# A comparison between the message embedded cryptosystem and the self-synchronous stream cipher Mosquito

Phuoc Vo Tan, Gilles Millérioux, Jamal Daafouz

# A comparison between the message embedded cryptosystem and the self-synchronous stream cipher Mosquito

Phuoc Vo Tan, Gilles Millérioux and Jamal Daafouz
Research Center on Automatic Control (CRAN)
Nancy University - CNRS UMR 7039, France
Email: {phuoc.vo-tan,gilles.millerioux}@esstin.uhp-nancy.fr, jamal.daafouz@ensem.inpl-nancy.fr

***Abstract*— It is widely admitted that most of chaotic cryptosystems belong to the class of symmetric key ciphers but few of them have really been compared with standard existing ones. As a consequence, their design often resorts to empirical approaches with trial-and-error settings. This paper must be considered as an attempt to handle such a situation. An in-depth comparison between a fully-fledged self-synchronous stream cipher called Mosquito and one of the most promising chaotic cryptosystem, namely hybrid message-embedding, is carried out. It is given a correspondence between the design parameters involved in the two respective schemes, in particular the size of the secret key, the required memory, the dimension of the system. Furthermore, we put a special emphasis on the link between the notion of relative degree of a dynamical system and the concept of pipelining.**

## I. Introduction

When looking into conventional cryptography (see the book of Menezes [6] for a pretty good account), it turns out that symmetric-key ciphers are extensively used in secure transmission requiring high throughput. Among a variety of symmetric-key ciphers, the self-synchronous stream ciphers (denoted SSSC for short) are of special interest. They admit at the transmitter side, the recursion

$$\begin{cases} z_k = f_K(c_{k-M}, \ldots, c_{k-1}) \\ c_k = e(z_k, m_k) \end{cases} \tag{1}$$

where $f_K$ is a function parameterized by the secret (also called static) key $K$. $f_K$ is called the key stream generator function. It depends on a fixed number of past ciphertexts $c_{k-i}$ $(i = 1, \ldots, M)$ of $c_k$ and delivers $z_k$. The sequence $\{z_k\}$ is called the key stream and $z_k$ is named the running key because, as opposed to the static key $K$, it is time-varying. The encryption function $e$, which depends on $z_k$, converts the plaintext $m_k$ into the ciphertext $c_k$. SSSCs are interesting because they get inherent ability to self-synchronizing ([6]). Thereby the communication does not require any additional synchronization flags or interactive protocols for recovering lost synchronization induced for instance by bit slips.

On the other hand, it is well admitted that performances of many chaotic cryptosystems in terms of speed and security are still rather poor and serious cryptanalytic works have revealed strong weakness. The reader may refer to [1] for getting acquainted with some relevant attacks and the related references. One of the main reasons stems from the fact that not enough attention has been payed on the basic rules borrowed from standard cryptography a chaos-based encryption scheme should obey. Furthermore, even though most of chaotic cryptosystems belong to the class of symmetric key ciphers, few of them have really been compared with standard existing ones. As a result, their design often resorts to empirical approaches with trial-and-error settings. A first connection has been recently brought out in [7] between the Hybrid Message-Embedded (HME for short) chaotic algorithms and the conventional SSSCs. Although the principle of the HME is well-known, let us briefly recalled it. At the transmitter part, the plaintext $m_k$ is injected (or, as it is also usually said, embedded) in a chaotic dynamics $f_\theta$ after a pre-ciphering via a function $\nu_e$. The resulting cipher turns into a non autonomous system of the form:

$$\begin{cases} x_{k+1} = f_\theta(x_k, u_k) \\ y_k = h'_\theta(x_k) \\ u_k = \nu_e(x_k, m_k) \end{cases} \tag{2}$$

The main result stated in [7] is that the HME is equivalent to a conventional self-synchronizing stream cipher under *flatness conditions*. The notion of flatness will be recalled later on in this paper. Nevertheless, the work was restricted to a structural comparison without any quantitative consideration like, to mention a few, the size of the memory, the size of the secret key, the dimension of the system. Such quantities will be called hereafter the *design parameters*.

The objective of the present paper is to refine the comparison by investigating a fully-fledged hardware-oriented SSSC called Mosquito [2]. Then is explained how such a connection enables to derive some basic rules leading to relevant design parameters settings. Mosquito is a very interesting SSSC scheme for comparative studies. Indeed, it has been elected through the eSTREAM project[1] which confers great credibility on it and thereby can be considered as an algorithm of reference. This paper is organized as follows. Section II sums up the essential of Mosquito. Section III brings out the connection between the design parameters of the Hybrid Message Embedding and those of Mosquito with special emphasis on the link between the notion of relative degree borrowed from the control theory and the concept of pipelining often involved in conventional cryptography. An example illustratingis then provided.

## II. Mosquito

### A. Maurer's design principle

The conventional SSSC Mosquito highly relies on a former design principle first suggested by Maurer [5]. This design is intended to guarantee complex and so secure key stream generator function $f_K$. The basic idea consists in replacing standard feedback shift registers by finite state automata. The encryption part of an SSSC is thereby specified as follows:

$$\begin{cases} q_{k+1} = g_K(q_k, c_k) \\ z_k = h_K(q_k) \\ c_k = z_k \oplus m_k \end{cases} \tag{3}$$

where $\oplus$ denotes the bitwise XOR operator and acts as the encryption function $e$. $q_k$ is the internal state of the automaton. Then a method which guarantees that the finite state automaton has a finite input memory $M$ is provided. If so, (3) can be rewritten into the canonical form (1). Actually, Mosquito partially follows such a principle with some additional refinements.

[1] Available online at http://www.ecrypt.eu.org/stream/

| | $v$ | $w$ |
|---|---|---|
| $(i+j)\ mod\ 3 = 0$ | $j-4+(i\ mod\ 2)$ | $j-2$ |
| $(i+j)\ mod\ 3 = 1$ | $j-6+(i\ mod\ 2)$ | $j-2$ |
| $(i+j)\ mod\ 3 = 2$ | $j-5+(i\ mod\ 2)$ | 0 |
| $(i+j)\ mod\ 3 = 5$ | 0 | $j-2$ |

TABLE I

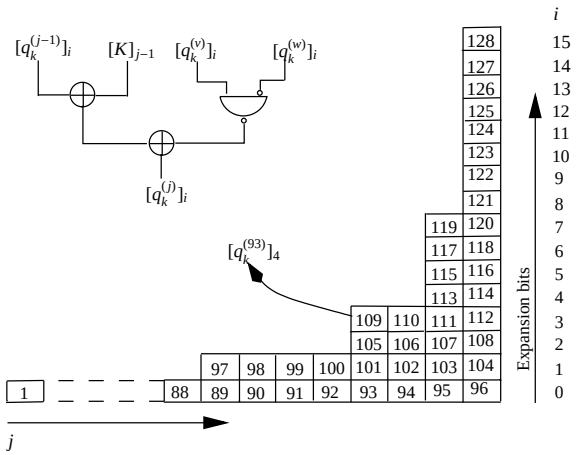


Fig. 1. The arrangement of 128 memory cells $[q_k^{(j)}]_i$ and the hardware architecture of Eq. (4).

### B. State transition function of Mosquito

Similarly to the Maurer's principle, the state transition function $g_K$ of Mosquito gets the form of a finite state automaton with internal state $q_k$. The dimension of $q_k$ is $M = 96$. The static key $K$ consists of 96 bits $[K]_0 \dots [K]_{95}$. An important point is that the components $q_k^{(j)}$ $(j = 1, \dots, 96)$ may involve more than one bit. The so-called *expansion* obeys the following rule: no expansion for $j = 1, \dots, 88$, expansion with 2 bits for $j = 89, \dots, 92$, with 4 bits for $j = 93, \dots, 94$, with 8 bits for $j = 95$ and with 16 bits for $j = 96$. As a result, the memory assigned to $q_k$ involves 128 bit cells. Let $[q_k^{(j)}]_i$ denote the $i^{th}$ bit of $q_k^{(j)}$. The recursion to which each bit cell $[q_k^{(j)}]_i$ obeys reads:

$$[q_k^{(j)}]_i = [q_{k-1}^{(j-1)}]_i \oplus [K]_{j-1} \oplus (([q_{k-1}^{(v)}]_i)([q_{k-1}^{(w)}]_i + 1) + 1) \qquad (4)$$

where $0 \le v, w < j-1$, $1 \le j \le 96$. The arrangement of the memory as well as the combinatorial circuit associated to (4) are depicted on Fig. 1. According to the pair $(i, j)$ assigned to a bit cell $[q_k^{(j)}]_i$, the value of $v$ and $w$ may differ and obeys the rule of Table I.

When omitting the bit index $i$ and taking into account that $[q_{k-1}^{(0)}]_0 = c_{k-1}$, (4) can be rewritten, for $1 \le j \le 96$:

$$q_k^{(j)} = q_{k-1}^{(j-1)} + E_K(q_{k-1}^{(j-2)}, \dots, q_{k-1}^{(1)}, c_{k-1}) \qquad (5)$$

with $E_K$ a non-linear function parameterized by $K$. Eq. (5) describes the state-transition function $g_K$ of a finite state automaton. Besides, in order to guarantee that such automaton has a finite input memory, the state transition function $g_K$ is chosen to be "triangular" in the sense that the $j^{th}$ component of $q_k$ only depends on the previous components $l < j$ at time $k-1$. "Triangularity" confers the property that after a finite transient time of length $M$, $q_k$ does no longer depend on the initial condition $q_0$. Therefore, there exists a function $l_K$ such that, for $k \ge M$:

$$q_k = l_K(c_{k-M}, \dots, c_{k-1}) \qquad (6)$$

### C. Output function of Mosquito

Unlike the Maurer's principle, Mosquito pays a great attention on the output function $h_K$. Its design is carried out through the concept of *pipelining*. A "pipeline" is somewhat similar to the round transformation in a block cipher. It involves some functions assigned to subsequent "stages". However, unlike block ciphers, "pipeline" functions are not necessarily permutations. Therefore, the dimension of the pipeline input can be different from the dimension of the pipeline output.

For Mosquito, the "pipeline" involves 9 stages. Every stage implements a function $s_i$ $(i = 0, \dots, 8)$ in a form of a combinatorial circuit. Let us notice that none of the functions $s_i$ depend explicitly on the static key $K$. The result of the function $s_i$ is stored in a variable $a_{<i>}$ (a register in the hardware implementation). The function $s_0$ of the first stage merely initializes the variable $a_{<0>}$ as follows:

$$(a_{<0>})_{k+1} = q_k$$

where $(a_{<i>})_{k+1}$ denote the variable $a_{<i>}$ at time instant $k+1$. Consequently, the input of the pipeline corresponds to the 128 bits of the internal state $q_k$. The results of the functions $s_i$, for $i = 1, \dots, 7$ (not detailed here) are stored in the variables $a_{<i>}$ of dimension 53 for $i = 1, \dots, 5$, dimension 12 for $i = 6$ and dimension 3 for $i = 7$. Finally, the last function $s_8$ is a mere addition between $a_{<7>}^{(0)}$, $a_{<7>}^{(1)}$ and $a_{<7>}^{(2)}$. As a consequence, the *output function* $h$ for Mosquito results from $b_s$ compositions and reads:

$$z_{k+b_s} = s_8(s_7(\dots(s_0(q_k)))) = h(q_k) \qquad (7)$$

$b_s$ is the *cipher function delay* due to the subsequent operations in the "pipeline" and $b_s = 9$.

Finally, the overall description of Mosquito reads:

$$\begin{cases} q_{k+1} = g_K(q_k, c_k) \\ z_{k+b_s} = h(q_k) \\ c_{k+b_s} = z_{k+b_s} \oplus m_k \end{cases} \qquad (8)$$

Besides, combining (6) and (7) yields

$$\begin{aligned} z_{k+b_s} &= h(l_K(c_{k-M}, \dots, c_{k-1})) \\ &= f'_K(c_{k-M}, \dots, c_{k-1}) \end{aligned} \qquad (9)$$

As a result, Mosquito can be equivalently described in the canonical form :

$$\begin{cases} z_{k+b_s} = f'_K(c_{k-M}, \dots, c_{k-1}) \\ c_{k+b_s} = z_{k+b_s} \oplus m_k = e(z_{k+b_s}, m_k) \end{cases} \qquad (10)$$

with $f'_K$ acting as the key stream generator function. The only, but major, distinction between (8) and (3) or between (10) and (1), lies in that the plaintext and the ciphertext are delayed by $b_s$ time-steps because of the "pipelining" structure. On the other hand, the output function can be thereby significantly complexified. Figure 2 depicts the block diagram of Mosquito.

## III. Comparative study

### A. The connection between the design parameters

We must first recall two important definitions.

*Definition 1:* The relative degree of a dynamical system with respect to its input $m_k$ is the required number $r$ of iterations of the output $y_k$ so as $y_{k+r}$ depends on $m_k$ in an explicit way.

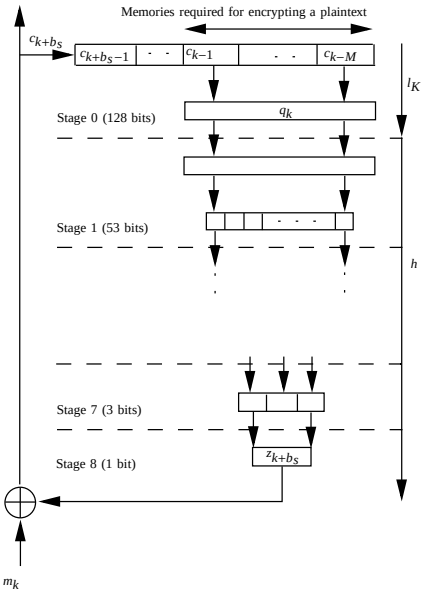


Fig. 2. Block diagram of Mosquito

In particular for (2), a relative degree $r > 0$ means that after iterating $r$ times the state vector $x_k$, its output $y_{k+r}$ will read

$$y_{k+r} = h'_\theta(f^r_\theta(x_k, \nu_e(x_k, m_k))) = \mathcal{H}_\theta(x_k, m_k) \tag{11}$$

$$\begin{aligned} \text{with} \quad f^i_\theta(x_k, u_k) &= x_k \quad when\ i = 0 \\ &= f_\theta(f^{i-1}_\theta(x_k, u_k), u_{k+i-1}) \quad \forall i \geq 1. \end{aligned}$$

and where $m_k$ appears explicitly into the expression (11).

*Definition 2:* ([3]) A dynamical system is said to be *flat* if there exists a set of independent variables referred to as flat outputs, such that all system variables can be expressed as a function of the flat output and a finite number of its backward and/or forward iterates.

In particular for (2), under flatness condition, there exists a function $\mathcal{F}_\theta$ such that its state vector $x_k$ can be rewritten as:

$$x_k = \mathcal{F}_\theta(y_{k-\mathcal{K}(r)}, \ldots, y_{k-\mathcal{K}'(r)}) \tag{12}$$

where $\mathcal{K}(r)$ and $\mathcal{K}'(r)$ are $\mathbb{Z}$-valued integers which may depend on the relative degree $r$ of the system.

Finally, if the Hybrid Message-Embedded cryptosystem (2) is flat and has a relative degree $r > 0$, it can be described into the form:

$$\begin{cases} x_k = \mathcal{F}_\theta(y_{k-\mathcal{K}(r)}, \ldots, y_{k-\mathcal{K}'(r)}) \\ y_{k+r} = \mathcal{H}_\theta(x_k, m_k) \end{cases} \tag{13}$$

Identifying (13) with (10) leads to the following proposition:

*Proposition 1:* If the Hybrid Message Embedded scheme (2) is flat and has a relative degree $r > 0$, it is equivalent to a pipelined SSSC with the correspondence

- Secret key $\theta \equiv K$
- Key stream generator function $\mathcal{F}_\theta \equiv f'_K$
- Encryption function $\mathcal{H}_\theta \equiv e$
- Running key $x \equiv z$
- Ciphertext $y \equiv c$
- Number of past ciphertexts $|\mathcal{K}'(r) - \mathcal{K}(r) + 1| \equiv M$
- Cipher function delay $r \equiv b_s$

where $\equiv$ stands for "plays the role of". A central and interesting conclusion is that a flat dynamical system with relative degree $r > 0$ acts as pipelined encryption scheme but the pipeline is actually virtual. Let us notice that most often, $|\mathcal{K}'(r) - \mathcal{K}(r) + 1| = n$ no matter $r$ is.

Besides, it is worth comparing the memory sizes which are respectively involved in Mosquito and in the HME. For Mosquito, 128 bit cells are needed to store $q_k$ and $53 \cdot 5 + 12 + 3 + 1 = 281$ bits cells are needed to store the variables $a_{<i>}$ of the nine stages of the pipeline. As a result, it can be claimed that Mosquito requires $M_d = 128 + 281 = 409$ bit cells for the dynamical variables. As for the static key $K$, $M_d = 96$ bit cells are required. For the HME, the following proposition holds.

*Proposition 2:* Assuming that the components of $x_k$ are $m$ bits words and that $dim(x_k) = n$, the memory size for the dynamical variables fulfills

$$M'_d = n \cdot m \text{ bits} \tag{14}$$

Assuming that the components $\theta^{(i)}$ of the static key parameter vector $\theta$ are encoded with $n_m$ bits and that $dim(\theta) = n_\theta$, the memory size for the static key fulfills

$$M'_s = n_\theta \cdot n_m \text{ bits} \tag{15}$$

It is worth stressing that for the HME, the memory requirement for the dynamical variables is independent of $r$ since the pipeline is "virtual". Actually, the memory size is only related to $x_k$.

### B. Simulation Example

We choose a basic example in order to clearly illustrate, first, the two major notion of flatness and relative degree for the HME and secondly, the correspondences between the HME and the standard SSSC Mosquito design parameters. To this end, we resort to a dynamical system with low dimension and with basic nonlinearities, namely congruential operations and dimension $n = 3$

$$\begin{cases} x_{k+1} = Ax_k + Bu_k \\ y_k = Cx_k + Du_k \\ u_k = \nu_e(x_k, m_k) \end{cases} \tag{16}$$

The operations are performed modulo 256. The entries of the matrices $A$, $B$, $C$ and $D$ are integers encoded with $n_m = 8$ bits and so range between 0 and 255. The supposed secret static key is the vector $\theta = [38\ 7\ 4]$ which actually corresponds to the first column of $A$ written in a companion form and $n_\theta = 3$. The components of $x_k$ are also 8-bit words ranging between 0 and 255, that is $m = 8$. Numerically, the matrices read

$$A = \begin{bmatrix} 38 & 1 & 0 \\ 7 & 0 & 1 \\ 4 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \quad D = 0.$$

The function $\nu_e$ is chosen to be a bitwise XOR between the components of $x_k$ denoted $x^{(i)}_k$ and the plaintext $m_k$:

$$u_k = x^{(1)}_k \oplus x^{(2)}_k \oplus x^{(3)}_k \oplus m_k.$$

$x^{(i)}_k$ and $m_k$ are meant here to be the corresponding 8-bit representation.

Let us first focus on the relative degree. It is recalled (from [4] for instance) that, for linear systems written in a state space form, the

relative degree is 0 if $D \neq 0$ or corresponds to the smallest integer $r$ such that $CA^{r-1}B$ is different from 0 if $D = 0$. The same definition still holds for linear congruential systems. Here, since $D = 0$ and $CB = 1$, the relative degree of the system is $r = 1$. The relative degree $r$ of the system being 1, we must compute $y_{k+1}$.

$$\begin{aligned} y_{k+1} &= CAx_k + CBv_e(x_k, m_k) \\ &= 38x_k^{(1)} + x_k^{(2)} + x_k^{(1)} \oplus x_k^{(2)} \oplus x_k^{(3)} \oplus m_k \end{aligned} \tag{17}$$

We now focus on flatness. After some rather basic manipulations, $x_k$ can be expressed into the form (12) with $\mathcal{F}_\theta$ obeying

$$\left\{ \begin{array}{lcl} x_k^{(1)} &=& y_k \\ x_k^{(2)} &=& 7y_{k-1} + 4y_{k-2} \\ x_k^{(3)} &=& 4y_{k-1} \end{array} \right. \tag{18}$$

Equation (18) clearly corroborates that the system is flat. Besides, it provides the actual values $\mathcal{K}'(1) = 0$, $\mathcal{K}(1) = 2$.

We are now in position of stating that the system (16) is equivalent to an SSSC with key stream generator $\mathcal{F}_\theta$ corresponding to Eq. (18), encryption function $\mathcal{H}_\theta$ corresponding to Eq. (17), running key $x$, ciphertext $y$, secret static key $\theta = [38\ 7\ 4]$, number of past ciphertexts $|\mathcal{K}'(1) - \mathcal{K}(1) + 1| = 3$, cipher function delay $r = 1$. (16) operates as it would get a pipeline with one stage but the pipeline is actually virtual.

The original image to be encrypted is the well-known Lena picture. Figure 3 depicts respectively the results of the encrypted and the decrypted image through the HME cryptosystem while Figure 4 depicts respectively the results of the encrypted and the decrypted image through Mosquito.

Fig. 3. Left: the encrypted image by the Hybrid Message Embedding. Right: the decrypted image by the Hybrid Message Embedding.

At first glance, both algorithms perform in a similar way. However, we should not content with this mere investigation which cannot really reveal some underlying weakness. Actually, the security of a chaotic cryptosystem lies both on a relevant setting of the design parameters and on a suitable choice of the state transition and output functions. Based on the previous results, we show throughout this illustrative example how the first requirement can be fulfilled for the HME. Indeed, the memory size of Mosquito is $M_d = 409$ for the dynamical variables, $M_s = 96$ for the static key and the cipher function delay (related to the size of the pipeline) is $b_s = 9$. Those parameters can be viewed as some reference values. For the HME considered in this example, the memory size for dynamical variables is $M'_d = n \cdot m = 3 \cdot 8 = 24$ while the memory size for the static key is $M'_s = n_\theta \cdot n_m = 3 \cdot 8 = 24$ bit cells. When comparing with Mosquito, by virtue of Proposition 1 and 2, it can be stressed that the design parameters of the HME (16) are not actually appropriate. But even more is true, Proposition 1 and 2 provide a guidance for a relevant setting even though we must have in mind that the choice of the chaotic map will also deserve attention. If we give importance to the size of the dynamical memory, it is reasonable to set $M'_d = M_d = 409$ and $M'_s = M_s = 96$. Thus, according to (14) and (15), the dimension $n$ of the HME should verify $n = M_d/m \approx 51$, the number of key parameters should be $n_\theta = M_s/n_m = 12$. If we give also importance to the number of past ciphertexts $M$, since $|\mathcal{K}'(r) - \mathcal{K}(r) + 1| = n$, as long as $M'_d = M_d = 409$ and $M'_s = M_s = 96$, we should set $n = 96$ and so $m = M'_d/n \approx 4$. Finally, if we give importance to the pipeline size, the relative degree should be $r = b_s = 9$. Strict equalities may certainly be relaxed but the values should lie on close ranges.

Fig. 4. Left: the encrypted image by Mosquito. Right: the decrypted image by the Mosquito.

## IV. Conclusion

In this paper, an in-depth comparison between a fully-fledged self-synchronous stream cipher called Mosquito and one of the most promising chaotic cryptosystem, namely hybrid message-embedding, has been carried out. The work enables to highlight the correspondence between the design parameters involved in the two respective schemes as the dimension of the system, the size of the secret key, of the running key, of the pipeline as well as the required memory. We expect that such parallelism, considering that Mosquito is an algorithm of reference, may provide some relevant information for setting the design parameters of HME chaotic cryptosystems in a way which makes really sense in practice and that should render them competitive. Henceforth we can expect that HME ciphers may be able to provide the same performances in terms of security or speed as any conventional self-synchronizing stream ciphers. As a matter of fact, not only those performances rest on the design parameters but also on a suitable choice of the chaotic functions. The present paper enables to guarantee the first requirement, the second one deserves obviously further investigation and should constitute a challenging task.

## References


[1] F. Anstett, G. Millerioux, and G. Bloch, "Chaotic cryptosystems: Cryptanalysis and identifiability," *IEEE Trans. on Circuits and Systems*, vol. 53, no. 12, pp. 2673–2680, December 2006.

[2] J. Daemen and P. Kitsos, "The self-synchronizing stream cipher mosquito: estream documentation, version 2," *eSTREAM, ECRYPT Stream Cipher Project, Report 2005/018*, April 2005, available online at http://www.ecrypt.eu.org/stream.

[3] M. Fliess, J. Levine, P. Martin, and P. Rouchon, "Flatness and defect of non-linear systems: introductory theory and examples," *Int. Jour. of Control*, vol. 61, no. 6, pp. 1327–1361, 1995.

[4] A. Isidori, *Nonlinear control systems*, ser. Communications and control engineering series. Springer, 1995.

[5] U. M. Maurer, "New approaches to the design of self-synchronizing stream cipher," *Advance in Cryptography, In Proc. Eurocrypt '91, Lecture Notes in Computer Science*, pp. 548–471, 1991.

[6] A. J. Menezes, P. C. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, October 1996.

[7] G. Millérioux, J. M. Amigó, and J. Daafouz, "A connection between chaotic message-embedding and conventional self-synchronizing stream ciphers," in *Proc. of the 2006 International Symposium on Nonlinear Theory and its Applications (NOLTA 2006)*, Bologna, Italy, September 2006.