Revisiting LFSMs

François Arnault, Thierry Berger, Marine Minier and Benjamin Pousse

Abstract—Linear Finite State Machines (LFSMs) are particular primitives widely used in information theory, coding theory and cryptography. Among those linear automata, a particular case of study is Linear Feedback Shift Registers (LFSRs) studied and implemented in many cryptographic applications such as design of stream ciphers or pseudo-random generation. LFSRs could be seen as particular LFSMs without inputs.

In this paper, we give first a general representation of LFSMs using traditional matrices representation linking this definition together with a new polynomial representation leading to sparse representations and implementations. As direct applications, we focus our work on the LFSRs case and show how the new LFSMs representation leads to a powerful design for LFSRs called Ring LFSRs efficient in both hardware and software.

We also study a particular LFSRs subcase called windmill LFSRs used for example in the E0 stream cipher and we generalize their representation leading to better hardware performances.

Index Terms—LFSMs, LFSRs, m-sequences.

I. INTRODUCTION

Linear Finite State Machines (LFSMs) are a building block of many information theory based applications such as synchronization codes, masking or scrambling codes. They are also used for white noise signals in communication systems, signal sets in CDMA (code division multiple access) communications, key stream generators in stream cipher cryptosystems, random number generators in many cryptographic primitive algorithms, and as testing vectors in hardware design.

A Linear Finite State Machine is a linear automaton composed of memories defined over a particular finite set \( \mathbb{F}_m \) (typically a finite field) and where the only operation updating cells is the addition \([1],[2],[3]\). At each clock, it inputs \( n \) elements of \( \mathbb{F}_m \) and outputs at least one element computed using its current state and additions. Two main classes of LFSMs could be defined: autonomous (without inputs in the updating process) and non-autonomous.

Autonomous and non-autonomous LFSMs are traditionally defined through a matrices representation. In this paper, we transpose the classical LFSRs representation defined using rational polynomials to the general case of autonomous and non-autonomous LFSMs and present the link between those two approaches.

As particular study cases, we put the stress on LFSRs. LFSRs are popular automata in many cryptographic applications and are particularly used as updating functions of stream ciphers and of pseudo-random generators. Their large popularity is due to their very simple design efficient both in hardware and in software and to the proved properties of the generated sequence (statistical properties, good periods,...) if the associated polynomial is primitive. In this last case, LFSRs generate \( m \)-sequences.

In this paper, we apply the matrices representation of LFSMs to the LFSRs case and give the equivalence between those two representations. This new model allows to give other characterizations of LFSRs beyond the traditional Galois and Fibonacci representations \([1] \). This leads to LFSRs called ring LFSRs that achieve better diffusion and better implementation both in hardware and in software. For the hardware case, we show theoretical bounds on the number of gates required to implement a ring LFSR compared with the equivalent Galois and Fibonacci LFSRs and we compare the associated traditional properties. For the software case, we compare the properties and the performances of our ring LFSRs with the LFSR used in the stream ciphers SNOW v2.0 \([4] \) and finalist of the NESSIE project \([5] \).

We finally give a generalization of some particular LFSRs called windmill LFSRs defined by Smeets and Chambers in \([6] \). Those LFSRs are based upon particular polynomials producing in parallel \( v \) subsequences of a given LFSR sequence. Four windmill generators are used as parallel updating functions in the stream cipher E0 \([7] \). The windmill constructions have been first extended in \([8] \). In this paper, we show other extensions of those particular constructions always using matrices representation.

This paper is organized as follows: Section II gives some backgrounds about Finite State Machines (FSMs) and introduces notations. Section III studies LFSMs using both the matrices representation and the new polynomial representation. Section IV puts the stress on the LFSRs case and shows how the new matrices representation leads to LFSRs with better properties and better implementations for hardware and software designs. Section V studies the particular windmill LFSRs and generalizes this last approach. Finally, Section VI concludes this paper.

II. BACKGROUNDS

In this paper we focus on how to build an automaton outputting a linear sequence. To reach this aim we must first make some general recall on sequences and on automata.

A. Linear sequences

The results in this paper are presented over \( \mathbb{F}_2 \) for convenience. They could be freely generalised to \( \mathbb{F}_q \) where \( q \) is a power of a prime number. We denote \( \mathbb{F}_2[X] \) the ring of polynomials over \( \mathbb{F}_2 \), i.e. \( \mathbb{F}_2[X] = \{ \sum_{i=0}^{n} a_i X^i, \ n \in \mathbb{N}, a_i \in \mathbb{F}_2 \} \).
We also denote $F_2[[X]]$ the ring of power series over $F_2$, i.e. $F_2[[X]] = \{ \sum_{i=0}^{\infty} a_i X^i, \ a_i \in F_2 \}$.

Let consider a sequence $s = (s_i)_{i \in \mathbb{N}}$ over $F_2$. This sequence is said linearly recurrent if there exists $q_1, \ldots, q_d \in F_2$ such that for all $n \geq d$, $s_n = q_1 s_{n-1} + \cdots + q_d s_{n-d}$.

The theory built upon the linearly recurrent sequences leads to a different point of view based upon power series. This relation is explained by the following theorem [1]:

**Theorem 2.1:** Let $s = (s_i)_{i \in \mathbb{N}}$ a sequence over $F_2$. The following statements are equivalent:
- $s$ is a linearly recurrent sequence.
- There exist polynomials $f(X), g(X) \in F_2[[X]]$ with $\deg f < \deg g$ and $g(0) = 1$ such that the power series $f(X)/g(X)$ is equal to $\sum_{i \in \mathbb{N}} s_i X^i$ in $F_2[[X]]$.

According to this theorem a correspondence can be built between power series and sequences. Given a power series $s(X) = \sum_{i \in \mathbb{N}} s_i X^i$ we denote $seq(s(X))$ the associated sequence $(s_i)_{i \in \mathbb{N}}$.

Remark that the statistical, random properties of a sequence of the form $f(X)/g(X)$ are characterised by $g(X)$, while $f(X)$ characterises the starting point of the sequence. More precisely, the following theorem stands [1]:

**Theorem 2.2:** Let $s(X) = f(X)/g(X)$ be a recurrent sequence, with $\gcd(f(x), g(x)) = 1$:
- The period of $seq(s(X))$ is equal to the order of $X$ in $F_2[[X]]/g(X)$.
- If $g(X)$ is primitive, i.e. $X$ is primitive in $F_2[[X]]/g(X)$, $seq(s(X))$ is called a $m$-sequence. Then $seq(s(X))$ is a punctured de Bruijn sequence.
- If $g(X)$ is primitive, there exists $N \in \mathbb{N}$ such that $\sum_{i \geq N} s_i X^{i-N} = 1/g(X)$.

**B. Automata**

We first give a general definition for an automaton.

**Definition 2.3:** A deterministic finite state machine (FSM) is defined by:
- $Q$ a finite set of states.
- $\Sigma$ an alphabet.
- $s_0$ in $Q$, called initial state.
- $F$ a subset of $Q$. Its elements are called final states.
- $\delta$ a transition function from $\Sigma \times Q$ to $Q$.

This definition is used in the automata theory. Given a deterministic FSM $A$ its behaviour could be totally described:

1. The automaton is initialised in state $s_0$ which is its current state at time $t \leftarrow 0$.
2. The automaton updates its state using $\delta$ and the symbol $i_t$ inputed at time $t$. More precisely $s_{t+1} \leftarrow \delta(i_t, s_t)$.
3. If $s_{t+1} \in F$, the automaton stops. Else it goes in 2.1 with $t \leftarrow t + 1$.

The steps 2.1 and 2.2 are called a clock. However this definition is often completed to agree with a more practical definition of automata. In particular it is usual for an automaton to output data at each clock. Then two models exist presented in [9] and [10]:

**Definition 2.4:** [9] A deterministic FSM in the Moore model is a deterministic FSM $A = (Q, \Sigma, s_0, F, \delta)$ and an additional function $\omega'$ from $Q$ to $\Sigma$. Then the step 2.1 describing the behaviour of the automaton is replaced by:

2.1. $s_{t+1} \leftarrow \delta(i_t, s_t)$ and the automaton outputs the symbol $a_t \leftarrow \omega'(s_t)$.

**Definition 2.5:** [10] A deterministic FSM in the Mealy model is a deterministic FSM $A = (Q, \Sigma, s_0, F, \delta)$ and an additional function $\omega''$ from $\Sigma \times Q$ to $\Sigma$. Then the step 2.1 describing the behaviour of the automaton is replaced by:

2.1. $s_{t+1} \leftarrow \delta(i_t, s_t)$ and the automaton outputs the symbol $a_t \leftarrow \omega''(i_t, s_t)$.

The difference between these two models is the fact that for Moore the outputed symbol at time $t$ does not depend of the inputed symbol at time $t$, while for Mealy, it does. Each model has its advantages. In practice Moore’s automata are synchron, and Mealy’s are not.

Now let introduce linear finite state machine:

**Definition 2.6:** A deterministic FSM $A = (Q, \Sigma, s_0, F, \delta)$ is called a linear finite state machine (LFSM) if $\delta$ is a linear function from $\Sigma \times Q$ to $Q$. In particular $Q$ and $\Sigma$ are vector spaces.

For a Moore (respectively Mealy) LFSM $\omega'$ (respectively $\omega''$) must also be linear.

**C. Notations**

We denote $w_H$ the Hamming weight of an element. For example the Hamming weight of a matrix $T = (t_{i,j})_{0 \leq i, j < n}$ is the cardinal of the set $\{ (i, j), \ t_{i,j} \neq 0 \}$. The Hamming weight of a polynomial is equal to the number of non-null coefficients.

Every time dependent element is denoted $(\cdot)_t$. For example consider a time dependent vector of bits $m^{(t)}$ in $F_2^n$. Its coordinate are denoted $m^{(t)}_i$ for $0 \leq i < n$.

In this paper we will classically construct from a time depending element the associated power series. Given a sequence $a = (a_i)_{i \in \mathbb{N}}$ in a ring $\mathcal{R}$ we construct $\sum_{i \geq 0} a_i X^{i-t_0}$, i.e. $a_{t_0} X^0 + a_{t_0+1} X^1 + \ldots$ in $\mathcal{R}[[X]]$. This power series is denoted $A^{(t_0)}$, i.e. we use the same letter than for the associated sequence capitalized, and we put in exponent the starting index used to build the series.

As an other example consider the vector $m^{(t)} = (m_0^{(t)}, \ldots, m_{n-1}^{(t)})$ in $F_2^n$. We denote $M_{t_0}^{(t)} = \sum_{i \geq t_0} m_i^{(t)} X^{i-t_0}$ in $F_2[[X]]$ and $M^{(t_0)} = \sum_{i \geq t_0} m_i^{(t_0)} X^{i-t_0}$ in $(F_2[[X]])^n$, i.e. $M^{(t_0)} = (M_0^{(t_0)}, \ldots, M_{n-1}^{(t_0)})$, i.e. the vector of the sequences produced by each cell from the time $t_0$.

We recall a linear algebra notation. Given a square matrix $M$, we denote $\text{Adj}(M)$ the adjugate matrix of $M$, i.e. the transpose of the cofactor matrix of $M$. We have the following relation $M \text{Adj}(M) = \text{Adj}(M) M = \det(M) I$. In particular, if $M$ is invertible, $M^{-1} = \frac{1}{\det(M)} \text{Adj}(M)$.

**III. LFSMs**

In this section, we will consider LFSMs inputting and outputting bits at each clock over $F_2$. We choose to use the Moore model to represent such automata. As we focus in this paper on LFSMs over $F_2$, LFSM means $F_2$-LFSM. Moreover
the automata in this paper formally do not end, i.e. the set of final states \( F = \emptyset \).

A. Definitions

According to Definition \ref{def:lfsm} a LFSM is an automaton composed of \( n \) binary memories. These memories are updated at each clock using linear (over \( \mathbb{F}_2 \)) function. At each clock the LFSM inputs \( k \) bits and outputs \( l \) bits. More precisely:

**Definition 3.1:** A Linear Finite State Machine (LFSM) \( \mathcal{L} \) with \( k \) inputs, \( l \) outputs and of length \( n \) over \( \mathbb{F}_2 \) is a triplet of matrices \( (A, B, C) \) over \( \mathbb{F}_2 \) of respective sizes \( n \times n, n \times k, k \times l \). Given the current state vector \( m(t) \in \mathcal{M}_{n,1}(\mathbb{F}_2) \), input vector \( u(t) \in \mathcal{M}_{k,1}(\mathbb{F}_2) \), the next state vector \( m(t+1) \in \mathcal{M}_{n,1}(\mathbb{F}_2) \) and the present output vector \( v(t) \in \mathcal{M}_{1,1}(\mathbb{F}_2) \) are expressed as:

\[
\begin{align*}
    m(t+1) &= Am(t) + Bu(t) \\
    v(t) &= Cm(t)
\end{align*}
\]

In the previous definition the matrix \( A \) is called the transition matrix of \( \mathcal{L} \).

B. Outputted sequences

We give in the following theorem a description of the sequences produced by a LFSM. This theorem is a generalization of the classical Theorem 4.3

**Theorem 3.2:** Let \( \mathcal{L} = (A, B, C) \) be a LFSM. The vectors \( M(t_0) \) and \( V(t_0) \) verify:

\[
\begin{align*}
    M(t_0) &= \frac{\text{Adj}(I - X \cdot A)}{\text{det}(I - X \cdot A)} \cdot (m(t_0) + X \cdot B \cdot U(t_0)) \\
    V(t_0) &= C \cdot \frac{\text{Adj}(I - X \cdot A)}{\text{det}(I - X \cdot A)} \cdot (m(t_0) + X \cdot B \cdot U(t_0))
\end{align*}
\]

**Proof:** With the previous notation we have the following relations:

\[
\begin{align*}
    M(t_0+1) &= A \cdot M(t_0) + B \cdot U(t_0) \\
    M(t_0) &= X \cdot M(t_0+1) + m(t_0)
\end{align*}
\]

Equation 1 is by definition \ref{eq:trans_matrix}  Equation 2 is by definition of \( M(t_0) \). It leads to the following relation:

\[
\text{det}(I - X \cdot A) \cdot M(t_0) = \text{Adj}(I - X \cdot A) \cdot (m(t_0) + X \cdot B \cdot U(t_0))
\]

Note that \( \text{det}(I - X \cdot A) \neq 0 \), this leads to \( M(t_0) = \frac{\text{Adj}(I - X \cdot A) \cdot m(t_0)}{\text{det}(I - X \cdot A)} + X \cdot B \cdot U(t_0) \) in \( \mathbb{F}_2[[X]] \).

We conclude the proof using the relation:

\[
V(t_0) = CM(t_0)
\]

**C. Similar LFSMs**

Note that these automata are defined by classical linear theory. In particular here we use similar matrices to define similar LFSMs \( \mathcal{L}' \).

**Definition 3.4:** Given two LFSM \( \mathcal{L} = (A, B, C) \) and \( \mathcal{L}' = (A', B', C') \), \( \mathcal{L} \) and \( \mathcal{L}' \) are said similar if there exists a nonsingular matrix \( P \) over \( \mathbb{F}_2 \) such that:

\[
A' = P^{-1} AP \quad B' = P^{-1} B \quad C' = CP
\]

**Theorem 3.5:** Let \( \mathcal{L} \) and \( \mathcal{L}' \) be two similar LFSMs. Assume that their initial state vectors satisfy \( m(0) = P^{-1} m(0) \) and their input vectors are the same. Then:

- \( \mathcal{L} \) and \( \mathcal{L}' \) have the same retroaction polynomial.
- \( M(0) = P^{-1} M(0) \). In particular \( m(t) = P^{-1} m(t) \) holds for each \( t \geq 0 \).
- The sequences outputted by \( \mathcal{L} \) and \( \mathcal{L}' \) are equal: \( v'(t) = V(0) \). In particular \( v(t) = v(t) \) holds for each \( t \geq 0 \).

**Proof:**

- The first point results from \( Q'(X) = \text{det}(I - X \cdot P^{-1} AP) = \text{det}(P^{-1} (I - X \cdot A) P) = \text{det}(I - X \cdot A) = Q(X) \).
- The proof is direct using Theorem 3.2 and the relation \( \text{Adj}(I - X \cdot P^{-1} AP) = \text{Adj}(P) \text{Adj}(I - X \cdot A) \text{Adj}(P^{-1}) = P^{-1} \text{Adj}(I - X \cdot A) P \):

\[
M(0) = \frac{\text{Adj}(I - X \cdot P^{-1} AP)}{\text{det}(I - X \cdot P^{-1} AP)} \cdot (P^{-1} m(0) + X \cdot P^{-1} B \cdot U(0)) = P^{-1} \frac{\text{Adj}(I - X \cdot A)}{\text{det}(I - X \cdot A)} \cdot P \cdot P^{-1} \cdot (m(0) + B \cdot U(0)) = P^{-1} M(0)
\]

- With the same arguments we have:

\[
V'(0) = C' \cdot M(0) = CP \cdot P^{-1} M(0) = V(0)
\]

**D. Polynomial representation**

We want to give a simpler description of a LFSM. Without loose of generality, we will consider in the two following subsections the case of LFSM with one output bit. The result presented here could be easily generalized to the case of a LFSM with more than one output but for simplicity reasons, we only present the proofs with one output bit.

The generic case of LFSM (i.e. with more than one output) could lead to better rational design.

Consider a LFSM \( \mathcal{L} = (A, B, C) \) with one output. Theorem 3.2 gives the relation:

\[
V(t_0)(X) = C \cdot \frac{\text{Adj}(I - X \cdot A)}{\text{det}(I - X \cdot A)} \cdot m(t_0) + \left( X \cdot C \cdot \frac{\text{Adj}(I - X \cdot A)}{\text{det}(I - X \cdot A)} \cdot B \right) \cdot U(t_0)
\]
First note that $C \cdot \text{Adj}(I - X \cdot A)/\det(I - X \cdot A)$ is a vector of size $n$ of the form $(P_0(X)/Q(X), \ldots, P_{n-1}(X)/Q(X))$ with $\deg P_0, \ldots, \deg P_{n-1} < n$ by definition of the adjugate matrix. So the term $C \cdot \frac{\text{Adj}(I - X \cdot A)}{\det(I - X \cdot A)} \cdot m^{(t)}$ is of the form $P^{(t)}(X)/Q(X)$ with $\deg P^{(t)} < n$.

By a similar reasoning the term $(X \cdot C \cdot \frac{\text{Adj}(I - X \cdot A)}{\det(I - X \cdot A)} \cdot B) \cdot U^{(t)}$ is of the form $(X \cdot R_0(X)/Q(X), \ldots, X \cdot R_{l-1}(X)/Q(X)) \cdot U^{(t)}$ with $\deg R_i < n$ for $i = 0, \ldots, l-1$.

Using those facts we propose a polynomial representation of a LFSM. The link between the inputs and the output of a LFSM is characterized by:

- The vector $(R_0(X)/Q(X), \ldots, R_{l-1}(X)/Q(X))$ which describes the linear transformations applied to the input bits.
- The rational serie $P^{(t)}(X)/Q(X)$ which is its initial state at time $t$.

A LFSM describes using this interpretation is called a Polynomial Linear Finite State Machine, denoted PLFSM.

**Definition 3.6:** A Polynomial Linear Finite State Machine (PLFSM) $L$ with $l$ inputs, one output and of size $n$ is a polynomial $Q(X)$ and a row vector $(R_0(X)/Q(X), \ldots, R_{l-1}(X)/Q(X))$ in $F_2[[X]]$ verifying $Q(0) \neq 0$ and $\deg R_0, \ldots, \deg R_{l-1} < n$, $\deg Q \leq n$.

The current state is a power series $P^{(t)}(X)/Q(X)$ verifying $\deg P^{(t)} < n$. The relation between the input vector $U^{(t)}$ and the outputted power series $V^{(t)}(X)$ is:

$$V^{(t)}(X) = \frac{P^{(t)}(X)}{Q(X)} + X \cdot \left(\frac{R_0(X)}{Q(X)}, \ldots, \frac{R_{l-1}(X)}{Q(X)}\right) \cdot U^{(t)}$$

With this definition we do not care about how the outputted sequences are computed by the automaton. In particular there are different possible implementations to achieve the same PLFSM. For example consider similar LFSMs according to Theorem 3.3.

Note that the size of the PLFSM represents the number of binary memories in the implementation.

Considering a PLFSM we also give the following relation:

$$V^{(t+1)}(X) = \frac{\hat{P}^{(t)}(X)}{Q(X)} + \left(\frac{R_0(X)}{Q(X)}, \ldots, \frac{R_{l-1}(X)}{Q(X)}\right) \cdot U^{(t)}$$

with $\deg \hat{P}^{(t)}(X) < n$.

**E. Example**

The reader could consider the example presented in Figure 1. This corresponds to a LFSMs with a single input, a single output and of length $d$. More precisely:

$$A = \begin{pmatrix} q_1 & 1 \\ q_2 & 1 & (0) \\ \vdots & (0) & \ddots \\ q_{d-1} & (0) & \cdots & 1 \\ q_d & 0 & 0 & \cdots & 0 \end{pmatrix}, B = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{d-1} \\ r_d \end{pmatrix}$$

and $C = (1, 0, \ldots, 0)$.

Considering an input $U^{(t)}(X) \in F_2[[X]]$ and using Theorem 3.2, this automaton outputs $R(X)/Q(X) \cdot U^{(t)}(X)$ with $R(X) = \sum_{i=1}^{d} r_i X^{i-1}$ and $Q(X) = \sum_{i=0}^{d} q_i X^i$ with $q_0 = 1$.

**IV. LFSRs**

In Section IV we present general results and representations concerning LFSMs. In this section, we specialize our work on LFSMs with no input and several outputs called Linear Feedback Shift Registers (LFSRs). These automata are widely used in cryptographic applications and thus have been well studied [11, 2, 6, 111, 12, 6]. We use the traditional matrix LFSM representation in the context of LFSRs to construct new efficient LFSRs both in hardware and in software respecting the traditional design criteria.

**A. Definitions**

A Linear Feedback Shift Register (LFSR) is a LFSM with no input and several outputs, i.e. an autonomous LFSM. These memories are updated at each clock using linear (over $F_2$) function. At each clock the LFSR outputs $l$ bits. More precisely:

**Definition 4.1:** A Linear Feedback Shift Register (LFSR) $L$ with $l$ outputs of length $n$ over $F_2$ is a couple of matrix $(A, C)$ over $F_2$ of size $n \times n$ and $l \times n$ respectively. Given the current state vector $m^{(t)} \in M_{n,1}(F_2)$, the next state vector $m^{(t+1)} \in M_{n,1}(F_2)$ and the present output vector $v^{(t)} \in M_{l,1}(F_2)$ are expressed as:

$$m^{(t+1)} = Am^{(t)}$$

$$v^{(t)} = Cm^{(t)}$$

As for the LFSM case, the matrix $A$ is called the transition matrix of $L$.

**Remark 4.2:** According to the previous definition it is clear that LFSRs are particular LFSMs. More precisely the LFSR $(A, C)$ could be identified to the LFSM $(A, 0, C)$.
B. Outputted sequences

We can characterize the outputted sequences as for the LFSMs case using the well-known theorem given in [1]. More precisely, we classically know that linear relations stand between the sequences outputted by a LFSR. However those linear relations are not often explicitly written. We give here a complete and simple expression using power series.

**Theorem 4.3**: Let \( L = (A, C) \) a LFSR. The vectors \( M(t_0) \) and \( V(t_0) \) verify:

\[
\begin{align*}
M(t_0) &= \frac{\text{Adj}(I - X \cdot A) \cdot m(t_0)}{\text{det}(I - X \cdot A)} \\
V(t_0) &= C \cdot \frac{\text{Adj}(I - X \cdot A) \cdot m(t_0)}{\text{det}(I - X \cdot A)}
\end{align*}
\]

**Proof**: The proof is direct using Theorem 3.2.

**Definition 4.4**: Let \( L = (A, C) \) a LFSR. The polynomial \( \text{det}(I - X \cdot A) \) is called the feedback polynomial of \( L \). It is denoted \( Q_L(x) \) or \( Q(x) \).

Remark that the degree of \( Q(x) \) is less or equal to \( n \), with equality if \( \text{det} A \neq 0 \).

C. Similar LFSRs

As previously done in the LFSMs case, we can define similar LFSRs:

**Definition 4.5**: Given two LFSRs \( L = (A, C) \) and \( L' = (A', C') \). \( L \) and \( L' \) are said similar if there exists a non-singular matrix \( P \) over \( \mathbb{F}_2 \) such that:

\[
A' = P^{-1} AP \quad \text{and} \quad C' = CP
\]

**Theorem 4.3** gives the following corollary for similar LFSRs:

**Corollary 4.6**: Let \( L \) and \( L' \) be two similar LFSRs. Assume that their initial state vectors satisfy \( m(0) = \text{det}(I - X \cdot A) \). Then:

- \( L \) and \( L' \) have same retroaction polynomial.
- \( M(t_0) = P^{-1} M(0) \). In particular \( m(t) = P^{-1} m(t) \) holds for each \( t \geq 0 \).
- The sequences outputted by \( L \) and \( L' \) are equal: \( V(t) = V'(t) \). In particular \( v(t) = v'(t) \) holds for each \( t \geq 0 \).

**Proof**: The proof is identical to the LFSM case.

D. Design criteria

We aim to build automata outputting linear recurring sequences. We give two criteria, one upon the sequences and one upon the automata, to choose good automata.

As introduced in Section III \( m \)-sequences are particular linear recurring sequences with good properties \( \mathbb{F}_2 \). For example we give some properties for \( m \)-sequences of degree \( n \) over \( \mathbb{F}_2 \):

- A \( m \)-sequence has maximal period: a \( m \)-sequence verifying a linear relation of degree \( n \) has a period of \( 2^n - 1 \).
- A \( m \)-sequence has a punctured de Bruijn sequence.

- A \( m \)-sequence has a period of \( 2^n - 1 \).
- A \( m \)-sequence has the run property: a run is a sub-sequence of 1 or 0. Half of the run are of length 1, a quarter of length 2, an eighth of length 3, etc. Until the run of length \( n \).

- A \( m \)-sequence is a punctured de Bruijn sequence.
as shown below. Moreover, Galois and Fibonacci LFSRs have a diffusion of \( n - 1 \) when of length \( n \), while with a Ring LFSR lower diffusion could be reached.

1) Properties: Using classical linear algebra we prove that a given feedback polynomial \( Q(X) \) the associated Galois and Fibonacci LFSRs are similar.

**Theorem 4.9:** The matrix \( T_G \) and \( T_F \) presented in Figure 2 are similar.

**Proof:** First we need some classical results:

**Definition 4.10:** The cotranspose of a matrix \( A = (a_{i,j})_{1 \leq i,j \leq n} \) is the matrix \( ^sA \) obtained by symmetry with respect to the second diagonal: \( ^sA = (a_{n-j+1,n-i+1})_{1 \leq i,j \leq n} \).

The following lemma stands:

**Lemma 4.11:** Given a matrix \( A \) of size \( n \times n \). The cotranspose of \( A \) and the transpose of \( A \) are similar.

**Proof:** Consider the matrix \( J \) of size \( n \times n \):

\[
J = \begin{pmatrix}
0 & \cdots & 1 \\
\vdots & \ddots & \vdots \\
1 & \cdots & 0
\end{pmatrix}
\]

The reader could verify the following relation:

\[
^sA = J \, A \, ^tJ
\]

Moreover note that \( J^{-1} = J \), this concludes the proof.

Note also the classical property:

**Lemma 4.12:** A matrix and its transpose are similar.

Finally note that \( T_G = T_F^s \). Using Lemma 4.11 and 4.12 concludes the proof.

The following theorem gives a strong link between generic Ring LFSR and Galois and Fibonacci LFSRs:

**Theorem 4.13:** If \( L \) is a LFSR with irreducible feedback polynomial \( Q(X) \), then \( L \) is similar to a Fibonacci LFSR and to a Galois LFSR.

**Proof:** As \( Q(X) \) is the minimal polynomial of \( A \), it follows from a well known linear algebra result that the matrix \( A \) is similar to the row companion matrix of \( Q(X) \). This shows the first assertion. The second follows from Theorem 4.9.

According to the results presented on similar LFSRs and to this theorem, a LFSR can be implemented on the one hand as a Fibonacci LFSR or a Galois LFSR as usual or on the other hand using our more general Ring representation. In this last case, we are able to build sparse matrices with arranging form to reach better hardware and software implementations whereas the underlying polynomial and thus the produced \( m \)-sequences are always the same.

2) Examples: We give a complete example of these automata. Consider the primitive feedback polynomial \( Q(X) = X^8 + X^6 + X^5 + X^3 + 1 \). Denote \( L_0 \) the associated Galois LFSR, \( L_1 \) the associated Fibonacci LFSR and \( L_2 \) a generic Ring LFSR with feedback polynomials \( Q(X) \). We present the respective transition matrices \( T_0 \), \( T_1 \) and \( T_2 \) of \( L_0 \), \( L_1 \), and \( L_2 \) in Figure 3.

We give in Figure 4 the implementation of \( L_0 \), \( L_1 \) and \( L_2 \) whereas we give in Table 1 the states of the automata \( L_0 \), \( L_1 \) and \( L_2 \) during 8 clocks with the same initial state.

The reader could notice that from the same initial state the outputted sequences (consider the content of the cell \( m_0 \)) are different. However they are all part of the \( m \)-sequences defined by \( Q(X) = X^8 + X^6 + X^5 + X + 1 \) according to Theorem 4.3, i.e. there exists three different polynomials \( P_0(X), P_1(X), P_2(X) \) of degree less than 8 such that the outputted sequences of \( L_0 \), \( L_1 \) and \( L_2 \) are \( P_0(X)/Q(X), P_1(X)/Q(X) \) and \( P_2(X)/Q(X) \).

All the three LFSRs presented here agree with our design criteria:

- The feedback polynomial \( Q(X) = X^8 + X^6 + X^5 + X + 1 \) is primitive.
- The diffusion of \( L_0 \) and \( L_1 \) are 7. The diffusion of \( L_2 \) is 6.
Critical path: The number of logic gate must be as small as possible to lower consumption.

We focus on these parameters because lowering these values allows to increase the frequency of the automata, consequently it allows to increase the throughput.

Previous works have been done to lower those parameters. For example in [14] the authors proposed top-bottom LFSR: a Ring LFSR divided in two parts: a Fibonacci part and a Galois part. More precisely, the transition matrix is of the form:

$$A = \begin{pmatrix} g_1 & 1 & 0 \\ & \ddots & \ddots \\ & 1 & 0 \\ 1 & \cdots & \cdots & \cdots & 1 \\ & \cdots & \cdots & \cdots & f_i & f_{i+1} & \ldots & f_n \\ \end{pmatrix}$$

In [15] the authors proposed a method which from a LFSR constructs a similar LFSR with a lower cost.

Our approach is based on the construction of good transition matrix for a LFSR. First we give expressions of the previous constraints relying to the transition matrix of a Ring LFSR in Table II. Consider a Ring LFSR \( \mathcal{L} \) of length \( n \) with transition matrix \( A \). We compute its feedback polynomial \( Q(X) \) and consider the associated Galois LFSR \( \mathcal{L}_G \) and Fibonacci LFSR \( \mathcal{L}_F \). We note the column of \( A \) \( Col_0, \ldots, Col_{n-1} \) and its rows \( Row_0, \ldots, Row_{n-1} \). We note \( w := w_H(Q(X)) \).

Galois LFSR are optimal for the critical path, while Fibonacci LFSR are optimal for the fan-out. A Ring LFSR can be built to reach these two values. More precisely a Ring LFSR with at most a Hamming weight of 2 for its columns and its rows will have optimal critical path and fan-out with a good diffusion as summed up in Table II.

We also give in Table II the associated values of the LFSRs presented in Figure 5. Note that the Ring LFSR \( \mathcal{L}_2 \) is better than the Galois and the Fibonacci for all criteria.

However we do not have an algorithm that construct a LFSR with a given feedback polynomial, we just can pick random transition matrix with good properties.

This algorithm picks random feedbacks positions and compute the associated feedback polynomial. This algorithm is probabilistic. We expect picking a random matrix of size \( n \) and computing its feedback polynomial is equivalent to pick a random polynomial of degree \( n \). More precisely we know

The diffusion for \( \mathcal{L}_2 \) is only one less than for Galois and Fibonacci. The difference here is low because of the small length of the considered LFSRs.

**F. Efficient hardware design**

We show in this subsection how to achieve good hardware design and we first introduce the constraints required to achieve such a design:

Critical path: The shorter longest path must be small to raise frequency.

Fan-out: A given signal should drive minimal gate number as exposed in [13].

<table>
<thead>
<tr>
<th></th>
<th>Galois</th>
<th>Fibonacci</th>
<th>Ring LFSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical path</td>
<td>( \lceil \log_2(w - 1) \rceil )</td>
<td>( \max \lceil \log_2(w_H(\text{Row}_i)) \rceil )</td>
<td>( \max \lceil \log_2(w_H(\text{Row}_i)) \rceil )</td>
</tr>
<tr>
<td>Fan-out</td>
<td>( w - 1 )</td>
<td>( w - 2 )</td>
<td>( w_H(\text{Col}_i) )</td>
</tr>
<tr>
<td>Cost</td>
<td>( w - 2 )</td>
<td>( w - 2 )</td>
<td>( w_H(\text{Col}_i) - n )</td>
</tr>
<tr>
<td>Diffusion</td>
<td>( n - 1 )</td>
<td>( n - 1 )</td>
<td>( \leq n - 1 )</td>
</tr>
</tbody>
</table>

### Table II

Critical path, fan-out, cost and diffusion of Galois, Fibonacci and generic Ring LFSRs.
Table III
CRITICAL PATH, FAN-OUT, COST AND DIFFUSION OF L₀, L₁ AND L₂

<table>
<thead>
<tr>
<th></th>
<th>L₀</th>
<th>L₁</th>
<th>L₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical path</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Fan-out</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Cost</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Diffusion</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

Require: $n$ the length of the Ring LFSR to seek. $f \leq n$ the number of feedbacks to place.

Ensure: A transition matrix $A$ with a critical path of 1, a fan-out of 2 and a cost of $f$ logic gates and such that its feedback polynomial is primitive of degree $n$.

repeat

$A \leftarrow (a_{i,j})_{0 \leq i,j < n}$ with $a_{i,j} = \begin{cases} 1 & \text{if } j \equiv i + 1 \mod n \\ 0 & \text{otherwise} \end{cases}$

while $w_H(A) < n + f$ do

$(i,j) \leftarrow \text{Random}([0, n] \times [0, n])$

if $w_H(\text{Row}_i) = 1$ AND $w_H(\text{Column}_j) = 1$ then

$a_{i,j} \leftarrow 1$

end if

end while

$Q(X) \leftarrow \det(I - X \cdot A)$

until $Q(X)$ is primitive

return $A$

Fig. 6. Algorithm to pick randomly a Ring LFSR with a good hardware design

that the feedback polynomial as its constant coefficient equal to 1, so the number of possibly constructed polynomials is $2^{n-2}$. The number of primitive polynomials of degree $n$ over $\mathbb{F}_2$ is $\frac{2^{n-1}}{n}$ where $\phi$ is the Euler’s function. We expect Algorithm 6 to be successful after $\frac{2^{n-2}}{\phi(2^{n-1})/n}$ tries as presented in Fig. 7.

The time complexity of this algorithm is driven by the time it takes to compute $\det(I - X \cdot A)$ which is roughly $O(n^3)$.

For a hardware oriented LFSR, each feedback is freely placed. Using this property we can lower the complexity of the previous algorithm. First recall the following property:

Proposition 4.14: Given a matrix $A$ over a ring $R$ of size $n \times n$. Note $E_{i,j}$ the matrix with a single 1 in position $i, j$. Then $\det(A + \lambda E_{i,j}) = \det(A) + \lambda \text{Cof}_{i,j}$ where $\text{Cof}_{i,j}$ denotes the cofactor of $i,j$ of the matrix $A$.

Note that the cofactor matrix of a matrix is equal to the transposition of its adjugate matrix, which could be computed with classical inversion algorithms.

Using this proposition, we propose a more efficient algorithm:

The complexity of this algorithm is driven by the computation of the cofactor matrix and its determinant (which could be in practice achieved by a single algorithm). But for one cofactor matrix computation, we test roughly $n^2 - n \cdot f$ polynomials. So the average complexity is about $O(n)$. 

Example: We give in Appendix A an example of a hardware oriented LFSR of length 128 found using this last algorithm. This LFSR has a primitive feedback polynomial which has an Hamming weight of 65. Note that the diffusion of this LFSR is only 27 whereas the corresponding diffusion for a Galois or a Fibonacci LFSR would be 127.

G. Efficient software design

The question is how to find a way to build a software oriented LFSR. The main difference between hardware and software is the atomic data size. In hardware we operate on single bit, but in software bits are natively packed in word such that working on single bit is not natural and need additional operations. The word size depends on the architecture of the
processor: 8 bits, 16 bits, 32 bits, 64 bits or more. To benefit of this architecture we propose to use LFSRs acting on words. Previous works have been done for the LFSR case to optimize software performances.

Firstly, the Generalized Feedback Shift Registers were introduced in [16] to increase the throughput. The main idea here was to parallelize \( w \) Fibonacci LFSRs. More formally, the corresponding matrix of such a construction is:

\[
A = \begin{pmatrix}
0 & I_w & I_w & (0) \\
0 & 0 & I_w & (0) \\
\vdots & \vdots & \ddots & \vdots \\
I_w & a_{n-2}I_w & \ldots & a_2I_w & a_1I_w & a_0I_w
\end{pmatrix}
\]

where \( I_w \) represents the \( w \times w \) identity matrix over \( \mathbb{F}_2 \) and where the \( a_i \) for \( i \) in \([0, \ldots, n-2]\) are binary coefficients. The matrix \( A \) could be seen at bit level but also at \( w \)-bits word level, each bit of the \( w \)-bits word is in fact one bit of the internal state of one Fibonacci LFSR among the \( w \) LFSRs.

In [2], Roggeman applied the previous definition to LFSRs to obtain the Generalized Linear Feedback Shift Registers but in this case the matrix \( T \) is always defined at bit level. In 1992, Matsumoto in [17] generalized this last approach considering no more LFSR at bit level but at vector bit level (called word). This representation is called Twisted Generalized Feedback Shift Register whereas the same kind of architecture was also described in [18] and called the Mersenne Twister. In those approaches, the considered LFSRs are in Fibonacci mode seen at word level with a unique linear feedback. The corresponding matrices are of the form:

\[
A = \begin{pmatrix}
0 & I_w & I_w & (0) \\
0 & 0 & I_w & (0) \\
\vdots & \vdots & \ddots & \vdots \\
I_w & 0 & 0 & L & 0 & 0
\end{pmatrix}
\]

where \( I_w \) represents the \( w \times w \) identity matrix and where \( L \) is a \( w \times w \) binary matrix. In this case, the matrix is defined over \( \mathbb{F}_2 \) but could also be seen at \( w \)-bits word level. This is the first generalization of LFSR specially designed for software applications due to the word oriented structure.

The last generalization was introduced in 1995 in [19] with the Multiple-Recursive Matrix Method and used in the Xorshift Generators described in [20] and well studied in [12]. In this case, the used LFSRs are in Fibonacci mode with several linear feedbacks. The matrix representation is:

\[
A = \begin{pmatrix}
0 & I_w & I_w & (0) \\
0 & 0 & I_w & (0) \\
\vdots & \vdots & \ddots & \vdots \\
A_1 & A_2 & A_3 & \ldots & A_{r-2} & A_{r-1} & A_r
\end{pmatrix}
\]

where \( I_w \) is the identity matrix and where the matrices \( A_i \) are software efficient transformations such as right or left shifts at word level or word rotation. The main advantage of this representation is its word-oriented software efficiency but it also preserves all the good LFSRs properties if the underlying polynomial is primitive.

An other way to construct software oriented LFSRs is to consider LFSRs over \( \mathbb{F}_{2^w} \) as done in [4], [21]. The SNOW LFSR is given in Appendix [B]. This interpretation allows to use table-lookup optimization and gives good results. Remark that those automata could be interpreted as linear automata over \( \mathbb{F}_2 \) because of the mapping \( \mathbb{F}_{2^w} \to (\mathbb{F}_2)^w \). In particular they can be consider as special case of our proposal.

As for the hardware case our approach focuses on the construction of a software oriented transition matrix. To do so, we will use transition matrices defined by block. In the next algorithm, \( A \) will define a block matrix, i.e. \( A \) is taken in \( \mathcal{M}_{n/k}(\mathcal{M}_k(\mathbb{F}_2)) \) for a matrix of size \( n \) divided in blocks of size \( k \) over \( \mathbb{F}_2 \). When a LFSR is being defined by block, we call it a word-LFSR.

Moreover we will use the right and left shift operation (denoted \( \gg \) and \( \ll \)) which is fast and implemented at word level. Given a word size \( k \) we define the matrix \( L \) of left shift as the matrix \( k \times k \) with ones on its sub-diagonal and zeros elsewhere. Similarly the matrix \( R \) of right shift is defined as the matrix \( k \times k \) with ones on its sub-diagonal and zeros elsewhere, such that we have:

\[
L \cdot (x_0, x_1, \ldots, x_{k-1})^t = (x_1, \ldots, x_{k-1}, 0)^t
\]

\[
R \cdot (x_0, x_1, \ldots, x_{k-1})^t = (0, x_0, x_1, \ldots, x_{k-2})^t
\]

Remark that LFSRs over \( \mathbb{F}_{2^w} \) can be expressed as word-LFSRs where used operations are multiplications on \( \mathbb{F}_{2^w} \) seen as a space vector over \( \mathbb{F}_2 \), i.e. there exists a bijection between \( \mathbb{F}_{2^w} \) and \((\mathbb{F}_2)^w\).

According to the previous discussion we propose the following algorithm to build software efficient LFSRs:

**Require:** \( k \) the word size, \( n \) the length of the LFSR to seek with \( k/n, f \leq n/k \) the number of word-feedbacks to place.

**Ensure:** A transition matrix \( A \) define by block with a cost of \( f \) shift and xor operations and such that its feedback polynomial is primitive of degree \( n \).

**repeat**

\[
A \leftarrow (a_{i,j})_{0 \leq i,j < n/k}
\]

with \( a_{i,j} = \begin{cases} I_k & \text{if } j \equiv i + 1 \mod n/k \\ 0 & \text{otherwise} \end{cases} \)

**From** \( \leftarrow \text{Random}(0, n/k]^f \)

**To** \( \leftarrow \text{Random}(0, n/k]^f \)

**Shift** \( \leftarrow \text{Random} \left(\lceil -k/2, k/2 \rceil \setminus \{0\}\right)^f \)

**for** \( l \leftarrow 0 \) to \( f - 1 \) do

\[
a_{T0[l],From[l]} \leftarrow a_{T0[l],From[l]} + \begin{cases} L_{Shift[l]}^t & \text{if } Shift[l] > 0 \\ R_{-Shift[l]} & \text{otherwise} \end{cases}
\]

**end for**

\( Q(X) \leftarrow \det(I - X \cdot A) \)

**until** \( Q(X) \) is primitive

**return** \( A \)

Fig. 9. Algorithm to pick randomly a LFSR with a good software design.
This algorithm picks random word-feedbacks positions and shift values, and compute the associated feedback polynomial. The complexity of this algorithm is about the same than Algorithm 4 because we have not been able to use the block structure of the matrix to lower the determinant complexity.

1) Example: We give in Figure 10 an example of a LFSR with an efficient software design with $n = 40$ and $k = 8$. The corresponding hardware implementation of this LFSR is also very good due to the intrinsic structure (a fan out of 2, a critical path of 1 and a cost of 19 adders). The diffusion of this LFSR is 27.

Let compare a word oriented LFSR to the SNOW2.0 LFSR defined in [4]. Those two LFSRs are described in Appendix B and C.

These two LFSRs outputted $m$-sequences of degree 512. We compare the diffusion and the throughput in software for those two LFSRs:

- The diffusion of the SNOW LFSR is 49 compared to 33 for our LFSR.
- The cost of one clock is 8 cycles for the SNOW LFSR using the sliding window implementation as proposed in [4]. The cost for this LFSR implemented using classical implementation is 20 cycles. The cost for our LFSR is 33 cycles.

As presented the diffusion is better for our LFSR. However the cost of one clock is higher in our case. This relies to the fact that the SNOW LFSR is sparse (three feedbacks) while ours has 8 feedbacks. Moreover, the computations are made using precomputed tables which leads to a better cost. However, the hardware implementation of our own LFSR has a really low cost (it fills the hardware design criteria we require in the previous section: critical-path of 1, fan-out of 2) whereas the SNOW2.0 LFSR could not be efficiently implemented in hardware due to the precomputed tables.

Note also that sliding window technique is only available for Fibonacci LFSR allowing better software implementation.

V. REVISITING WINDMILL LFSRS

In this section we propose a new representation of LFSRs using Definition 3.6. In Section III we built LFSMs and give a polynomial representation of such automata. The idea is to combine different LFSMs to construct a LFSR. We first give an example by considering windmill LFSRs which were introduced by Smeets and Chambers [6] and studied in [8]. Then we give a formal definition for such automaton and the underlying constraints.

A. Windmill LFSRs

Windmill LFSRs could be defined as LFSRs with no input and several outputs. They are introduced in [6] as a cyclic cascade connection of $v$, $v \geq 1$, LFSRs. Each LFSR is called a vane of the windmill. The classical representation of those LFSRs is the Fibonacci one. However in the rest of this section, we will show them using the equivalent Galois representation because it is more suitable for a better understanding. The windmill LFSRs are characterised by their feedback and feedforward connections. These feedback and feedforward connections are identical for all vanes, but as the length of the LFSRs may be different they could be shifted in different LFSRs. Figure 11 presents a generic vane in Galois mode.

Windmill LFSRs were introduced to achieve parallel generation of sequences. Consider a sequence $S = (s_n)_{n \in \mathbb{N}}$. While a classical automaton outputs $s_0$ at the first clock, $s_1$ at the second, and so on, a parallel automaton outputs $v$ bits at each clock: $(s_0, s_1, \ldots, s_{v-1})$ at the first clock, $(s_v, \ldots, s_{2v-1})$ at the second, \ldots. More precisely a parallel automaton has $v$ outputs and products the sequences $S^v := (s_{v+i})_{n \in \mathbb{N}}$ where $0 \leq i < v$. Note that our study focus on characterising the sequences $S^v$ and not the reconstructed sequence $S$.

Consider the windmill presented in Figure 11 which is used in the stream cipher E0 [7]. It is constituted of one vane of length 7 and three identical vanes of length 6. No feedback connection appears. Feedforward connections appears, for example from cell $m_{13}$ to cells $m_{12}$, $m_{10}$, $m_9$ and $m_7$.

Until now, only windmill LFSRs with a single vane repeated several times have been studied. We generalize this definition allowing different vanes in a windmill. We also give a new description of this windmill which will be more compact. More precisely we want to consider outputted sequences of cells $m_0$, $m_7$, $m_{10}$ and $m_{19}$, and characterise each vane by a polynomial. This leads to the interpretation presented in Figure 12.

Driven by this representation we introduce a new definition for LFSRs.
B. Formal definition

Definition 5.1: A Linear Feedback Shift Register (LFSR) $L$ with length $n$ over $\mathbb{F}_2$ is a matrix $A$ over $\mathbb{F}_2[[X]]$ of size $n \times n$. Given the current state vectors $m(t) \in M_{n,1}(\mathbb{F}_2)$ and $m'(t) \in M_{n,1}(\mathbb{F}_2[[X]])$, the next state vectors are expressed as:

$$
\begin{align*}
  m(t+1) &= Am(t) + m'(t) \pmod{X} \\
  m'(t+1) &= Am'(t) + m(t) \div X
\end{align*}
$$

where $P(X) \div X = \frac{P(X)}{Q(X)} \mod X$.

With this definition the LFSR described in Figure 12 as the following transition matrix:

$$
\begin{pmatrix}
  0 & X & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  1 & 0 & 0 & 0
\end{pmatrix}
$$

We give in Table IV the values of $m(t)$ and $m'(t)$ during 8 clocks.

According to Definition 5.1, windmills as introduced by Smeets and Chambers agree with the following definition:

Definition 5.2: A windmill LFSR with polynomials $\alpha(X), \beta(X)$ with $\beta(0) \neq 0$ and $v$ vanes is a LFSR of length $v$ with matrix $A$ over $\mathbb{F}_2[[X]]$ of the form:

$$
\begin{pmatrix}
  0 & \alpha(X) \beta(X) \cdot X^{i_0} \\
  \vdots & \ddots & \ddots \\
  \alpha(X) \beta(X) \cdot X^{i_{v-2}} & \alpha(X) \beta(X) \cdot X^{i_{v-1}} & 0 & \ldots & 0
\end{pmatrix} \quad (0)
$$

where $0 \leq i_0, \ldots, i_{v-1}$.

With this representation each row represents a vane of the windmill. In particular as described in the following section the length of the vane $j$ is equal to $\max(\deg(\alpha(X) \cdot X^{i_j}), \deg(\beta(X)))$.

As previously we are able to describe the outputted sequences:

Theorem 5.3: Let $L = A$ a LFSR. The vectors $M(t)$ verifies:

$$
M(t) = (I - X \cdot A)^{-1} \cdot (m(t) + Xm(t))
$$

Proof: With the previous notion we have the following relations:

$$
\begin{align*}
  M^{(t+1)} &= A \cdot M^{(t)} + m^{(t)} \\
  M^{(t)} &= X \cdot M^{(t+1)} + m^{(t)}
\end{align*}
$$

Equation 5 is by Definition 5.1. Equation 6 is by definition of $M^{(t)}$. It leads to the following relation:

$$(I - X \cdot A) \cdot M^{(t)} = m^{(t)} + Xm^{(t)}$$

Note that $(I - X \cdot A)$ is invertible in $M_{n}(\mathbb{F}_2[[X]])$ this leads to $M^{(t)} = (I - X \cdot A)^{-1} \cdot (m^{(t)} + Xm^{(t)})$ in $\mathbb{F}_2[[X]]$.

C. Underlying constraints

Definition 5.1 allows to describe numerous LFSRs impracticable because it allows to consider $A$ and $m'$ over $\mathbb{F}_2[[X]]$. But an element in $\mathbb{F}_2[[X]]$ can require infinite memories to be represented by an automaton. To avoid these cases we need to focus on a subset of $\mathbb{F}_2[[X]]$. The natural set to consider is the set of rational power series, i.e. $Q = \{ P(X)/Q(X) \in \mathbb{F}_2[[X]] \mid P(X), Q(X) \in \mathbb{F}_2[X], Q(0) \neq 0 \}$. Definition 3.6 and Theorem 3.2 and 4.3 show that those elements can be produced by automata. More precisely $A$ and $m'$ must be in $Q$.

Second the vector $m^{(t)}$ is linked to the choice of $A$. Each row of $A$ could be interpreted as describing one LFSM with one output bit using Relation 3 and Relation 5. Consider $L$, the PLFSM described by the row $i$ of $A$, i.e. $L_i$ is characterized by the row vector $Row_i(A)$. With this point of view $m^{(t)}$ is the state of $L_i$. We have the following constraints:

- $n_i$ is the number of binary memories in $L_i$.
- The $i$-th row of $A$ must be of the form $(P_0/X/Q(X), \ldots, P_{n-1}/X/Q(X))$ with $\deg P_0, \ldots, \deg P_{n-1} < n_i$ and $\deg Q_i \leq n_i$.
- $m^{(t)}_i + X \cdot m^{(t)}_i$ is of the form $\tilde{P}^{(t)}_i(X)/Q_i(X)$ with $\deg P_i < n_i$.

D. First example

Consider the PLFSM $L^1$ defined by the following matrix:

$$
A = \begin{pmatrix}
  X^2 & X \\
  0 & 1
\end{pmatrix}
$$

We compute $(I - X \cdot A)^{-1}$ to characterize the outputted sequences:

$$(I - X \cdot A)^{-1} = \begin{pmatrix}
  X^2 + 1 & X^2 + X + 1 \\
  X^2 + X & X^2 + 1
\end{pmatrix}
$$

Figure 13 presents an implementation of this automaton built upon three LFSMs. One for each non-zero coefficient of $A$. Those LFSMs are built using a Galois vane architecture as presented in Figure 1.

Note that according to the notation in Figure 13 $L^1$ could be expressed as the LFSR $(A', C')$ with:

$$
A' = \begin{pmatrix}
  0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 \\
  1 & 0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1 & 0
\end{pmatrix},
C' = \begin{pmatrix}
  1 & 0 & 0 & 0 & 1 \\
  0 & 0 & 1 & 0 & 0
\end{pmatrix}
$$
In particular we have the following relations according to Theorem 4.3.

\[ V(t) = \frac{1}{X^4 + X^3 + 1} \cdot \left( \begin{array}{cccc}
1 & X & X^2 & X^3 + 1
\end{array} \right) \cdot m(t) \]

This implementation is not optimal because it requires seven memories cells where four are enough (it outputs sequences of the form \( P(X)/(X^4 + X^3 + 1) \) with \( \deg P(X) < 3 \)). In particular \( \det(I - X \cdot A') = X^6 + X^3 + X^2 + X + 1 \), i.e. this automaton could output \( m \)-sequences of the form \( P(X)/(X^6 + X^3 + X^2 + X + 1) \) using a different matrix \( C' \) because \( X^6 + X^3 + X^2 + X + 1 \) is primitive.

A better implementation is given considering one LFSM per line. To do so, note that \( \frac{X}{X^2 + X + 1} = \frac{X^2 + X}{X^2 + 1} \). This leads to the implementation presented in Figure 14.

As previously this leads to this relation:

\[ V(t) = \left( \begin{array}{cccc}
X^3 + X^2 + 1 & X & X^2 + X^3 + 1 & X^3 + X^2 + 1
\end{array} \right) \cdot m(t) \]

**E. Second example**

Consider the PLFSM \( L^2 \) defined by the following matrix:

\[ A = \left( \begin{array}{cccc}
X & X^2 + 1 & 0 & 0
X^3 + X^2 & X^2 & 0 & 0
0 & X & X^3 + 1 & 0
0 & 0 & X^3 + 1 & 1
\end{array} \right) \]

Figure 15 presents an implementation of this automaton built upon six LFSMs. One for each non-zero coefficient of \( A \). Those LFSMs are built using a Galois vane architecture as presented in Figure 1.
we have shown our ring LFSRs have better diffusion and better performances.

In further works, we aim at more precisely looking at the case of an LFSM with \( l \) output bits to give equivalent and general representations. We also want to generalize those new results to Finite State Machines (FSMs) that are no more linear. The same kind of generalization could be efficiently applied to Feedback with Carry Shift Registers (FCSRs) or to Algebraic Feedback Shift Registers (AFSRS).

**REFERENCES**


APPENDIX

A. Example of a Ring LFSR of size 128 bits

We describe a Ring LFSR of size 128 bits. The transition matrix $A = (a_{i,j})$ is given by:

$$
\begin{align*}
    a_{i,i+1} &= 1 \text{ for all } 0 \leq i < 127 \\
    a_{127,0} &= 1 \\
    a_{i,j} &= 1 \text{ for } (i,j) \in \mathcal{F}
\end{align*}
$$

where $\mathcal{F}$ is the set:

$$
\begin{pmatrix}
    (4, 78), & (5, 19), & (8, 44), & (9, 106), \\
    (10, 70), & (12, 14), & (14, 115), & (15, 55), \\
    (17, 82), & (21, 64), & (22, 12), & (25, 127), \\
    (27, 107), & (28, 112), & (31, 59), & (34, 111), \\
    (35, 48), & (37, 36), & (38, 23), & (39, 88), \\
    (43, 37), & (44, 26), & (46, 60), & (47, 100), \\
    (49, 24), & (50, 25), & (51, 2), & (51, 27), \\
    (55, 124), & (57, 113), & (59, 71), & (61, 29), \\
    (69, 123), & (72, 52), & (73, 118), & (77, 46), \\
    (80, 74), & (81, 83), & (83, 98), & (87, 53), \\
    (88, 73), & (91, 47), & (93, 10), & (94, 21), \\
    (95, 93), & (97, 13), & (98, 117), & (99, 50), \\
    (100, 3), & (101, 104), & (104, 1), & (105, 114), \\
    (106, 108), & (107, 105), & (109, 4), & (111, 28), \\
    (112, 68), & (113, 42), & (114, 31), & (119, 18), \\
    (120, 49), & (121, 32), & (123, 94), & (124, 6)
\end{pmatrix}
$$

This LFSR has a primitive feedback polynomial. It has a cost of 64 adders, a fan-out equal to 2 and a critical path of 1, and a diffusion of 27.

B. Description of the LFSR in SNOW 2.0 over $\mathbb{F}_2$

We give here a description of the LFSR used in SNOW 2.0 seen as a LFSR over $\mathbb{F}_2$.

First this LFSR is defined as a Fibonacci LFSR over $\mathbb{F}_{2^{32}}$. The field $\mathbb{F}_{2^{32}}$ is defined as an extension of $\mathbb{F}_2$ to allow an efficient implementation and to prevent guess-and-determine attack presented in $[22]$. The implementation is based upon the multiplication by $\alpha \in \mathbb{F}_{2^{32}}$ verifying $\alpha \cdot (c_3 \alpha^3 + c_2 \alpha^2 + c_1 \alpha + c_0) = (c_2 \alpha^3 + c_1 \alpha^2 + c_0 \alpha) + c_4 \cdot V$ with $V$ an element in $\mathbb{F}_{2^{32}}$. We denote $M_\alpha$ the matrix of this linear application seen over $\mathbb{F}_2^{32}$:

$$
M_\alpha = \begin{pmatrix}
    0 & 0 & 0 \\
    I_8 & 0 \\
    I_8 & I_8 \\
    (0) & I_8
\end{pmatrix}
\begin{pmatrix}
    V_0 \\
    V_1 \\
    \ldots \\
    V_7
\end{pmatrix}
$$

where

$$
\begin{align*}
    V_0 &= t'(0x1E19FCF13) \\
    V_1 &= t'(0x6B973726) \\
    V_2 &= t'(0xD6876E4C) \\
    V_3 &= t'(0x05A7DC98) \\
    V_4 &= t'(0x0AE71199) \\
    V_5 &= t'(0x1467229B) \\
    V_6 &= t'(0x28CE449F) \\
    V_7 &= t'(0x50358897)
\end{align*}
$$

Then the transition matrix of the LFSR of SNOW2.0 is presented in Figure [18].

C. Example of a word-oriented LFSR of size 512 bits

We give in Figure [18] a description of a word-oriented LFSR of length 512 with words of 32 bits. The grid in the matrix is drawn for readability.
Fig. 17. Transition matrix of SNOW2.0

Fig. 18. Transition matrix of a word oriented LFSR