Non-triangular self-synchronizing stream ciphers
Julien Francq, Loic Besson, Paul Huynh, Philippe Guillot, Gilles Millérioux, Marine Minier

To cite this version:

HAL Id: hal-03081725
https://hal.science/hal-03081725
Submitted on 20 Jul 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Non-Triangular Self-Synchronizing Stream Ciphers

Julien Francq, Loïc Besson, Paul Huynh, Philippe Guillot, Gilles Millerioux and Marine Minier

Abstract—In this paper, we propose an instantiation, called Stanislas, of a dedicated Self-Synchronizing Stream Cipher (SSSC) involving an automaton with finite input memory using non-triangular state transition functions. Previous existing SSSC are based on automata with shifts or triangular functions ($T$–functions) as state transition functions. Our algorithm Stanislas admits a matrix representation deduced from a general and systematic methodology called Linear Parameter Varying (LPV). This particular representation comes from the automatic theory and from a special property of dynamical systems called flatness.

Hardware implementations and comparisons with some state-of-the-art stream ciphers on Xilinx FPGAs are presented. It turns out that Stanislas provides bigger throughput than the considered stream ciphers (synchronous and self-synchronizing) when straightforward implementations are considered. Moreover, its self-synchronization delay is much smaller than the SSSC Moustique (40 clock cycles instead of 105) and the standard approach CFB1-AES128 (40 clock cycles instead of 128).

Index Terms—Self-Synchronizing Stream Ciphers, automata with finite input memory, flatness, hardware implementations.

I. Introduction

Self-Synchronizing Stream Ciphers (SSSC) were patented in 1946. The basic principle of such ciphers is to encrypt every plaintext symbol with a transformation that only involves a fixed number of previous ciphertext symbols. Therefore, every ciphertext symbol is correctly deciphered provided that previous symbols have been properly received. This self-synchronization property has many advantages and is especially relevant to group communications. In this respect, since 1960, specific SSSCs have been designed and are still used to provide bulk encryption (for Hertzian line, RNIS link, etc.) in military applications or governmental radio mobile networks [1], [2].

The canonical form of the SSSC combines a shift register, which acts as a state register with the ciphertext as input, together with a filtering function that provides the running key stream. The cryptographic complexity of the canonical form of the SSSC lies in the filtering function. In the early 90s, studies have been performed [1], [3] to propose secure designs of SSSCs. These works have been followed by effective constructions ([4], [5], [6]). What motivates the present proposal for a new SSSC is that, till now, all of these SSSC schemes have been broken ([7], [8], [9], [10], [11]). And up to our knowledge, since 10 years, no other proposals of SSSCs has been made. Clearly, SSSCs can be naturally built using a block cipher by applying the Cipher Feedback (CFB) mode. However, the computational cost of CFB is one full block cipher operation per digit. So for single-bit digits, it is $n$ times less efficient than synchronous stream encryption modes such as Output Feedback (OFB) or Counter (CTR) Mode, with $n$ the block length. For example, AES in single-bit CFB mode (as defined as “CFB1-AES128” in NIST SP 800-38a [12]) is 128 times less efficient than AES in CTR mode. As a consequence, it seems interesting to propose dedicated SSSCs and consider them as a category of primitives on their own.

The aim of the present paper is to propose a new framework, along with an instantiation called Stanislas, to design SSSCs. The design approach is based on both the special feature of Finite State Machines admitting a matrix representation, called LPV (Linear Parameter Varying) automata and on a property of dynamical systems named flatness coming from control theory. Flatness is interpreted in terms of the structure of the adjacency matrix of a graph associated to the automaton from which self-synchronization can be easily characterized. The matrix representation is thus a generalization of Rational Linear Finite State Machines [13], [14]. The use of flatness for the sake of cryptography has been first proposed in [15]. It has been shown that flatness characterizes the self-synchronization property. Moreover, the matrix representation allows to design automata which can be more general than $T$–functions as it was the case over the past years (see [16], [6] as examples). One of the benefits of this approach is that we could introduce nonlinearities with proved properties in the matrix representation, what a shift register does not always permit. Due to this peculiarity, the class of admissible automata to act as SSSCs is thereby enlarged.

In the present paper, a complete cipher, called Stanislas (for Secure Transmission Algorithm with Non triangular Iterative Structure Looking After Self-synchronization) and designed from the LPV framework, is described. The Key Schedule, the design rationale and the security analysis are provided. Next, hardware implementation results on Xilinx FPGA platforms of Stanislas are performed and compared with some state-of-the-art competitors: some
Synchronous Stream Ciphers coming from eSTREAM portfolio (Trivium [17] and Grain [18]), one SSSC (Moustique) and the only known feedback mode (the NIST-standardized CFB1-AES128). Interestingly, Stanislas provides the highest throughput on Xilinx Spartan-6 XC6SLX75T FPGAs compared to its stream ciphers and SSSC competitors, implemented in a straightforward manner. Moreover, with the same comparison conditions, its synchronization delay is much smaller than the SSSC Moustique (40 clock cycles instead of 105) and the standard approach CFB1-AES128 (40 clock cycles instead of 128), which provides a decisive advantage for applications when low-latency synchronization is required (e.g., Telecom).

The paper is organized as follows. Section II recalls the theoretical results linking together flatness and SSSC. Section III presents the new SSSC Stanislas. The design rationale and the security analysis are detailed in Section IV. Finally, Section V provides hardware implementation results and comparisons. Section VI concludes this paper.

II. THEORETICAL FOUNDATIONS AND FLATNESS

After few generalities on stream ciphers, it is recalled in this section the main results of [15] concerning the design of SSSC based on the property of flatness.

A. Generalities on Stream Ciphers

For a stream cipher, it must be given an alphabet $A$, that is, a finite set of basic elements named symbols. The set $A$ stands in this paragraph as a general notation without any specific alphabet. Typically, $A$ could be composed of 1 or several bits elements. Hereafter, the index $t \in \mathbb{N}$ will stand for the discrete-time. On the transmitter part, the plaintext (also called information or message) $m \in \mathcal{M}$ ($\mathcal{M}$ is the message space) is a string of plaintext symbols $m_t \in A$. Each plaintext symbol is encrypted, by means of an encryption (or ciphering) function $e$, according to:

$$c_{t+r} = e(z_{t+r}, m_t),$$

where $z_t \in A$ is a so-called keystream (or running key) symbol delivered by a keystream generator. The function $e$ is invertible for any prescribed $z_t$. The resulting symbol $c_t \in A$ is the ciphertext symbol. The integer $r \geq 0$ stands for a potential delay between the plaintext $m_t$ and the corresponding ciphertext $c_{t+r}$. This is explained by computational or implementation reasons, see [16] for example. Consequently, for stream ciphers, the way how to encrypt each plaintext symbol changes on each iteration. The resulting ciphertext $c \in \mathcal{C}$ ($\mathcal{C}$ is called the ciphertext space), that is the string of symbols $c_t$, is conveyed to the receiver through a public channel.

At the receiver side, the ciphertext $c_t$ is deciphered according to a decryption function $d$ which depends on a running key $\hat{z}_t \in A$ delivered, similarly to the cipher part, by a keystream generator. The decryption function $d$ obeys the following rule. For any two keystream symbols $\hat{z}_{t+r}, z_{t+r} \in A$, it holds that

$$\hat{m}_{t+r} := d(c_{t+r}, \hat{z}_{t+r}) = m_t \text{ whenever } \hat{z}_{t+r} = z_{t+r}.$$  \hspace{1cm} (2)

Equation (2) means that the running keys $z_t$ and $\hat{z}_t$ must be synchronized for a proper decryption. The generators delivering the keystreams are parametrized by a secret key denoted by $K \in \mathcal{K}$ ($\mathcal{K}$ is the secret key space). The distinct classes of stream ciphers (synchronous or self-synchronizing) differ each other by the way on how the keystreams are generated and synchronized. Next, we detail the special class of stream ciphers called Self-Synchronizing Stream Ciphers.

B. Keystream Generators for Self-Synchronizing Stream Ciphers

A well-admitted approach to generate the keystreams has been first suggested in [1]. It is based on the use of so-called finite state automata with finite input memory as described below. This is typically the case in the cipher Moustique [19]. At the ciphering side, the automaton delivering the keystream takes the form:

$$\begin{cases}
    x_{t+1} = f_K(x_t, m_t), \\
    z_{t+r} = h_K(x_t)
\end{cases}$$  \hspace{1cm} (3)

where $x_t \in A$ is the internal state, $f$ is the next-state transition function parametrized by $K \in \mathcal{K}$. As previously stressed, the delay $b$ is introduced to cope with special situations, in particular when the computation of the output (also called filtering) delivered by the function $h$ involves $r$ successive operations processed at time instants $t, \ldots, t+r$. Those operations will be here matrix multiplications as detailed later in Equation (14). Substituting $m_t$ by its expression (2) yields an automaton described by

$$\begin{cases}
    x_{t+1} = g_K(x_t, c_{t+r}), \\
    z_{t+r} = h_K(x_t)
\end{cases}$$  \hspace{1cm} (4)

If such an automaton has finite input memory, it means that, by iterating (4) a finite number of times, there exists a function $\ell_K$ and a finite integer $M$ such that

$$x_t = \ell_K(c_{t+r-1}, \ldots, c_{t+r-M})$$  \hspace{1cm} (5)

and thus,

$$z_{t+r} = h_K(\ell_K(c_{t+r-1}, \ldots, c_{t+r-M})).$$  \hspace{1cm} (6)

Actually, the fact that the keystream symbol can be written in the general form

$$z_{t+r} = \sigma_K(c_{t-r}, \ldots, c_{t-\ell}),$$  \hspace{1cm} (7)

with $\sigma_K$ a function involving a finite number of past ciphertexts from time $t-\ell$ to $t-\ell'$ ($\ell, \ell' \in \mathbb{Z}$), is a common feature of the SSSC. Equation (7) is called the canonical equation.

Remark 1: The benefits of implementing the recursive forms (3) or (4) instead of directly implementing the canonical form (7) is that we can obtain nonlinear functions $\sigma_K$ of high complexity by implementing simpler nonlinear functions $f_K$ or $g_K$. The complexity results from the successive iterations which act as composition operations.
At the deciphering side, the automaton takes the form

\[
\begin{align*}
\hat{x}_{t+1} &= g_K(\hat{x}_t, c_{t+r}), \\
\hat{z}_{t+r} &= h_K(\hat{x}_t)
\end{align*}
\]  

(8)

where \( \hat{x}_t \) is the internal state. Similarly to the cipher part, the automaton having a finite input memory, it means that, by iterating Equation (8) a finite number of times, one also obtains

\[
\hat{x}_t = \ell_K(c_{t+r-1}, \ldots, c_{t+r-M}),
\]

and thus,

\[
\hat{z}_{t+r} = h_K(\ell_K(c_{t+r-1}, \ldots, c_{t+r-M})).
\]

Hence, it is clear that after a transient time of maximal length equal to \( M \), it holds that, for \( t \geq M \),

\[
\hat{x}_t = x_t \quad \text{and} \quad \hat{z}_{t+r} = z_{t+r}.
\]

(9)

In other words, the generators synchronize automatically after at most \( M \) iterations. Hence, the decryption is automatically and properly achieved after at most \( M \) iterations too. No specific synchronizing protocol between the cipher and the decipher is needed. This explains the terminology Self-Synchronizing Stream Ciphers. The integer \( M \) is called the synchronization delay.

Hereafter, the considered automata will be assumed to operate on the \( q \) elements finite field \( \mathbb{F} = \mathbb{F}_q \) where \( q \) is a prime power.

C. Flat LPV Automata and SSSC

For the automaton described by (3) (or the equivalent automaton described by (4) after substitution) to get a finite input memory feature (see (5)), the solutions proposed in the open literature call for state transition functions \( g_K \) in the form of shifts or more generally \( T \)-functions (\( T \) for Triangle). It is recalled that \( T \)-functions are functions that propagate dependencies in one direction only. Till now, none of the proposed SSSCs has involved non-triangular state transition functions although \( T \)-functions are known to suffer from weaknesses [9]. Indeed, \( T \)-functions induce a propagation of differential properties which make them easier to cryptanalyse. It is explained by the fact that no systematic methodology for constructing finite automata with finite input memory and involving general non-triangular state transition functions was proposed so far. Actually, in [15], a particular class of automata called flat LPV (Linear Parameter-Varying) has been introduced and it has been shown that this class allows to define such an expected systematic methodology. Let us recall what is a flat automaton.

**Definition 1:** An automaton described by the dynamics \( f \) verifying

\[
x_{t+1} = f(x_t, m_t)
\]

(10)

where \( x_t \in \mathbb{F}^n \) is the state, \( m_t \in \mathbb{F} \) is the input, is said to be flat, if there exists a function

\[
h : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}
\]

\[
c_t = h(x_t, m_t)
\]

such that all system variables can be expressed as a function of \( c_t \) and a finite number of its backward and forward shifts.

The output \( c_t \) is called the flat output. Hence, by definition, there exists a function \( \mathcal{F} \) such that

\[
x_t = \mathcal{F} (c_{t+t_0}, \ldots, c_{t+t_1})
\]

(11)

where \( t_0 \) and \( t_1 \) are \( \mathbb{Z} \)-valued integers. A central remark is that (11) is nothing but the canonical equation of an SSSC (compare with (5)). As a direct consequence, a flat automaton is equivalent to being an SSSC and thus is central for design perspectives.

LPV automata, defined over a field \( \mathbb{F} \), are described by the following state space representation:

\[
x_{t+1} = A_{\rho(t)} x_t + B m_t
\]

(12)

\( x_t \in \mathbb{F}^n \) is the state vector, \( m_t \in \mathbb{F} \) is the input. The matrices \( A \in \mathbb{F}^{n \times n} \) and \( B \in \mathbb{F}^{n \times 1} \) are respectively the dynamical matrix and the input matrix. The output \( c_t \) is defined as

\[
c_t = C x_t
\]

(13)

with \( C \in \mathbb{F}^{1 \times n} \) the output matrix. The matrix \( B \) is the input matrix and defines the component \( x_s \) on which the symbol \( m_t \) is added. Let us note that \( m_t \) can be added to several components. Such a system is called Linear Parameter-Varying because it is written with a linear dependency with respect to the state vector. The set of all varying parameters of \( A \) are collected on a vector denoted by

\[
\rho(t) = [ \rho^1(t), \rho^2(t), \ldots, \rho^L(t) ] \in \mathbb{F}^L
\]

where \( L \) is the total number of non-zero (possibly varying) entries. Such automata can exhibit nonlinear dynamics. Indeed, the nonlinearity is obtained by defining the varying parameters \( \rho^i(t) \) as nonlinear functions \( \varphi^i : \mathbb{F}^{s+1} \to \mathbb{F} \) of the output \( c_t \) (or a finite number of shifts) \( \rho^i(t) = \varphi^i(c_{t}, c_{t-1}, \ldots, c_{t-s}) \) with \( s \) a natural number. Let us notice that the notation \( \rho^i(t) \) (usual in the literature for LPV systems) is somehow abusive because it does not reflect an explicit dependency with respect to the time \( t \) but on quantities, here \( c_t \), indexed with \( t \). Furthermore, the LPV structure is suitable to construct non triangular state transition functions. Indeed, because of the varying entries, the state transition function is not triangular if there does not exist a common and constant triangularization basis over the whole set of matrices \( A_{\rho(t)} \) with \( \rho(t) \in \mathbb{F}^L \). Hence, it suffices to select the position of the varying parameters \( \rho^i(t) \) in the matrix \( A \) accordingly.

As a simple illustration, the automaton governed by Equation (12) with the setting

\[
A_{\rho(t)} = \begin{pmatrix}
a & 0 & 1 \\
a & \rho^1(t) & 0 \\
a & 1 & \rho^2(t)
\end{pmatrix}, \quad B = \begin{pmatrix}
1 \\
0 \\
0
\end{pmatrix}
\]

with \( a \) a constant element in \( \mathbb{F} \), \( \rho^1(t) = c_t \cdot c_{t-1} \) and \( \rho^2(t) = (c_{t-2})^2 \), is an LPV automaton and does not admit a constant triangularization basis.
A flat LPV automaton is an LPV automaton of which state vector $x_t$ verifies (11).

Let us recall from [15] the main proposition which allows to define a family of SSSCs based on LPV automata. For brevity, we introduce the following notation. For $t_2 \geq t_1$, denote by $\prod_{r=t_2}^{t_1} A_{\rho(r)}$ the product of matrices $A_{\rho(r)}$ from $t_2$ to $t_1$. For $t_2 < t_1$, define $\prod_{r=t_2}^{t_1} A_{\rho(r)} = 1_n$ (the identity matrix of dimension $n$). Finally, let $T$ be the scalar defined by $T = C \prod_{r=t+1}^{t+r-1} A_{\rho(r)} B$.

Proposition 1: If the LPV finite state automaton defined by (12) is flat, defining the keystream with delay $r$ as

$$z_{t+r} = C \prod_{t+l=r-1}^{t+r-1} A_{\rho(l)} x_t,$$

and the ciphering function as

$$c_{t+r} = z_{t+r} + T m_t,$$

the set of equations (12), (14) and (15) define the ciphering part of an SSSC.

On the other hand, consider the finite state automaton with internal state $\tilde{x}_t$ with dynamics given by

$$\begin{cases} \tilde{x}_{t+1} = P_{\rho(t:t+r)} \tilde{x}_t + B T^{-1} c_{t+r} \end{cases}$$

with

$$P_{\rho(t:t+r)} = A_{\rho(t)} - B T^{-1} C \prod_{l=t+r-1}^{t} A_{\rho(l)}$$

along with the keystream $\tilde{z}_t$ defined as

$$\tilde{z}_{t+r} = C \prod_{t+l=r-1}^{t+r-1} A_{\rho(l)} \tilde{x}_t$$

and the deciphering function obeying

$$\hat{m}_{t+r} = T^{-1} c_{t+r} - \tilde{z}_{t+r}.$$

Then, the set of equations (16-19) define the deciphering part of an SSSC.

The proof given in [15] consists in showing that, if the LPV finite state automaton defined by (12) is flat, then there exists an integer $M$ such that the synchronization error $x_k - \tilde{x}_{t+r}$ reaches zero after a finite transient time of length $M$. The integer $M$ is the synchronization delay. Actually, it is shown that flatness is equivalent to the existence of an integer $M$ such that for all $t \geq 0$,

$$P_{\rho(t+M-1:t+M-1+r)} P_{\rho(t+M-2:t+M-2+r)} \cdots P_{\rho(t:t+r)} = 0$$

where the product (20) results from the composition of the state transition functions of the deciphering automaton. Let us note that $T$ and $r$ are independent.

This LPV framework for the design of SSSC is new regarding the literature devoted to the design of SSSC. In particular, it really differs from the serial and parallel constructions proposed in the 90s by Maurer [1].

According to Remark 1, implementing the recursive form (12) and (16) instead of the canonical form (5) is more efficient from a computational point of view.

It is recalled that the non-linearity is obtained by defining the values of every varying parameters $\rho^i(t)$ ($i = 1, \ldots, L$) involved in the matrices of (12-19) as non-linear functions $\varphi^i$ of a finite number of past cryptograms $(\rho^i(t) = \varphi^i(c_1, c_{t-1}, \cdots, c_{t-s}))$. Those functions will be implemented in the form of S-boxes denoted $S$.

$$\varphi^i : \mathbb{F}^{s+1} \rightarrow \mathbb{F}$$

$$(c_1, c_{t-1}, \cdots, c_{t-s}) \mapsto S(c_1, c_{t-1}, \cdots, c_{t-s}, SK_i)$$

where $SK_i$ is the subkey number $i$ derived from the secret key denoted with $K$.

The point is that the LPV automaton defined by (12) must be flat for any secret key $K$ and any realization of $\rho(t)$. In other words, flatness must be a generic property of (12). Designing an LPV automaton (12) which is generically flat relies on an admissible realization of a corresponding structured linear system. A structured linear system is a linear system only defined by the sparsity pattern of the state space realization matrices. In other words, for a structured linear system, we distinguish between the entries that are fixed to zero and the other ones that can take any value in $\mathbb{F}$, including the ones which are time-varying.

Hence, a structured linear discrete-time system, denoted by $\Sigma_A$, is a system that admits the form:

$$\Sigma_A : \begin{align} x_{t+1} & = I_A x_t + I_B m_t. \end{align}$$

The entries of the matrices of (22) are ‘0’ or ‘1’. In particular, the entries $A(i, j)$ of $I_A$ (resp. $B(i)$ of $I_B$) that are ‘0’ mean that there are no relation (dynamical interaction) between the state $x_{t+1}$ at time $t + 1$ and the state $x_t$ at time $t$ (resp. the state $x_{t+1}$ at time $t + 1$ and the input $m_t$ at time $t$). The entries that are ‘1’ mean that there is a relation. As a simple example, let us consider again the LPV system with the setting

$$A_{\rho(t)} = \begin{pmatrix} a & 0 & 1 \\ a & \rho^1(t) & 0 \\ a & 1 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

where $a$ is a constant element in $\mathbb{F}$, $\rho^1(t)$ and $\rho^2(t)$ are varying parameters in $\mathbb{F}$. The dynamical matrix and the input matrix $I_A$ and $I_B$ of the corresponding structured linear system read:

$$I_A = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \quad I_B = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

As a consequence, if the structural linear system (22) derived from (12) is flat, the flatness will hold for any $\rho(t)$ or equivalently any nonlinearity $\varphi^i$ (any S-box will be admissible). Hence, the challenge is to define a methodology to construct flat linear structural systems. It is the purpose of the graph-based approach provided in [20] which follows the steps recalled in Appendix B. Roughly speaking, given a triplet $(n, r, n_a)$ with $n$ the dimension of the state, $r$ the delay and $n_a$ the number of non-zero entries of the matrix A, a digraph $G(\Sigma_A)$ is constructed according to given rules and the matrices $I_A$ and $I_B$ are derived.

The triplet $(n, r, n_a)$, the number of non-linear functions $\varphi^i$ and their locations in the matrix $I_A$ determine a family of flat LPV-based SSSC. Next subsection aims at
summarizing the steps needed for the design of such a family. Then, a particular instantiation, leading to the SSSC called Stanislas, is given in next section.

1) Summary for the construction of SSSC from a flat LPV-based automaton: Choose a triplet \((n, r, n_a)\) with \(n\) the dimension of the state, \(r\) the delay and \(n_a\) the number of non-zero entries of the matrix \(A\).

**Step S1:** Choose a component \(x_i^t\) on which the plaintext symbol \(m_t\) is added. It follows that \(B = (0 \ldots 1 \ 0 \ldots 0)^t\) (the entries 1 is located at column \(i\)).

**Step S2:** Choose a component \(x_i^t\) (\(i \in \{1, \ldots, n\}\)) as the desired flat output \(y_t = x_i^t\). It follows that \(C = (0 \ldots \cdot 0 \ 0 \ldots 0)\) (the only entry 1 is located at the \(i\)-th column of \(C\)). It can be shown that for the special case when \(B = (1 \ 0 \ldots 0)^t\), \(i\) must be equal to \(r\) for \(y_t = x_i^t\) to be a flat output.

**Step S3:** Construct the corresponding digraph \(G(\Sigma_A)\) according to Step 1-5 given in Appendix B and derive the matrices \(I_A\) and \(I_B\) of the structured linear system \(\Sigma_A\).

**Step S4:** Replace some of the non-zero entries of \(I_A\) by a nonlinear function \(\rho^1(t) = \varphi^1(c_t, c_{t-1}, \cdots, c_{t-s})\) to construct the matrix \(A_{\rho}(t)\) of (12) and set \(B = I_B\). Not all ‘1’ entries of \(I_A\) must be assigned to a non-linear function. Some of them can be merely constant. The choice must obey a trade-off between complexity of the architecture and security (a matter discussed in next section). Since the construction ensures structural flatness, any choice will preserve the self-synchronization property.

**Step S5:** Complete the design by deriving the equations of Proposition 1. In particular, calculate the matrix (17) governing the state transition function of the automata (16) ensuring the deciphering.

**Example:** Consider the triplet \((n = 2, r = 2, n_a = 5)\).

Choose
\[
B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 & 1 \end{pmatrix}.
\]

The particular setting of the matrix \(C\) means that the component \(x_2^t\) of the state vector of the LPV system (12) is the desired flat output.

For \(n_a = 5\), Steps 1-5 given in Appendix B give
\[
I_A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad I_B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.
\]

Let us keep constant and equal to 1 the first three entries of \(A\) and let the fourth entry be a nonlinear function. It is denoted by \(\rho^1(t)\). This finally leads to the following matrix \(A_{\rho}(t)\):
\[
A_{\rho}(t) = \begin{pmatrix} 1 \\ 1 \\ \rho^1(t) \end{pmatrix}.
\]

Calculate \(P_{\rho(t+2)} = A_{\rho}(t) - BCA_{\rho(t+1)}A_{\rho}(t)\). One obtains
\[
P_{\rho(t+2)} = \begin{pmatrix} -\rho^1(t+1) \\ 1 \\ -\rho^1(t)\rho^1(t+1) \\ \rho^1(t) \end{pmatrix}.
\]

Next sections are devoted to the specifications, design rationale and security analysis of a complete SSSC based on flat LPV dynamical systems as described above. The cipher is called Stanislas for Secure Transmission Algorithm with Nontriangular Iterative Structure Looking After Self-synchronization.

### III. Specification of the flat LPV-based SSSC Stanislas

Stanislas operates over the 16 elements of the finite field \(\mathbb{F}_{16}\) defined as: \(\mathbb{F}_{16} = \mathbb{F}_2[X]/(X^4 + X + 1)\), the addition being the componentwise exclusive or, simply denoted \(+\), and the multiplication, denoted by \(\cdot\), being the polynomial multiplication modulo the primitive polynomial \(X^4 + X + 1\).

#### A. Equations of Stanislas

The internal state \(x_t \in \mathbb{F}_{16}^{40}\) of the cipher consists in a vector of dimension \(n = 40\) with 4-bit components considered as elements in \(\mathbb{F}_{16}\). The input \(m_t\) and the output \(c_t\) of the ciphering function are 4-bit respective elements in \(\mathbb{F}_{16}\). See Fig. 1 for a graphical representation of Stanislas.

1) Ciphering equations: The ciphering equation defines the next internal state \(x_{t+1} \in \mathbb{F}_{16}^{40}\) and the cipher output \(c_t \in \mathbb{F}_{16}\). Note also that all elements that compose the matrix \(A_{\rho}(t)\) and the vectors \(B\) and \(C\) are elements of \(\mathbb{F}_{16}\). The ciphering equation obeys, for \(t \geq 0\),

\[
\text{cipher: } \begin{cases} 
  x_{t+1} = A_{\rho}(t)x_t + Bm_t \\
  c_{t+1} = Cx_{t+1} 
\end{cases} \tag{24}
\]

where \(B\) is the column vector equal to \(B = (1_{\mathbb{F}_{16}}, 0_{\mathbb{F}_{16}}, \ldots, 0_{\mathbb{F}_{16}})^T\) and \(C\) is the row vector equal to \(C = (0_{\mathbb{F}_{16}}, 0_{\mathbb{F}_{16}}, 1_{\mathbb{F}_{16}}, 0_{\mathbb{F}_{16}}, \ldots, 0_{\mathbb{F}_{16}})\) with the \(1_{\mathbb{F}_{16}}\) component located at column \(r = 3\). In other words, the only non-zero component of \(C\) which equals \(1_{\mathbb{F}_{16}}\) is the third component. Hence, the ciphertext symbol consists in the third component of the internal state. Let us note that in general, the ciphertext obeys Equation (13). Hence, the ciphertext results from a linear combination of the state vector components. Here, we propose a construction for which the linear combination reduces to the selection of one component. To guarantee a self-synchronization property (otherwise stated, to ensure Equation (20)), the component that is selected must coincide with \(r\), the delay of the system. This is why here, \(r = 3\).

The matrix \(A_{\rho}(t)\) is a \(40 \times 40\) dimensional matrix (see Fig. 2 of Appendix ). The entries \(a_{ij}\) of \(A_{\rho}(t)\) are either 0 of \(\mathbb{F}_{16}\), or 1 of \(\mathbb{F}_{16}\) or a nonlinear function of \(c_t\). Among them, \(n_a = 115\) entries are non-zero coefficients and \(L = 80\) entries correspond to nonlinear functions \(\varphi^i\) \((i = 1, \ldots, 80)\). Each function \(\varphi^i\) depends on the current ciphertext symbol \(c_t\) \((s = 0\) in Equation (21)) and on a subkey \(SK_i\) of \(K\). This subkey \(SK_i\) thus defines the corresponding function \(\varphi^i\) which depends on only one ciphertext symbol \(c_t\) as shown on Fig. 1.

\[
\varphi^i : \mathbb{F}_{16} \rightarrow \mathbb{F}_{16} \quad c_t \rightarrow S(c_t \oplus SK_i) \tag{25}
\]

where the subkey \(SK_i\) is a 4-bit word derived from the secret key \(K\) as described in the Key Schedule of subsection III-A3 detailed further on. The function \(S\) is the bijective S-Box extracted from Piccolo [21]. It is defined on 4-bit words
by Table I. The entries $a_{ij}$ according to their location (row $i$, column $j$ for $i, j \in \{1, \ldots, 40\}$) are given in Appendix A in a symbolic manner. In the symbolic representation of $A_{p(t)}$, denoted $A_S$, the functions $\varphi^i$ ($i = 1, \ldots, 80$) are assigned to the entries of $S$. Thus, the first line of the Fig. 1 corresponds to the first row of the matrix $A_S$ applied to the internal state $x_t$ to produce the corresponding coefficient of the internal state $x_{t+1}$ (combined with $m_t$ to produce $c_{t+1}$).

**TABLE I**

<table>
<thead>
<tr>
<th>S-box in hexadecimal notation.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S(x)$</td>
<td>2</td>
<td>4</td>
<td>B</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>A</td>
<td>1</td>
<td>F</td>
<td>6</td>
<td>C</td>
<td>5</td>
<td>D</td>
</tr>
</tbody>
</table>

In other words, taking into account the particular structure of the matrices $B$ and $C$ with only one non-zero coefficient, the ciphering process is governed by:

\[
\begin{align*}
x^1_{t+1} &= \sum_{j=1}^{n} a_{1j}x^j_t + m_t \\
x^i_{t+1} &= \sum_{j=1}^{n} a_{ij}x^j_t, \quad i = 2, \ldots, n \\
c_{t+1} &= Cx_{t+1} = x^3_{t+1} \quad (26)
\end{align*}
\]

2) Deciphering equations: The deciphering is governed by two equations defined, for $t \geq 0$, by

\[
\begin{align*}
\hat{x}_{t+1} &= P_{p(t+\rho)}[\hat{x}_t + BT^{-1}c_{t+r}] \\
\hat{m}_{t+r} &= T^{-1}(c_{t+r} - C \prod_{\ell=t-r+1}^{r-1} A_{p(\ell)}[\hat{x}_\ell]) \\
\end{align*}
\]

where the $40 \times 40$-dimensional matrix $P_{p(t+\rho)}$ over $\mathbb{F}_{16}$ verifies $P_{p(t+\rho)} = T^{-1}(c_{t+r} - C \prod_{\ell=t-r+1}^{r-1} A_{p(\ell)}[\hat{x}_\ell])$ and $T = C \prod_{\ell=t-r+1}^{r-1} A_{p(\ell)}$. The second equation ensures the recovery of the plaintext symbol $m_t$. The self-synchronization between the internal states $x_t$ and $\hat{x}_t$ of the cipher and the decipher automatons is thus a proper decryption are guaranteed, by construction, after a finite transient time.

Let $X[i]$ denote the $i$-th row of a matrix $X$. Taking into account the particular structure of the matrices $A$, $B$ and $C$, the delay $r = 3$, and noticing that $T$ equals one, the deciphering process is governed by:

\[
\begin{align*}
\hat{x}^1_{t+1} &= (P_{p(t+3)}[1] \cdot \hat{x}_t) + c_{t+3} \\
\hat{x}^i_{t+1} &= (P_{p(t+3)}[i] \cdot \hat{x}_t), \quad i = 2, \ldots, 40 \quad (28)
\end{align*}
\]

where

\[
\begin{align*}
P_{p(t+3)}[1] &= A_{p(t)}[1] - (A_{p(t+2)} A_{p(t+1)} A_{p(t)})[3] \\
P_{p(t+3)}[i] &= A_{p(t)}[i], \quad 2 \leq i \leq n. \quad (29)
\end{align*}
\]

3) Key Schedule: The matrix $A$ consists of $n_a = 115$ non-zero entries, and among them, $L = 80$ are functions $\varphi^i$ depending on the subkey $SK_i$. Thus, the Ciphering process aims at generating $80$ subkeys of 4-bit length: $SK_1, \ldots, SK_{80}$ from the 80-bit master key $K$ arranged as 20 words $K_1, \ldots, K_{20}$ of 4-bit length. To do so, the ciphering equations (26) with $m_t = 0$ are used. During this process, the parameters $SK_i$ involved in the functions $\varphi^i$ (see Equation (25)) are set to zero. The internal state $x_0$ is initialized by duplicating the master key $K$ as $(x^0_0, \ldots, x^{20}_0) = (K_1, \ldots, K_{20})$ and $(x^1_0, \ldots, x^{40}_0) = (K_1, \ldots, K_{20})$. Then, the initial internal state $x_0$ is updated ten times by using equations (26) with $m_t = 0$ for $t = 0, \ldots, 9$.

After those ten iterations, the ten subkeys $SK_1, \ldots, SK_{10}$ are respectively initialized with the following components of the internal state $x_{10}: x^1_{10}, x^2_{10}, x^3_{10}, x^4_{10}, x^5_{10}, x^6_{10}, x^7_{10}, x^8_{10}, x^9_{10}, x^{10}_{10}$. This process is repeated 7 more times to initialize the other subkeys $SK_{10+1}, \ldots, SK_{10+10}$, for $i = 0, \ldots, 7$.

**B. Ciphering Process**

The plaintext consists of $\ell$ elements of $\mathbb{F}_{16}$: $m = m_0 \ldots m_{\ell-1}$. The initial state $x_0$ is first initialized with 40 random elements of $\mathbb{F}_{16}$. These initial values are kept secret and are not transmitted to the decipher even if it is possible to recover the secret key. The way those elements are randomly picked is out of the scope of this paper. Only consider that you have a source of randomness. $x_0$ could be considered as a secret nonce.

Then randomly pick $n = 39$ elements $m_{-39}, \ldots, m_{-1}$ of $\mathbb{F}_{16}$ as the synchronization sequence which is placed before the plaintext. It is recalled that at most $n = 40$ iterations are needed for the self-synchronization to be achieved.

Then, because of the parameter value $r = 3$ that induces a delay, randomly pick $r - 1 = 3 - 1 = 2$ more elements $m_t, m_{t+1}$ of $\mathbb{F}_{16}$ that will be placed at the end of the plaintext sequence $m$ to process the two last plaintext symbols. Finally, the sequence that must feed the cipher is $m^* = m_{-39}, \ldots, m_0, \ldots, m_t, m_{t+1}$.

The resulting ciphertext consists of the sequence $c_{-39}, \ldots, c_{t+1}$ of $(\ell + 41)$ symbols in $\mathbb{F}_{16}$, computed with the ciphering Equations (26) using Matrix $A_S$ given in Appendix A for $t$ in $-39, \ldots, \ell + 1$:

\[
\begin{align*}
x^1_{t+1} &= S(x^0_t \oplus SK_0) \oplus S(x^{20}_t \oplus SK_1) \\
&\quad \oplus S(x^{21}_t \oplus SK_2) \oplus S(x^{40}_t \oplus SK_3) \oplus m_t \\
x^2_{t+1} &= x^1_t \oplus S(x^2_t \oplus SK_4) \oplus S(x^{30}_t \oplus SK_5) \\
&\quad \oplus S(x^{31}_t \oplus SK_6) \\
x^3_{t+1} &= x^2_t \oplus S(x^3_t \oplus SK_7) \\
x^4_{t+1} &= S(x^4_t \oplus SK_8) \oplus S(x^5_t \oplus SK_9) \\
&\quad \vdots \\
c_{t+1} &= x^3_{t+1}
\end{align*}
\]

Those equations are the ones given in Fig. 1 corresponding with the coefficients of the Matrix $A_S$.

Note that the Matrix $A_S$ given in Appendix A could be seen as one round of a block cipher applied on a state with 40 many 4-bit words where after each round the 4-bit word $x^3$ is outputted whereas the 4-bit word $x^0$ is completely updated by other 4-bit words. At each clock, each 4-bit word crosses at least one S-box, except the 4-bit word $x^1$ that could be considered as a temporary variable (as it receives the 4-bit message $m_t$). Note also that the updated rule for most of the variables $x^i$ ($5 \leq i \leq 40$) could be written as $x^i_{t+1} = S(x^{i-1}_t \oplus SK_k) \oplus f(x^{i-2}_t)$ or
The plaintext sequence is recovered as the last 41 symbols.

**A. Design Rationale**

Take into account practical considerations, regarding in internal state, the delay \( r \) and where \( f \) is the identity or the S-box \( S \). It could be seen as a generalization of the so-called \( L \)-scheme with a circular permutation used in the block cipher MISTY1 with balanced inputs/outputs. The complete ciphering process of Stanislas is illustrated on Fig. 1.

**C. Deciphering Process**

The decipherer receives the cryptogram consisting of \( \ell + 41 \) symbols \( c_{39}, \ldots, c_{\ell+1} \) in \( \mathbb{F}_16 \). The internal state \( \hat{x}_0 \) is initialized to an arbitrary value, for example the zero value.

Then, the deciphering Equations (28)-(29) are applied to recover a \( \ell + 41 \)-length message \( \hat{m} = \hat{m}_{-41}, \ldots, \hat{m}_{-1} \). The plaintext sequence is recovered as the last \( \ell \) symbols \( m = m_0, \ldots, m_{\ell-1} \).

**Remark 2:** It could be surprising that a part of the ciphering process directly depends on a secret nonce (i.e. \( x_0 \)). Instead, we could imagine that an 80-bit \( IV \) is used to generate the first value \( x_0 \) adding an \( IV \) schedule to the Key Schedule process. First, generate the subkeys using the Key Schedule, then initialize the internal state with the concatenation of the \( IV \) and of the key. Then, apply again the Key Schedule process (but including the generated subkeys \( SK_i \) in the S-boxes) to initialize the internal state \( x_0 \). But, note that in this case, the particular property that the internal states (that must be kept secret) of the ciphering part and of the deciphering part are not required to be equal is lost. Indeed, in this case where an \( IV \) is used, we will suppose that the internal state will be computed in the same way in both sides.

The matrix \( A \) consists of \( n_a = 115 \) non-zero entries, and among them, \( L = 80 \) are functions \( \phi^t \) depending on the subkey \( SK_i \). Thus, the Key Schedule process aims at generating 80 subkeys of 4-bit length: \( SK_1, \ldots, SK_{80} \) from the 80-bit master key \( K \) arranged as 20 words \( K_1, \ldots, K_{20} \) of 4-bit length. To do so, the ciphering equations (26) with \( m_t = 0 \) are used. During this process, the parameters \( SK_i \) involved in the functions \( \phi^t \) (see Equation (25)) are set to zero. The internal state \( x_0 \) is initialized by duplicating the master key \( K \) as \( (x_0^1, \ldots, x_0^{20}) = (K_1, \ldots, K_{20}) \) and \( (x_0^1, \ldots, x_0^{40}) = (K_1, \ldots, K_{20}) \). Then, the initial internal state \( x_0 \) is updated ten times by using equations (26) with \( m_t = 0 \) for \( t = 0, \ldots, 9 \).

**IV. DESIGN RATIONALE AND SECURITY ANALYSIS**

In this section, we motivate the choices of the field on which the cryptosystem operates, the dimension \( n \) of the internal state, the delay \( r \) and the structure of the matrix \( A \). Most of the choices rest on security criteria, others ones take into account practical considerations, regarding in particular the hardware implementation issues.

**A. Design Rationale**

**a) Field on which the cryptosystem operates: Galois field \( GF(16) \):** Any quantities \( m_t, \epsilon_t \), components of \( x_t \) and \( \hat{x}_t \) and non-linear functions \( \phi^t \) (S-boxes) inputs are 4-bit data. It is motivated by the fact that the cryptosystem is intended to be implemented on a digital equipment. Hence, field extensions and so, power of two are required. On the other hand, 8-bit would be too heavy for an embedded algorithm. In particular, S-boxes would involve too many logic gates.

**b) Dimension \( n: \) 40:** As the internal state components are 4-bit words, a dimension \( n = 40 \) provides an internal state of 160 bits and thus a security level of 80 bits. Indeed, to prevent time-memory trade-off attack \([22] \) (an attack which is a trade-off between exhaustive search and table look-up), the internal state must be two times longer than the key length which defines the security level. This level is compatible with a real-world application.

**c) Delay \( r: \) 3:** The more the delay, the more the algebraic degree of the entries of \( P_{p(t:t+r)} \), recalling that \( P_{p(t:t+r)} \) involves the product of \( r \) matrices (see (29)). Thus, for a good resistance against an algebraic attack \((23)\), the algebraic degree should be as large as possible. On the other hand, the more the delay, the more the complexity of implementation and the less the computational performances. The delay \( r = 3 \) results from a trade-off between security with respect to algebraic attacks (to increase the overall algebraic degree) and ease of implementation (especially the implementation of the \( P \) deciphering matrix which involves several S-box multiplications (see Equation (17) in the general case (29) for Stanislas)).

**d) Structure of the matrices \( A \) and \( P \):** We recall that the matrix \( A \) (and thus \( P \) from the computation (29)) is derived from the construction of a digraph (see Appendix B) from which an adjacency matrix \( I_A \) is extracted. More precisely, the adjacency matrix \( I_A \) determines the entries of \( A_{p(t)} \) that are zero and the others that are possibly non-zero. The number \( n_a \) of edges in the digraph \( G(\Sigma_A) \) corresponds to the number of non-zero entries of \( I_A \). Hence, the number \( n_a \) also determines the number of non-zero entries of the state transition matrix \( P \). Beyond the number of non-zero entries, their location (row and column) must also be chosen. Finally, it must be decided whether a non-zero entry will be 1 or will correspond to an S-box. All those issues have been addressed by considering several criteria regarding the security, in particular the algebraic degree of the entries of \( A_{p(t)} \) and of \( P_{p(t)} \) denoted by \( A_S \) and \( P_S \) where the coefficients of \( A_S \) and \( P_S \) belong to \( \mathbb{Z}[S] \), \( S \) representing any non-linear function. The following considerations on \( A_S \) and \( P_S \) can be made.

**Diffusion Delay and Depth.** The diffusion delay and the depth are properties related to the consideration of the powers \( A_S^p \) of \( P_S^p \) as \( p \), a natural integer, increases. Indeed, the power of matrices results from the successive iterations of the ciphering and the deciphering process. Let \( p \) denotes the power of a matrix \( Z \in M_n(GF(16)) \). The diffusion delay, introduced in [14], is the smallest value, denoted by \( d_0 \), of \( p \) such that \( Z^p \) does not have any zero coefficient. In other words, it is the smallest value of \( p \) such that each element of the initial internal state \( x_0 \)
has influenced every element of $x_t$ for $t \geq d_0$. The depth, introduced in [24], is the smallest value, denoted by $d_1$, of $p$ such that any entry of $Z^p$ are polynomials of degree at least 1. We are looking for the smallest values of $d_0$ and $d_1$.

**Algebraic Degree.** Considering the matrix resulting from successive powers of $A_S$ and $P_S$, we are first interested in ensuring that at least one entry has the largest algebraic degree. To this end, we must add a cycle on the $r$-th vertex of the digraph $\mathcal{G}(\Sigma_A)$, which equivalently means that the entry of $A_S$ and $P_S$ located at row $r$ and column $r$ must be an S-box. Furthermore, the evolution of this quantity, after successive iterations, must meet an ideal shape: it must increase by one at each iteration, remain constant and equal to its maximum value as long as possible and finally must drop down to zero (let us recall that after 40 iterations, due to (20), the product reaches exactly zero).

**Full-Rank Matrix.** The fact that $A_S$ is a full-rank matrix is a necessary condition to ensure a full diffusion of the internal state and to maximize the dependency between the involved terms at time $t$ and the involved terms at time $t+1$. Moreover, this condition guarantees that the encryption process does not collapse. By construction (see Appendix B) to ensure flatness, every element of the subdiagonal of $I_A$ from the $r$-th one is 1. Hence, for $A_S$ to be full-rank, each column and each row must contain at least one non-zero element. And yet, by construction, only the $r - 1$ first elements of the last column can be non-zero. Hence, one of them must be non-zero. From the digraph point of view, it means that at least one of the $r - 1$ first vertices must be connected to the last vertex.

The symbolic matrix $A_S$ which has been finally selected is given in Appendix A. It has been obtained after 700000 random runs performed under the aforementioned constraints: best diffusion delay and depth, algebraic degree (especially increase by 1 at each iteration), full-rank matrix. Several matrices correspond to the best choices (we add a sum indicator without weighing) and we finally chose the one with the best implementations for $A_S$ and for $P_S$. The symbolic matrix $P_S$ can be directly obtained by considering Equations (29). The matrix $A_S$ involves $n_a = 120$ non-zero entries and its number of S-boxes is $L = 80$. The matrix extracted from $A_S$ and $P_S$ by removing the first $r$ rows and columns have a special feature. The lower subdiagonal is full of coefficients $S$ (corresponding to S-boxes and denoted $SB$ in the symbolic representation) and the subsubdiagonal just above is full of 1. Thus, each line has at least one S-box to ensure that the internal state $x_t$ is updated in a non-linear way: this corresponds to a non-linear shift register.

The corresponding diffusion delay, $d_0 = 7$, could be considered as a good diffusion delay with regard to the synchronization delay that equals 40 (the system dimension). Indeed, the worst diffusion delay is equal to 40 and the best diffusion delay of 1 could only be reached with a matrix full of non-zero coefficients. Thus, we consider that a diffusion delay of 7 is a sufficiently good compromise between the best diffusion delay and a reasonable number of non-zero coefficients.

Let us notice that the dimension $n$ of the system is the upper bound of the diffusion delay. Indeed, due to (20), the product reaches exactly zero in $GF(16)$ since the synchronization is achieved after at most 40 iterations. The depth is $d_1 = 7$ and is the best that we manage to achieve.

e) **S-box:** Various classifications of 4-bit S-boxes exist in the literature [25], [26], [27]. For the sake of hardware
optimization, the same S-box has been used to define the \( L = 80 \) nonlinear functions \( \varphi^i \). Two kinds of criteria have been considered: theoretical and practical. On one hand, we have selected S-boxes to satisfy the maximum differential probability and the maximum (absolute) linear bias \( 2^{-2} \), the algebraic degree 3, and no fixed-point. Four S-boxes that satisfy those criteria and that have simple algebraic expressions (i.e. a minimal number of non-linear transformations) have been selected, each one corresponding respectively to the four classes proposed in [27].

From an implementation point of view, it turns out that the S-box depicted in Table I induces the smallest gate count. It is the Piccolo S-box ([21]). The area is around 23 Gate Equivalents (GEs).\(^1\) It involves four NOR gates, three XOR gates and one XNOR gate. Let us notice that the masking method can be applied using only three shares, making this S-box suitable for efficient threshold (i.e. side-channel protected) implementations.

f) Key Schedule: The Key Schedule has been chosen to reuse existing circuit while sufficiently mixing together the key words. To do so, we use the already implemented matrix \( A \) by applying it a sufficient number of times when looking at the diffusion degree and at the induced algebraic degree. The extracted 4-bit words of the internal state \( x_t \) at positions 1, 2, 7, 10, 17, 18, 22, 26, 28, 34 - have been chosen to be among the ones that depend of the maximum number of other elements of \( x_t \) to ensure to maximize the diffusion effect of the initial state \( x_0 \).

From a theoretical point of view, if we consider that each S-box behaves as a random function (i.e. it has a behavior sufficiently near the one of a true random function) and using the direct extension of Lemma 9 and Theorem 7 of [28], we could say, that after \( d_0 + 2 \) of the matrix \( A \) on the input \( x_0 \), the applied transformation behaves as a random function. In other words, the operations done to fulfill the subkey words behaves as a random function.

B. Security Analysis

The following section focuses on the security of Stanislas against known attacks. We claim a 80-bit security level which corresponds to the key length (we could not have a security level greater than the key length). Moreover, this security level is also achieved regarding the length of the main register: 160 bits. Indeed, the Guess-and-Determine attacks described in [22] imply to double the length of the used register compared to the key length to achieve a security level corresponding to the key length. Thus, we try to derive security bounds for all known attacks against Stanislas and we found no attacks that work with a complexity smaller than \( 2^{80} \) operations which corresponds with our security claim.

Moreover, it has been proven in [29] that the canonical form of an SSSC is secure against Chosen Plaintext Attacks (IND-CPA secure) but not against Chosen Ciphertext Attacks (IND-CCA security). We do not claim any security result in this last model. Moreover, we suppose that the attacker has no access to the key and to the initial value of the internal state \( x_0 \). To prevent collision search attack, we limit the size of each plaintext to \( 2^{64} \) 4-bit words.

First, it seems very difficult to analyze the security of Stanislas in its true settings (i.e. the 160-bit internal state \( x_0 \) is a secret nonce). In those settings, we could only say that:

- the time-memory-data trade-off attacks described in [30], [31], [22] apply when the internal state is smaller than two times the key length. This is why we choose the length of the internal state to be twice the key length to prevent this kind of attacks.
- Guess-and-Determine Attacks [22] consist in guessing a part of the state to further determine the remaining part of the state. Thus, at time \( t \), suppose that we know \( x_t^1 \), \( x_t^j \) and of course, \( c_t \). Thus, we could suppose that we observe \( n \) consecutive outputs from \( c_t \) to \( c_{t+n} \). Thus, how much does it cost to recover the induced subkey \( S_{K_j} \)? First, from the equation \( x_{t+1}^1 = x_t^1 \oplus S(x_t^1 \oplus SK_7) \), we could directly compute \( SK_7 \). Thus, we could derive the successive value of \( x_t^2 \) from \( t \) to \( n \). Thus, from \( x_t^2 \) we derive non-linear equations where the unknowns are \( x_t^1 \), \( x_t^{31} \), \( x_t^{35} \), \( SK_5 \), \( SK_4 \) and \( SK_6 \) and the known terms are \( x_t^2 \) to \( x_{t+n}^2 \). Thus, as the Algebraic Normal Form (ANF) of the S-box has 2 equations with 4 terms, 1 equation with 9 terms and 1 equation with 7 terms, each new \( x_t^n \) will induce a system of \( 4 \times 3 \times (n + 1) + 3 \times 4 = 12n + 24 \) unknown binary variables with 4 equations with in total 24 non-linear terms. Thus, we could not solve the system without guessing a part of it whatever the \( n \) value. Even considering that we guess a part of the system, the combinatorial explosion seems clear because each value of the internal state depend on at least one other value. Thus, we conjecture here that Stanislas is safe against this type of attacks since, even if a part of the internal state is guessed, each 4-bit word of this state is updated through a XOR with a 4-bit word of subkey and an S-box. Moreover, since the diffusion delay of \( A_S \) is \( d_0 = 7 \), this leads to guess after 7 outputs all the key. Thus, guessing a part of the state allows to guess a part of the key which, in then, amounts to a guess-and-determine attack with a complexity greater than the key exhaustive search.

Thus, instead, when looking at classical attacks, we will use the model described in Remark 2 where the initial state is derived in its first components after 20 iterations of the matrix \( A_S \) applied on the concatenation of an IV and of the master key \( K \) and in its last components after 14 more iterations. Those settings could be considered as a degrading mode of the original Stanislas Specifications. Thus considering that the attacker has full access to the IV, we obtain the following bounds against classical attacks.

Differential / Linear Cryptanalysis: we compute the lower bounds on the minimal number of active S-boxes for the computation of the internal state with Remark 2.
To do so, we implement the model of Remark 2 using Constraint Programming to explore all the possible paths in the induced graph. Then, we obtain that, for the differential case, after 7 iterations, a minimal number of 38 S-boxes has been crossed, after 10 iterations, 46, after 14 iterations, 65. As the differential probability of the chosen S-box is equal to $2^{-2}$, we could guarantee that the 80-bit key exhaustive search is less expensive than passing through more than 40 S-boxes, which is the case after 10 iterations of the $A_S$ matrix. In the same way, for the linear case, we obtain after 7 iterations, 35 active S-boxes, after 10 iterations, 41 and after 14 iterations, 59. Thus, for the same reason, after 10 iterations, a 80-bit key exhaustive search is more efficient.

**Algebraic Attacks:** This kind of attacks [23] is possible when the overall degree of the induced system of equations does not sufficiently increase at each clock. Especially, if the overall degree $d$ in each of the $n$ unknown variables (the key variables for example) of the system is such that $(n^d)^{2.5}$ is lower than the security bounds, it means that it is faster to solve the induced system by Gaussian elimination (considering that each new monomial is a new unknown variable) than trying all the keys of the system. Thus, we want to prevent this attack from happening as described below. The algebraic degree of each S-box component is the best one: equal to 3. Thus, each passing through an S-box increases the degree in the equations describing the internal state and in the equations describing the key. Even if the $SK_i$ linearly depends on the master key bits $K_1, \cdots, K_{80}$ (there are 80 key bits that are unknown), if we write the number of variables after crossing the first S-box, we have $(80)^{3} = 80000$ monomials depending on the unknown key bits, after the second pass we obtain $(80)^{6} = 20736$ monomials also depending on the unknown key bits and so on. Thus, if we apply those estimations using the bounds on the number of active S-boxes of the differential/linear case, after 10 iterations, we have 46 active S-boxes, which means that the number of unknowns (considering that each new monomial is a new unknown) is lower bounded after 10 iterations by $(80)^{3} \times 46 \approx 2^{57.21}$ where 80 are the unknowns coming from the master key, 3 is the algebraic degree of the S-box and 46 is the number of crossed S-boxes. Thus, we conjecture that the complexity of the best algebraic attack is greater than the 80-bit key exhaustive search.

**Cube Attacks:** As established in [32], a cipher is vulnerable to cube attacks if an output bit can be represented as a sufficiently low degree polynomial over $GF(2)$ of key and input bits. It works by summing an output bit value for all possible values of a subset of public input bits, chosen such that the resulting sum is a linear combination of secret bits. Repeated application of this technique gives a set of linear relations between secret bits that can be solved to discover these bits. In [33], the authors analyzed this kind of attacks on the block cipher Piccolo, especially its S-box. They stated that after 8 rounds, no relation with 63 input bits could be found. The minimal number of S-boxes crossed for 8 Piccolo rounds is 58 whereas in our case, it is 46 after 10 iterations. So we conjecture that we cross a sufficient number of S-boxes after few iterations to prevent having low degree relations between secret key bits and public input bits.

In summary, we conjecture that most of the usual attacks which apply in the stream cipher context have a complexity greater than the exhaustive key search for **Stanislas**.

V. Hardware Performance and Implementation Aspects

We give hereafter the implementation results of a straightforward implementation of **Stanislas**. It produces one 4-bit word of ciphertext per clock cycle. Subkeys are computed in the initialization step using the Key Schedule and stored in dedicated registers, before the cipher state processing. The same material is used for the cipher state and the Key Schedule processes. The hardware implementation of **Stanislas** is not an optimized version targeting any specific performance. The main area occupation comes from the matrix update as it carries big registers during all the calculations. Those registers are the internal state which are mixed with the subkeys either with binary addition or multiplication. This means updating a $40 \times 4$ bits register all along the matrix update. During ciphering process, the matrix update is solely made of S-boxes and XOR of 4-bit words. The lowest line of the matrix in terms of area occupation is made of 1 S-box and 2 additional XORS, the biggest of 4 S-boxes and 8 additional XORs. The deciphering process implies adding 23 multiplications, 28 S-boxes and 39 XORs to the previous total which makes deciphering heavier in terms of area occupation. The matrix is implemented line by line, calculated straightforwardly following the equations as depicted in Fig. 1. The S-boxes are implemented in a Look-Up-Table (LUT) way, so we let the compiler do its own optimizations. Our **Stanislas** implementation combines both the encryption and decryption process in order to ease comparisons with other (synchronous or self-synchronizing) stream ciphers.

We implemented **Stanislas** in VHDL and we provide in Table II FPGA hardware implementations and performance comparisons with synchronous SCs **Trivium** [17] and Grain [18], final members of the eSTREAM portfolio, another SSSC Moustique and the AES-based SSSC CFB1-AES128 as defined in NIST SP 800-38a [12]. The chosen FPGA platform for our benchmark is the Xilinx Spartan-6 XC6SLX75T, package FGG676. High effort of the Xilinx ISE Design Suite has been put on area reduction. Post-place-and-route results are provided.

To get **Trivium**, Grain, Moustique and CFB1-AES128 implementation results, we have designed our own VHDL straightforward reference implementations, without further optimization in mind. For this latter, we have implemented the potential S-boxes as LUTs, to be consistent with the S-boxes LUT implementations of **Trivium**.

At first sight, we can check that some well-known properties are visible in the results. For example, **Trivium** and Grain are very compact, which can be explained by their low combinatorial gate counts. Moreover, the number of initialization cycles needed for **Trivium**, Grain
and Moustique is consistent with the specifications: e.g., Trivium needs a warm-up phase of minimum 1152 steps. This number is really low for Stanislas where it just consists in a Key Schedule and it is equal to 0 for CFB1-AES128 where the key is processed on the fly.

Surprisingly, straightforward implementation of Stanislas provides the best throughput (TP) compared to the other stream ciphers, even self-synchronizing. The reason is that one 4-bit word is processed by clock cycle, so the throughput is given by: $95 \times 4 = 380$ Mbps. That justifies our design choice of processing 4-bit words instead of individual bits. Compared to the standard approach CFB1-AES128, the time needed for synchronization is also shorter, along with a smaller area.

This encouraging result has to be mitigated if we consider the combined metric TP/Area as shown on Table III. This latter allows to estimate the cost of optimized parallel implementations, where many bits can be processed in parallel, at the expense of additional area.

### VI. Conclusion

An instantiation, called Stanislas, of a dedicated Self-Synchronizing Stream Ciphers (SSSC) has been proposed. Its main peculiarity comes from the fact that it involves an automaton with finite input memory using non-triangular state transition functions. The construction is based on a general and systematic methodology that uses automata (called Linear Parameter Varying, LPV) admitting a matrix representation and a special property called flatness. The security analysis allows to conjecture that most of the usual attacks which apply in the stream cipher context have a complexity greater than the key exhaustive search for Stanislas. But, Stanislas could not be considered having a small hardware footprint.

However, when straightforward implementations are considered, Stanislas provides bigger throughput than the considered stream ciphers, and its intrinsic synchronization delay is much smaller than the SSSC Moustique (40 clock cycles instead of 105) and the standard approach CFB1-AES128 (40 clock cycles instead of 128).

Moreover, the number of surviving Self-Synchronizing Stream Ciphers after a phase of public crytanalytic time is equal to zero. So, we hope that Stanislas will be the first one and we encourage the symmetric key cryptographic community to cryptanalyze it.

### References


<table>
<thead>
<tr>
<th>TABLE II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XILINX SPARTAN-6 XC6SLX75T (FPGA) STRAIGHTFORWARD IMPLEMENTATION RESULTS.</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Area (slices)</th>
<th>Init. (cycles)</th>
<th>Synchro. (cycles)</th>
<th>Freq. (MHz)</th>
<th>TP (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivium</td>
<td>47</td>
<td>1056</td>
<td>0</td>
<td>184</td>
<td>394</td>
</tr>
<tr>
<td>Grain</td>
<td>48</td>
<td>256</td>
<td>0</td>
<td>355</td>
<td>355</td>
</tr>
<tr>
<td>Moustique</td>
<td>166</td>
<td>105</td>
<td>105</td>
<td>309</td>
<td>309</td>
</tr>
<tr>
<td>CFB1-AES128</td>
<td>265</td>
<td>0</td>
<td>192</td>
<td>77</td>
<td>349</td>
</tr>
<tr>
<td>Stanislas</td>
<td>701</td>
<td>66</td>
<td>40</td>
<td>95</td>
<td>880</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE III</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMBINED METRIC TP/AREA (MBPS/SLICE ON XILINX SPARTAN-6 XC6SLX75T FPGA).</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Area (slices)</th>
<th>TP/Area (Mbps/slice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivium</td>
<td>47</td>
<td>2.00</td>
</tr>
<tr>
<td>Grain</td>
<td>48</td>
<td>7.39</td>
</tr>
<tr>
<td>Moustique</td>
<td>166</td>
<td>2.80</td>
</tr>
<tr>
<td>CFB1-AES128</td>
<td>265</td>
<td>1.14</td>
</tr>
<tr>
<td>Stanislas</td>
<td>701</td>
<td>0.34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE IV</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>THEORETICAL IMPLEMENTATION RESULTS OF SOME (SS)SSCS ON XILINX SPARTAN-6 XC6SLX75T FPGA FOR 4-BIT VERSIONS</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Area (slices)</th>
<th>TP (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivium</td>
<td>198</td>
<td>764</td>
</tr>
<tr>
<td>Grain</td>
<td>192</td>
<td>1140</td>
</tr>
<tr>
<td>Moustique</td>
<td>664</td>
<td>1236</td>
</tr>
<tr>
<td>CFB4-AES128</td>
<td>2980</td>
<td>4306</td>
</tr>
<tr>
<td>Stanislas</td>
<td>1701</td>
<td>3800</td>
</tr>
</tbody>
</table>


Julien Francq received the Ph.D. degree in Computer Science domain in 2009 from the Université de Montpellier (France). He is now working in Naval Group company as a crypography, data science and hardware security expert in the Naval Cyber Laboratory. One of his main research interests is the security and the efficiency of (hardware/software) implementations of cryptography against (mathematical/physical) attacks.


Paul Huyhn has started his Ph.D. at Université de Lorraine in 2017. He works on lightweight cryptography.

Philippe Guillot received the Ph.D. degree in 1999, from the Université de Caen. He is an Assistant Professor at Université Paris 8 and he is interested in symmetric cryptography and their hardware implementations.

Gilles Millérioux received his Ph.D. from INSA Toulouse in 1997 and his French Habilitation in 2004 from Université Henri Poincaré - Nancy 1. Between 1998 and 2005 he has been Associate professor at Université Henri Poincaré - Nancy 1. Since 2005, he is Professor at Polytech Nancy and at the Centre de Recherche en Automatique de Nancy (CRAN). His main research interest concerns control theory and its application to cryptography.

Marine Minier received the Ph.D. degree in 2002, from the Université de Limoges and the French Habilitation from the Université de Lyon in 2012. In 2005, she joined the INSA de Lyon and the C²TI Laboratory, as an Assistant Professor. Since 2016, she is professor at Université de Lorraine and at the LORI Lab. Her research interests include Symmetric Key Cryptography and Security in WSNs.
APPENDIX A
The Matrix $A_S$
The matrix $A_S$ is given in Fig. 2.

APPENDIX B
Construction of the Matrices of the SSSC
A digraph $G(\Sigma_A)$ describing the structured linear system associated to the state equations (12), is the combination of a vertex set $V$ and an edge set $E$. The vertices represent the states and the input components of $\Sigma_A$ while the edges describe the dynamic relations between these variables. One has $V = X \cup \{m\}$ where $X$ is the set of state vertices defined as $X = \{x_1, \ldots, x^n\}$ and $m$ is the input vertex. The edge set is $E = E_A \cup E_B$, with $E_A = \{(x_i, x_j) | A(i, j) \neq 0\}$ and $E_B = \{(m, x_i) | B(i) \neq 0\}$. The entries of $A_{p(i)}$ correspond to the weights of the edges in the digraph. For convenience, we will denote by $v_i$, $(j = 0, \ldots, n)$ a vertex of the digraph $G(\Sigma_A)$ regardless of whether it is the input or a state vertex.

Given a triplet $(n, r, n_a)$ with $n$ the dimension of the state, $r$ the delay and $n_a$ the number of non-zero entries of the matrix $A$, the construction of the digraph $G(\Sigma_A)$ related to the system $\Sigma_A$ involves the following steps.

The system $\Sigma_A$ is of dimension $n$ and thus, the digraph $G(\Sigma_A)$ involves $n + 1$ vertices. The input is assigned to the vertex denoted by $v_0$. The other $n$ vertices are denoted by $v^1, \ldots, v^n$. Let $v^r$ be the vertex that corresponds to the flat output.

**Step 1:** For, $i = 0, \ldots, n - 1$, add the edges $(v^i, v^{i+1})$. There are $r$ edges which connect $v^i$ to $v^r$. Hence, the delay of the automaton is $r$.

After Step 1, this line topology corresponds to quite trivial dynamical systems since it corresponds to state transition functions in the form of simple shifts. Let us recall that we aim at designing an automaton possibly involving state transition functions more general than $T$–functions. A shift is a special and trivial $T$–function. To this end, the following steps provide a way of adding edges $(v^i, v^j)$ while guaranteeing flatness.

**Step 2:** Add the edges $(v^{i+1}, v^{i+1})$ for $i = 1, \ldots, n - r - 1$. Step 2 allows vertex $v^i$, $j = r + 1, \ldots, n$ to have a predecessor. Indeed, if not so, the dynamics of the corresponding vertex $v^i$ would reduce to $x_{k+1}^j = 0$ and would be clearly useless. The resulting path is a so-called main directed path and is depicted in Figure 3.

**Step 3:** Add the edges $(v^r, v^i)$, $i = 1, \ldots, n$ that connect the vertex $v^r$ to any other vertices of the graph (except the vertex $v^0$ related to the input).

**Step 4:** For every vertex $v^i$, $i = 1, \ldots, r - 1$, add the directed edge $(v^i, v^j)$ for $j = 1, \ldots, i$.

The graph obtained after Step 1-4 is depicted in Figure 4.

**Step 5:** For every vertex $v_i$, $i = r + 1, \ldots, n$, add the directed edge $(v_i, v_j)$ for $j = 1, \ldots, r$ and $j = i + 2, \ldots, n$.

The resulting digraph after completion of Step 1-5 is depicted in Figure 5.

To sum up, the digraph $G(\Sigma_A)$ is parametrized by the triplet $(n, r, n_a)$. The number of vertices of the digraph is equal to $n + 1$. Indeed, there are $n$ vertices assigned to the state components and one assigned to the input. The delay $r$ is the number of edges in the main directed path. The integer $n_a$ defines the desired number of edges in the digraph $G(\Sigma_A)$. It must satisfy $n_a \leq n_M$, where $n_M$ is the maximal number of edges resulting from the construction Step 1-5. A simple counting leads to:

$$n_M = \frac{n(n + 1)}{2} + r. \quad (30)$$

During the construction, at each step, we can decide whether we actually add the edges or not. That introduces flexibility in the perspective of providing distinct graphs and thus, distinct SSSC as detailed in Subsection II-C1.

Finally, the matrices $I_A$ and $I_B$ of the structural system $\Sigma_A$ can be extracted from the adjacency matrix, denoted by $I$, associated to the digraph $G(\Sigma_A)$. Indeed, the adjacency matrix $I$ associated to the digraph $G(\Sigma_A)$ is the $(n + 1) \times (n + 1)$ matrix

$$I = \begin{pmatrix} 0 & I_A & \cdots & I_B \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots \\ 0 & \cdots & \cdots & 0 \end{pmatrix} \quad (31)$$

where $I_A$ and $I_B$ stands respectively for the transpose of the structured matrices $I_A$ and $I_B$. The entries $I_{ij}$ are defined as follows for $1 \leq i, j \leq n$

$$I_{ij} = \begin{cases} 1 & \text{if there exists an edge from } v^i \text{ to } v^j \\ 0 & \text{otherwise}. \end{cases} \quad (32)$$

The adjacency matrix associated to $G(\Sigma_A)$, obtained after completion of Step 1-5, is given by

The open source software Sagemath [34] has been used to elaborate the digraph $G(\Sigma_A)$ corresponding to the triplet
Fig. 2. The Matrix $A_S$.  

$v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{r-1} \rightarrow v_r \rightarrow \cdots \rightarrow v_n$  

Fig. 3. Digraph obtained after completion of Step 1-2. The vertex $v^r$ corresponds to the flat output  

$v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{r-1} \rightarrow v_r \rightarrow \cdots \rightarrow v_n$  

Fig. 4. Graph obtained after Step 1-4.  

$(n = 40, r = 3, n_a = 120)$. The construction has been performed on an Intel CORE i7 CPU 2.26 GHz running Linux Ubuntu 14.04. All experiments ran single-threaded on the processors. It took 21 ms on the computer to obtain the digraph $\mathcal{G}(\Sigma_\Lambda)$.  

Fig. 5. Graph obtained after completion of Step 1-5.