X-FCSR: a new software oriented stream cipher based upon FCSRs

François Arnault[1], Thierry P. Berger[1], Marine Minier[2], and Cédric Lauradoux[3]

[1] XLIM, Faculté des Sciences de Limoges
23 avenue Albert Thomas, F-87060 Limoges Cedex - France
FirstName.Name@unilim.fr
[2] CITI - INSA de Lyon - Bâtiment Léonard de Vinci
21 Avenue Jean Capelle, 69621 Villeurbanne Cedex - France
marine.minier@insa-lyon.fr
[3] INRIA - projet CODES, B.P. 105, 78153 Le Chesnay Cedex - France
cedric.lauradoux@insa-lyon.fr

Abstract. Feedback with Carry Shift Registers (FCSRs) are a promising alternative to LFSRs in the design of stream cipher. The previous constructions based on FCSRs were dedicated to hardware applications [3]. In this paper, we will describe X-FCSR a family of software oriented stream cipher using FCSRs. The core of the system is composed of two 256-bits FCSRs. We propose two versions: X-FCSR-128 and X-FCSR-256 which output respectively 128 and 256 bits at each iteration. We study the resistance of our design against several cryptanalyses. In this way, we achieve a high throughput and secure stream ciphers suitable for software applications (6.3 cycles/byte).
Keywords: stream cipher, FCSRs, software design, cryptanalysis.

Introduction

Following the recent development of algebraic attacks [6, 12, 13], it seems difficult to design good stream ciphers using combined or filtered LFSRs. A FCSR is similar to LFSRs, but it performs operations with carries, and so its transition function is not linear. Such an automaton computes the 2-adic expansion of some 2-adic rational number p/q. This can be used to prove several interesting properties of FCSRs: proved period, non-degenerated states, good statistical properties [23, 24, 17]. The high non-linearity of FCSR feedback function provides an intrinsic resistance to algebraic attacks, and seems also to prevent correlation attacks. There exists a hardware efficient family of stream ciphers based on FCSRs: the filtered FCSR or F-FCSR [1, 4, 2, 3]. In these propositions, the internal state of the FCSR is filtered by a linear function to provide from 1 to 16 output bits at each iteration. At the present moment, the F-FCSR-H and F-FCSR-16 are selected for the third and last phase of the European Project eSTREAM for the Profile 2 (i.e. hardware profile) [28].

While F-FCSR stream ciphers have good performances in hardware, they are extremely slow in software since they requires many bit manipulation instructions to output only a few bits. In this paper, we propose an efficient way to design a fast stream cipher by generating many keystream bits from the same internal state. Our design is based on two 256-bit FCSRs and on a mechanism of extraction with a 16×256 bits or 16×128 bits memory. The input of the extraction function is the bitwise Xor of the full contents of the two FCSRs main registers. This function and the IV setup use some classical and well-known techniques of block cipher design such as the "ShiftRows" and the "MixColumns" operations of the AES [14, 16].

In this paper, we present two versions of our stream cipher. The first one X-FCSR-256 outputs 256 bits at each iteration. This version is probably riskier but is the most efficient (6.5 cycles/byte). It can be considered as a challenge for cryptanalysts. The second one X-FCSR-128 outputs 128 bits at each iteration. It seems more robust, and have also good performances (8.2 cycles/byte).

Section 1 presents the background on FCSRs and their most useful properties for cryptographic use. The stream ciphers X-FCSR-256 and X-FCSR-128 are described in Section 2. We present an analysis of security in Section 3. We give in conclusion detailed results on the performance of our designs.

1 Background on FCSRs and 2-adic sequences

1.1 The FCSR automaton

The Feedback with Carry Shift Registers (FCSRs) were first introduced by A. Klapper and M. Goresky in [23] (see also [24, 17, 1, 4]). In this section, we only recall the main results needed to understand the principle of a FCSR automaton.

Let p and q, q odd be two integers. The 2-adic rational number p/q is the formal power series $\sum_{i=0}^{\infty} a_i 2^i$ such that $p = q \times \sum_{i=0}^{\infty} a_i 2^i$. This series can be computed by performing the division of p by q following the increasing powers of 2.

In the sequel, we suppose that q is a negative prime and p satisfies $0 \leq p < |q|$. In that situation the sequence $S = (a_i)_{i \in \mathbb{N}}$ is periodic with period T where T is the order of 2 modulo q. This period is maximal if

$T = |q| - 1$. In that case the sequence S is called a ℓ-sequence. We define an optimal FCSR as an FCSR generating ℓ-sequences.

We suppose that the size of q is $n+1$, i.e. $2^n < -q < 2^{n+1}$. Let $d = (1-q)/2$ and $d = \sum_{i=0}^{n-1} d_i 2^i$, $d_i \in \{0,1\}$, $d_{n-1} = 1$. For a fixed q (the so-called connection integer of the automaton), the 2-adic rational number p/q can be computed by a FCSR for any p, $0 \leq p < |q|$. This automaton is composed of two registers (sets of cells): a main register M and a carry register C.

The main register M contains n binary cells where each bit is denoted by $m_i(t)$ $(0 \leq i \leq n-1)$. We call the integer $m(t) = \sum_{i=0}^{n-1} m_i(t) 2^i$ the content of M.

The carry register contains ℓ cells where $\ell+1$ is the number of nonzero d_i digits, i.e. the Hamming weight of d. More precisely, the carry register contains one cell for each nonzero d_i with $0 \leq i \leq n-2$. We denote $c_i(t)$ the binary digit contained in this cell. We put $c_i(t) = 0$ when $d_i = 0$ or when $i = n-1$. We call the integer $c(t) = \sum_{i=0}^{n-2} c_i(t) 2^i$ the content of C. The Hamming weight of the binary expansion of $c(t)$ is at most ℓ. Note that, if $d_i = 0$, then $c_i(t) = 0$ for all t. At cell level, the transition function of an FCSR is given by:

$$\begin{aligned} m_i(t+1) &= m_{i+1}(t) \oplus d_i c_i(t) \oplus d_i m_0(t) \\ c_i(t+1) &= d_i \left(m_{i+1}(t) c_i(t) \oplus c_i(t) m_0(t) \oplus m_0(t) m_{i+1}(t) \right) \end{aligned}$$

where $\oplus$ denotes bitwise Xor.

Note that $m_0(t)$ is the least significant bit of $m(t)$ and represents the feedback bit. The integers $m(t)$, $c(t)$ and d are integers of bit-size n (or less).

If at time $t = t_0$ the automaton is in the state (m, c), i.e. $m = m(t_0)$ and $c = c(t_0)$, then the observed sequence in the cell $(m_0(t_0+i))_{i \in \mathbb{N}}$ is the 2-adic expansion of p/q, with $p = m + 2c$.

Moreover, for every cell of the main register, there exists an integer $p^{(j)}$ such that the observed sequence $(m_j(t_0+i))_{i \in \mathbb{N}}$ is the 2-adic expansion of $p^{(j)}/q$. This number $p^{(j)}$ can be explicitly computed from the knowledge of m and c. The reader can refer to [4, 5] for more details.

1.2 Loss of entropy and TMD attacks against FCSR

The knowledge of entropy evolution when applying the transition function of a stream cipher is very important as a loss of entropy could be seen as a potential weakness: such a loss allows to improve time/memory/data attacks [11, 19].

Usually, except in some particular cases such as LFSRs of length n (where the entropy is maximal and is equal to $\log(2^n - 1)$), the entropy is not always known and computable. In the FCSR case, we have the following property [3]:

Proposition 1. *If the order of* $2 \bmod q$ *is* $T = |q| - 1$*, the size of the final cycle of a component of the transition function graph is exactly* T *(except for two degenerated cases:* $m + 2c = 0$ *or* $m + 2c = -q$*).*

Moreover, as soon as two initial states (m, c) and (m', c') verify $m + 2c = m' + 2c'$, the automaton converges in less than $n + 4$ iterations to the same state of the final cycle. To avoid any problems and prevent any loss of entropy, a good method of initialization is to put $c = 0$ and take a random m from the set $[1, \cdots, 2^n - 1]$.

Using this procedure, two distinct initializations cannot converge to the same state after a same number of iterations. The entropy of the FCSR state remains equal to $\log(2^n - 1)$ after any number of iterations.

1.3 Distinguishing attacks using diffusion of differences

Let us consider two initial states (m, c) and (m', c') which differ on only one bit in the cell number i (of the main register or of the carry register). Then, after k transitions, there will always a difference at position $i - k$, while a difference may subsist at positions $i-n+1$, $i-n+2$, $i-n+j$ for j small. However, if any feedback bit is affected (i.e. k is larger than i) then the whole state change but some statistical biases may remain if the chosen initial values of m and m' or of c and c' have a very low Hamming weight located on few concomitant cells.

If the setup of the initial value (IV) is not carefully designed, this property can be exploited to mount distinguishing attacks or resynchronization attacks (cf. [21, 22]).

To avoid this problem, we need to design an IV setup which unables any attacker to master a difference between two internal states by choosing some pairs of IV. Such a procedure can be obtained by essentially two ways:

- The first one consists in performing more than $n + 4$ iterations before to output any value [3]. Unfortunately this solution is slow for software applications.
- The second solution consists to digest the key and the IV using an efficient function such as an hash function or a block cipher. In our design, we have chosen to use an AES-like block cipher: some subkeys

are derived from the secret master key and the IV is the plaintext of the block cipher. The encrypted message is then used to initialize the main registers of the FCSRs.

1.4 Algebraic attacks on FCSR automata

Roughly speaking, an algebraic attack consists in generating and solving a system of non-linear equations which relates the secret key or the values of the internal state to known values [12, 13]. In the case of additive stream ciphers, these known values are the output keystream bits at each iteration.

Let us consider the example of a filtered LFSR of size n. We denote by $X(t) = (x_1(t), \ldots, x_n(t)) \in GF(2)^n$ the value of the internal state of the LFSR automaton at time t. Let L be the transition function from $GF(2)^n$ into itself. Since the automaton is an LFSR the function L is linear. By definition, we have $X(t+1) = L(X(t))$. Suppose that at each iteration, we extract k bits by a function f from $GF(2)^n$ into $GF(2)^k$. The known output at time t is then $S(t) = f(X(t)) \in GF(2)^k$. Set $X = (x_1, \ldots, x_n) = X(t_0)$. For any r iterations, the unknowns x_i are solutions of the system of $k \times r$ equations $f(L^j(X)) = S(t_0 + j)$ for $0 \leq j < r$. The degree and the difficulty to solve such a system depends only on the function f.

For a filtered FCSR, the transition function Q is no more linear but quadratic. The system becomes $f(Q^j(X)) = S(t_0 + j)$ for $0 \leq j < r$. In this case, the complexity of the problem depends essentially on the function Q^j and the function f could be linear as done in the F-FCSR stream ciphers.

In [9], the authors studied this problem for the F-FCSRs. In particular, they gave some experimental values for the degree of the equations for a register of length 128 bits. After 7 initial iterations, the observed degree of the equation is $d = 10$ and the number of monomials is 7148. Moreover, in the same article, the authors precise that they are unable to compute algebraic equations beyond this bound due to the large degree of such equations implying a huge number of monomials. Clearly, as soon as r is sufficiently large, the degree of the equations grows up to n and the number of monomials grows exponentially. The difficulty is then to compute the corresponding system. The limit seems close to 10 iterations, even for FCSRs of small sizes (typically 64 or 128).

1.5 More properties of xored 2-adic sequences

In [17], M. Goresky and A. Klapper studied the general behavior of a sequence of the form: $s = \frac{p}{q} \oplus \frac{p'}{q'}$. They gave some results concerning the properties of the bitwise or of two maximum period FCSR sequences. We apply those results for the integers q_a and q_b chosen here and given in Appendix A. If we define $s = \frac{p_a}{q_a} \oplus \frac{p_b}{q_b}$, $0 < p_a < |q_a|$ and $0 < p_b < |q_b|$ and $S = (s_i)$ with $s = \sum_{i=0}^{\infty} s_i 2^i$, then we have the following properties:

- the period of S is $T = (|q_a| - 1)(|q_b| - 1)/2$.
- Fix $k \geq 0$. Let $Q = |q_a| - 1 \bmod 2^k$ and $R = |q_b| - 1 \bmod 2^k$. Define $w = (\min(Q, R) - \max(0, Q + R - 2k))/2$. The number of occurrences of a block $e = (e_0, e_1, \cdots, e_{k-1})$ of size k in the sequence S varies at most by w as the block e varies over all 2^k possibilities.
- As a corollary, the sequence S is balanced and the distribution of consecutive pairs in S is uniform.

Until now, there is no known method more efficient than the exhaustive search to recover p/q and p'/q' from the knowledge of s.

2 Design of X-FCSR-128 and X-FCSR-256

In this section, we propose two new stream ciphers inspired by the construction of the summation generator proposed in [31] and in [25] on the 2-adic numbers. Our proposal is dedicated to the 2-adic case and is a tweaked modified version of this generator to improve the general security of the scheme. So, X-FCSR is a new binary additive synchronous stream cipher design: the keystream Z is added bitwise to the plaintext P to produce the ciphertext C.

The two proposals, X-FCSR-128 and X-FCSR-256, differ on the extraction function. For X-FCSR-128, 128 bits of keystream are generated at each iteration, while the extraction of X-FCSR-256 produces 256 bits of keystream. Clearly, the later version is riskier since it outputs 256 bits - the size of one FCSR main register - and can be viewed as a challenge for cryptanalysts.

Both stream ciphers admit a secret key K of 128-bit length and a public initialization vector IV of bitlength ranging from 64 to 128 as input. These parameters conform to the requirements given in the eSTREAM initial call for Stream ciphers for software applications [28].

2.1 General overview and parameters of the FCSR automata

The core of the design is constituted of two optimal 256 bit FCSRs: The first one with the connection integer q_a is right-clocked whereas the second one is left-clocked with connection integer q_b. q_a and q_b, given in Appendix A, are primes and produces ℓ-sequences, i.e. the corresponding FCSRs are optimal.

At time t, we denote by $M_a(t)$ and $M_b(t)$ the contents of the two main registers. At each iteration, the value $X(t) = M_a(t) \oplus M_b(t)$ feeds an extraction function with memory which computes the output from the current value and from the value obtained at time $t - 16$. As noticed in Appendix A, the connection integers have been chosen such that, if we denote $X(t)$ by $(x_0(t), \ldots, x_{255}(t))$ at bit level, then at least one of the two FCSRs has a carry bit between $x_i(t)$ and $x_{i+1}(t)$ for each i, $0 \leq i < 255$.

2.2 Extraction function of X-FCSR-128

The extraction function is constituted of a function $Round_{128}$ working on 128-bit input/output words, and a memory of 16 128-bit words which stores the output of $Round_{128}$ that will be used 16 iterations later. More formally, the full extraction function holds as follows:

- compute the 128-bit word $Y(t) = X^{(0)}(t) \oplus X^{(1)}(t)$,
  with $X(t) = \left(X^{(0)}(t)||X^{(1)}(t)\right)$, where $||$ denotes the concatenation.
- Compute $Z(t) = Round_{128}(Y(t))$.
- Store $Z(t)$ in memory (keep it during 16 iterations).
- Output the 128-bit word $Output_{128}(t) = Y(t) \oplus Z(t - 16)$.

$Round_{128}$ is a one-round function from $\{0,1\}^{128}$ into itself that could be written as $Round_{128}(a) = Mix_{128}(SR_{128}(SL_{128}(a)))$. If the 128-bit word a is represented at byte level by a 4×4 matrix M where each byte is represented by the word $a_{i,j}$ with $0 \leq i, j \leq 3$, then the function $Round_{128}$ could be described as follows:

- the first transformation $SL_{128}()$ consists in a S-box layer applied at byte level that transform each byte $a_{i,j}$ into an other byte $b_{i,j}$ such as $b_{i,j} = S(a_{i,j})$ where S is the S-box given in Appendix B chosen for its good properties (the differential and linear probabilities are low, the algebraic degree is equal to 7, the nonlinear order is equal to 6, the I/O-degree is equal to three).
- The $SR_{128}()$ operation corresponds with the ShiftRows() operation of the AES described in [16] and consists in Shifting each row of the current matrix on the left at byte level: by 0 for the first row, by one for the second, by two for the third and by three for the fourth one.

- the $Mix_{128}()$ operation is the one used in [18] computed using the operations over $GF(2)$. More precisely for each column of a, we compute $\forall j, 0 \leq j \leq 3$:

$$Mix_{128}\begin{pmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{pmatrix} = \begin{pmatrix} a_{3,j} \oplus a_{0,j} \oplus a_{1,j} \\ a_{0,j} \oplus a_{1,j} \oplus a_{2,j} \\ a_{1,j} \oplus a_{2,j} \oplus a_{3,j} \\ a_{2,j} \oplus a_{3,j} \oplus a_{0,j} \end{pmatrix}.$$

  Even if this function is not fully optimal for a diffusion purpose, its branch number is however equal to 4 and its computation is significantly faster than the MixColumns of the AES: Mix_{128} can be computed with only 6 32-bit bitwise Xors.

2.3 Extraction function of X-FCSR-256

For X-FCSR-256, the extraction function works directly on the 256-bit word $X(t)$ with the function $Round_{256}$ from $\{0,1\}^{256}$ into itself, and a memory of 16 256-bit words which stores the output of $Round_{256}$ that will be used 16 iterations later. More formally, the full extraction function holds as follows:

- Compute the 256-bit word $W(t) = Round_{256}(X(t))$.
- Store $W(t)$ in memory (keep it during 16 iterations).
- Output the 256-bit word $Output_{256}(t) = X(t) \oplus W(t-16)$.

$Round_{256}$ is the same round function than the previous one but from $\{0,1\}^{256}$ into itself that could be written as
$Round_{256}(a) = Mix_{256}(SR_{256}(SL_{256}(a)))$. If the 256-bit word a is represented at byte level by a 4×8 matrix M where each byte could be written $a_{i,j}$ with $0 \leq i \leq 3$ and $0 \leq j \leq 7$, then the function $Round_{256}$ could be described as follow:

- the first transformation $SL_{256}()$ consists in the same S-box layer applied at byte level that transforms each byte $a_{i,j}$ into an other byte $b_{i,j}$ such as $b_{i,j} = S(a_{i,j})$ using always the S-box S.
- The $SR_{256}()$ operation corresponds with the ShiftRows() operation of Rijndael described in [14] and consists in Shifting each row of the current matrix on the left at byte level: by 0 for the first row, by one for the second, by three for the third and by four for the fourth one.
- the $Mix_{256}()$ operation is similar to $Mix_{128}()$ but there are here 8 columns to consider.

2.4 Key and IV injection

As done in [7], we have split the initialization process into two steps to speed up the IV injection:

- The key schedule, which processes the secret key but does not depend on the IV.
- The IV injection, which uses the output of the key schedule and the IV.

This initializes the stream cipher internal state. Then, the IV setup for a fixed key is less expensive than a complete key setup, improving the common design since changing the IV is more frequent than changing the secret key.

Key schedule The key setup process used here corresponds to a classical key schedule of a block cipher and is inspired by the one of the DES [27] due to its good resistance against related key attacks [10] and against related key rectangle attacks [20]. The key expansion produces 25 128-bit subkeys denoted $K_0, \cdots, K_{24}$. It works as follow:

- the subkey K_0 is deduced from the master key: $K_0 = (Round_{128}(K))_{<<<23}$ where ${<<<_j}$ denotes a 128-bit left rotation of j positions.
- then K_i is deduced from K_{i-1}: $K_i = Round_{128}((K_{i-1})_{<<<j})$ where $j = 23$ if $i \equiv 3 \bmod 4$ and $j = 11$ otherwise.

IV injection If necessary the IV is extended to a 128-bit word by adding leading zeros. Then, this value is considered as a plaintext that is enciphered first 12 times using the $Round_{128}$ function followed by an X-or at byte level with the subkey of the round K_j. More precisely, the process is the following if we denote by V_i the ciphertext after the round i:

- $V_0 = IV \oplus K_0$
- for i from 1 to 24 do $V_i = Round_{128}(V_{i-1}) \oplus K_i$

Then, the values V_{12}, V_{16} V_{20} and V_{24} are used to initialize the main registers as follows: $M_a(0) = (V_{12}||V_{20})$ and $M_b(0) = (V_{16}||V_{24})$ whereas the carry registers C_a and C_b are initialized to zero. The two FCSRs are then clocked 16 times to fill the sixteen memory registers of the extraction function. Note that for X-FCSR-256, the registers are directly filled with the 256-bit value $W(t_0) = Round_{256}(M_a(t_0) \oplus M_b(t_0))$ whereas for X-FCSR-128 this value is folded at 128-bit level using the 128-bit word $Y(t_0)$.

3 Design rationale and security analysis

Objectives of key and IV injection In [8], the authors demonstrate that to be secure the key and IV setup (parametrized by the key K) of an IV-dependent stream cipher must be a pseudo-random function. We have tried to achieve this goal designing our key and IV setup as a block cipher using the round function $Round_{128}$. It is also important to note that, under those conditions, the secret key of the cipher cannot be easily recovered from the initial state of the generator. Once the initial state is recovered, the attacker is only able to generate the output sequence for a particular key and a given IV.

This particular mechanism already used in [7] also prevents our stream cipher from the distinguishing differential attacks described in Section 1.3: the values of the two main registers are key dependent and, for a given secret key, no difference could be mastered with a sufficient probability between two distinct IV values and the contents of the main registers. Moreover, and as explained in Section 1.2, the two carry registers are initialized to 0 to avoid any loss of entropy and prevent TMD attacks.

Role of the core FCSR automata As noticed in [8], to be secure the keystream generation of an IV-dependent stream cipher must rely on a pseudo-random number generator. Following this requirement and the results of Section 1.5, we have based our stream cipher on the Xor of two independent 2-adic sequences that provides performing pseudo-random sequences. As explained in [4], any cell of the main register of a FCSR automaton provides a 2-adic sequence. Except if there is no feedback bits between two cells, the theoretical dependencies between these sequences cannot be exploited easily. The main idea of X-FCSR is to directly Xor the contents of two distinct FCSRs of size 256 to provide in parallel 256 xored 2-adic sequences, denoted at time t by $X(t) = (x_0(t), \ldots, x_{255}(t))$ in § 2.1.

However, the content of $X(t)$ cannot be directly outputted because in this case we obtain a system of 256 equations with 512 variables at time t that could be easily solved: the condition "q_a and q_b have been chosen such that there is always at least a feedback bit between two consecutive xored cells x_i and x_{i+1} (as explained in Appendix A)" is necessary but not sufficient.

Moreover, we could also build a guess and determine attack using the knowledge of $X(t)$: first choose an indice i such that there is no carry between $m_{a\,i}$ and $m_{a\,i+1}$, and between $m_{a\,i+1}$ and $m_{a\,i+2}$. So there are carries between $m_{b\,i+1}$ and $m_{b\,i}$ and between $m_{b\,i+2}$ and $m_{b\,i+1}$. Then, guess the contents of the 9 cells $m_{a\,i}$, $m_{a\,i+1}$, $m_{a\,i+2}$, $m_{b\,i}$, $m_{b\,i+1}$, $m_{b\,i+2}$, $c_{b\,i+1}$, $c_{b\,i+2}$ and $m_{b\,255}$ at time t. Using the transition formula, derive the corresponding values at time $t+1$ from the known outputs $x_i(t)$, $x_{i+1}(t)$ and $x_{i+2}(t)$. Repeating this process, we then obtain the consecutive values of the feedback bit $m_{b\,255}$ to deduce the content of the first register and we then could guess the cell values of the second register. In the case of X-FCSR-128 (cf. § 2.2), the 128-bit output $Y(t)$ computed from $X(t)$ with the formula $y_i(t) = x_i(t) \oplus x_{i+128}(t)$ prevents this attack from holding even if the information provided by $Y(t)$ seem to be too strong to directly output this value.

An other constraint must be respected: the two registers must be clocked in opposite way because if the two automata are both right-clocked (for example), the values $x_i(t)$ and $x_{i+1}(t+1)$ are correlated according the values of the two feedback bits.

Role of the extraction function The two round functions $Round_{128}$ and $Round_{256}$ have been chosen for their good diffusion and non-linear properties: they ensure a good resistance against the residual correlations present between the bits of the two main registers of the FCSR automata. Their use also prevent attacks that is derived from the ones previously described.

The use of 16 memory registers is a good compromise between a better security and a limited performance cost. First, it increases the number of unknown variables depending on the cells of the main register from 2×256 to 16×256 for X-FCSR-256 (or to 16×128 for X-FCSR-128): solving such a system becomes more expensive than the exhaustive key search. This memory could also be seen as four FCSR automata, since at each operation, the output depends on $X(t)$ and $X(t-16)$. Even if there exists dependencies between $X(t)$ and $X(t-16)$, it is computationally infeasible to determine the values of the main registers at time $t+16$ from the values at time t using the transition functions as noticed in Section 1.4.

Resistance against known attacks The good statistical properties (period, balanced sequences and so on) of our constructions are provided by the xored 2-adic properties. We experimentally verified some of them by applying the NIST statistical test suite [29] with success to our two constructions.

As previously mentioned, differential distinguishing attacks, algebraic attacks, Time/Memory/Data trade off attacks and guess and determine attacks are discarded due to the properties described in Section 1 and due to the previous remarks.

Then, we focus on correlation and fast correlation attacks. In those attacks, the cryptanalyst tries to exploit an existing correlation between some internal bits of the automaton and some output bits using in general linear relations [32, 26]. Since the FCSR description has appeared in [4] two years ago, no correlation attack have been exhibited against this new construction. This is essentially due to the non-linearity induced by the carries propagation. Even if there exists some 2-adic correlations in a FCSR, those non-linear relations are destroyed by the action of the xor. Specifically, in the case of X-FCSR, the residual correlations between the neighbor cells of the main registers are stopped by the use of the *Round* functions.

To sum up all the previous analyses, we think that traditional attacks against stream cipher that exploit linear relations built upon the transition function are not realistic in our case. Thus, wanting to cryptanalyse FCSRs leads to create new attacks exploiting other sorts of relations.

4 Conclusion

We have integrated the X-FCSR-128 and X-FCSR-256 stream ciphers to the eSTREAM benchmarking suite [15]. We run the benchmark on an Opteron 250 (1.4Ghz) with GCC 4.1.1 (*-O3-funroll-all-loops -fomit-frame-pointer*) and gather the results in Table 1.

	cycles/byte				cycles/key	cycles/IV
Algorithm	Keystream speed	40 bytes	576 bytes	1500 bytes	Key setup	IV setup
X-FCSR-256	6.5	50	9.5	7.6	1093	1636
X-FCSR-128	8.21	51	11	9.3	1096	1651
AES-CTR	18.23	22.98	18.3	18.3	172.23	11.74

Table 1. X-FCSR performance on an Opteron with the eSTREAM benchmark suite.

Our new design X-FCSR-128 and X-FCSR-256 are significantly faster than the AES-CTR except for small plaintexts. The performance of the two proposed stream ciphers are promising. We hope that we have shown how to use FCSRs for software applications. FCSRs have many other advantages such as a simple software implementation or proved properties. In such a design, two important parameters have to be considered: the size of the FCSR automaton, and the size of the output of the extraction function. They have opposite impacts on the throughput and on the security, thus a compromise has to be found. The two stream ciphers presented here correspond to some choice for these parameters. However, the problem of the optimal choice for them remains open.

References

1. F. Arnault and T.P. Berger. F-FCSR: design of a new class of stream ciphers. In *Fast Software Encryption - FSE 2005*, v. 3557 of *Lecture Notes in Computer Science*, p. 83–97. Springer-Verlag, 2005.
2. F. Arnault, T.P. Berger, and C. Lauradoux. The FCSR: primitive specification and supporting documentation. ECRYPT - Network of Excellence in Cryptology, Call for stream Cipher Primitives 2005. `http://www.ecrypt.eu.org/stream/`.
3. F. Arnault, T.P. Berger, and C. Lauradoux. Update on F-FCSR stream cipher. ECRYPT - Network of Excellence in Cryptology, Call for stream Cipher Primitives - Phase 2 2006. `http://www.ecrypt.eu.org/stream/`.
4. F. Arnault and T. P. Berger. Design and properties of a new pseudorandom generator based on a filtered FCSR automaton. *IEEE Trans. Computers*, 54(11):1374–1383, 2005.
5. F. Arnault, T.P. Berger and M. Minier. On the security of FCSR-based pseudorandom generators. In *ECRYPT Network of Excellence - SASC Workshop*, 2007. Available at `http://sasc.crypto.rub.de/files/sasc2007_179.pdf`.
6. G. Ars and J.-C. Faugère. An algebraic cryptanalysis of nonlinear filter generators using Gröbner bases. Research Report INRIA Lorraine, number 4739, 2003.
7. C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert. Sosemanuk: a fast oriented software-oriented stream cipher. ECRYPT - Network of Excellence in Cryptology, Call for stream Cipher Primitives - Phase 2 2005. `http://www.ecrypt.eu.org/stream/`.
8. C. Berbain and H. Gilbert. On the security of IV dependent stream ciphers. In *Fast Software Encryption - FSE 2007*, v. 4593 of *Lecture Notes in Computer Science*, to appear. Springer-Verlag, 2007.
9. T.P. Berger and M. Minier. Two algebraic attacks against the F-FCSRs using the iv mode. In *INDOCRYPT 2005*, v. 3797 of *Lecture Notes in Computer Science*, p. 143–154. Springer-Verlag, 2005.
10. E. Biham. New types of cryptoanalytic attacks using related keys (extended abstract). In *EUROCRYPT 93*, v. 765 of *Lecture Notes in Computer Science*, p. 398–409. Springer-Verlag, 1994.
11. A. Biryukov and A. Shamir. Cryptanalytic Time-Memory-Data Trade-Offs for Stream Ciphers. In *Advances in Cryptology - ASIACRYPT 2000*, v. 1976 of *Lecture Notes in Computer Science*, p. 1–14. Springer-Verlag, 2000.
12. N. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology - CRYPTO 2003*, v. 2729 of *Lecture Notes in Computer Science*, p. 177–194. Springer-Verlag, 2003.
13. N. Courtois and W. Meier. Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology - EUROCRYPT 2003*, v. 2656 of *Lecture Notes in Computer Science*, p. 345–359. Springer-Verlag, 2003.

14. J. Daemen and V. Rijmen. AES proposal: Rijndael. In *The Second Advanced Encryption Standard Candidate Conference*. N.I.S.T., 1999. available at `http://csrc.nist.gov/encryption/aes/`.
15. C. de Cannières. eSTREAM Optimized Code HOWTO, 2005. `http://www.ecrypt.eu.org/stream/perf`.
16. FIPS 197. Advanced Encryption Standard. Federal Information Processing Standards Publication 197, 2001. U.S. Department of Commerce/N.I.S.T.
17. M. Goresky and A. Klapper. Periodicity and distribution properties of combined FCSR sequences. In Guang Gong, Tor Helleseth, Hong-Yeop Song, and Kyeongcheol Yang, editors, *SETA*, v. 4086 of *Lecture Notes in Computer Science*, p. 334–341. Springer, 2006.
18. L. Granboulan, E. Levieil, and G. Piret. Pseudorandom permutation families over abelian groups. In Matthew J. B. Robshaw, editor, *FSE*, v. 4047 of *Lecture Notes in Computer Science*, p. 57–77. Springer, 2006.
19. J. Hong and W.-H. Kim. Tmd-tradeoff and state entropy loss considerations of streamcipher mickey. In Subhamoy Maitra, C. E. Veni Madhavan, and Ramarathnam Venkatesan, editors, *INDOCRYPT*, v. 3797 of *Lecture Notes in Computer Science*, p. 169–182. Springer, 2005.
20. S. Hong, J. Kim, S. Lee, and B. Preneel. Related-key rectangle attacks on reduced versions of shacal-1 and aes-192. In Henri Gilbert and Helena Handschuh, editors, *FSE*, v. 3557 of *Lecture Notes in Computer Science*, p. 368–383. Springer, 2005.
21. E. Jaulmes and F. Muller. Cryptanalysis of ecrypt candidates F-FCSR-8 and F-FCSR-H. ECRYPT Stream Cipher Project Report 2005/046, 2005. `http://www.ecrypt.eu.org/stream`.
22. E. Jaulmes and F. Muller. Cryptanalysis of the F-FCSR stream cipher family. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, v. 3897 of *Lecture Notes in Computer Science*, p. 20–35. Springer, 2005.
23. A. Klapper and M. Goresky. 2-adic shift registers. In *Fast Software Encryption - FSE'93*, v. 809 of *Lecture Notes in Computer Science*, p. 174–178. Springer-Verlag, 1993.
24. A. Klapper and M. Goresky. Feedback Shift Registers, 2-Adic Span, and Combiners with Memory, In *J. Cryptol.*, vol. 10(2), pp. 111-147. 1997.
25. J. L. Massey and R. A. Rueppel. Method of, and apparatus for, transforming a digital sequence into an encoded form. U.S. Patent No. 4,797,922, 1989.
26. W. Meier and O. Staffelbach. Fast correlation attack on certain stream ciphers. In *J. Cryptol.*, vol. 1(3), pp. 159-176, 1989.
27. National Bureau of Standards (USA). Data Encryption Standard. *Federal Information Processing Standard*, 1977. Publication 46.
28. Network of Excellence in Cryptology ECRYPT. Call for stream cipher primitives. `http://www.ecrypt.eu.org/stream/`.
29. National Institute of Standards and Technology. the statistical test suite (v.1.8), 2005. `http://csrc.nist.gov/rng/rng2.html`.
30. R. Rivest. The RC4 encryption algorithm. RSA Data Security, 1992.
31. R. A. Rueppel. Correlation immunity and the summation generator. In Hugh C. Williams, editor, *CRYPTO*, v. 218 of *Lecture Notes in Computer Science*, p. 260–272. Springer, 1985.
32. T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Computers*, C-34(1):81–84, 1985.

A The connection integers

The X-FCSR primitives are composed of two FCSRs of bitlength 256 with the following connection integers:

1. The value of q_a is

   $$-23158373676191642998087032666622460867207843241572527691478170790314536991 7947$$

2. The value of q_b is

   $$-17187700518600281458145539366740823721204558315634632365649000473737223260 1307$$

3. The corresponding value of d_a is (in hexadecimal notation):

   ffffdfff fb7d9f7f dfefd8ef dfbef7fe 6bfebf9f fffeeffd fecb9def ed3dedfe

4. The corresponding value of d_b is (in hexadecimal notation):

   bdff77ff fcffbdf7 efdfdfaf ff53d9ff fdfebbfc faffffdf 47d6d7ff 7fbfe76e

5. The binary Hamming weight of d_a and d_b is 210.
6. The first FCSR is right-clocked whereas the second one is left-clocked.
7. More precisely, the first one is updated with an Hamming weight equal to 210 using the notations of Section 1 and of Section 2.1 by the formulas written at registers level:

   $$\begin{aligned} M_a(t+1) &= (M_a(t))_{<<1} \oplus C_a(t) \oplus m_{a\,0}(t) d_a \\ C_a(t+1) &= (M_a(t))_{<<1} \otimes C_a(t) \oplus C_a(t) \otimes m_{a\,0}(t) d_a \\ &\quad \oplus m_{a\,0}(t) d_a \otimes (M_a(t))_{<<1} \end{aligned}$$

   where $\oplus$ denotes the bitwise Xor, $\otimes$ denotes bitwise AND, and $<< 1$ is a simple shift to the left. Note also that $m_{a\,0}(t)$ is the least significant bit of $M_a(t)$ and represents the feedback bit.
8. The second FCSR is updated on the right using the formulas:

   $$\begin{aligned} M_b(t+1) &= (M_b(t))_{>>1} \oplus C_b(t) \oplus m_{b\,255}(t) d'_b \\ C_b(t+1) &= (M_b(t))_{>>1} \otimes C_b(t) \oplus C_b(t) \otimes m_{b\,255}(t) d'_b \\ &\quad \oplus m_{b\,255}(t) d'_b \otimes (M_b(t))_{>>1} \end{aligned}$$

   where d'_b is the reverse of d_b in binary expansion: $d'_b = \sum_{i=0}^{255} d_{b\,255-i} 2^i$ and where $m_{b\,255}(t)$ is the most significant bit of $M_b(t)$ and represents the feedback bit.
9. The numbers q_a and q_b have been chosen in such a way that $d_{a\,i}$ and $d_{b\,255-i}$ are not simultaneously equal to 0.

B The S-box S

We have designed our S-box using the requirements and the method defined in [18] except that all the steps are performed here on $GF(2)^8$ into itself. The S-box is given in tab. 2 in hexadecimal notation. It has been chosen to have a good resistance against differential and linear cryptanalyzes, an high algebraic degree, an high nonlinear order and a degree between inputs and outputs equal to three. The S-box was generated using the key schedule algorithm KSA of RC4 ([30]) algorithm initialized with a key of 26 bytes length equal to the string "To design our steam cipher". Then after 48574 iterations of this algorithm, we obtain the following S-box:

52	c3	45	ce	9	cf	a8	f8	fd	ab	b8	6d	95	2	31	8
56	f4	cb	40	61	7	12	39	62	bb	ef	5d	3a	a9	fb	2c
78	ad	75	77	10	ca	55	66	9e	65	7b	9b	13	76	c7	1c
71	d	18	3f	50	6c	28	64	a3	b7	d0	be	e6	9c	b9	94
fc	bc	a1	cd	3b	48	4c	99	cc	3e	79	24	f2	c1	da	d8
de	f	e8	67	2e	16	53	c4	9d	57	c0	4f	f0	d6	4e	81
69	8a	ae	f9	8b	ee	43	3d	e4	23	97	68	b	32	e1	b2
ec	e9	59	1	c2	34	b5	1f	2a	29	d7	d5	b0	96	11	c6
7d	91	2d	72	8f	87	1d	e7	ba	19	25	15	5e	d9	98	70
4a	ed	51	a6	88	86	58	c5	5f	eb	49	0	ff	1b	2f	6a
82	1a	af	9f	8c	6b	a2	f1	e	5	7f	73	92	3c	f5	d2
54	14	ac	83	20	90	c9	22	fa	74	d3	27	37	38	a5	33
85	6	4	b3	e2	5b	e3	47	1e	8d	4b	b1	36	46	bd	35
dc	6e	d1	7c	a7	41	c	42	a0	aa	26	5a	4d	e5	5c	80
21	3	f3	63	ea	44	dd	89	8e	7e	b4	30	a	a4	60	f6
bf	fe	e0	f7	c8	d4	9a	db	84	7a	6f	2b	b6	17	93	df

Table 2. the chosen S-box

The chosen S-box has the following properties:

- The best differential trail is $DP(S) = \mathrm{Max}_{a,b \in (GF(2)^8)^2 \setminus \{0,0\}} \#\{x | S(x \oplus a) \oplus S(x) = b\} = 10$.
- The best linear trail is $LP(S) = \mathrm{Max}_{a,b \in (GF(2)^8)^2 \setminus \{0,0\}} |\#\{x | a \cdot S(x) = b \cdot x\} - 128| = 32$.
- The algebraic degree is equal to 7.
- The non-linear order is equal to 6.
- The degree between inputs and outputs is equal to three. There is no equation of degree two.