ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **21**, Number 1, 2018, 18–33

Dynamically changing the secret key of an FPGA chaos-based cipher

Octaviana Datcu[1], Radu Hobincu[1], and Lucian Petrică[1]

[1]Politehnica University of Bucharest, Bucharest, Romania
Email: od@elcom.pub.ro

Abstract. As a continuation of previous work, where a discrete-time Baptista-type chaos-based cipher was implemented in a Xilinx FPGA module, this paper proposes a more complex scheme involving hybrid dynamics. A continuous chaotic evolution is utilized to dynamically and randomly change the secret key of the discrete-time part. This work describes the design of a continuous-time chaotic transmitter meant to embed the secret key as well as a high order sliding mode differentiator also designed such that it is capable of estimating the dynamics of the transmitter, and Baptista's algorithm secret key, conceptualized as its unknown input.

Key-words: Baptista type algorithms; chaos-based cryptography; high order sliding mode observers; estimation.

1. Introduction

In a previous work [1] a Baptista's chaos-based cryptosystem [2] was implemented in a Field Programmable Gate Array. This paper improves the communication scheme by adding an additional key-exchange procedure involving a continuous-time chaotic Colpitts oscillator as a transmitter of its secret key and a high order sliding mode differentiator [3] on the receiving end, designed to estimate the states of the transmitter and recover the data. The data is then used as a decryption key.

Chaos systems are based on chaotic mathematical functions that have the property that they have a very strong dependency on the initial values, meaning that even a miniscule variation in the initial conditions will trigger a completely different system evolution. Their definitions are often recursive [10], [11], [9] and the dynamics of the systems are so complex that they can not be predicted. They are considered to be effectively pseudo-random. These properties make chaotic systems an interesting alternative to classic cryptography and pseudo-random number generators [13], [12].

One of the more famous albeit not very performant chaos-based cipher is Baptista's implementation that uses the Logistic map given in (1) as a chaotic function, with a and b real parameters chosen such that the chaotic behavior is enabled. There have been many publications that criticise the security of the cipher, one of the main complaints being that the distribution of the cipher text is not uniform in its definition space. We have published [14] in which the authors propose a modification of the cipher that will address this issue. This paper will use the modified and improved version. The encryption and decription algorithms themselves will be described in sections 2.5. and 2.6..

$$x[k+1] = bx[k](1 - x[k]) \tag{1}$$

One other important issue to address is the key space. The Logistic Map has two parameters that influence the dynamics of the system: the parameter b and the initial value $x[0]$. Baptista has used both these values as keys and since both have a strict definition interval for which the system manifests chaotic behaviour, the size of key space is strictly related to the precision of the system. As an initial proof of concept, we will use the key-exchange mechanism to only transfer the value of b from the transmitter to the receiver and we will consider $x[0]$ fixed. We will show that the size of the key space is directly linked to the quality of the estimator described in section 5.

Next, we will present the system implementation in section 2., we will show simulation results for an encryption of a data block in section 2.7. and we will finish with conclusions and suggestions for future work in section 3..

2. Implementation

2.1. System Overview

The proposed chaos based secret communication scheme has several components that will be described in the following sections. The system diagram is shown in figure 1. The system behaviour is described below.

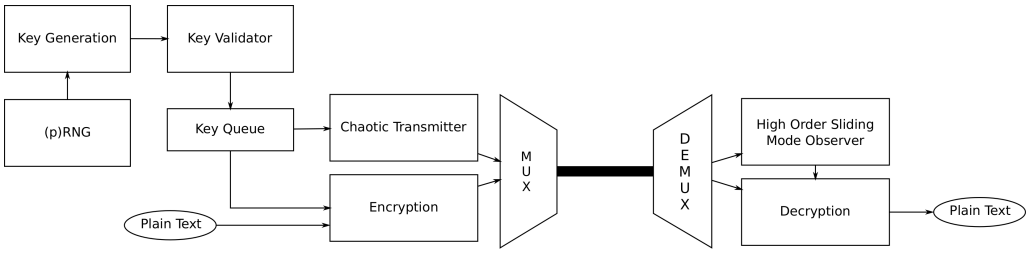

Fig. 1. System diagram

The Random Number Generator (RNG) block is used to create random keys for the system. This RNG will generate uniform values in range $[0; 1]$. For now, a state-of-the art RNG can be used, but the authors plan on designing a chaotic based RNG.

The key used for the chaos based encryption is the value for parameter b of the Logistic Map (1). We will show in section 2.2. that values for b from 0 to 3.57 are not useful, because

they do not exibit chaotic behaviour, thus a key needs to be selected in range $[3.57; 4]$. The *Key Generation* block translates the RNG value to the required interval.

However, there are subintervals in $[3.57; 4]$ that are still not suited for encryption. The role of the *Key Validator* block is to analyze the generated value for b and decide if it is secure enough to use in the encryption stage. If the value is useful, it is written to the *Key Queue*, else it is discarded.

Every 1024 bytes (the chosen size for the data block), a new key is extracted from the *Key Queue* and sent by the *Chaotic Transmitter* (described in section 2.3.) to the receiver side. The key is also used by the *Encryption* block to encipher data which is then transmitted to the receiver side. The channel is shared by the key and data transmission through a pair of mux-demux.

On the receiver side, the *High Order Sliding Mode Observer* (described in section 5) will recover the key and send it to the *Decryption* stage which will use it to decipher the following kilobyte of data.

2.2. Key Generation and Validation

The encryption key that is transmitted through the analog Colpits oscilators consists of a fractional number: the value for the Logistic map parameter b. Technically, the interval for this parameter is $(0; 4)$, however, some subintervals show non-chaotic behavior which is not suited for cryptographic applications. Figure 2 shows the distribution of output for $b \in (0; 3.6]$ - the bifurcation diagram. There can easily be seen that for values of $b \in (0; 3]$, x only has one solution and for $b \in (3; 3.5]$, it has two. Even for $b \in (3.57; 3.678]$ shown in figure 3, there can be seen a large gap for x around 0.7. This is problematic because due to the nature of the Baptista's cipher, which is used for encryption, there will be some values of the plain text that will be impossible to encrypt.

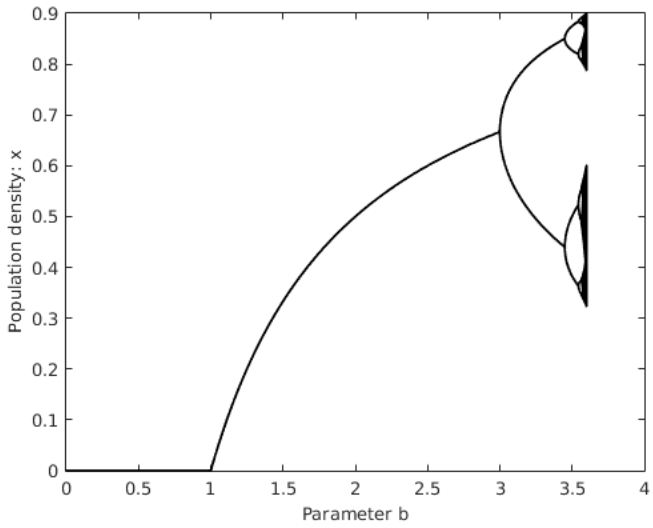

Fig. 2. Bifuration diagram for b in range (0; 3.6)

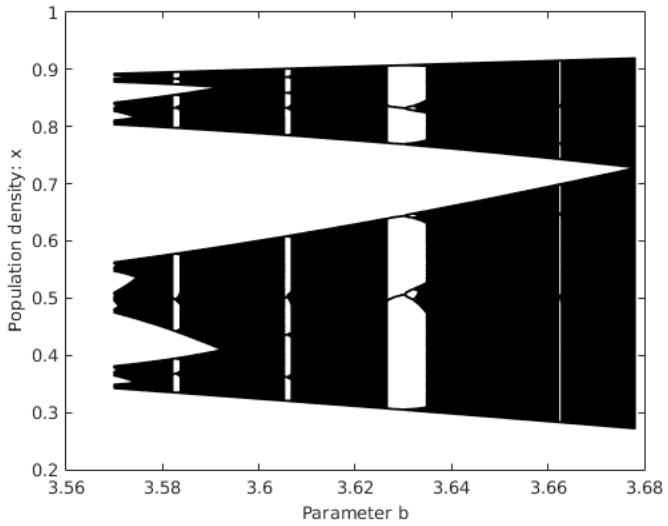

Fig. 3. Bifuration diagram for b in range (3.57; 3.678)

Starting with $b = 3.678$, there is a continuous interval for x between 0.3 and 0.9 that we can use, as shown in figure 4. b still has some gaps though: a small one around 3.75 and a large one around 3.85. It is possible to zoom in and analyse the range for b between 3.678 and 3.738 which appears continuous in figure 4. Figure 5 however shows an additional gap at $b \approx 3.702$. This is because the distribution of the x values follows the rules of a fractal function: zooming in will produce new patterns with new gaps, making it even more difficult to choose a valid key.

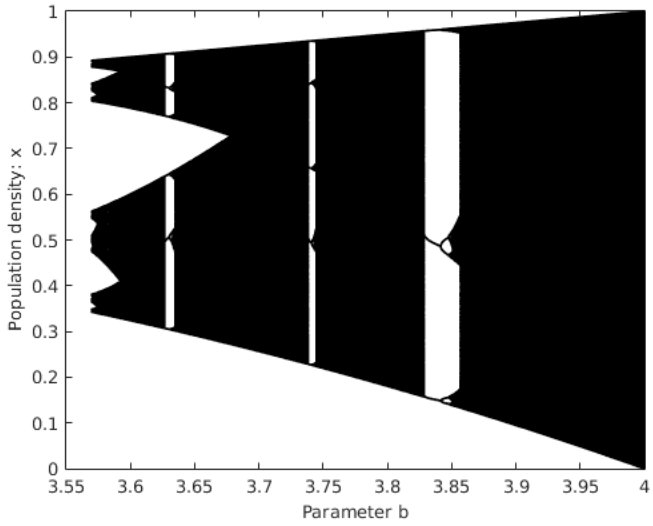

Fig. 4. Bifuration diagram for b in range (3.57; 4.00]

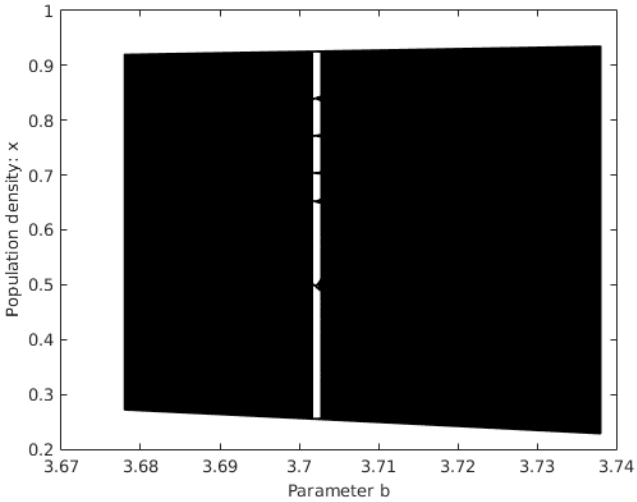

Fig. 5. Bifuration diagram for b in range (3.678; 3.738)

In order to address this issue, a process different from encryption will attempt to generate and validate key values. Fortunately, due to the nature of the key exchange algorithm, these keys don't need to be reproducible on the receiver side, since they are transmitted. They don't even need to be reproducible on the transmitter side since they are not generated from a password. In this case, a completely random sequence if values for b will actually increase the security for the system. Validation for the generated keys imply checking if the value doesn't fall into one of the gaps where the distribution of output values has a very small number of items. To check for this, there are two options: a) compute the Lyapunov exponent and check if it positive, or b) compute the bifurcation diagram by skipping the transitory time of 250 iterations and then advance the map a large number of times (we have chosen 1000) and check if the number of different values is close to 1000.

Because the second option implies a level of uncertainty, we have chosen to implement the *Key Validation* block using the Lyapunov exponents. The expression for this exponent is given in (3).

$$\lambda(x,k) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \log |x'[k_i]| \tag{2}$$

with x being the Logistic map given in (1). Replacing x we obtain

$$\lambda(x,k) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \log |b(1-2k_i)| \tag{3}$$

In practice, a value for n must of course be finite, and the system will increase i until the exponent converges (the new term is less than 10^{-9}). If the result is greater than 0, then the value of b is appropriate for encryption. It was shown that there is a dense set of parameters for which

the Lyapunov exponent is negative. This means, we don't find a single interval in $[0; 4]$ on which the Lyapunov exponent is positive, which affirms the fractal nature of the bifurcation diagram.

2.3. Chaotic Transmitter

The secret communication key, called b in the remaining of the paper, is embedded using the inclusion method [15], in the output signal of a Colpitts oscillator, configured in such a way that this output signal exhibits chaotic behaviour. The output is sampled by a analog-to-digital converter and multiplexed to the output channel. The modeling equations are given in (4). The gain G was added only to differentiate between the evolution of the Colpitts oscillator with ($G = 1$) and without ($G = 0$) the secret key included.

$$\begin{aligned} \dot{y}_1 &= A(-e^{-y_2} + y_3 + 1) + Gb \\ \dot{y}_2 &= Ay_3 \\ \dot{y}_3 &= -\frac{k}{A}(y_1 + y_2) + By_3 \end{aligned} \tag{4}$$

Some particular values for parameters $A = g/[q(1-k)]$ and $B = -1/Q$ are given in Table 1 in correspondence with the behavior manifested by the Colpitts oscillator, indicated by the sign of the Lyapunov exponents computed according to the algorithm in [8]. At least one positive exponent marks the existence of aperiodic, dense trajectories and small perturbation sensitive, thus pseudo-random (chaotic), dynamics.

Table 1. Lyapunov exponents for the Colpitts oscillator.

g	Q	λ_1	λ_2	λ_3
0.46	1.38	-0.1392	-0.1551	-0.4298
0.80	1.38	-0.0844	-0.0745	-0.5767
1.46	1.38	-0.5185	-0.2548	-0.4179
2.46	1.38	0.1339	-0.3597	-0.4988
3.46	1.38	0.1564	-0.4227	-0.4583
4.46	1.38	0.1867	-0.4043	-0.5069
4.46	0.38	0.1368	-0.2959	-2.4724
4.46	0.50	0.1544	-0.3673	-0.6619
4.46	1.00	0.2033	-0.5414	-0.6619
4.46	5.00	-0.0169	-0.0203	-0.1626

2.4. The High Order Sliding Mode Observer

The receiver, a high order sliding mode observer [5], from now on abreviated HOSMO, for the sake of simplicity, is given by (5). The information HOSMO dynamically gets from the transmitter is contained in the signal y_2.

$$\begin{aligned} z_1 &= y_2 \\ z_2 &= \dot{y}_2 = Ay_3 \\ z_3 &= \ddot{y}_2 = A\dot{y}_3 = -k(y_1 + y_2) + ABy_3 \\ z_4 &= \dddot{y}_2 = -k(\dot{y}_1 + \dot{y}_2) + AB\dot{y}_3 = \\ &= -k\dot{y}_1 - kz_2 + Bz_3 = \\ &= kAe^{-z_1} - 2kz_2 + Bz_3 - kA - kb \end{aligned} \tag{5}$$

From system (5), the estimated states are as in:

$$\begin{aligned} \hat{y}_2 &= z_1 \\ \hat{y}_3 &= z_2/A \\ \hat{y}_1 &= \frac{-kz_1 + Bz_2 - z_3}{k} \\ \hat{b} &= Ae^{-z_1} - 2z_2 + \frac{B}{k}z_3 - \frac{1}{k}z_4 - A \end{aligned} \tag{6}$$

Therefore, HOSMO's main task is to reproduce the evolution of the transmitter, by only knowing the vector of values corresponding to the signal y_2. It uses the dynamical behavior of system (5), as in:

$$\begin{aligned} \dot{z}_1 &= z_2 + C_1 \\ \dot{z}_2 &= z_3 + C_2 \\ \dot{z}_3 &= z_4 + C_3 \\ \dot{z}_4 &= z_5 + C_4 \\ \dot{z}_5 &= kAe^{-z_1}({z_2}^2 - z_3) - 2kz_4 + Bz_5 + C_5 \end{aligned} \tag{7}$$

where C_1, C_2, C_3, C_4 are called correction factors as described in [5] and expressed by:

$$\begin{aligned} C_1 &= -8M^{1/5}|z_1 - y_2|^{4/5}\text{sign}[L(z_1 - y_2)] \\ C_2 &= -5M^{1/4}|z_2 - \dot{z}_1|^{3/4}\text{sign}[L(z_2 - \dot{z}_1)] \\ C_3 &= -3M^{1/3}|z_3 - \dot{z}_2|^{2/3}\text{sign}[L(z_3 - \dot{z}_2)] \\ C_4 &= -1.5M^{1/2}|z_4 - \dot{z}_3|^{1/2}\text{sign}[L(z_4 - \dot{z}_3)] \\ C_5 &= -1.1M\text{sign}[L(z_5 - \dot{z}_4)] \end{aligned} \tag{8}$$

The correction factors are chosen such that the errors between the actual evolution of HOSMO and its expected dynamics $\{\dot{z}_1, \dot{z}_2, \dot{z}_3, \dot{z}_4, \dot{z}_5\}$ are fed back to the HOSMO. They apear with a properly chosen magnitude and sign such that the error space $\{e_{z_1}, ez_2, e_{z_3}, e_b\}$ slides on a zero-error surface in a finite time.

The fifth state, z_5 was added in (7) to reduce the influence of the noise intrinsec to HOSMO estimation, the chattering (see [6]), on the values we are interested in $\{z_2, z_3, z_1, b\}$. The order chosen to mention the estimated values is that respecting cronollogy, from the one the HOSMO

receives, z_1, to the one having the greatest relative degree to the output of the transmitter, $y_2 = z_1$. See [7] to better follow the implications of relative degrees in clasic sliding-mode observers and, in particular, in high-order sliding mode observers. The chattering noise is generated by the high-frequency control switching given by the sign function used in order to solve uncertainity in the HOSMO.

2.5. Encryption

The encryption block receives a stream of bytes for encryption. Baptista's approach was to split the codomain interval for the Logistic map into 256 subintervals, each corresponding to a possible input value. Then, the system is advanced until the output of the function falls into the corresponding subinterval for the input. The cipher text is the number of iterations the function was advanced to reach that value. The problem with this approach, as already stated in the introduction, is that the distribution of the cipher text is not uniform. In order to address this issue, we use the output of the Logistic map for the previously enciphered byte to perform an exclusive-or (XOR) operation with the Baptista's cipher text producing a new cipher text value which distribution is uniform, like in figure 6. The blue sections represent the addition to the original Baptista cipher which is depicted in black. When a value of b is selected for encryption, a new Logistic map system is initialized with a fixed values for $x[0]$ and the value of b, then it is advanced with 250 steps in order to pass transitory time, after which the actual encryption starts.

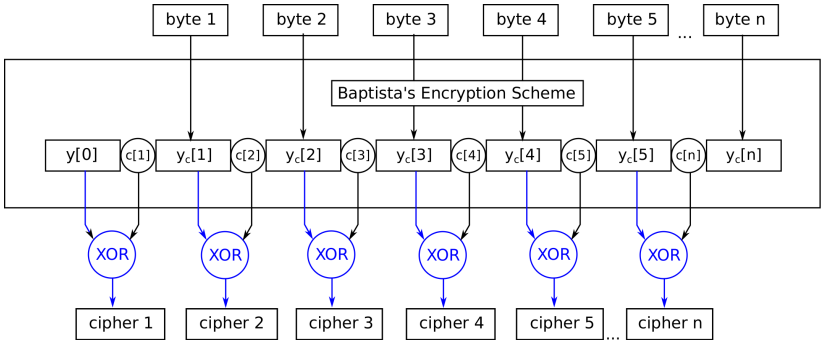

Fig. 6. Encryption process diagram

Another problem with Baptista's approach is that the number of iterations necessary to reach the required subinterval can be rather large, to the order of thousands. This means that the cipher text is 16-bits for every byte of clear text, effectively doubling the size of the data and also that it takes a large number of operations in order to encrypt a single character. These will not be addressed in this paper.

2.6. Decryption

Decryption follows the same algorithm, only reversed. Given the parameter b, the system starts from the initial value $x[0]$. The received cipher text is XOR'ed with $x[0]$ and the resulting number represents a number of iterations with which the system is advanced. The subinterval for the resulting function output value gives the corresponding plain text. The operation repeats for an entire block of 1024 bytes.

The security of the cipher greatly depends on the key sensitivity and fortunately chaotic systems depend greatly on the initial condition and parameters. As seen in Fig.7, a variation of 10^{-15} of the decryption key makes the data impossible to discern, resulting in a quasi-uniform distribution. In the example, the encryption key is $b = 3.99$ and the decryption key is $b' = 3.99 + 10^{-15}$.

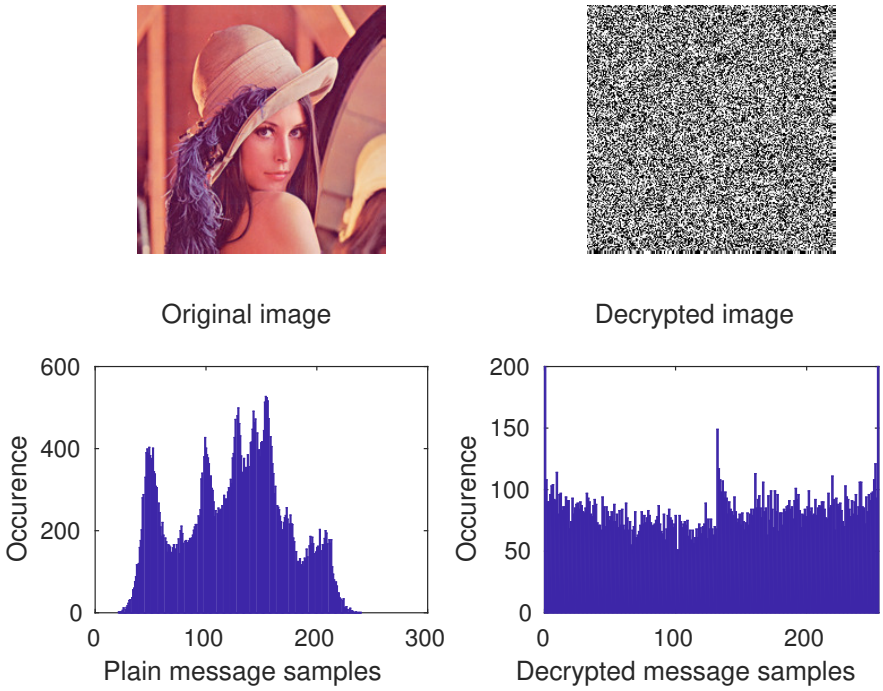

Fig. 7. Decryption with different key

2.7. An example. Simulation results and discussion

2.7.1. Setup and parameters

To exemplify the algorithm described above, the ordinary differential equations solver 1 (Euler) with fixed step 10^{-6} was used in a Matlab Simulink simulation. For the Colpitts transmitter the parameters were chosen $g = 4.46; Q = 1.38$. The initial conditions of the estimator are $\{z_1(0), z_2(0), z_3(0), z_4(0), z_5(0)\} = \{y_1(0),\ y_2(0), y_3(0), 0, 0\} = 0.81472368639317; 0.905791937075619; 0.126986816293506; 0; 0$. A Butterworth second order low pass filter with cutoff frequency $\omega_0 = 2\pi/0.12$ was added to the HOSMO in order to remove the chattering from the estimated states. The same filter was used in the transmitter to compare the original states and the estimated ones as in figures 10, 12 and 14. The evolution of the reconstructed parameter $\hat{b}$ and its constant value $b = 4$ are illustrated in figure 8.The final b value used in the decryption block is obtained by averaging this evolution over the last 10^5

samples. The results are shown in table 2 for $b \in [3.56; 4]$ with a step of 10^{-2}. In this example, the HOSMO is only able to recover b with 2 exact decimal values as it can be seen in Fig. 9. The precision in state estimation decreases with a factor of $\approx 10^{-3}$ with each further estimation step, as it can be observed in Fig. 13, Fig. 15 and Fig. 11.

Table 2. Estimation of the secret key $b \in [3.56; 4]$ by the HOSMO with parameters $L = 10^{40}$ and $M = 10^5$.

b	$\hat{b}$	b	$\hat{b}$	b	$\hat{b}$
3.56	3.5580351508858	3.71	3.7079979620259	3.86	3.8579228779057
3.57	3.5680328807786	3.72	3.7179900482914	3.87	3.8679515066995
3.58	3.5780322749782	3.73	3.7279853331485	3.88	3.8779429277126
3.59	3.5880269030275	3.74	3.7379599534386	3.89	3.8879416773573
3.60	3.5980588032151	3.75	3.7479636094201	3.90	3.8979316809814
3.61	3.6079991118783	3.76	3.7579963713988	3.91	3.9078934595980
3.62	3.6180608344815	3.77	3.7679870212014	3.92	3.9179170173659
3.63	3.6280037968218	3.78	3.7779855720617	3.93	3.9279145906426
3.64	3.6379951757411	3.79	3.7879587653470	3.94	3.9378812589547
3.65	3.6479925077161	3.80	3.7979441927598	3.95	3.9479074269695
3.66	3.6579957119919	3.81	3.8079141349108	3.96	3.9578759024316
3.67	3.6680142681080	3.82	3.8179399760569	3.97	3.9679320438056
3.68	3.6780306160224	3.83	3.8279441244635	3.98	3.9778819997400
3.69	3.6879928167361	3.84	3.8379429783991	3.99	3.9879133410858
3.70	3.6979969154444	3.85	3.8479302386131	4.00	3.9978609373831

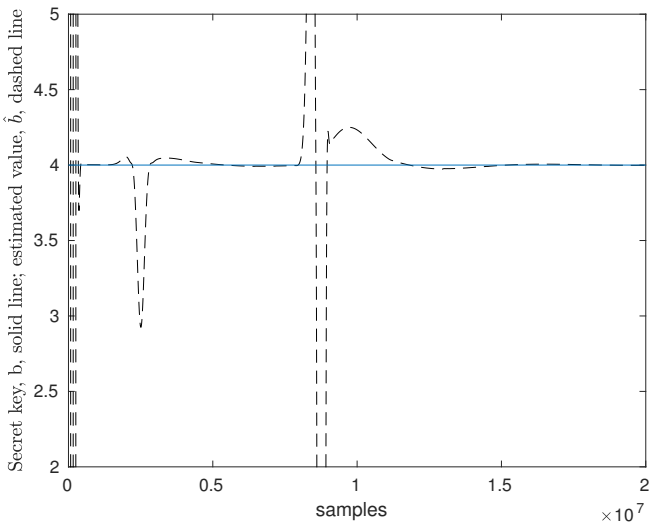

Fig. 8. Estimated value for parameter b - the Decryption Key

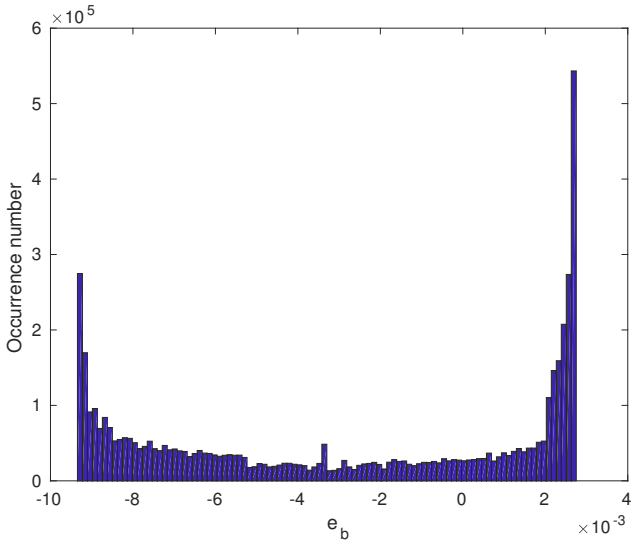

Fig. 9. Error for parameter b - the Decryption Key

2.7.2. Comparison with the state-of-the-art

The state-of-the-art secret communication scheme involves two algorithms: a public-private (asymmetric) key encryption (like Rivest-Shamir-Adleman) used to negotiate a secret key that is then utilized for a symmetric encryption algorithm (like Advanced Encryption Standard). Both RSA and AES are currently approved standards by the National Institute of Standards and Technology.

Altough the proposed communication scheme is a lot different from the state-of-the-art and it is difficult to make a meaningful comparison, especially since the system was only implemented in simulation, we will attempt to discuss the differences, both quantitatively and qualitatively.

Quantitatively, we will evaluate the performance of the encryption system in floating point operations per byte of clear text. Since decryption and encryption require the same operations, only in reverse, it is enough to estimate the number of operations required for encryption. In order to advance the logistic map one iteration, given (1), the operations required are: one subtraction and two multiplications in floating point arithmetic; that is three Floating Point Operations (FLOPs). In [2], the average number of iterations required for encrypting one character is 6000. Our improvement of the cipher requires an additional XOR operation per encrypted byte. The total number of operations required is: 18k FLOPs + 1 Integer Operation. The latter is much faster so it can be ignored.

The processor used for evaluation is a *Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz*. According to [16], this processor model can execute a number of $14.33\ GFLOPS$. Dividing that with $18 \cdot 10^3 FLOPs$ yelds $0.76 MB/s$.

At the same time, running the OpenSSL evaluation tool on the same processor yelds the following single core results:

type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
aes−128 cbc	113357.03k	123624.69k	126561.16k	127446.22k	127860.96k
aes−192 cbc	95368.48k	103951.46k	106232.38k	106234.85k	105129.75k
aes−256 cbc	83489.82k	88777.48k	90334.24k	90573.31k	90535.34k

The conclusion is that the proposed system is 500 times slower than 256-bits CBC AES. However, the comparison might not be completely fair due to the fact that most modern processor have dedicated instructions for AES and RSA encryption. Moreover, the most significant slowdown is due to Baptista's algorithm that requires 6000 iterations for encrypting one byte of clear text, not the key exchange itself, which is the main contribution of this paper. As future work, we will attempt to replace the encryption scheme with a stream cipher implemented with a chaos based pseudo-random number generator.

Qualitatively, the security of the cipher needs to be evaluated. This is, of course, critical for practical applications and will be addressed thoroughly though cryptanalysis in a following research. The security is influenced by two factors: a) the security of the key exchange and b) the robustness of the cipher when the key is unknown to the attacker. We will only discuss the first item, especially considering that the security of Baptista's cryptosystem has been thoroughly analyzed in several papers.

It was shown already that the value for the key b is extremely sensitive: a variation of even 10^{-15} will generate a entirely different dynamic of the system and thus it will make decryption impossible. So it is sufficient to show that a HOSMO configured with a wrong set of parameters will not be able to estimate the key with enough precision. Currently, the estimator can only obtain the key correctly up to the second decimal, and that is with the best values for parameters L and M and the initial states z_{1-5}. Since the key transmitter is itself a chaotic system, we know of no other ways to predict the evolution of the system without knowing the initial state.

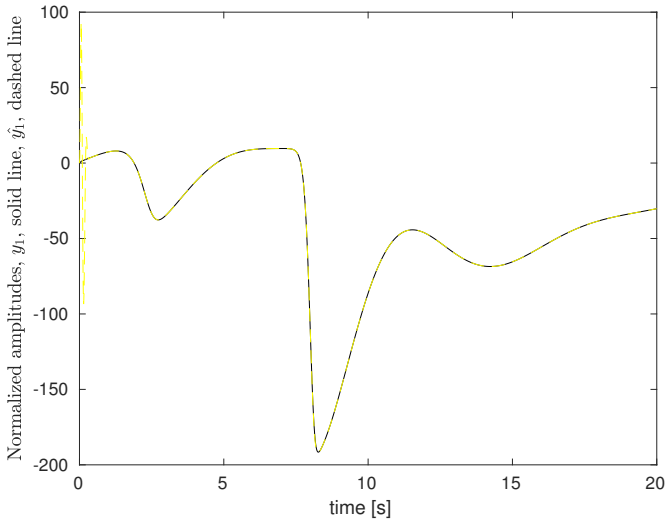

Fig. 10. Estimated value for state y_1

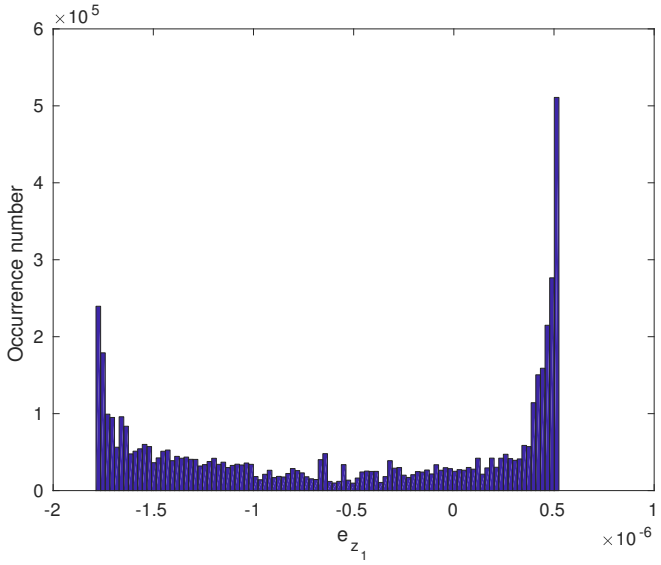

Fig. 11. Estimation error for y_1

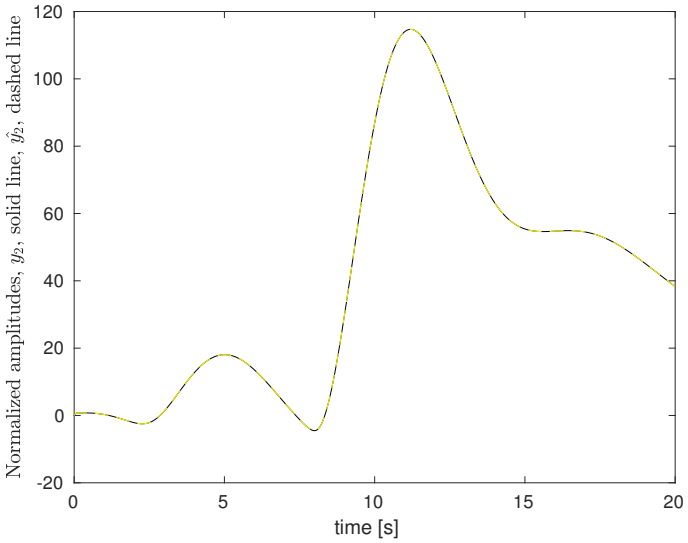

Fig. 12. Estimated value for state y_2

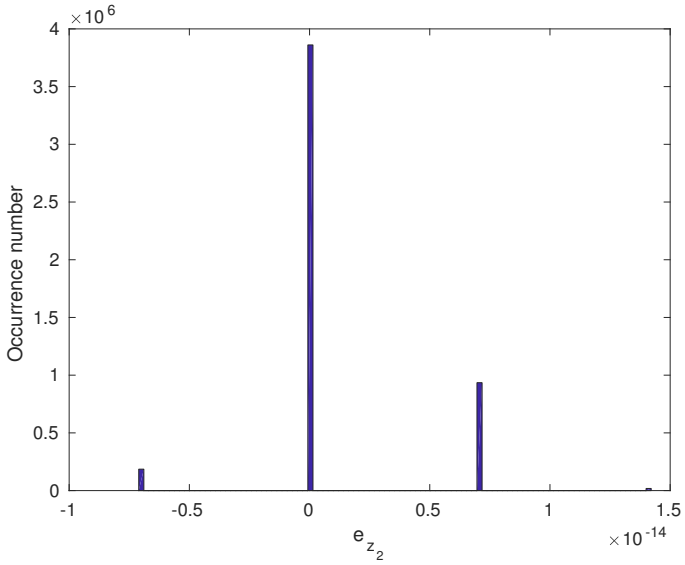

Fig. 13. Estimation error for y_2

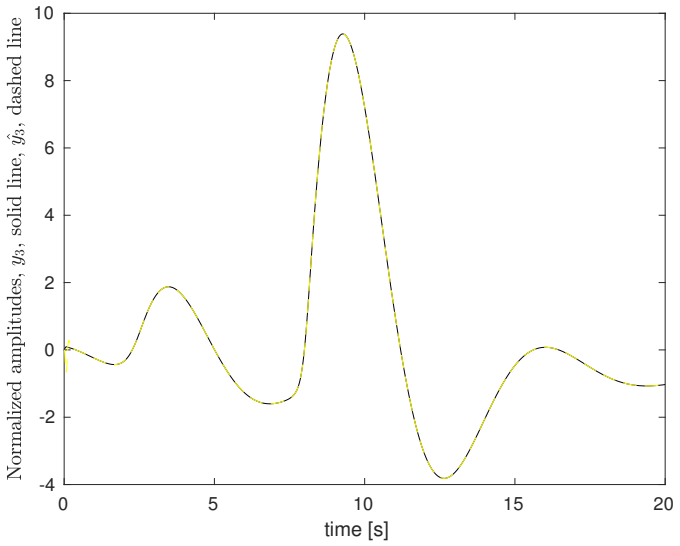

Fig. 14. Estimated value for state y_3

.

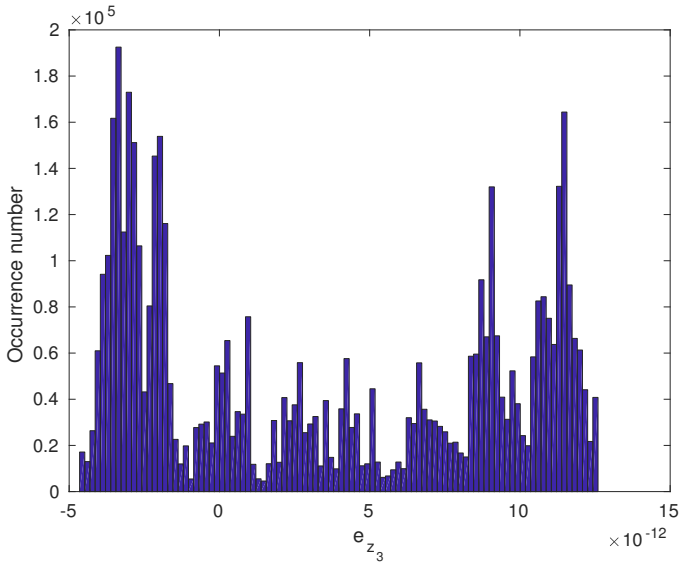

Fig. 15. Estimation error for y_3

3. Conclusions

This paper proposes an encryption scheme that utilizes a Colpitts oscillator and a high order sliding mode observer to implement a secure key exchange system. The key exchange was used in a larger mechanism to encrypt and transmit blocks of 1KB of data. The key is generated randomly and is validated using values for the logistic map Lyapunov exponents. The experiments show that the key can be transmitted and recovered completely for values up to two fractional digits. The key space size is limited and increasing it is a goal for future research, as is the reduction of the cipher text size in Baptista style encryption.

Acknowledgements. This work has been funded by University Politehnica of Bucharest, through the Excellence Research Grants Program, UPB GEX 2017. Identifier: UPB- GEX2017, Ctr. No. 36 /25.09.2017 (CNP).

References

[1] O. DATCU, R. HOBINCU, L. PETRICĂ, *Baptista's chaos-based cipher implemented in a field programmable gate array*, 2017 International Semiconductor Conference, Proceedings, pp. 191-194, 2017.

[2] M. S. BAPTISTA, *Cryptography with chaos*, Phys. Lett. A 240, 1998.

[3] O. de FEO and G. M. MAGGIO, *Bifurcations in the Colpitts oscillator: From theory to practice*, International Journal of Bifurcation and Chaos, vol. 13, no. 10, pp. 2917–2934, 2003, doi: 10.1142/S0218127403008338.

[4] G. M. MAGGIO, O. De FEO and M. P. KENNEDY, *Nonlinear analysis of the Colpitts oscillator and applications to design*, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 46, no. 9, pp. 1118–1130, 1999.

[5] Higher-order sliding modes, differentiation and output-feedback control, Int. J. of Control, 76 (9/10), 924-941, Special issue on Sliding-Mode Control, 2003.

[6] A. LEVANT, M. LIVNE, D. LUNZ -*On Discretization of High Order Sliding Modes*, in Barbot,Fridman,Plestan (eds) "Recent Trends in Sliding Mode Control", 177-202,IET,2016

[7] A. LEVANT, *Finite Time Stability and High Relative Degrees in Sliding Mode Control*, Lecture Notes in Control and Information Sciences No. 412, pp. 59-92, 2012.

[8] A. WOLF, J.B. SWIFT, H. L. SWINNEY, J. A. VASTANO, *Determining Lyapunov exponents from a time series*, Physica D: Nonlinear Phenomena, Vol. 16, Issue 3, July 1985, Pages 285-317.

[9] M. H*É*NON, *A two-dimensional mapping with a strange attractor*, Communications in Mathematical Physics. 50 (1): 6977, 1976.

[10] K.T. ALLIGOOD, T.D. SAUER and J.A. YORKE, Chaos: An Introduction to Dynamical Systems, Berlin: Springer-Verlag, 1996.

[11] DEVANEY, ROBERT L. (1988), *Fractal patterns arising in chaotic dynamical systems*, in Peitgen, Heinz-Otto; Saupe, Dietmar, The Science of Fractal Images, Springer-Verlag, pp. 137168, doi:10.1007/978-1-4612-3784-6_3.

[12] A. VLAD, A. LUCA, O. HODEA, R. TĂTARU, *Generating Chaotic Secure Sequences Using Tent Map and a Running-Key Approach*, The Publishing House Proceedings of the Romanian Academy, Series A, Vol. 14, Special Issue 2013, pp. 295 - 302.

[13] G ALVAREZ, S LI, *Some basic cryptographic requirements for chaos-based cryptosystems*, International Journal of Bifurcation and Chaos 16 (08), 2129-2151.

[14] O. DATCU, R. HOBINCU, M. STANCIU, R. A. BADEA, *Encrypting multimedia data using modified Baptista's chaos-based algorithm*, 3rd EAI International Conference on Future Access Enablers of Ubiquitous and Intelligent Infrastructures, 2017.

[15] M. L'HERNAULT, J.-P. BARBOT, A. OUSLIMANI, *Sliding Mode Observer for a Chaotic Communication System: Experimental Results*, IFAC Proceedings Volumes, Vol. 39, no. 8, 2006, Pages 401-406, doi:20060628-3-FR-3903.00071.

[16] Project Asteroids@home, online, visited on 28/01/2018, https://asteroidsathome.net/boinc/cpu_list.php