Synchronization of hybrid systems for secure multimedia streaming

Jeremy Parriaux, Gilles Millérioux

Synchronization of hybrid systems for secure multimedia streaming

J. Parriaux*, G. Millerioux*
*Nancy University, Research Center on Automatic Control of Nancy (CRAN), France
jeremy.parriaux@esstin.uhp-nancy.fr, gilles.millerioux@esstin.uhp-nancy.fr

***Abstract*— The aim of this paper is to propose some solutions to the problems that occur when one tries to implement cryptosystems with high throughput performances. The paper describes theoretical and practical aspects regarding an FPGA implementation of a secure multimedia streaming setup. The class of hybrid dynamical systems is addressed with special emphasis on piecewise linear dynamical systems.**

I. Introduction

Nowadays, electronic devices are everywhere and often need to interact with each other. They sometimes process critical information that needs to be securely exchanged for various purposes. This requirement has led to the development of cryptology. Cryptology is a topic that gathers cryptography and cryptanalysis. Cryptography is the art of secrecy, it deals with authentication and confidentiality of digital communications. The algorithms used for these purposes are parametrized with a secret (often called the key) that only the legitimate part must know. Cryptanalysis essentially consists in seeking for possible weakness of cryptosystems. One has to point out that the security of an algorithm has to rely entirely on the key. As a result, every single part of the algorithm is supposed to be publicly known. This assumption is known as *Kerckhoffs' principle* [2].

Cryptographers have developed several algorithms to securely exchange information. They are based on the confusion and diffusion principles stated by Shannon in [4]. *Diffusion* is a property that makes the relation between the key and the ciphertext complex and is such that a small change (as small as one bit) in the key or in the plaintext results in a completely different cryptogram. *Confusion* is a property that makes the relationship between the plaintext, the ciphertext and the key very complex. The first difference that can be made among the cryptographic algorithms is that some of them need the same key for ciphering and deciphering and the other ones use a different key. In the first case they are called symmetric (or private key) algorithms and in the second one they are called asymmetric (or public key) algorithms.

Symmetric algorithms can be divided in two subcategories: block ciphers and stream ciphers. Block ciphers are algorithms that take fixed size inputs and produce fixed size outputs through a fixed transformation called ciphering function. One problem that arises for such a class of algorithms lies in that the same plaintext results in the same ciphertext. This might be a problem in some situations, redundancy on plaintexts for example, because even though an attacker does not know the exact meaning of the data exchanged, he can extract partial information. One way to solve this problem is to use a stream cipher. In a stream cipher, the successive ciphertexts are ciphered by mean of a function which not only depends on the secret key but also on a time-varying quantity called running key. Such a time-varying key is a function of the internal state of the cipher. Therefore the ciphering function is a set of transformations indexed by the current state of the cipher. Unlike block ciphers, stream ciphers require an internal memory to store the state. Usual stream ciphers are RC4 (used in the WEP encryption in WIFI), CSA (used in the digital TV), A5 (used in mobile phones), etc. Another advantage of stream ciphers is that they usually have less hardware complexity which makes them faster. They are especially well suited when one needs to cipher data at a high throughput. For stream ciphers, the internal state of the decipher has to be kept synchronized with the internal of the cipher for proper decryption. Therefore when using stream ciphers one must resort to a synchronization scheme between the cipher and the decipher. To this end, one can insert synchronization flags in the communication stream or resorting to resynchronization protocols. Among the stream ciphers, there is a special kind of algorithms called self-synchronizing stream ciphers (SSSC for short). One of their important properties is that they do not require any external synchronization. Indeed, if for any reason the cipher and the decipher lose their synchronization, assuming a correct transmission for a short while, the two systems will resynchronize automatically: it is an intrinsic property. One can easily understand the relevance of this class of ciphers.

The rest of the document will discuss how to implement the special SSSC presented in [5] and how to build the corresponding circuit. In order to show that it is possible to build an implementation that is able to work fast enough for a real application, we will use an FPGA-based circuit to cipher and decipher a video stream on the fly.

The outline of the document is the following. In Section II, after giving some generalities on SSSC, we focus on the special algorithm presented in [5]. In Section III, we discuss the points, which have been omitted in [5], having in mind in the present paper a practical implementation. Section IV is devoted to the hardware description along with performances. We ends up by suggesting a list of open problems that could be used as hints for further studies to improve the performances of the circuit.

II. Theoretical description of the algorithm

A. Self-synchronizing stream ciphers in a nutshell

The canonical description of a Self-Synchronizing Stream Ciphers admits the equations:

$$\begin{cases} x_k = \sigma_\theta^{ss}(y_{k-l}, \ldots, y_{k-l-M}) \\ y_k = e(x_k, u_k) \end{cases} \quad (1)$$

u_k, y_k and x_k stand respectively for the plaintext, the ciphertext and the running key. σ_θ^{ss} is the function that generates the time-varying key x_k. It is parametrized by θ which acts as the secret key. l is a non-negative integer standing for a possible delay and M is called the *delay of memorization*. e is the ciphering function. Insofar as σ_θ^{ss} depends on a fixed number of past values of y_k, it is clear that SSSC do not require any external synchronization protocol between the state of the cipher and the state of the decipher. The synchronization is an intrinsic property of the algorithm. The self-synchronization property potentially offers two advantages. First, because there is no need for any synchronization flags, fast throughput can be obtained. Furthermore, it is possible to implement one way communication scenarios. For instance, it is possible to broadcast a ciphered stream (say a video stream) over a large area to low power end-terminals. These low power devices can't necessary send information to the emitter to synchronize their state (and even if it could, the emitter can't necessary handle the synchronization of thousands of devices).

B. Background on piecewise linear dynamical systems

The algorithm to be implemented is deeply described in [5]. It involves a special class of hybrid systems, namely piecewise linear systems. It uses theoretical concepts borrowed from automatic control. This Section aims at recalling some backgrounds.

Dynamical systems are commonly used to describe the evolution in time of a system. Linear systems can be described by the set of equations, called state space representation:

$$\begin{cases} x_{k+1} = Ax_k + Bu_k \\ y_k = Cx_k + Du_k \end{cases}, \quad (2)$$

with

- A a matrix of dimension n called the dynamical matrix;
- B a vector (for a single input system) of dimension $n \times 1$ called the input vector,
- C a vector (for a single output system) of dimension $1 \times n$ called the output vector,
- D a scalar (for a single input/output system) called the direct transfer matrix,
- u_k a scalar corresponding to the input of the system,
- x_k a vector of dimension n standing for the internal state of the system,
- y_k a scalar corresponding to the output of the system.

The first equation of (2) describes the evolution of the internal state of the system and is named dynamical equation. The second one describes the value of the output y_k with respect to the current internal state x_k and possibly the input u_k (whenever D is not 0).

For switched linear dynamical system, a special class of hybrid systems, the matrix A, B, C and D may change at each iteration. They belong to a finite set of size J. They can be described by the state space representation

$$\begin{cases} x_{k+1} = A_{\sigma(k)}x_k + B_{\sigma(k)}u_k \\ y_k = C_{\sigma(k)}x_k + D_{\sigma(k)}u_k \end{cases} \quad (3)$$

with $\sigma : \mathbb{N} \to \{1, \ldots, J\}$; $\sigma(k) = i$; σ is called the switching function (or commutation rule) and i is called the mode.

Three properties of dynamical systems play an central role when a cryptographic application is sought.

- the *relative degree*. It is an integer value denoted hereafter r. r is the smallest quantity so that the output y_{k+r} at time $k + r$ depends explicitly of the input u_k at time k.
- *left-invertibility*. It is a property that ensures to recover in a unique way the input u_k from the knowledge of a finite sequence of outputs $y_k, y_{k+1}, \ldots, y_{k+l}$ with $l < \infty$.
- *flatness*. It is a property that guarantees that the internal state x_k can be expressed by mean of a function whose arguments are finite number of the output y_k and/or its backward and forward iterates.

C. Ciphers based on piecewise linear dynamical systems

The class of piecewise linear systems (PWL) has been largely proposed in the literature to design cryptosystems. Indeed, when $u_k = 0$, (3) turns into an autonomous system which can exhibit, for suitable choices of parameters (entries of the dynamical matrices A_i) and suitable switching rule σ, a chaotic behaviour. If so, the sequence of the successive x_k and so of the successive y_k is a random-looking sequence even though it is generated by deterministic equations. The sequence can be used to scramble information. The simplest scrambling is the mere addition. This is typically what happens in the well-known additive masking proposed for the first time in [6]. The ciphering consists in considering (3) with $u_k = 0$ and performing:

$$c_k = y_k + m_k$$

where m_k is the plaintext (the information to be encrypted) and c_k stands for the corresponding ciphertext.

Resorting directly to (3) as a ciphering algorithm is an alternative. For this scheme, the plaintext is precisely u_k and the corresponding ciphertext is y_k. Such a cipher is called message-embedding. Indeed, the plaintext u_k is embedded (injected) into a dynamics which is expected to be chaotic. A central question arises. Indeed, chaos, in particular its major property of sensitivity on initial conditions, makes sense for autonomous systems. On the other hand, chaos makes no longer sense for non autonomous systems unless the magnitude of u_k with respect to the state x_k is small. Indeed, if so, embedding

u_k into the dynamics can be merely considered as a small modulation. "Chaoticity", and so complexity of the ciphering, is thereby more or less preserved. Let us note that few works have been carried out to really bring out a correspondence between chaoticity (in terms for example of Lyapunov exponents) and security. Hence, the relevance of resorting to the message-embedding involving chaotic dynamics is somewhat questionable.

The recent paper [3] provides a survey on the different chaotic ciphering proposed so far in the literature and brings out a connection with standard cryptography. It has been shown therein that, regardless the dynamics (chaotic or not), the message embedding can be structurally equivalent to a standard Self-Synchronizing Stream Cipher. Such an equivalence is guaranteed under the conditions that the dynamical system has finite relative degree, is left invertible and flat. It has been particularized in [5] for piecewise linear systems. Thorough theoretical developments on left invertibility and flatness for discrete-time piecewise linear systems can be found in [7]. Let us note that the problem of left-invertibility in connection with chaos synchronization problem has been addressed also for SISO discrete-time systems in [8]. The result on equivalence rises a new interest on the message-embedding. Indeed, the result is interesting regarding the discussion on chaoticity and security. Since (3) can reduce to a standard SSSC, all the tools assessing the security of standard SSSC still apply for (3). In particular, since the equivalence is structural, it is independent from the dynamics (chaotic or not) and the assumption stating that u_k must be "small" can be relaxed. It is of first importance if (3) is no longer defined on the set of real numbers $\mathbb{R}$ but on discrete sets. Yet observe that in digital applications, precisely our context, (3) is defined on finite sets.

The dynamical system (3) being considered as a cipher, we must define the equations of the decipher. It is shown in [5] that, under the condition of finite relative degree, left invertibility and flatness, the equations of the decipher read:

$$\begin{cases} \hat{x}_{k+r+1} = P^r_{\sigma(k)}\hat{x}_{k+r} + B_{\sigma(k)}(\mathcal{T}^{r,0}_{\sigma(k)})^{-1}y_{k+r} \\ \hat{u}_{k+r} = -(\mathcal{T}^{r,0}_{\sigma(k)})^{-1}C_{\sigma(k+r)}A^{\sigma(k+r-1)}_{\sigma(k)}\hat{x}_{k+r} \\ \qquad\quad + (\mathcal{T}^{r,0}_{\sigma(k)})^{-1}y_{k+r} \end{cases} \quad (4)$$

with r the relative degree of the system,

$$P^r_{\sigma(k)} = A_{\sigma(k)} - B_{\sigma(k)}(\mathcal{T}^{r,0}_{\sigma(k)})^{-1}C_{\sigma(k+r)}A^{\sigma(k+r-1)}_{\sigma(k)} \quad (5)$$

and

$$\begin{aligned} A^{\sigma(k_1)}_{\sigma(k_0)} &= A_{\sigma(k_1)}A_{\sigma(k_1-1)}\cdots A_{\sigma(k_0)} \quad \text{if } k_1 \geqslant k_0 \\ &= \mathbb{1}_n \quad \text{otherwise} \end{aligned} \quad (6)$$

and

$$\begin{cases} \mathcal{T}^{i,j}_{\sigma(k)} = C_{\sigma(k+i)}A^{\sigma(k+i-1)}_{\sigma(k+j+1)}B_{\sigma(k+j)} & \text{if } j \leqslant i-1 \\ \mathcal{T}^{i,j}_{\sigma(k)} = D_{\sigma(k+i)} & \text{otherwise} \end{cases}. \quad (7)$$

III. Practical issues

The equations (3)-(7) describe the cryptosystem setup. However in order to implement this algorithm, one has to further address the following matters:

- How to involve the key, hereafter denoted θ, in the system?
- What are the possible structures for the matrices A_i, B_i, C_i and D_i?
- Which relevant function to use for the commutation rule σ?
- On which set the dynamical systems must be defined?

From a security point of view we could also investigate the following issues:

- What should be the dimension of the system?
- How many modes should the system have for the system to be secure?
- What should be the key size?

Now, let us try to answer the first set of questions.

A. How to involve the key bits in the system?

The only places where the key bits of θ can be involved are the matrices A_i, B_i, C_i and D_i and/or the commutation rule. A more detailed answer to this question is given in the following sections when discussing the matrices and commutation rule structures.

B. Matrices structures

We have previously stressed that for the dynamical system (3) to guarantee the existence of an inverse system and the self-synchronizing property, altogether left-invertibility, flatness and finite relative degree must apply. An important point to consider is the number of operations required to compute the output (ciphertext) of the system. It is easy to understand that using sparse matrices (especially for A_i) will reduce the number of required operations, thus reducing the area, the power and the time required. Taking into account these constraints, we suggest to use an horizontal companion structure for the A_i matrices. They are built as follows: the A_is entries are 1 on the superdiagonal, all the other entries are zero except for the last row. For the special dimension $n = 4$ chosen for our circuit, they read as:

$$A_i = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\alpha_i & -\beta_i & -\gamma_i & -\delta_i \end{pmatrix} \quad (8)$$

Actually, α_i, β_i, γ_i and δ_i, and more generally the entries of the last row, are the coefficients of the characteristic polynomials of the matrices A_i. For the sake of computational cost, the matrices B_i and C_i and D_i are chosen to be constant and they read

$$B_{\sigma(k)} = \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}^T \quad \forall\ k$$

and

$$C_{\sigma(k)} = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} \text{ and } D_{\sigma(k)} = \begin{pmatrix} 0 \end{pmatrix} \quad \forall\ k$$

Besides the sparsity and the related computational efficiency, this choice has been made because it is interesting insofar as left-invertibility and flatness are ensured for any values of the characteristic polynomial coefficients and any switching rule σ. Finally, the equality $\mathcal{T}_{\sigma(k)} = 1$ which is a further advantage from a computational point of view always applies.

C. Commutation Rule

The aim of the commutation rule is to generate the index i corresponding to the mode. First, to guarantee self-synchronization, the sequence generated by the commutation rule has to be the same in the cipher as in the decipher (at least after a few iterations). Second, the sequences of modes must not be directly accessible for any unauthorized party (eavesdropper). Therefore, a solution consists in choosing a commutation rule $\sigma(k)$ of the generic form:

$$\sigma(k) = f(\theta, [y_k, \ldots, y_{k-l}]), \tag{9}$$

where f is a function that depends on θ (the secret key) and on a finite number y_k (the outputs of the cipher) fixed by a positive integer l. The function f has to be chosen so that if the secret key θ is unknown, it is a hard task to get $\sigma(k)$ even though the successive y_k are accessible (indeed, they are conveyed through a public channel). The proposed function which fulfills these constraints is the dot product between the elements of the key and the past outputs, that is

$$\sigma(k) = \theta \cdot [y_k, \cdots, y_{k-l}]^T. \tag{10}$$

D. Finite sets

Another problem that must be solved when trying to implement the algorithm is the set of input symbols used and output symbols generated. Usually, when dealing with dynamical systems the quantities being manipulated are real numbers. This is typically what happens for chaotic systems (see the discussion provided in Subsection II-C). On the other hand, it is not possible for the intended application because digital processing require a representation of the input and output symbol with a bounded number of bits. This means that we can't use real numbers as input or output symbols when implementing this system. An alternative could be resorting to floating point numbers but this is a poor solution because the quantities are inevitably rounded. The behaviour would be machine dependent and it is clearly redhibitory.

In order to overcome this problem we look into mathematics and in particular algebra. Algebra defines structures called fields that are of particular interest. A field $\mathbb{F}$ is a set together with two operations usually called addition and multiplication and respectively denoted by $+$ and $\cdot$ so that the following axioms hold:

1) For all a and b in $\mathbb{F}$ both $a + b$ and $a \cdot b$ are in $\mathbb{F}$,
2) For all a, b and c in $\mathbb{F}$, associativity holds: $a + (b + c) = (a + b) + c$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$,
3) For all a and b in $\mathbb{F}$, commutativity holds: $a + b = b + a$ and $a \cdot b = b \cdot a$,
4) There exists an element of $\mathbb{F}$, called the additive identity element and denoted by 0 such that for all a in $\mathbb{F}$, $a + 0 = a$. Likewise there is an element, called the multiplicative identity element and denoted by 1, such that for all a in $\mathbb{F}$, $a \cdot 1 = a$. The element denoted 0 and the one denoted 1 have to be different,
5) For every a in $\mathbb{F}$, there exists an element $-a$ in $\mathbb{F}$ such that $a + (-a) = 0$. Similarly, for any a in $\mathbb{F}$ other than 0, there exists an element a^{-1} in $\mathbb{F}$ such that $a \cdot a^{-1} = 1$,
6) For all a, b and c in $\mathbb{F}$ distributivity of the multiplication over the addition holds: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

A finite field is a field with a finite number of elements. These are the ones that we can use in digital processing. The following subsections detail two of them.

1) Prime fields: Interestingly, the set of integers modulo an integer n, that is the structure $\mathbb{Z}/n\mathbb{Z}$, is a field if n is prime. And since it has a finite number of elements it could be used for our purpose. But using this kind of field might be not suitable because its order (the number of elements) has to be a prime which means that except for a field with two elements we can't use all the possible bit combinations. For instance, if one chooses a field with five elements, one would need $\lceil log_2(5) \rceil = 3$ bits to encode the different numbers. Hence, we would only use five out of eight bit combinations, which is clearly a non optimal use of the bits. Moreover, we are dealing with a hardware implementation. This means that every basic operation can have a great impact on the efficiency of the algorithm especially if we target a highly parallelized circuit as it is the case here. Indeed, except the case when one chooses $p = 2$, the parallel hardware implementation requires a huge area and has long time critical pathes that limit the maximum frequency at which the circuit can operate. This is mainly due to the modulus reduction which is time consuming and quite heavy in term of circuit area if p is not a power of two. The next section suggests the use of another kind of field which allows the use of all the bit combinations of any k bits length word.

2) Field extensions: A field extension is a field based on a smaller field. Elements of a field extension are vectors whose components belong to the underlying field (the smaller field). If the base field is of order p and that the elements of the field extension are of dimension n then, there are p^n possible combinations. This gives an idea about how to build the set of elements. In order for that set to be a field, we also need to define an addition and a multiplication such that the field axioms 1)-6) are fulfilled. To this end, we consider elements of the field extension as polynomials where each component of the vector corresponds to the coefficient of a monomial of the polynomial. For instance if we consider the element (α, β, γ) of the extension field $\mathbb{F}_{p^3}$ it can be seen as the polynomial $\alpha \cdot x^2 + \beta \cdot x + \gamma \cdot 1$ where α

and β and γ belong to the field $\mathbb{F}_p$. It is therefore quite natural to use the polynomial addition and multiplication to define the two field operations. We will see in the following paragraphs that adding two elements is straightforward and that it is a bit more tricky to multiply them. The examples in the next paragraphs are given with $p = 2$ and $n = 3$ but the same kind of result holds for any prime p and any positive integer n.

a) Addition: The polynomial addition only consists in adding each monomial component over the field they are defined on. For instance, we can take the two following elements $A = 110$ (or $A = x^2 + x$) and $B = 101$ (or $B = x^2 + 1$) of $\mathbb{F}_{2^3}$. Applying the polynomial addition and performing the operation of the coefficients of the monomials in $\mathbb{F}_2$ yields

$$\begin{array}{cccc} & 1 & 1 & 0 \\ + & 1 & 0 & 1 \\ \hline & 0 & 1 & 1 \end{array} \quad \text{or} \quad \begin{array}{cccc} & x^2 & +x & +0 \\ + & x^2 & +0 & +1 \\ \hline & 0 & +x & +1 \end{array} .$$

It is easy to see that whatever the elements of this structure are, the result of the addition remains in the structure. We will see that the multiplication has not this property and requires an additional operation.

b) Multiplication: Let us turn to the polynomial multiplication involving the two elements of the previous example:

$$\begin{array}{ccccccc} & & & 1 & 1 & 0 \\ \times & & & 1 & 0 & 1 \\ \hline & & & 1 & 1 & 0 \\ & & 0 & 0 & 0 & . \\ + & 1 & 1 & 0 & . & . \\ \hline & 1 & 1 & 1 & 1 & 0 \end{array} .$$

The result of the polynomial multiplication is a polynomial of degree at most the sum of the degree of the two multiplied polynomials. Thus, in some cases the resulting polynomial won't be an element of the original structure. To tackle this problem, in the same way we build a field from the ring $\mathbb{Z}$ by reducing the elements modulus a prime, we can build a field from the polynomial ring $\mathbb{F}_2[X]$ by reducing the elements by an irreducible polynomial. An irreducible polynomial is the polynomial equivalent of a prime that is, a polynomial that can't be written as a product of two or more polynomial different from 1 and itself. In the example above the irreducible polynomial x^3+x+1 can be used. Applying the modular reduction with this polynomial, we get the following result:

$$\begin{array}{ccccccc} & 1 & 1 & 1 & 1 & 0 \\ - & 1 & 0 & 1 & 1 & . \\ \hline & 0 & 1 & 0 & 0 & 0 \\ - & & 1 & 0 & 1 & 1 \\ \hline & & 0 & 0 & 1 & 1 \end{array} .$$

The result of the modular reduction is now an element of $\mathbb{F}_{2^3}$. It can be shown that there exists at least one such polynomial for any n which means that it is possible to build a field with p^n elements for any n as long as p is a prime.

E. Key size

According to the points discussed earlier, we can derive the key size of the system. Let b be the number of bits of an element of the alphabet of the plaintext or ciphertext, let n be the dimension of the system and J the number of modes of the system. If one wants to use distinct values in each A_i matrix, there is a need of n times J values. Each value is b bits length, therefore the number of bits of the key is given by

$$Key_{size} = n \cdot J \cdot b. \tag{11}$$

In order for the algorithm to have a minimum security with respect to an eavesdropper, Key_{size} must be large enough so that an exhaustive search would not succeed in reasonable time. We could then consider choosing a key size ranging from 128 to 256 bits. That induces constraints on n, J and b.

IV. Circuit

As said in the introduction stream ciphers are particularly well suited when high throughputs are required. Therefore, we decided to build a device that is able to cipher and decipher a video stream on the fly. We used an Altera development kit based on the Cyclone III FPGA in order to build this device. An analog source broadcasts a video stream. This stream is then ciphered and deciphered. Then, a multiplexor selects the signal to be displayed on the screen (input stream, ciphered stream or deciphered stream).

A. Implementation and performances

The aim of the circuit is to build a high throughput secure streaming device. Therefore, we have chosen a fully parallel implementation of (3)-(7). When comparing the equations (3) (cipher) and (4) (decipher), it turns out that the structure of the cipher and of the decipher looks alike. However, the differences lies in the complexity which is mainly due to the computation of the matrices $P^r_{\sigma(k)}$ and $A^{\sigma(k_1)}_{\sigma(k_0)}$ (see Equations (5) and (6)) for the decipher.

In order to get an idea of the minimum throughput that the system should reach to build our video streaming system, one needs to compute the throughput of the input stream. The analog to digital converter produces 3×8 bits at frequency 25 MHz. For our application, we have chosen $b = 8$ so the system can only process 8 bits at a time. Therefore, it should work at least at frequency 75 MHz. And the throughput of the system should be $8 \times 75 \times 10^6 = 600\,000\,000$ bits per second. We succeeded to build the circuits (the cipher and the decipher) so that they work fast enough for our application. It terms of area, it uses 903 (respectively 969) logical elements for the cipher (respectively the decipher). A picture of the setup is shown on the Figure 1. It shows the analog source connected to the device. The output is displayed on the screen.

We then wondered what was the maximum frequency to which the circuit could work. At first, one should note that the structure of the circuit (and therefore the

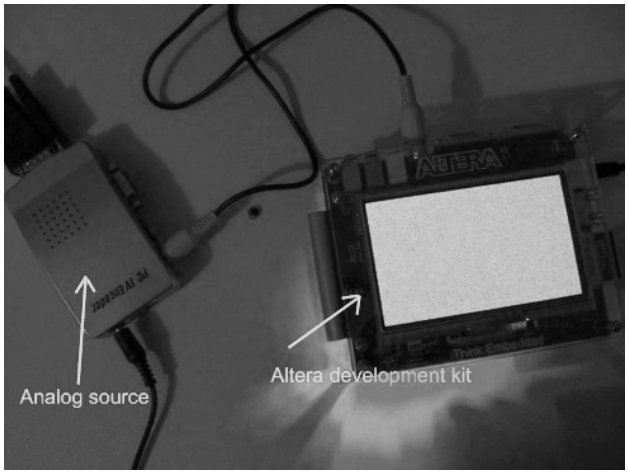

Fig. 1: Picture of the setup

n (dimension)	4	5	6	7	8
Cipher	216	186	194	178	168
Decipher	168	155	162	152	149

TABLE I: Maximum frequency (in MHz) of the system with $J = 4$ modes and symbols of $b = 8$ bits.

maximum frequency) heavily depends on the value of the three circuit parameters which are the dimension n of the dynamical system, the number J of modes and the number b of bits per symbol. Moreover, due to the obvious complexity of the decipher compared to the cipher, one can easily guess that these two circuits have different timing and area characteristics and that the maximum frequency of the decipher is much lower that the one of the cipher. Therefore, when investigating the performances of this cryptosystem, it is better to focus on the decipher which represents the worst case scenario. Table I shows different results of the system after having synthesized a circuit with $b = 8$ and $J = 4$ using Quartus for a Cyclone III FPGA. The option of the synthesizer gave the advantage to the time optimization over the area optimization.

Table I highlights what was expected about the performance differences between the cipher and the decipher. One recall that the size of the key associated with these systems can be calculated with the formula given by the Equation (11). For instance, for a system of dimension $n = 8$, with $J = 4$ modes and $b = 8$ bits per symbol, $Key_{size} = 8 \times 4 \times 8 = 256$ bits. One can now investigate the evolution of the throughput of the system according to the parameter n (the dimension of the system): there is no pipeline and the circuit is fully parallel, symbols are processed one at a time each clock cycle, meaning that the throughput of the system is simply the value of the maximum frequency times the number of bits per symbol. For instance, if one chooses again the system $n = 8, J = 4, b = 8$, the throughput is limited by the decipher and is $8 \times 149 \times 10^6 = 1.192$ Gbit/s[1].

[1]Note that this is a theoretical value. So far, no experiment has been done to validate this result.

B. Improvements and concluding remarks

A first and obvious improvement which can be done targets the $\mathbb{F}_{2^n}$ multipliers. Due to the structure of the algorithm they are heavily used and any small improvement can have a significant impact on the performances. So far, these components are implemented intuitively meaning that we perform a polynomial multiplication followed by a modular reduction. These blocks can be replaced with efficient ones widely described in the literature.

It may be also interesting to find another commutation rule. There are two reasons for this. The first one is due to the large area required to implement this part of the algorithm compared to the rest of it. Indeed, it takes more than half of the area of the decipher and even more in the cipher. The second reason is for cryptographic sake. The commutation rule is implemented as a dot product and the linear property of this function might be a problem. So far, there is no formal investigation concerning its impact on the security. Yet observe that huge part of the nonlinearity of the system relies on the commutation rule. This points requires thorough insights.

Finally, the difference on the complexity of the cipher and the decipher must be reduced. From an automatic control point of view, the decipher is a left inverse dynamical system. As a result, we must look into structures of dynamical systems playing the role of the cipher which leads to simple structure of left inverse. Actually, such a purpose is an open issue which deserves theoretical insights but with high impact for hardware sakes.

ACKNOWLEDGMENT

We would like to thank Philippe Guillot who gave us some precious advices. We also thank Phuoc Vo-Tan for his work. Finally, we express our gratitude to Yves Berviller, Serge Weber and Sébastien Calvi for their implication in the achievement of the test device.

REFERENCES

[1] Ahlquist, Brent Nelson and Rice "Optimal finite field multipliers for fpgas" In: *FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*. Springer-Verlag. London, UK. pp. 51–60, 1999.

[2] Delfs and Knebl "Introduction to cryptography: principles and applications". *Springer-Verlag New York, Inc.. New York, NY, USA*, 2001.

[3] Millérioux, Amigó and Daafouz "A connection between chaotic and conventional cryptography" *IEEE Trans. on Circuits and Systems I: Regular Papers*, 2008.

[4] Shannon "Communication theory of secrecy systems" *Bell Systems Tech. Journ.*, **28**, 657–715, 1949.

[5] Vo-Tan, Millérioux and Daafouz, "Left invertibility, flatness and identifiability of switched linear dynamical systems: a framework for cryptographic applications", *International Journal of Control*, **83**(1), 145–153, 2010.

[6] Wu and Chua, "A simple way to synchronize chaotic systems with applications to secure communications systems" *International Journal of Bifurcation and Chaos*, **3**(6), 1619–1627, 1993.

[7] G. Millérioux and J. Daafouz, "Flatness of switched linear discrete-time systems" *IEEE Trans. on Automatic Control* **54**(3), 615–619, 2009.

[8] I. Belmouhoub, M. Djemai, and J.-P. Barbot, "Observability Quadratic Normal Form for Discrete-Time Systems" *IEEE Trans. on Automatic Control*, **50**(7), 1031–1038, 2005.