efficient outputs against the serial execution of the algorithm.

5. Acknowledgments

This work is partially supported by the Scientific research fund of University of Shumen, Bulgaria, under the grant No. RD-08-110/22.02.2022.

6. References

[1] J. Katz, Y. Lindell, Introduction to modern cryptography, CRC press, 2020.
[2] D. J. Bernstein, The Salsa20 Family of Stream Ciphers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 84–97. doi:10.1007/978-3-540-68351-3_8.
[3] P. Rogaway, D. Coppersmith, A software-optimized encryption algorithm, J. Cryptol. 11 (1998) 273–287. doi:10.1007/ s001459900048.
[4] N. Munir, M. Khan, M. M. Hazzazi, A. Aljaedi, A. A. K. Haj Ismail, A. R. Alharbi, I. Hussain, Cryptanalysis of internet of health things encryption scheme based on chaotic maps, IEEE Access 9 (2021) 105678–105685. doi:10.1109/ACCESS.2021.3099004.
[5] G. He, W. Wu, L. Nie, J. Wen, C. Yang, W. Yu, An improved image multi-dimensional chaos encryption algorithm based on cuda, in: 2019 9th International Conference on Information Science and Technology (ICIST), 2019, pp. 183–187. doi:10.1109/ICIST.2019.8836920.
[6] J. Luo, H. Shi, Research of chaos encryption algorithm based on logistic mapping, in: 2006 International Conference on Intelligent Information Hiding and Multimedia, 2006, pp. 381–383. doi:10.1109/IIH-MSP.2006.265022.
[7] N. Nesa, T. Ghosh, I. Banerjee, Design of a chaos-based encryption

scheme for sensor data using a novel logarithmic chaotic map, Journal of Information Security and Applications 47 (2019) 320–328. doi:10.1016/j.jisa.2019.05.017.

[8] P. R. Krishna, C. Surya Teja, R. D. S., T. V., A chaos based image encryption using tinkerbell map functions, in: 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), 2018, pp. 578–582. doi:10.1109/ICECA.2018.8474891.

[9] J. Liu, D. Song, Y. Xu, A parallel encryption algorithm for dual-core processor based on chaotic map, in: S. S. Mahmoud, Z. Zeng, Y. Li (Eds.), Fourth International Conference on Machine Vision (ICMV 2011): Computer Vision and Image Analysis; Pattern Recognition and Basic Technologies, volume 8350, International Society for Optics and Photonics, SPIE, 2012, pp. 73 – 79. doi:10.1117/12.920226.

[10] W. Wang, X. Wang, D. Song, A parallel chaotic cryptosystem for dual-core processor, in: The 2nd International Conference on Information Science and Engineering, 2010, pp.920–923. doi:10.1109/ICISE.2010.5689747.

[11] J. Liu, H. Zhang, D. Song, G. Sun, W. Bi, M. K. Buza, A parallel encryption algorithm of the logistic map for multicore with openmp, in: Ifost, volume 2, 2013, pp. 47–50. doi:10.1109/IFOST.2013.6616857.

[12] D. Burak, Parallelization of the block encryption algorithm based on logistic map, Przegląd Elektrotechniczny 88 (2012) 198–200.

[13] M. J. Rostami, A. Shahba, S. Saryazdi, H. Nezamabadi-pour, A novel parallel image encryption with chaotic windows based on logistic map, Computers & Electrical Engineering 62 (2017) 384–400. doi:10.1016/j.compeleceng.2017.04.004.

[14] Ü. Çavuşoğlu, S. Kaçar, A novel parallel image encryption algorithm based on chaos, Cluster Computing 22 (2019) 1211–1223. doi:10.1007/s10586-018-02895-w.

[15] K. T. Alligood, T. D. Sauer, J. A. Yorke, D. Chillingworth, Chaos: an introduction to dynamical systems, Springer, New York, 1996.

[16] K. Ikeda, Multiple-valued stationary state and its instability of the transmitted light by a ring cavity system, Optics Communications 30 (1979) 257–261. doi:10.1016/0030-4018(79)90090-7.

[17] K. Ikeda, H. Daido, O. Akimoto, Optical turbulence: Chaotic behavior of transmitted light from a ring cavity, Physical Review Letters 45 (1980) 709–712. doi:10.1103/PhysRevLett.45.709.

[18] F.-P. W. Group, IEEE standard for floating-point arithmetic, IEEE Std 754-2019 (Revision of IEEE 754-2008) (2019) 1–84. doi:10.1109/IEEESTD.2019.8766229.

[19] G. Alvarez, S. Li, Some basic cryptographic requirements for chaos-based cryptosystems, International Journal of Bifurcation and Chaos 16 (2006)

2129–2151. doi:10.1142/S0218127406015970.

[20] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, A statistical test suite for random and pseudorandom number generators for cryptographic application, 2010. NIST Special Publication 800-22: Revision 1a, Lawrence E. Bassham III. doi:10.5555/2206233.

[21] C. Doty-Humphrey, Practrand: C++ library of pseudo-random number generators and statistical tests for rngs (2014). URL: http://pracrand.sourceforge.net.

[22] J. Walker, ENT: A pseudorandom number sequence test program, 2008. URL: http:// www.fourmilab.ch/random/.

[23] G. M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS '67 (Spring), Association for Computing Machinery, New York, NY, USA, 1967, p. 483–485. doi:10.1145/1465482.1465560.