mathematics

MDPI

Article

Security and Efficiency of Linear Feedback Shift Registers in $GF(2^n)$ Using n-Bit Grouped Operations

Javier Espinosa García [1], Guillermo Cotrina [2], Alberto Peinado [2,*] and Andrés Ortiz [2]

[1] Institute of Physical and Information Technologies (ITEFI), Spanish National Research Council (CSIC), 28006 Madrid, Spain; javier.espinosa@iec.csic.es
[2] Escuela Técnica Superior de Ingeniería de Telecomunicación, Universidad de Málaga, 29071 Málaga, Spain; gcotrinacuenca@uma.es (G.C.); aortiz@ic.uma.es (A.O.)
* Correspondence: apeinado@ic.uma.es

Abstract: Many stream ciphers employ linear feedback shift registers (LFSRs) to generate pseudorandom sequences. Many recent LFSRs are defined in $GF(2^n)$ to take advantage of the n-bit processors, instead of using the classic binary field. In this way, the bit generation rate increases at the expense of a higher complexity in computations. For this reason, only certain primitive polynomials in $GF(2^n)$ are used as feedback polynomials in real ciphers. In this article, we present an efficient implementation of the LFSRs defined in $GF(2^n)$. The efficiency is achieved by using equivalent binary LFSRs in combination with binary n-bit grouped operations, n being the processor word's length. This improvement affects the general considerations about the security of cryptographic systems that uses LFSR. The model also allows the development of a faster method to test the primitiveness of polynomials in $GF(2^n)$.

Keywords: LFSR; stream cipher; m-sequence; primitive polynomial; extended Galois field; symmetric encryption

MSC: 3304; 05C31; 90C23; 46B85; 6804

Citation: Espinosa García, J.; Cotrina, G.; Peinado, A.; Ortiz, A. Security and Efficiency of Linear Feedback Shift Registers in $GF(2^n)$ Using n-Bit Grouped Operations. *Mathematics* **2022**, *10*, 996. https://doi.org/10.3390/math10060996

Academic Editors: Víctor Gayoso Martínez and Antanas Čenys

Received: 24 January 2022
Accepted: 16 March 2022
Published: 19 March 2022

1. Introduction

The symmetric cryptographic systems known as stream ciphers base their operation on the generation of binary sequences that are combined with the message to be encrypted by binary addition (XOR operation). The sequence used in the transmitter must also be generated in the receiver to recover the message by applying the XOR operation to the received ciphertext [1]. The concept of perfect secrecy defined by Shannon [2] establishes as a condition that the binary sequences used to encrypt the message (ciphering sequences) are random, have a length greater than or equal to the message, and are one time use. Although excellent random generators exist, the need to reproduce the sequence in the receiver makes it necessary to use pseudo-random sequences instead of true random ones. Therefore, pseudorandom number generators (PRNGs) constitute the fundamental part of any stream cipher.

One of the simplest and most widely used methods to generate cryptographic pseudo-random sequences is the linear feedback shift register (LFSR) [3]. This generator stands out for its simplicity and for the good statistical properties of the generated sequences. In addition, the behavior of the LFSR is completely characterized by the polynomial that defines the applied feedback. Thus, if the polynomial is primitive, the generated sequence reaches the maximum length, which is known as the m-sequence. However, these sequences are easily predictable from $2L$ known elements of the sequence generated by an L-stage LFSR. This makes the sequences obtained from an LFSR not directly usable. Instead, non-linear filtering or non-linear combinations of various LFSRs have to be applied to ensure the cryptographic security of stream ciphers, such as those used in mobile and

wireless communication systems, e.g., Bluetooth [4], wireless area networks [5], or GSM [6]. On the other hand, although LFSRs are generically defined on a finite field $GF(q)$, practical implementations of these ciphers are carried out on $GF(2)$ to integrate with bitwise operations. Nevertheless, these operations are clearly inefficient when using current processors that work with 16-, 32-, or 64-bit words. For this reason, the extended Galois fields $GF(2^n)$, where 2^n matches the processor word length, have been analyzed to substitute $GF(2)$ in cryptographic applications [7–9]. In the particular case of LFSR-based stream ciphers, the SNOW 3G algorithm [10] is currently used in 3G and 4G mobile communication systems, and several proposals have recently appeared to be applied to 5G communications [11–13]. However, computations on an extended field are time consuming, much more than in the base field [14]. Although the bit generation rate improves n times the binary case, the overall performance of the system does not, sometimes being even lower. In fact, several methods have been developed to reduce computational time, such as precomputed tables, optimized algorithms for multiplication [15], or the use of representations of the $GF(2^n)$ elements in terms of $GF(2^m)$ elements with $m < n$. In contrast to binary case, where any primitive polynomial guarantees a sequence of maximum length, only certain primitive polynomials that facilitate its implementation are used in extended fields. This means that they do not require excessive resource consumption, as the SNOW 3G case does, defined to work on devices with 32-bit processors by combining operations with 32-bit and 8-bit arguments. On the other hand, the identification of primitive polynomials in $GF(2^n)$ requires a much higher computational effort than in the binary case.

This article presents two methods that reduce the execution time for the implementation of an LFSR and the search for primitive polynomials in the extended filed $GF(2^n)$. These methods are based on the model that establishes a direct relationship between the m-sequence generated by an LFSR in $GF(2^n)$ and the interleaving of n m-sequences generated by n LFSRs in $GF(2)$. This relationship was used by Komo and Lam [16] to build primitive polynomials in $GF(2^n)$ in terms of primitive polynomials in $GF(2)$, establishing the relationships that must hold between both polynomials. We propose to use these relationships in the opposite direction; that is, we propose to represent the primitive polynomial over $GF(2^n)$ in terms of binary LFSRs. In this way, the same sequences will be generated using only binary operations (XOR). However, moving from n-bit word operations to bit operations would be back to square one, since the main reason for using LFSR on extended fields is to take advantage of the capabilities of n-bit processors where bit operations are inefficient. Taking into account that the n binary LFSRs that allow generating the same sequence as the LFSR in $GF(2^n)$ have all the same primitive feedback polynomial in $GF(2)$, the previous obstacle can be easily overcame. Thus, the calculation of the bit operations of the n binary LFSRs can be performed jointly, giving rise to XOR operations between n-bit words and eliminating the inefficiency generated by single-bit operations in this type of processor. The efficiency improvement provided by the proposed implementation turns it into a method especially suitable for cryptanalysis tasks where any execution time reduction in the systems under analysis is very appreciated. Therefore, security assessments performed to cryptosystems based on LFSR in extended fields must take into account the proposed implementation to report a more realistic security level. Additionally, the proposed method allows any primitive polynomial in $GF(2^n)$ to be used as feedback polynomial of an LFSR, thus overcoming the current limitations.

The rest of the paper is organized as follows. In Section 2, the mathematical background and notation are introduced, with special emphasis on the LFSR fundamentals and the relationships between the m-sequences generated in extended and base fields, $GF(q^n)$ and $GF(q)$. Section 3 describes the proposed implementation for the particular case of LFSRs defined in $GF(2^n)$ through a different and more efficient way. Section 4 contains the algorithm proposed to test the primitiveness of polynomials making use of the same relationships. Finally, discussion about security and efficiency of the implementations and conclusion are included in Sections 5 and 6, respectively.

2. Mathematical Background

Let $GF(q)$ be the finite field of q elements, q being a prime, and $GF[q,x]$ the set of all polynomials with coefficients in $GF(q)$. Equivalently, let $GF(q^n)$ be the finite field of q^n elements, and $GF[q^n,x]$ the set of all polynomials with coefficients from $GF(q^n)$. A generator of the cyclic group of a finite field is called a primitive element of that field. Hence, a polynomial $p(x) \in GF[q,x]$ of degree $m \geq 1$ is called primitive over $GF(q)$ if it is the minimal polynomial over $GF(q)$ of a primitive element $\alpha \in GF(q^n)$. A primitive polynomial $h(x)$ of degree n allows one to construct $GF(q^n)$ in such a way that:

$$GF(q^n) \approx GF[q,x]/h(x) \tag{1}$$

The addition and multiplication in $GF(q^n)$ are the ones in $GF[q,x]$, but performing the module $h(x)$ reduction, as all the elements in $GF(q^n)$ can be represented as polynomials of degree less than n with coefficients in $GF(q)$. On the other hand, any element $a \in GF(q^n)$ can be expressed in terms of a basis $\{\alpha^{n-1}, \ldots, \alpha^2, \alpha, 1\}$, α being a root of $h(x)$ in $GF(q^n)$. Consequently, any element $a \in GF(q^n)$ can be written as the vector $(a_{n-1}, \ldots, a_1, a_0)$ where:

$$a = a_{n-1}\alpha^{n-1} + \cdots + a_1\alpha + a_0 1. \tag{2}$$

For $q = 2$, it is very common to use the hexadecimal notation as a compact representation of the elements in $GF(q^n)$. Thus, if we use $h(x) = x^8 + x^4 + x^3 + x^2 + 1$ to construct $GF(2^8)$, α being a root of $h(x)$, as any element in $GF(2^8)$ can be represented as a power of α, we can write the element α^3 as $(0,0,0,0,1,0,0,0)$ or $0x08$ and the element $\alpha^{10} = \alpha^6 + \alpha^5 + \alpha^4 + \alpha^2$ as $(0,1,1,1,0,1,0,0)$ or $0x74$. Note that the powers of α correspond to the vector components in descending order, beginning from the left, to facilitate the conversion to and from hexadecimal values.

2.1. Linear Feedback Shift Registers

An LFSR defined over $GF(q)$ is a collection of L memory cells b_i, $0 \leq i \leq L-1$, whose contents belong to that field and are updated synchronously by a master clock, by the following equations:

$$\begin{aligned} &\gamma = b_{L-1}c_1 + b_{L-2}c_2 + \cdots + b_1c_{L-1} + b_0c_L, \\ &b_i = b_{i+1}, \quad 0 \leq i \leq L-2, \\ &b_{L-1} = \gamma, \end{aligned} \tag{3}$$

giving rise to the sequence $D = d_0d_1d_2\cdots$, which is completely determined by the initial state of the cells, named *seed*, and the feedback coefficients $c_i \in GF(q)$, $1 \leq i \leq L$ according to the linear recurrence:

$$d_i = d_{i-1}c_1 + d_{i-2}c_2 + \cdots + d_1c_{L-1} + d_0c_L, \tag{4}$$

where $d_0, d_1, \cdots, d_{L-1}$ correspond to the seed (see Figure 1). The length of the sequence D can be analyzed in terms of the connection polynomial $p(x)$ composed with the feedback coefficients:

$$p(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{L-1}x^{L-1} + c_Lx^L, \tag{5}$$

in such a way that the maximal sequence length $q^L - 1$ is achieved when $p(x)$ is primitive. In such a case, the sequence is called m-sequence and is independent from the chosen seed.

Stream ciphers are mainly based on LFSRs defined over finite fields with $q = 2$ [1]. Hence, the cell content is one bit, and the addition and multiplication correspond to XOR and AND operations, respectively. However, for efficiency reasons, in the generation process, LFSRs defined in $GF(2^n)$ are also being used in current communication systems. When LFSR is defined in this extension field, the cells contain n-bit words, where n matches the processor's word length. Although the equations that govern the LFSR are the same,

as described above, addition and multiplication are defined as polynomial addition and multiplication modulus $h(x)$, the polynomial used to construct the $GF(2^n)$. From now on, we shall use the notation $< L, p(x) >$ to refer to an LFSR with L cells in $GF(2)$ and connection polynomial $p(x) \in GF[2, x]$ of degree L. The form $< L, p(x), n >$ is for an LFSR of L cells in $GF(2^n)$ and connection polynomial $p(x) \in GF[2^n, x]$ of degree L.

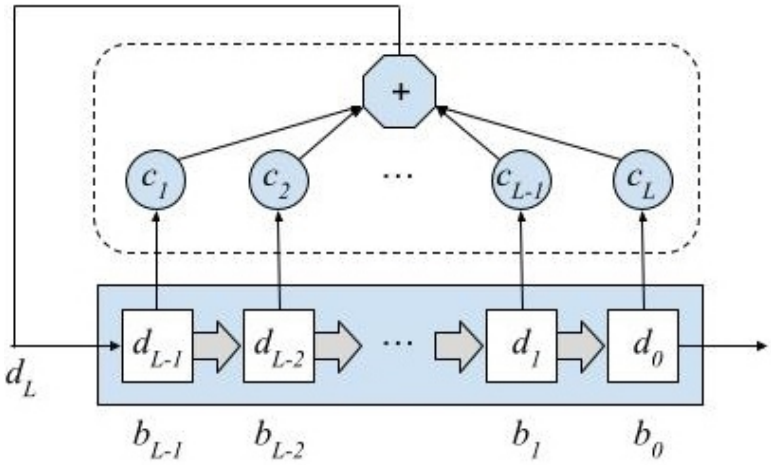

Figure 1. Linear feedback shift register.

2.2. Binary Equivalent Model

Komo and Lam [16] proposed a method to generate primitive polynomials in $GF(q^n)$ using primitive polynomials in $GF(q)$ based on the relationship previously discovered by Park and Komo [17] between the m-sequences produced in $GF(q^n)$ and $GF(q)$, in such a way that an m-sequence in $GF(q^m)$ can be decomposed into n m-sequences in $GF(q)$. More precisely:

Theorem 1 (*cf.* [17], th 7). *Let $p(x)$ be a primitive polynomial of degree $m \cdot n$ in $GF[q, x]$. Let $f(x)$ be one of the n primitive polynomials of degree m in $GF[q^n, x]$ into which $p(x)$ factors when viewed in $GF[q^n, x]$. Let $D = d_0, d_1, \cdots$ be an m-sequence over $GF(q^n)$ generated by $f(x)$. If*

$$d_i = d_{i,0}\lambda_0 + d_{i,1}\lambda_1 + \cdots + d_{i,n-1}\lambda_n - 1 \tag{6}$$

where $\{\lambda_0, \lambda_1, \cdots, \lambda_{n-1}\}$ is a basis for $GF(q^n)$ over $GF(q)$, then the sequence $d_{0,j}, d_{1,j}, \cdots$ is an m-sequence of length $q^{nm} - 1$ over $GF(q)$.

As one can observe, the sequence $d_{0,j}, d_{1,j}, \cdots$ is composed by the j-th component of each element d_i in the sequence D. Equivalently, the sequence $d_{0,j}, d_{1,j}, \cdots$ can be considered as a decimation sequence obtained from D giving rise to the following set of decimated sequences, as it is shown in Figure 2:

$$D^j = d_{0,j}, d_{1,j}, d_{2,j}, \cdots, \tag{7}$$

Furthermore, as it is stated in [16], since the sequences D^j, for $0 \leq j \leq n-1$, are generated by the same polynomial $f(x)$, all of them are shifted versions of the same m-sequence. Hence, taking D^0 as the reference, we can define θ_j as the shift of D^j respect to D^0. In [16], a method to obtain a primitive polynomial $f(x) \in GF[q^n, x]$ in terms of a given primitive polynomial $p(x) \in GF[q, x]$ is proposed by means of the computation of the shifts θ_j that satisfy the relationship between the m-sequences in the extended and base fields.

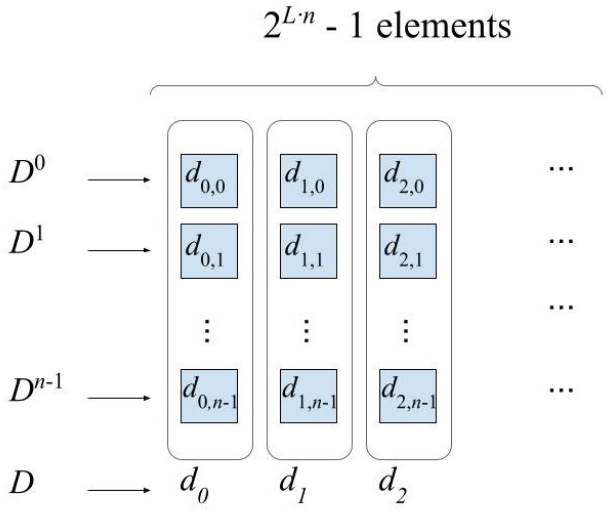

Figure 2. Relationship between m-sequences in the extended and base fields.

3. Efficient LFSR Implementation

The relationship between the m-sequences on $GF(2^n)$ and the m-sequences on $GF(2)$, as described in the previous section, also allows us to establish an equivalence between the LFSRs that generate them. This section presents a practical and efficient method to obtain such LFSRs, i.e., to obtain the feedback polynomial $p(x)$ and the seeds of each LFSR $< Ln, p(x) >$ that allow us to generate the same sequence as the LFSR $< L, f(x), n >$ from a given seed. Note that, according to the equivalent model, the n LFSRs in $GF(2)$ have the same connection polynomial $p(x)$ but different seeds, all of them related to the seed in $GF(2^n)$. Hence, in order to efficiently implement the LFSR $< L, f(x), n >$ using the equivalent model, it is necessary to solve three main questions: the computation of $p(x)$, the computation of the seeds, and how to speed up the performance of the binary LFSRs.

Since the connection polynomial $f(x)$ is primitive, the minimal polynomial $p(x)$ of the decimated sequences D^i is also primitive and unique for $0 \leq i \leq n-1$. Hence, $p(x)$ can be obtained analyzing a decimated sequence using the Massey–Berlekamp algorithm [18]. Consequently, the following Algorithm 1 is defined.

Algorithm 1: Computation of connection polynomial $p(x)$.

input : LFSR $< L, f(x), n >$
output: $p(x)$ in LFSR $< Ln, p(x) >$

1 *Implement the LFSR* $< L, f(x), n >$;
2 *Generate* $2Ln + 1$ *elements, at least, using any nonzero seed*;
3 *Obtain a decimated sequence* D^i *for any* $i, \quad 0 \leq i \leq n-1$;
4 *Obtain the minimal polynomial* $p(x)$ *of the* D^i *generated in step* 3

Once $p(x)$ has been determined, we can construct the n LFSRs $< Ln, p(x) >$, but we need their respective seeds in order to generate the same sequence as that generated from the initial seed $s = (d_0, d_1, \ldots, d_{L-1})$ in the extended field. As it is derived from the equivalent model, the L known elements in $GF(2^n)$ that compose the initial seed only provide us with nL bits, while n^2L bits are needed to complete the n seeds in $GF(2)$ (see Figure 3). Thus, the seed s^j of D^j can be partially written in terms of the seed s of D as:

$$s^j = d_{0,j}, d_{1,j}, \ldots, d_{L-1,j}, d_{L,j}, \ldots, d_{n \cdot L-1,j} \tag{8}$$

where $d_{L,j}, \ldots, d_{n \cdot L-1,j}$ are unknown for $0 \leq j \leq n-1$. However, the shifts θ_i that relate the binary sequences to each other allow us to obtain the seeds completely. The following subsections describes the computation of θ_j and s^j.

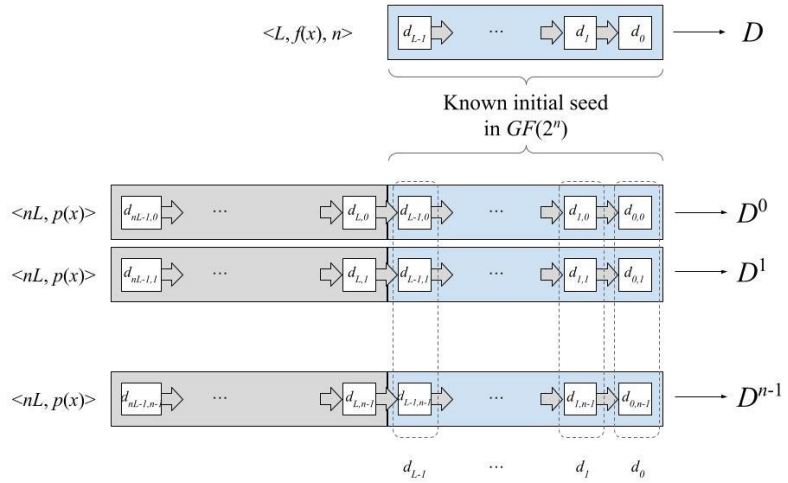

Figure 3. LFSR seeds in the binary equivalent model.

3.1. Computation of Shifts

All decimated sequences D^j have the same primitive minimal polynomial $p(x)$, obtained using Algorithm 1. Hence, all of them are shifted versions of the others. The shift θ_j of D^j with respect to D^0 allows one to obtain the state of the j-th LFSR (the one that generates D^j) from the state of the 0-th LFSR as follows:

$$(d_{0,j}, d_{1,j}, \ldots, d_{n\cdot L-1,j}) = (d_{0,0}, d_{1,0}, \ldots, d_{n\cdot L-1,0})A^{\theta_j}, \tag{9}$$

where A is the connection matrix of $p(x)$, that is:

$$A = \begin{bmatrix} 0 & 0 & \cdots & 0 & c_{n\cdot L} \\ 1 & 0 & \cdots & 0 & c_{n\cdot L-1} \\ 0 & 1 & \cdots & 0 & c_{n\cdot L-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & c_1 \end{bmatrix} \tag{10}$$

The computation of θ_j is not an easy task. Furthermore, the computation of A^{θ_j} is time-consuming. Instead, we can obtain the matrix $A^{(j)} = A^{\theta_j}$ solving the linear equation system of Equation (9). Note that we have n matrices $A^{(j)}$ to compute and thus n linear systems to solve. The linear system in Equation (9), stated using the first nL element, the seed, has nL equations and n^2L^2 unknowns. Each new element of the sequence defines nL new equations with the same unknowns. Hence, the $2Ln + 1$ elements of the m-sequence in $GF(2^n)$ generated in Algorithm 1 to obtain $p(x)$ provide enough equations to solve the systems. As an example, we consider a 3-stage LFSR defined in $GF(2^4)$, that is, the LFSR $< 3, f(x), 4 >$, where $f(x) = 1 + x + (0x9)x^3$ is primitive over $GF(2^4)$ and the primitive polynomial $x^4 + x + 1$ has been used to construct $GF(2^4)$. From the initial seed $(1, 1, 1, 1)$, $(0, 0, 0, 0)$, $(0, 1, 1, 0)$ or, equivalently, $(1 + \alpha + \alpha^2 + \alpha^3, 0, \alpha + \alpha^2)$, α being a root of $x^4 + x + 1$, the LFSR $< 3, f(x), 4 >$ generates the following sequence:

$$\begin{array}{l} 0010000011001110100100000\cdots \\ 1010010110000000001010001\cdots \\ 1011101000000001111001000\cdots \\ 0011100000010001011000011\cdots \end{array} \tag{11}$$

where the elements of $GF(2^4)$ are represented in columns with the least significant bit at the top. The four decimated sequences are all generated by the same primitive polynomial

$p(x) = x^{12} + x^6 + x^5 + x^3 + 1$ and, hence, all of them are shifted versions of the same m-sequence. Solving the system in Equation (9), we obtain the matrices $A^{(1)}, A^{(2)}, A^{(3)}$:

$$A^{(1)} = \begin{bmatrix} 100000000110 \\ 010000000011 \\ 101000000001 \\ 110100000000 \\ 011010000000 \\ 001101000000 \\ 000110100110 \\ 000011010101 \\ 000001101010 \\ 000000110011 \\ 000000011001 \\ 000000001100 \end{bmatrix} A^{(2)} = \begin{bmatrix} 000000101010 \\ 000000010101 \\ 100000001010 \\ 110000000101 \\ 111000000010 \\ 011100000001 \\ 101110101010 \\ 010111111111 \\ 001011111111 \\ 000101010101 \\ 000010101010 \\ 000001010101 \end{bmatrix} A^{(3)} = \begin{bmatrix} 000111111111 \\ 000011111111 \\ 000001111111 \\ 000000111111 \\ 100000011111 \\ 010000001111 \\ 001111111000 \\ 100000000011 \\ 110000000001 \\ 111111111111 \\ 011111111111 \\ 001111111111 \end{bmatrix} \tag{12}$$

Since the decimated sequences have a small period of $2^{12} - 1 = 4095$, we have also compared them to obtain the shifts $\theta_1 = 3276$, $\theta_2 = 3549$, and $\theta_3 = 3822$. In a real scenario, θ_j is not going to be computed. It is important to point out that the calculation of the $A^{(j)}$ matrices can be performed prior to the normal operation of the LFSR since they do not depend on the seeds.

3.2. Computation of Seeds

Let us consider that the shift matrices $A^{(j)} = A^{\theta_j}$ have already been precomputed. For any given seed $(d_0, d_1, \ldots, d_{L-1})$, the seeds of the binary LFSRs can be represented as:

$$(d_{0,j}, d_{1,j}, \ldots, d_{n \cdot L-1,j}) = (d_{0,0}, d_{1,0}, \ldots, d_{n \cdot L-1,0}) A^{(j)}, \tag{13}$$

where $d_{L,j}, \ldots, d_{n \cdot L-1,j}$ are unknowns for every $0 \leq j \leq n-1$. The values $d_{L,0}, \ldots, d_{n \cdot L-1,0}$ can be obtained solving the linear system composed with the $(n \cdot L - L)$ equations with the known values $d_{0,j}, d_{1,j}, \ldots, d_{L-1,j}$, for $0 \leq j \leq n-1$, i.e.:

$$d_{i,j} = \sum_{k=0}^{n \cdot L-1} d_{k,0} a_{k,i}^{(j)}, \quad 0 \leq i \leq L-1, \quad 1 \leq j \leq n-1, \tag{14}$$

where $a_{k,i}^{(j)}$ are the components of the matrix $A^{(j)}$. The remaining seeds $(d_{0,j}, d_{1,j}, \ldots, d_{n \cdot L-1,j})$, with $1 \leq j \leq L-1$, are calculated using Equation (13). Considering the precomputed matrices in Equation (12) of the previous example, the Equation (13) can be written as:

$$\begin{aligned} (1, 0, 1, d_{3,1}, \ldots, d_{11,1})) &= (0, 0, 1, d_{3,0}, \ldots, d_{11,0}) A^{(1)} \\ (1, 0, 1, d_{3,2}, \ldots, d_{11,2})) &= (0, 0, 1, d_{3,0}, \ldots, d_{11,0}) A^{(2)} \\ (0, 0, 1, d_{3,3}, \ldots, d_{11,3})) &= (0, 0, 1, d_{3,0}, \ldots, d_{11,0}) A^{(3)} \end{aligned} \tag{15}$$

Solving for $(d_{3,0}, \ldots, d_{11,0})$, we obtained the complete seed for the sequence D^0, that is, $(0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0)$. Next, using the matrices $A^{(1)}$, $A^{(2)}$ and $A^{(3)}$, the complete seeds of the sequences D^1, D^2, and D^3 are obtained. Hence, we have:

$$\begin{aligned} seed_0 &= (0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0) \\ seed_1 &= (1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0) \\ seed_2 &= (1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0) \\ seed_3 &= (0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0) \end{aligned} \tag{16}$$

3.3. Grouped Operations

Once the n binary LFSRs have been constructed, it is time to generate the sequence. Instead of running n independent instances of the binary LFSR, which require 1-bit operations with an n-bit processor, we propose to group the n LFSRs into a unique LFSR with connection polynomial $p(x)$ but using n-bit cells. The result is not an LFSR over $GF(2^n)$ but a parallel implementation of n LFSR over $GF(2)$ using only one processor. Since the addition and multiplication in the binary LFSRs correspond to the XOR and AND bitwise operations, respectively, instead of applying the XOR to 1-bit values, we apply it over n-bit values. The processor takes the same time to perform the XOR operation with 1-bit values than with n-bit values because the word length is n, thus saving a lot of execution time. Hence, Equation (4) can be redefined as follows:

$$\begin{bmatrix} d_{i,0} \\ \vdots \\ d_{i,n-1} \end{bmatrix} = c_1 \begin{bmatrix} d_{i-1,0} \\ \vdots \\ d_{i-1,n-1} \end{bmatrix} + c_2 \begin{bmatrix} d_{i-2,0} \\ \vdots \\ d_{i-2,n-1} \end{bmatrix} + \cdots + c_L \begin{bmatrix} d_{i-Ln,0} \\ \vdots \\ d_{i-Ln,n-1} \end{bmatrix}, \tag{17}$$

in such a way that the n sequences stated in Equation (7) are simultaneously generated using a unique polynomial $p(x)$ (see Equation (5)).

From the practical implementation perspective, there is no difference at all with respect to preforming a classical binary LFSR, which includes coding and execution, since the 1-bit XOR operation is actually performed taking n-bit operands. Hence, the n-bit grouped operation proposed in this paper is a way of not wasting the capacity of the operations of the n-bit processors. As a consequence, this implementation method increases the bit generation net rate by n because the generation of a new element of a 32-bit m-sequence takes the same amount of time as a new element of a 1-bit m-sequence.

4. Primitiveness Test

As mentioned in Section 2.2, the m-sequences generated by an LFSR in $GF(2^n)$ can be decomposed into n m-sequences generated by n LFSRs in $GF(2)$, so that when the feedback polynomial of the LFSR in the extended field is primitive, all LFSRs in $GF(2)$ have the same feedback polynomial, and it is also primitive. This relationship is what allows us to propose an algorithm to check if a polynomial is primitive over $GF(2^n)$.

In general terms, to check if a polynomial $f(x)$ over $GF(2^n)$ is primitive, we propose to build an LFSR whose feedback polynomial is $f(x)$ and generate $2 \cdot m \cdot n + 1$ elements, at least. The sequence generated by concatenating all generated elements is decomposed into n binary sequences by decimating by n, as it is stated in Equation (7). Next, the n sequences are processed to obtain the minimal polynomials of the binary LFSRs that generate them (This can be achieved using the Berlekamp–Massey algorithm). If all the n sequences are generated by the same polynomial and it is also primitive, then $f(x)$ is primitive over $GF(2^n)$. Algorithm 1 can be extended including one more step (step 5) to perform the check, resulting in Algorithm 2.

As an example, let us consider the degree 6 polynomial $f(x) = 0x1 + 0x2 \cdot x + 0x4 \cdot x^2 + 0x8 \cdot x^3 + 0x5 \cdot x^4 + 0x9 \cdot x^5 + x^6 \in GF[2^4, x]$, where the primitive polynomial $h(x) = x^4 + x + 1$ has been used to construct $GF(2^4)$. In order to check if $f(x)$ is primitive, we consider $f(x)$ as the feedback polynomial of an LFSR in the $GF(2^4)$ of 6 cells. For a random seed $(0x9, 0x4, 0x5, 0x9, 0x5, 0x1)$, we generate $2mn + 1 = 49$ elements, giving rise to the following decimated sequences:

$$\begin{aligned} D^0 &= 0010011011100000011010110001000000101011001111100 \\ D^1 &= 0101100111010011110000001010111101111001011010001 \\ D^2 &= 0000000011000101110101100100010010011000001000101 \\ D^3 &= 1111011111000011001111010101101110100100010100001 \end{aligned} \tag{18}$$

Algorithm 2: Primitiveness test.

input : polynomial $f(x)$ of degree m over $GF(2^n)$
output: primitive/non primitive

Implement the LFSR $< m, f(x), n >$;
Generate $2mn + 1$ *elements, at least, using any nonzero seed*;
Obtain the decimated sequences $D^i, 0 \leq i \leq n - 1$;
Obtain the minimal polynomial $p_i(x)$ *of each decimated sequence* D^i;
if $p_i(x) = p_j(x), \forall i, j$ *and degree of* $p_i(x)$ *is* mn **then**
  $f(x)$ is primitive
else
  $f(x)$ is non primitive
end

The same minimal polynomial $p(x)$ is obtained for each D^i sequence by means of the Massey–Berlekamp algorithm [18]. The polynomial is the following:

$$p(x) = x^{24} + x^{23} + x^{22} + x^{21} + x^{20} + x^{19} + x^{15} + x^{14} + x^9 + x^8 + x^7 + x^6 + 1 \quad (19)$$

Since $p(x)$ is primitive over $GF(2)$, we can conclude that $f(x)$ is primitive over $GF(2^4)$.

5. Efficiency and Security

The implementation presented in Section 3 considerably reduces the execution time of the LFSRs defined in $GF(2^n)$. Specifically, an implementation of the LFSR used in the SNOW 3G stream cipher has been performed and compared with the implementation provided in the technical specification of the protocol [10]. This is an LFSR $< 16, f(x), 32 >$. Hence, the equivalent model is based on 32 LFSRs $< 512, p(x) >$. The polynomial $p(x)$ is built from 1024 elements generated using the official implementation [10] following the steps established in Algorithm 1. The result is that the 32 decimated sequences have the same minimal polynomial $p(x)$ of degree 512, whose coefficients are represented below in compact hexadecimal format:

$$84009C624D4F75F17EDA41C663C5DFDED8A535DA1C5F70824152A7 \quad (20)$$

$$C23EDB90D572852A765FF5F2012A64F5D3FD361B005ADBA45A1995 \quad (21)$$

$$E64E48362706D62606828 \quad (22)$$

Despite the fact that $p(x)$ has 250 non-zero coefficients, and therefore 250 XOR operations are required to generate the next element in the sequence, the execution time is 3.3 times lower than the original implementation. The computation of the matrices $A^{(j)}$ is not considered, since this is performed prior to the normal operating of the LFSR. The times have been calculated by taking the average of 10 repetitions of each generation of 1000, 10,000, 20,000, 50,000, and 100,000 elements. Both implementations have been made in Python 3.9 language and have been executed on an Intel(R) Core(TM) i7-10510U 64-bit processor with 16 GB of RAM. Although the SNOW 3G algorithm has been designed to be executed on 32-bit platforms, the tests carried out on a 64-bit processor are completely valid since the greater word length of the processor compared to the algorithm does not affect the normal execution of our implementation. Note that the goal is to achieve an implementation of an algorithm defined in $GF(2^n)$ using n-bit operations. The case of working with processors of more than n bits offers the possibility of developing new faster implementations to make the most of their capacity, but requires adapting the algorithms to the new processor architecture. This is outside the scope of this work.

As mentioned in the Section 3, the net bit generation rate is increased by n when compared to a single binary LFSR implementation, that is, to a binary LFSR using the same connection polynomial. However, the improvement observed in the tests on SNOW 3G

does not reach the net rate. This is mainly due to the number of non-zero coefficients in $p(x)$ that slow down the computation. Therefore, in the case of equivalent binary polynomials $p(x)$ with a few nonzero coefficients, the real rate will reach the net rate. In the general case, the fewer non-zero coefficients in $p(x)$, the greater the improvement to the generation rate.

The efficiency of the proposed implementations, shown in Section 3 affects the security of the cryptosystems used by these LFSRs, although, in general, they are not intended to replace those currently used by most devices, such as smartphones, but rather for use in mobile devices without any type of restriction, such as personal computers or servers. Using the equivalent model implies multiplying by n, the number of bits needed to generate a new element of the m-sequence over $GF(2^n)$, that is, to generate n bits. In the case of SNOW 3G, it goes from 512 to 16,384 bits. Therefore, the theoretical improvement of the execution time by a factor n is associated with an increase in the same factor n in the amount of memory needed to generate the same number of bits. As a consequence, this implementation provides a substantial improvement for the calculation of m-sequences that, although it could not be deployed on some devices, could always be used for cryptanalysis tasks. Regarding the cost of our proposal, an increment in the memory cells must be taken in mind, rising from nL to n^2L bits.

Regarding the algorithm for primitiveness testing, we can conclude that it has a better performance than O'Connor's algorithm, one of the most used ones. The main advantage is that the proposed algorithm has to generate $2mn + 1$ elements instead of the whole sequence or alternatively to perform as many divisions as elements in the maximal sequence over the extended field. The validation and comparison tests have been performed using Mathematica software, version 10.0.0, on a 64-bits Microsoft Windows platform running on a Intel(R) Core (TM) processor with i7-4510U CPU @ 2.00 GHz and 16 GB RAM. The processor's temperature and the amount of simultaneous running processes have been taken into account for the execution time comparison. Figure 4 shows the behavior of the algorithm with respect to O'Connor's in $GF(2^8)$ and $GF(2^{16})$, respectively.

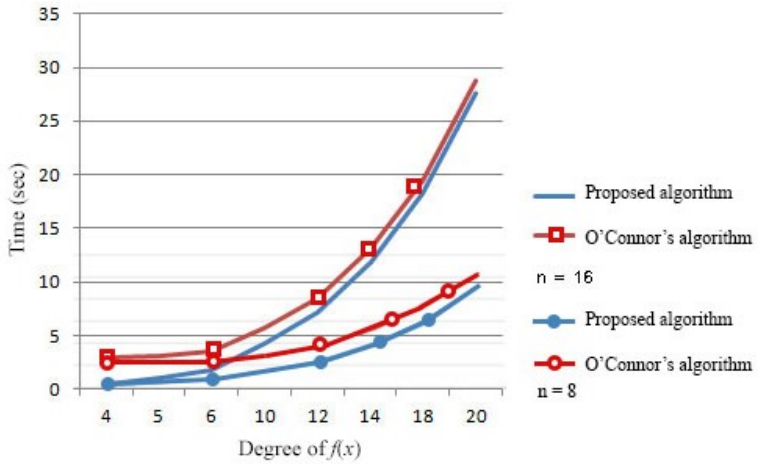

Figure 4. Execution time in $GF(2^8)$ and $GF(2^{16})$.

6. Conclusions

In this article, we have presented two real applications of the relationships between the m-sequences in $GF(2^n)$ and $GF(2)$. The first one is a new algorithm designed to verify the primitiveness of polynomials with coefficients in $GF(2^n)$ that improves the execution times of existing methods. The second one is the support of an efficient implementation of the LFSRs defined over the extended field $GF(2^n)$, which improves the other implementations. It enables better performance of the LFSR-based stream ciphers, often used in high speed communication systems. The improvement is achieved by a combination of the binary equivalent model of LFSRs in the extended field, which uses only binary operations, and

the n-bit grouped operation that take advantage of the n-bit processors. The feasibility of the implementation has been shown by applying it to the SNOW 3G stream cipher, whose execution time has been reduced by a factor of 3.3 with respect to the code provided in the technical specification of the protocol. These results can also be extended to cryptanalysis, making use of not only the grouped operations but of the underlying binary structure that may facilitate the parallelization of the operations. On the other hand, the proposed implementation is software-oriented, although the binary operations also allow hardware implementations. In this way, we provide a complete method to efficiently increase the bit generation rate.

Author Contributions: Writing of original draft and writing—review and editing, J.E.G., G.C., A.P. and A.O. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the research group BIOSIP (TIC-251) and by the Spanish State Research Agency (AEI) of the Ministry of Science and Innovation (MICINN), project P2QProMeTe (PID2020-112586RB-I00/AEI/10.13039/501100011033) co-funded by the European Regional Development Fund (ERDF, EU).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Menezes, A.J.; Van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*; CRC Press: Boca Ratón, FL, USA, 2020.
2. Shannon, C. Communication Theory of Secrecy Systems. *Bell Syst. Tech. J.* **1949**, *28*, 656–715. [CrossRef]
3. Golomb, S.W. *Shift Register Sequences*, 3rd Revised ed.; Aegean Park Press: Laguna Hills, CA, USA, 2017.
4. Padgette, J.; Bahr, J.; Batra, M.; Holtmann, M.; Smithbey, R.; Lily, C.; Scarfone, K. *Guide to Bluetooth Security*; NIST: Gaithersburg, MD, USA, 2017.
5. Jindal, P.; Singh, B. RC4 Encryption-A Literature Survey. *Procedia Comput. Sci.* **2015**, *46*, 697–705. [CrossRef]
6. Biham, E.; Dunkelman, O. Cryptanalysis of the A5/1 GSM stream cipher. In Proceedings of the International Conference on Cryptology in India, Calcutta, India, 10–13 December 2000; pp. 43–51.
7. Kiyomoto, S.; Tanaka, T.; Sakurai, K. K2: A stream cipher algorithm using dynamic feedback control. In Proceedings of the International Conference on Security and Cryptography, SECRYPT, Barcelona, Spain, 28–13 July 2007; Hernando, J., Fernández-Medina, E., Malek, M., Eds.; INSTICC Press: Lisboa, Portugal, 2007; pp. 204–213.
8. George, K.; Michaels, A.J. Designing a Block Cipher in Galois Extension Fields for IoT Security. *IoT* **2021**, *2*, 669–687. [CrossRef]
9. Panario, D.; Reis, L. The functional graph of linear maps over finite fields and applications. *Des. Codes Cryptogr.* **2019**, *87*, 437–453. [CrossRef]
10. ETSI/SAGE. *Specification of the 3GPP, Confidentiality and Integrity Algorithm UEA2 and UIA2; Document 2: SNOW 3G Specification*; ETSI: Sophia Antipolis, France, 2006.
11. Caforio, A.; Balli, F.; Banik, S. Melting SNOW-V: Improved lightweight architectures. *J. Cryptogr. Eng.* **2020**, *12*, 53–73. [CrossRef]
12. Ekdahl, P.; Johansson, T.; Maximov, A.; Yang, J. A new SNOW stream cipher called SNOW-V. *IACR Trans. Symmetr. Cryptol.* **2019**, *3*, 1–42. [CrossRef]
13. Ekdahl, P.; Maximov, A. SNOW-Vi: An Extreme Performance Variant of SNOW-V for Lower Grade CPUs. In Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec '21, Abu Dhabi, United Arab Emirates, 28 June–2 July 2021; pp. 261–272.
14. Avanzi, R.; Theriault, N. Effects of optimization for software implementations of small binary field arithmetic. In *International Workshop on the Arithmetic of Finite Fields*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 69–84.
15. Delgado-Mohatar, O.; Fúster-Sabater, A.; Sierra, J.M. Performance evaluation of highly efficient techniques for software implementation of LFSR. *Comput. Electr. Eng.* **2011**, *37*, 1222–1231. [CrossRef]
16. Komo, J.J.; Lam, M.S. Primitive Polynomials and m-sequences over $GF(q^m)$. *IEEE Trans. Inf. Theory* **1993**, *39*, 643–647. [CrossRef]
17. Park, W.J.; Komo, J.J. Relationships Between m-Sequences over $GF(q)$ and $GF(q^m)$. *IEEE Trans. Inf. Theory* **1989**, *35*, 183–186. [CrossRef]
18. Massey, J.L. Shift register synthesis and BCH decoding. *IEEE Trans. Inf. Theory* **1969**, *15*, 122–127. [CrossRef]