An Equivalence-Preserving Transformation of Shift Registers

Elena Dubrova

Royal Institute of Technology (KTH), Electrum 229, 164 40 Kista, Sweden
{dubrova}@kth.se

Abstract. The Fibonacci-to-Galois transformation is useful for reducing the propagation delay of feedback shift register-based stream ciphers and hash functions. In this paper, we extend it to handle Galois-to-Galois case as well as feedforward connections. This makes possible transforming Trivium stream cipher and increasing its keystream data rate by 27% without any penalty in area. The presented transformation might open new possibilities for cryptanalysis of Trivium, since it induces a class of stream ciphers which generate the same set of keystreams as Trivium, but have a different structure.

1 Introduction

Shift register-based cryptographic systems are the fastest and the most power-efficient cryptographic systems for hardware implementations [1]. The speed and the power are two crucial factors for future cryptographic systems, since they are expected to support very high data rates in 5G ultra-low power products and applications. The 5G is envisioned to have 1000 times higher traffic volume compared to current LTE deployments while providing a better quality of service [2]. Consumer data rates of hundreds of Mbps are expected to be available in a general scenario. In special scenarios, such as office spaces or dense urban outdoor environments, reliably achievable data rates of multi-Gbps are foreseen.

An n-bit shift register implements an n-variate mapping $\{0,1\}^n \rightarrow \{0,1\}^n$ of type

$$\begin{pmatrix} x_0 \\ \dots \\ x_{n-1} \end{pmatrix} \rightarrow \begin{pmatrix} f_0(x_0,\dots,x_{n-1}) \\ \dots \\ f_{n-1}(x_0,\dots,x_{n-1}) \end{pmatrix} \tag{1}$$

where each Boolean function f_i, $i \in \{0,1,\dots,n-1\}$, is of type:

$$f_i = x_{i+1} \oplus g_i(x_0,\dots,x_i,x_{i+2},\dots,x_{n-1}) \tag{2}$$

where "$\oplus$" is addition modulo 2 and "$+$" is addition modulo n.

Note that the function g_i in (2) does not depend on x_{i+1}. This is a necessary condition for invertibility of mappings implemented by a shift register [3]. A mapping $x \rightarrow f(x)$ on a finite set is called *invertible* if $f(x) = f(y)$ if and only if $x = y$. Stream ciphers usually use invertible mappings to prevent incremental reduction of the entropy of the state [4].

Another desirable property is long period. The *period* of a mapping is the length of the longest cycle in its state transition graph. Obviously, if we iterate a mapping a large number of times, we do not want the sequence of generated states to be trapped in a short cycle. Furthermore, as demonstrated by the cryptanalysis of A5, short cycles can be exploited to greatly reduce the complexity of the attack [5].

In this paper, we present a transformation which preserves both, invertibility and period, of a mapping. It makes possible to construct classes of shift registers which have structurally isomorphic state transition graphs and generate equivalent sets of output sequences. This is useful for optimizing the hardware performance of shift register-based stream ciphers [6–10] and hash functions [11]. We apply the presented transformation to Trivium [7] and show that it increases its keystream data rate by 27% without any penalty in area. The transformation can also be potentially useful for cryptanalysis since, within the class of shift registers generating equivalent sets of output sequences, some might be easier to cryptanalysize than others.

The presented transformation extends Fibonacci-to-Galois transformation of Non-Linear Feedback Shift Registers (NLFSR) [12] to the more general case of shift registers. Two main differences are:

1. The presented transformation can be applied to shift registers with both, feedback and feedforward connections (e.g. Trivium).
2. The presented transformation can be applied to any Galois NLFSR. The transformation [12] is applicable to uniform NLFSRs only[1].

The paper is organized as follows. Section 2 summarises basic notations used in the sequel. Section 3 describes previous work. Section 4 gives an informal description of the presented transformation. Section 5 formalizes the main result. Section 6 shows how the presented transformation can be applied to Trivium. Section 7 concludes the paper and discusses open problems.

2 Preliminaries

Throughout the paper, we use "$\oplus$" and "$\cdot$" to denote the *GF(2)* addition and multiplication, respectively.

The Algebraic Normal Form (ANF) [13] of a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ is a polynomial in *GF(2)* of type

$$f(x_0, x_1, \ldots, x_{n-1}) = \sum_{i=0}^{2^n-1} c_i \cdot x_0^{i_0} \cdot x_1^{i_1} \cdot \ldots \cdot x_{n-1}^{i_{n-1}},$$

where $c_i \in \{0,1\}$ and $(i_0 i_1 \ldots i_{n-1})$ is the binary expansion of i.

The *dependence set* [14] of a Boolean function is defined by

$$dep(f) = \{j \mid f(x_j = 0) \neq f(x_j = 1)\},$$

where $f(x_j = k) = f(x_0, \ldots, x_{j-1}, k, x_{j+1}, \ldots, x_{n-1})$ for $k \in \{0,1\}$.

[1] An n-bit NLFSR is *uniform* if, for all $i \in \{\tau, \tau+1, \ldots, n-1\}$, the largest index of variables of function g_i in (2) is smaller than or equal to τ, where τ is the maximal index such that, for all $j \in \{0,1,\ldots,\tau-1\}$, $g_j = 0$.

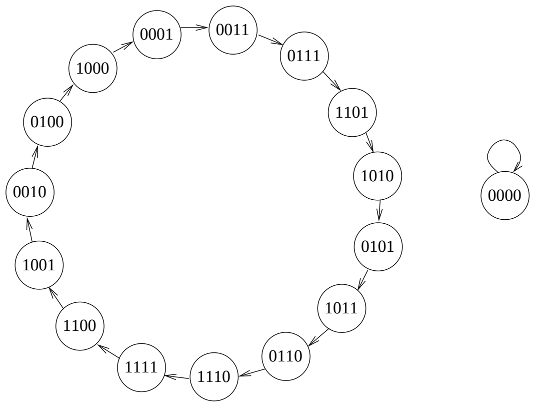

Fig. 1. The state transition graph of the mapping (3). Each 4-tuple represents a state (x_0, x_1, x_2, x_3).

Throughout the paper we also use the expression "dependence set of a monomial of the ANF". It should not create any ambiguity since each monomial of the ANF represents a Boolean function. For example, for $m = x_1 x_3$, $dep(m) = \{1,3\}$.

The *state* of an n-variate mapping $\{0,1\}^n \to \{0,1\}^n$ is any specific assignment of $\{x_0, x_1, \ldots, x_{n-1}\}$. The *State Transition Graph* (STG) is a directed graph in which the nodes represent the states and the edges show possible transitions between the states.

For example, the STG of the 4-variate mapping $\{0,1\}^4 \to \{0,1\}^4$:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \to \begin{pmatrix} x_1 \\ x_2 \\ x_3 \oplus x_1 x_2 \\ x_0 \oplus x_3 \end{pmatrix}. \tag{3}$$

is shown in Figure 1. This mapping is invertible. It has period 15.

Any n-variate mapping $\{0,1\}^n \to \{0,1\}^n$ can be implemented by an n-bit *shift register* shown in Figure 2. It consists of n binary storage elements, called *stages*, and n *updating functions* $f_i : \{0,1\}^n \to \{0,1\}$ which determine how the values of stages are updated [3]. At every clock cycle, the next state is computed from the current state by updating the values of all stages simultaneously to the values of the corresponding updating functions.

The *degree of parallelization* of a shift register is the number of bits of output which are produced at each clock cycle.

A shift register can be implemented either in the *Fibonacci* or in the *Galois* configuration [12]. In the former, all updating functions except f_{n-1} are of type $f_i(x) = x_{i+1}$, for $i \in \{0,1,\ldots,n-2\}$. In other words, feedback/feedforward connections are applied to the input stage of the shift register only. In the latter, feedback/feedforward connections can potentially be applied to every stage.

Two shift registers are *equivalent* if their sets of output sequences are equal.

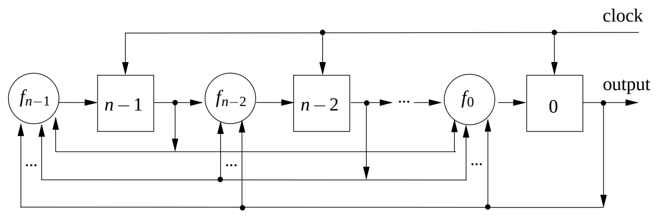

Fig. 2. The general structure of an n-bit shift register with updating functions.

3 Previous Work

For LFSRs, there exist a one-to-one mapping between the Fibonacci and the Galois configurations. The Galois LFSR generating the same sets of output sequences as a given Fibonacci LFSR can be obtained by reversing the order of the feedback taps and adjusting the initial state. Several transformations aiming to optimize the traditional LFSRs with respect to different parameters were presented, including [15–19].

For NLFSRs, however, the Galois configuration is not unique. Usually, there are many n-bit Galois NLFSRs which are equivalent to a given n-bit Fibonacci NLFSR. On the other hand, not every n-bit Galois NLFSR has an equivalent n-bit Fibonacci NLFSR. The latter is because, while an output sequence of every n-bit Fibonacci NLFSR can be described by a nonlinear recurrence of order n [3], for an n-bit Galois NLFSR such a recurrence may not exist. It was shown in [12] that a nonlinear recurrence of order n always exists for uniform n-bit NLFSRs. An algorithm for constructing a best uniform Galois NLFSR which is equivalent to a given Fibonacci NLFSR was presented in [20].

A interesting type of NLFSRs was introduced by Massey and Liu in [21]. Similarly to the Fibonacci NLFSRs, these NLFSRs have a single feedback function Boolean function, f, of the state variables $x_0, x_1, \ldots, x_{n-1}$. However, the output of $f(x_0, x_1, \ldots, x_{n-1})$ is fed not only to the bit $n-1$ but also to other bits, namely

$$\begin{array}{l} f_i(x_0, x_1, \ldots, x_{n-1}) = f(x_0, x_1, \ldots, x_{n-1}), \text{ for } i = n-1 \\ f_i(x_0, x_1, \ldots, x_{n-1}) = x_{i+1} \oplus w_i \cdot f(x_0, x_1, \ldots, x_{n-1}), \text{ for } i = \{0, 1, \ldots, n-2\} \end{array}$$

where $w_i \in \{0,1\}$. It was shown in [21] that every NLFSR of this type has an equivalent NLFSR in the Fibonacci configuration. Moreover, the mapping between the two configurations is one-to-one.

4 Intuitive Description

We start with an intuitive description of the presented transformation.

Consider an n-variate mapping of type (1). It can be represented by an n-bit ring with connections corresponding to the monomials of ANFs of functions f_i induced by the mapping[2]. Each connection has a single sink and one or more sources. The

[2] We use an n-bit ring as a simplification of an n-bit shift register which shows the structure of its feedback/feedforward connections. The gates implementing *GF(2)* addition (XORs) are

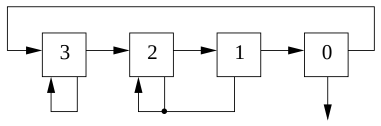

Fig. 3. The 4-bit ring with connections corresponding to the monomials of ANFs of Boolean functions induced by the mapping (3).

sources originate in the stages corresponding to the state variables of the monomial. The sink points to the stage i with the index of the updating function f_i represented by the ANF, $i \in \{0,1,\ldots,n-1\}$. The output is represented by an outgoing edge from the corresponding stage.

For example, if we assume that the output is taken from the stage 0, then the 4-variate mapping (3) is represented by the 4-bit ring shown in Figure 3. The connection with sources 1,2 and sink 2 corresponds to the monomial x_1x_2 of f_2.

The transformation presented in the paper moves a connection either left or right in the ring, without changing its length or shape, i.e. the sink and all sources are moved by the same number of stages. For example, if the monomial x_1x_2 of f_2 in the mapping (3) is moved one stage right, we get the mapping

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \begin{pmatrix} x_1 \\ x_2 \oplus x_0x_1 \\ x_3 \\ x_0 \oplus x_3 \end{pmatrix}. \tag{4}$$

Its STG is shown in Figure 4.

Indexes crossing the 0 to $n-1$ border of the ring are updated modulo n. So, if we move the monomial x_3 of f_3 in the mapping (3) one stage left, we get

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \begin{pmatrix} x_1 \oplus x_0 \\ x_2 \\ x_3 \oplus x_1x_2 \\ x_0 \end{pmatrix}. \tag{5}$$

Its STG is shown in Figure 5.

Three conditions should hold for the transformation to preserve the cycle structure of the STG. First, only the connections corresponding to the monomials of functions g_i in the equation(2), $i \in \{0,1,\ldots,n-1\}$, can be moved. The monomial x_{i+1} of functions f_i cannot be moved, where "$+$" is addition modulo n.

Second, sources of a connection can be moved k stages left/right if the functions f_i of the k stages on the left/right of each source do shifts only (i.e. no source crosses any of the sinks of connections related to g_is during its move). This condition makes sure that time dependencies in the computation are preserved.

omitted and the gates implementing *GF(2)* multiplication (ANDs) are represented by a dot. Everything unnecessary for structural analysis is removed.

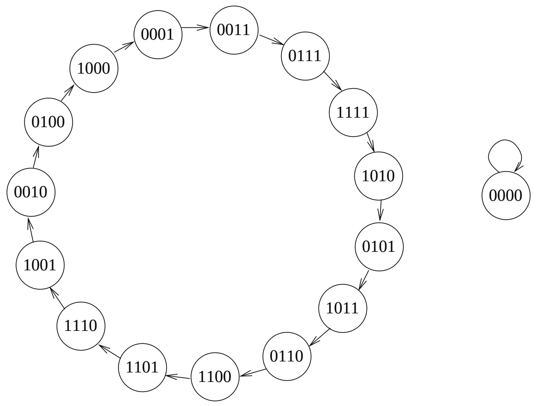

Fig. 4. The state transition graph of the mapping (4).

For example, the monomial x_3 of f_3 in the mapping (3) can be moved one stage left to f_0, or two stages left to f_1, but not one stage right to f_2 since $f_2 = x_3 \oplus x_1x_2$. Due to the circular structure of the ring, we can always reach any stage either from the left or from the right. It is sufficient that the condition is satisfied only in one of the directions. For example, although x_3 cannot be moved to f_1 from the right, it can be moved to f_1 from the left. So, we can move x_3 to f_1.

Third, the sink of a connection can be moved k stages left/right if k stages on the left/right of the sink do not serve as sources of any other connection of any g_i, $i \in \{0,1,\ldots,n-1\}$ (i.e. the sink does not cross any of the sources of connections related to g_is during its move). This condition makes sure that values of variables participating in the computation are correct. For example, the monomial x_1x_2 of f_2 in the mapping (3) cannot be moved to f_3 because x_3 is a variable of a monomial of g_3.

Suppose that, in addition to preserving the cycle structure of the STG of a mapping, we want to preserve the binary sequence generated by one of its functions, say f_i, for any $i \in \{0,1,\ldots,n-1\}$. This might be desirable because, for example, this sequence is used as a keystream and we do not want to change its properties. Then, in addition to the three conditions above, we need to add a condition that neither the sink nor the sources of a shifted connection cross the border between ith and $i-1$st modulo n stage of the ring.

For example, if the value computed by the function f_0 of the mapping (3) is used as an output, then, in order to preserve the output sequence after the transformation, neither the sink nor the sources of a shifted connection should cross the border between 0th and 3rd stage. This holds for the transformation from (3) to (4). Indeed, we can see from Figures 1 and 4 that, for the initial state $(x_0,x_1,x_2,x_3) = (0001)$, the functions f_0 of both mappings generate the periodic sequence [3] 000110101111001. However, this is not the case for the mapping (5). From its STG in Figure 5, we can see that the sequence generated by its function f_0, namely 000111101100101 is different from the sequence

[3] Note that in this case the initial states are the same but generally they can be different [22].

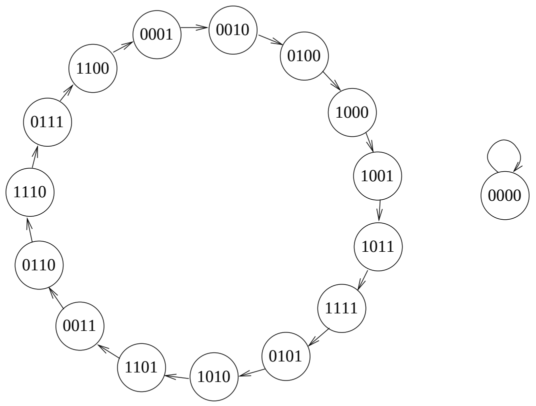

Fig. 5. The state transition graph of the mapping (5).

above for any initial state. This is because the shifted connection crosses the border between 0th and 3rd stage.

5 Formal Description

In this section, we give a formal description of the presented transformation.

Definition 1. *The shifting, denoted by $f_i \overset{m}{\to} f_j$, $i,j \in \{0,1,\ldots,n-1\}$, $i \neq j$, transforms an n-variate mapping of type (1) to another n-variate mapping in which the ANF monomial m of f_i is moved to f_j and each index $a \in dep(m)$ is changed to b defined by*

$$b = (a - i + j) \, mod \, n \tag{6}$$

For example, by applying shifting $f_2 \overset{x_1 x_2}{\to} f_1$ to the 4-variate mapping (3), we get the mapping (4).

Given a shifting $g_i \overset{m}{\to} g_j$, we denote by g_i^* the function $g_i^* = g_i \oplus m$.

Definition 2. *Given an n-variate mapping of type (1) in which the values computed by f_z, $z \in \{0,1,\ldots,n-1\}$ are used as an output sequence, a shifting $g_i \overset{m}{\to} g_j$, $i,j \in \{0,1,\ldots,n-1\}$, $i \neq j$, is valid if for each $a \in dep(m)$ and for b defined by (6) the following three conditions hold:*

1. *For each $c \in [a,b] \setminus \{i\}$, $g_c = 0$; if $i \in [a,b]$, $g_i^* = 0$.*
2. *For all $k \in [i,j]$:*
   (a) $k \notin dep(g_i^*)$;
   (b) for all $p \in \{0,1,...,i-1,i+1,...n-1\}$, $k \notin dep(g_p)$.
3. *Neither $[a,b]$ nor $[i,j]$ contains both, z and $(z-1)$ modulo n*

where

$$[a,b] = \{a, a-1, \ldots, b\} \text{ and } [i,j] = \{i, i-1, \ldots, j\} \text{ for } i > j$$
$$[a,b] = \{a, a+1, \ldots, b\} \text{ and } [i,j] = \{i, i+1, \ldots, j\} \text{ otherwise}$$

and "+" and "−" are addition and subtraction modulo n, respectively.

If the values of more than one stage z are used to compute the output sequence (e.g. as in Grain [6], Trivium [7], or other filter generators), then the condition 3 should hold for each pair z and $(z-1)$ modulo n.

For example, for the mapping (3) with f_0 as an output, shifting $g_2 \overset{x_1x_2}{\rightarrow} g_1$ is valid. However, shiftings $g_3 \overset{x_3}{\rightarrow} g_2$ and $g_2 \overset{x_1x_2}{\rightarrow} g_3$ are not valid since the former violates the condition 1 and the latter violates the condition 2 of Definition 2.

In the theorem below, we use $f(s)$ to denote the value of the function f evaluated for the vector s. We also use $f|_j$ $(f|_{-j})$ to denote the function obtained from f by adding (subtracting) j modulo n to (from) indexes of all variables of f. For example, if $f = x_1x_2 \oplus x_3$, then $f|_2 = x_3x_4 \oplus x_5$ and $f|_{-1} = x_0x_1 \oplus x_2$.

Theorem 1. *Let F be a mapping of type (1) and F' be a mapping obtained from F by applying a valid shifting $g_i \overset{m}{\rightarrow} g_j$, $i, j \in \{0,1,\ldots,n-1\}$, $i \neq j$. If F is initialized to the state $s = (s_0, s_1, \ldots, s_{n-1})$ and F' is initialized to the state $r = (r_0, r_1, \ldots, r_{n-1})$ such that*

$$\begin{array}{l} \text{if } i > j, \text{ then } r_k = s_k \oplus m|_{k-i-1}(s) \text{ for } k \in \{i, i-1, ..., j+1\} \\ \text{if } i < j, \text{ then } r_k = s_k \oplus m|_{k-j-1}(s) \text{ for } k \in \{i+1, i+2, ..., j\} \end{array} \tag{7}$$

and $r_k = s_k$ for all remaining $k \in \{0,1,\ldots,n-1\}$, then sequences of states generated by F and F' may differ only in bit positions $i, i-1, \ldots, j+1$ if $i > j$ and only in bit positions $i+1, i+2, ..., j$ if $i < j$.

Proof: First we show that Theorem 1 holds for the case of $i = j+1$. In this case, the equation (7) is reduced to $r_k = s_k \oplus m|_{-1}(s)$ for $k = j+1$.

Suppose that $m = x_{a_1}x_{a_2}\ldots x_{a_t}$, where $a_l \in \{0,1,\ldots,n-1\}$, for all $l \in \{1,2,\ldots,t\}$, and $a_1 > a_2 > \ldots > a_t$. For simplicity, let us assume that the values computed by f_0 are used as an output sequence of F. If the shifting $g_i \overset{m}{\rightarrow} g_j$ is valid, then, from the condition 3 of Definition 2, we can conclude that $a_t > 0$. Thus, after shifting, m changes to $x_{a_1-1}x_{a_2-1}\ldots x_{a_t-1}$. Furthermore, from the condition 2 of Definition 2 we can conclude that $\{j+1, j\} \not\subset dep(g^*_{j+1})$ and $\{j+1, j\} \not\subset dep(g_p)$ for all $p \in \{0,1,...,j,j+2,...n-1\}$. Therefore, F is of type

$$\begin{pmatrix} x_0 \\ \ldots \\ x_j \\ x_{j+1} \\ \ldots \\ x_{n-1} \end{pmatrix} \rightarrow \begin{pmatrix} x_1 \oplus g_0(x_0,\ldots,x_{j-1},x_{j+2},\ldots,x_{n-1}) \\ \ldots \\ x_{j+1} \oplus g_j(x_0,\ldots,x_{j-1},x_{j+2},\ldots,x_{n-1}) \\ x_{j+2} \oplus g^*_{j+1}(x_0,\ldots,x_{j-1},x_{j+2},\ldots,x_{n-1}) \oplus x_{a_1}x_{a_2}\ldots x_{a_t} \\ \ldots \\ x_{n-1} \oplus g_{n-1}(x_0,\ldots,x_{j-1},x_{j+2},\ldots,x_{n-1}) \end{pmatrix}$$

and F' is of type

$$\begin{pmatrix} x_0 \\ \ldots \\ x_j \\ x_{j+1} \\ \ldots \\ x_{n-1} \end{pmatrix} \rightarrow \begin{pmatrix} x_1 \oplus g_0(x_0,\ldots,x_{j-1},x_{j+2},\ldots,x_{n-1}) \\ \ldots \\ x_{j+1} \oplus g_j(x_0,\ldots,x_{j-1},x_{j+2},\ldots,x_{n-1}) \oplus x_{a_1-1}x_{a_2-1}\ldots x_{a_t-1} \\ x_{j+2} \oplus g^*_{j+1}(x_0,\ldots,x_{j-1},x_{j+2},\ldots,x_{n-1}) \\ \ldots \\ x_{n-1} \oplus g_{n-1}(x_0,\ldots,x_{j-1},x_{j+2},\ldots,x_{n-1}) \end{pmatrix}$$

Note that, due to the restriction imposed on the function g_i in equation (2), $j+2 \notin \{a_1, a_2, \dots, a_t\}$ and therefore $j+1 \notin \{a_1-1, a_2-1, \dots, a_t-1\}$. In addition, from the condition 1 of Definition 2 we can conclude that, for all $l \in \{1,2,\dots,t\}$, $g_{c_l} = 0$ for $c_l \in \{a_l, a_l-1\}$.

Suppose that F is initialized to a state $s = (s_0, s_1, \dots, s_{n-1})$ and F' is initialized to a state $r = (s_0, s_1, \dots, s_j, s_{j+1} \oplus s_{a_1-1}s_{a_2-1} \dots s_{a_t-1}, s_{j+2}, \dots, s_{n-1})$. On one hand, for F, the next state $s^+ = (s_0^+, s_1^+, \dots, s_{n-1}^+)$ is given by:

$$\begin{aligned}
&s_0^+ = s_1 \oplus g_0(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1})\\
&\dots\\
&s_j^+ = s_{j+1} \oplus g_j(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1})\\
&s_{j+1}^+ = s_{j+2} \oplus g_{j+1}^*(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1}) \oplus s_{a_1}s_{a_2}\dots s_{a_t}\\
&\dots\\
&s_{n-1}^+ = s_0 \oplus g_{n-1}(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1})
\end{aligned}$$

On the other hand, for F', the next state $r^+ = (r_0^+, r_1^+, \dots, r_{n-1}^+)$ is given by:

$$\begin{aligned}
&r_0^+ = s_1 \oplus g_0(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1})\\
&\dots\\
&r_j^+ = s_{j+1} \oplus s_{a_1-1}s_{a_2-1}\dots s_{a_t-1} \oplus g_j(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1}) \oplus s_{a_1-1}s_{a_2-1}\dots s_{a_t-1}\\
&\quad = s_{j+1} \oplus g_j(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1})\\
&r_{j+1}^+ = s_{j+2} \oplus g_{j+1}^*(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1})\\
&\dots\\
&r_{n-1}^+ = s_0 \oplus g_{n-1}(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1})
\end{aligned}$$

We can see that the next states of F and F' can potentially differ in the bit position $j+1$ only. They are the same for all other bits.

In order to extend this conclusion to a sequence of states, it remains to show that r_{j+1}^+ can be expressed as $r_{j+1}^+ = s_{j+1}^+ \oplus s_{a_1-1}^+ s_{a_2-1}^+ \dots s_{a_t-1}^+$. From

$$s_{j+1}^+ = s_{j+2} \oplus g_{j+1}^*(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1}) \oplus s_{a_1}s_{a_2}\dots s_{a_t}$$

we can derive $s_{j+2} = s_{j+1}^+ \oplus g_{j+1}^*(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1}) \oplus s_{a_1}s_{a_2}\dots s_{a_t}$. By substituting this expression into

$$r_{j+1}^+ = s_{j+2} \oplus g_{j+1}^*(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1})$$

and eliminating the double occurrence of $g_{j+1}^*(s_0, \dots, s_{j-1}, s_{j+2}, \dots, s_{n-1})$, we get

$$r_{j+1}^+ = s_{j+1}^+ \oplus s_{a_1}s_{a_2}\dots s_{a_t}.$$

Since $s_{a_1}s_{a_2}\dots s_{a_t} = s_{a_1-1}^+ s_{a_2-1}^+ \dots s_{a_t-1}^+$, we obtain $r_{j+1}^+ = s_{j+1}^+ \oplus s_{a_1-1}^+ s_{a_2-1}^+ \dots s_{a_t-1}^+$.

By exchanging the roles of r and s and of i and j in the proof above, we can show that the result also applies for the case of $i = j-1$. Since any shifting can be performed by repeatedly applying either $g_{j+1} \overset{m}{\to} g_j$ or $g_{j-1} \overset{m}{\to} g_j$ as many steps as required, Theorem 1 holds for the general case.

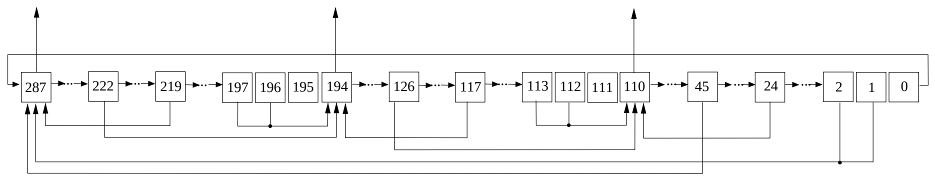

Fig. 6. The structure of Trivium.

□

The following result follows directly from Theorem 1.

Lemma 1. *Let F be a mapping of type (1). Any mapping F' obtained from F by applying a sequence of valid shiftings generates a set of output sequences equivalent to the one of F.*

6 Transforming Trivium

In this section, we show how the presented transformation can be applied to Trivium stream cipher.

Trivium [7] is defined by a 288-variate mapping in which all but 3 out of 288 of functions are of type $f_i = x_{i+1}$. The remaining 3 functions are given by:

$$\begin{aligned} f_{287} &= x_0 \oplus x_1 x_2 \oplus x_{45} \oplus x_{219} \\ f_{194} &= x_{195} \oplus x_{196} x_{197} \oplus x_{117} \oplus x_{222} \\ f_{110} &= x_{111} \oplus x_{112} x_{113} \oplus x_{24} \oplus x_{126} \end{aligned}$$

The structure of 288-bit ring representing Trivium is shown in Figure 6. The outputs from stages 110, 94 and 287 are added to get the keystream:

$$f_{output} = f_{287} \oplus f_{194} \oplus f_{110}.$$

There are many different possibilities for modifying Trivium. If the target is to minimize the propagation delay, then one possible solution obtained by applying the presented transformation is:

$$\begin{array}{lll} f_{287} = x_0 \oplus x_{219} & f_{194} = x_{195} \oplus x_{222} & f_{110} = x_{111} \oplus x_{24} \\ f_{218} = x_{219} \oplus x_{220} x_{221} & f_{131} = x_{132} \oplus x_{133} x_{134} & f_{21} = x_{22} \oplus x_{23} x_{24} \\ f_{210} = x_{211} \oplus x_{133} & f_{118} = x_{119} \oplus x_{134} & f_{17} = x_{18} \oplus x_{63} \end{array}$$

and the remaining functions of type $f_i = x_{i+1}$. The keystream is computed as previously. By Theorem 1, it is equivalent to the keystream generated by the original Trivium. The reader can easily see that, in the original Trivium, the propagation delay is given by:

$$d_{original} = 2d_{XOR} + d_{AND} + d_{FF}$$

Gate	Delay, ps
2-input AND	87
2-input XOR	115
flip-flop	221

Table 1. Propagation delays for a typical 90nm CMOS technology.

where d_{XOR}, d_{AND} and d_{FF} are the delays of the 2-input XOR, the 2-input AND, and the flip-flop, respectively. On the other hand, for the modified Trivium:

$$d_{modified} = d_{XOR} + d_{AND} + d_{FF}$$

By substituting d_{XOR}, d_{AND} and d_{FF} by values shown in Table 1, we get $d_{original} = 538$ ps and $d_{modified} = 423$ ps.

A shift register with the propagation delay of 538 ps can support data rates up to 1.86 Gbits/s. A shift register with the propagation delay of 423 ps can support data rates up to 2.36 Gbits/s. Note that 0.5 Gbits/s improvement (27%) comes without any penalty in area, since the number of gates before and after the transformation remains the same.

It should be noted that the transformation reduces the maximum possible degree of parallelization of Trivium from the original 64 to 8. The modified Trivium can generate up to 8 bits per clock cycle because no variables are taken from 7 consecutive stages after each sink and after outputs 110, 94 and 287. The modified Trivium with the degree of parallelization 8 can support data rates up to 18.88 Gbits/s. The original Trivium with the degree of parallelization 8 can support data rates up to 14.88 Gbits/s.

7 Conclusion

We presented a transformation which can be applied to an n-bit shift register to construct other n-bit shift registers which generate the same set of output sequences. Using the example of Trivium stream cipher, we demonstrated that this transformation is useful for optimizing its hardware performance.

Being able to construct different shift registers generating equivalent sets of output sequences might be potentially useful for cryptanalysis. Exploring this opportunity to cryptanalyze Trivium is a focus of our future works.

8 Acknowledgements

This work was supported in part the research grant No 621-2010-4388 from the Swedish Research Council and in part by the research grant No SM12-0005 from the Swedish Foundation for Strategic Research.

References

1. T. Good and M. Benaissa, "ASIC hardware performance," *New Stream Cipher Designs: The eSTREAM Finalists, LNCS 4986*, pp. 267–293, 2008.

2. Ericsson, "5G radio access - research and vision." White paper, 2013. http://www.ericsson.com/news/130625-5g-radio-access-research-and-vision_244129228_c.
3. S. Golomb, *Shift Register Sequences*. Aegean Park Press, 1982.
4. A. Klimov and A. Shamir, "A new class of invertible mappings," in *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES'02, (London, UK), pp. 470–483, Springer-Verlag, 2002.
5. A. B. Xu, D. K. He, and X. M. Wang, "An implementation of the GSM general data encryption algorithm A5," in *Proceedings of CHINACRYPT'94*, 1994.
6. M. Hell, T. Johansson, A. Maximov, and W. Meier, "The Grain family of stream ciphers," *New Stream Cipher Designs: The eSTREAM Finalists, LNCS 4986*, pp. 179–190, 2008.
7. C. Cannière and B. Preneel, "Trivium," *New Stream Cipher Designs: The eSTREAM Finalists, LNCS 4986*, pp. 244–266, 2008.
8. B. Gammel, R. Göttfert, and O. Kniffler, "Achterbahn-128/80: Design and analysis," in *SASC'2007: Workshop Record of The State of the Art of Stream Ciphers*, pp. 152–165, 2007.
9. B. Gittins, H. A. Landman, S. O'Neil, and R. Kelson, "A presentation on VEST hardware performance, chip area measurements, power consumption estimates and benchmarking in relation to the AES, SHA-256 and SHA-512." Cryptology ePrint Archive, Report 2005/415, 2005. http://eprint.iacr.org/2005/415.
10. B. M. Gammel, R. Göttfert, and O. Kniffler, "An NLFSR-based stream cipher," in *ISCAS*, 2006.
11. J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "Quark: A lightweight hash," *Journal of Cryptology*, vol. 26, no. 2, pp. 313–339, 2013.
12. E. Dubrova, "A transformation from the Fibonacci to the Galois NLFSRs," *IEEE Transactions on Information Theory*, vol. 55, pp. 5263–5271, November 2009.
13. T. W. Cusick and P. Stănică, *Cryptographic Boolean functions and applications*. San Diego, CA, USA: Academic Press, 2009.
14. R. K. Brayton, C. McMullen, G. Hatchel, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms For VLSI Synthesis*. Kluwer Academic Publishers, 1984.
15. M. Goresky and A. Klapper, "Fibonacci and Galois representations of feedback-with-carry shift registers," *IEEE Transactions on Information Theory*, vol. 48, pp. 2826–2836, 2002.
16. G. Mrugalski, J. Rajski, and J. Tyszer, "Ring generators - New devices for embedded test applications," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 9, pp. 1306–1320, 2004.
17. D. Kagaris, "A similarity transform for linear finite state machines," *Discrete Appl. Math.*, vol. 154, pp. 1570–1577, July 2006.
18. F. Arnault, T. Berger, M. Minier, and B. Pousse, "Revisiting LFSRs for cryptographic applications," *IEEE Transactions on Information Theory*, vol. 57, no. 12, pp. 8095–8113, 2011.
19. F. Arnault, T. P. Berger, and B. Pousse, "A matrix approach for FCSR automata," *Cryptography Commun.*, vol. 3, pp. 109–139, June 2011.
20. J.-M. Chabloz, S. Mansouri, and E. Dubrova, "An algorithm for constructing a fastest Galois NLFSR generating a given sequence," in *Sequences and Their Applications - SETA 2010* (C. Carlet and A. Pott, eds.), vol. 6338 of *Lecture Notes in Computer Science*, pp. 41–54, Springer Berlin / Heidelberg, 2010.
21. J. Massey and R.-W. Liu, "Equivalence of nonlinear shift-registers," *IEEE Transactions on Information Theory*, vol. 10, no. 4, pp. 378–379, 1964.
22. E. Dubrova, "Finding matching initial states for equivalent NLFSRs in the Fibonacci to the Galois configurations," *IEEE Transactions on Information Theory*, vol. 56, pp. 2961–2967, June 2010.