F-FCSR: design of a new class of stream ciphers

François Arnault and Thierry P. Berger

arnault@unilim.fr thierry.berger@unilim.fr
LACO, Université de Limoges, 123 avenue A. Thomas,
87060 Limoges CEDEX, France

Abstract

In this paper we present a new class of stream ciphers based on a very simple mechanism. The heart of our method is a Feedback with Carry Shift Registers (FCSR) automaton. This automaton is very similar to the classical LFSR generators, except the fact that it performs operations with carries. Its properties are well mastered: proved period, non-degenerated states, good statistical properties, high non-linearity.

The only problem to use such an automaton directly is the fact that the mathematical structure (2-adic fraction) can be retrieved from few bits of its output using an analog of the Berlekamp-Massey algorithm.

To mask this structure, we propose to use a filter on the cells of the FCSR automaton. Due to the high non-linearity of this automaton, the best filter is simply a linear filter, that is a XOR on some internal states. We call such a generator a Filtered FCSR (F-FCSR) generator.

We propose four versions of our generator: the first uses a static filter with a single output at each iteration of the generator (F-FCSR-SF1). A second with an 8 bit output (F-FCSR-SF8). The third and the fourth are similar, but use a dynamic filter depending on the key (F-FCSR-DF1 and F-FCSR-DF8). We give limitations on the use of the static filter versions, in scope of the time/memory/data tradeoff attack.

These stream ciphers are very fast and efficient, especially for hardware implementations.

Keywords: stream cipher, pseudorandom generator, feedback with carry shift register, 2-adic fractions.

1 Introduction

Linear Feedback Shift Registers (LFSR) are the most popular tool used to design fast pseudorandom generators. Their properties are well known, among them the fact that the structure of an LFSR can be easily recovered from his output by the Berlekamp-Massey algorithm. Many methods have been used to thwart the Berlekamp-Massey attack because the high speed and simplicity of LFSRs are important benefits.

Feedback with Carry Shift Registers (FCSR) were introduced by M. Goresky and A. Klapper in [7]. They are very similar to classical Linear Feedback Shift Registers (LFSR) used in many pseudorandom generators. The main difference is the fact that the elementary additions are not additions modulo 2 but with propagation of carries. This generator is almost as simple and as fast as a LFSR generator. The mathematical model for FCSR is the one of rational 2-adic numbers (cf. [9, 10]). This model leads to proved results on period and non degeneration of internal states of the generator. It inherits the good statistical properties of LFSR sequences.

Unfortunately, as for the LFSR case, it is possible to recover the structure of a sequence generated by an FCSR (cf. [8, 2],[1]). To avoid this problem, we propose to use a filter on the cells of the FCSR automaton. Since this automaton has good non linear properties, the filter is simply a linear function, i.e. a XOR on some cells. This method is very efficient for practical implementations.

First we describe the FCSR automaton and recall the properties of its output. For applications, we propose an automaton with a key of 128 bits in the main register.

Then we present the different versions of our generator with a detailed security analysis in each case. For the F-FCSR-SF1 version, we show that the algebraic attack is not possible and we describe some dedicated attacks. For the proposed parameters, this attack is more expensive than the exhaustive one. The main restriction to the use of this version is the fact that the cost of the time/memory/data tradeoffs attack is $O(2^{98})$, which is less than the exhaustive attack.

With the F-FCSR-SF8 version, we explain how our automaton can be filtered in order to obtain an 8-bit output at each iteration. The problem on designing a good filter in that situation is discussed. This leads to some problems on its design. This is why we recommend to use the F-FCSR-DF8 version of our generator to perform a 8-bit output system with high level of security.

In the dynamic filter versions of our generator, we substitute to the static filter a dynamic one, i.e. depending on the secret initialization key. This method increases the cost of the time/memory/data tradeoffs attack. This cost becomes $O(2^{162})$ for a 128-bit key. Moreover this dynamic filter avoids all 2-adic and algebraic attacks. In particular for the 8-bit output version, it avoids some attacks on filter combinations. For practical applications, we propose to use the S-box of Rijndael in order to construct the dynamic filter. This method is very efficient, and generally, this box is already implemented.

In the last section, we explain how it is possible to use our generators as stream ciphers with IV mode of size 64 bits. The 128-bit key is used to initialize the main register, and the initial vector is used to initialize the carries register. For some dedicated applications, we also propose to use a key of 96 bits with an IV of 64 bits.

2 The FCSR automaton

We first recall the properties of an FCSR automaton used to construct our pseudorandom generators: an FCSR automaton performs the division of two integers following the increasing powers of 2 in their binary decompositions. This mechanism is directly related to the theory of 2-adic fractions. For more theoretical approach, the reader could refer to [11, 7].

The main results used here are the following:

- Any periodic binary sequence can be expressed as a 2-adic fraction p/q, where q is a negative odd integer and $0 \leq p < |q|$.
- Conversely, if a periodic binary sequence is generated from a 2-adic fraction p/q, then the period of this sequence is known and is exactly the order of 2 modulo q.
- It is easy to choose a prime number q such as the order of 2 is exactly $T = |q| - 1$, and therefore the period generated by any initial value $0 < p < |q|$ is exactly T. So, in the rest of this paper, we suppose that q is such that $2^{128} < |q| < 2^{129}$ and that the condition on the order of 2 is always satisfied in order to guarantee a period greater than 2^{128}.
- If p and q are integers of "small size", i.e. 128 bits for p and 129 bits for q, the sequences p/q looks like random sequences of period T in terms of linear complexity (but it remains false for its 2-adic complexity (i.e. the size of q)).

¿From now, we suppose that the FCSR studied in this section verifies the following conditions: $q < 0 \leq p$, $p < -q$, $p = \sum_{i=0}^{k-1} p_i 2^i$, $q = 1 - 2d$ and $d = \sum_{i=0}^{k-1} d_i 2^i$.

p will be the initial (secret) state of the automaton whereas q will be the equivalent of the "feedback polynomial" of a classical LFSR.

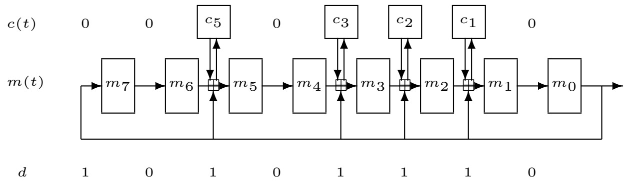
2.1 Modelization of the automaton

If q is defined as above, the FCSR generator with feedback prime q can be described as a circuit containing two registers:

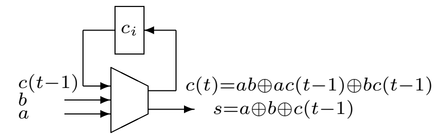
- The main register M with k binary memories (one for each cell), where k is the bitlength of d, that is $2^{k-1} \leq d < 2^k$.
- The carry register $\mathcal{C}$ with ℓ binary memories (one for each cell with a $\boxplus$ at its left) where $\ell + 1$ is the Hamming weight of d. Using the binary expansion $\sum_{i=0}^{k-1} d_i 2^i$ of d, we put $I_d = \{i \mid 0 \leq i \leq k-2 \text{ and } d_i = 1\}$. So $\ell = \#I_d$. We also put $d^* = d - 2^{k-1}$.

We will say that the main register contains the integer $m = \sum_{i=0}^{k-1} m_i 2^i$ when it contains the binary values $(m_0, \ldots, m_{k-1})$. The content m of the main register always satisfies $0 \leq m \leq 2^k - 1$. In order to use similar notation for the carry register, we can think of it as a k bit register where the $k - l$ bits of rank **not** in I_d are always 0. The content $c = \sum_{i \in I_d} c_i 2^i$ of the carry register always satisfies $0 \leq c \leq d^*$.

Example 1 *Let $q = -347$, so $d = 174 =$ 0xAE, $k = 8$ and $\ell = 4$. The following diagram shows these two registers:*



where $\boxplus$ denotes the addition with carry, i.e., it corresponds to the following scheme in hardware:



c_i
$c(t-1)$
b
a
$c(t)=ab\oplus ac(t-1)\oplus bc(t-1)$
$s=a\oplus b\oplus c(t-1)$

Transition function

As described above, the FCSR circuit with feedback prime q is an automaton with 2^{k+l} states corresponding to the $k + l$ binary memories of main and carry registers. We say that the FCSR circuit is in state (m, c) if the main and carry registers contain respectively the binary expansion of m and of c.

Suppose that at time t, the FCSR circuit is in state $(m(t), c(t))$ with $m = \sum_{i=0}^{k-1} m_i(t) 2^i$ and $c = \sum_{i=0}^{k-1} c_i(t) 2^i$. The state $(m(t+1), c(t+1))$ at time $t+1$ is computed using:

- For $0 \leq i \leq k-2$ and $i \notin I_d$
  $m_i(t+1) := m_{i+1}(t)$
- For $0 \leq i \leq k-2$ and $i \in I_d$

$m_i(t+1) := m_{i+1}(t) \oplus c_i(t) \oplus m_0(t)$
$c_i(t+1) := m_{i+1}(t)c_i(t) \oplus c_i(t)m_0(t) \oplus m_0(t)m_{i+1}(t)$

• For the case $i = k-1$
$m_{k-1}(t+1) := m_0(t)$.

Note that this transition function is described with (at most) quadratic boolean functions and that for all three cases $m_i(t+1)$ and $c_i(t+1)$ can be expressed with a single formula:

$$m_i(t+1) := m_{i+1}(t) \oplus d_i c_i(t) \oplus d_i m_0(t)$$

$$c_i(t+1) := m_{i+1}(t)c_i(t) \oplus c_i(t)m_0(t) \oplus m_0(t)m_{i+1}(t)$$

if we put $m_k(t) = 0$ and $c_i(t) = 0$ for i not in I_d.

We now study the sequences of values taken by the binary memories of the main register, that is the sequences $M_i = (m_i(t))_{t\in\mathbb{N}}$, for $0 \le i \le k-1$.

The main result is the following theorem:

Theorem 1 *Consider the FCSR automaton with (negative) feedback prime $q = 1-2d$. Let k be the bitlength of d. Then, for all i such that $0 \le i \le k-1$, there exists an integer p_i such that M_i is the 2-adic expansion of p_i/q. More precisely, these values p_i can be easily computed from the initial states $m_i(0)$ and $c_i(0)$ using the recursive following formulas:*

$$p_i = \begin{cases} qm_i(0) + 2p_{i+1} & \text{if } d_i = 0 \\ q\big(m_i(0) + 2c_i(0)\big) + 2(p_{i+1} + p_0) & \text{if } d_i = 1. \end{cases}$$

If we consider a prime divisor q such that the period is exactly $T = (|q|-1)/2$, the sequences M_i are distinct shifts of a same sequence (e.g. $1/q$), but each shift amount depends on the initial values of the main register and the carry register, and looks like random shifts on a sequence of period T (remember that, for applications $T \simeq 2^{128}$).

2.2 Hardware and software performances of the FCSR

2.2.1 Hardware realization

As we have just seen before, we could directly implement in hardware the structure of an FCSR using a Galois architecture. Even if the needed number of gates is greater, the speed of such a structure is equivalent to the one of an LFSR.

2.2.2 Software aspects

The transition function can also be described in pseudocode with the following global presentation expressing integers $m(t), c(t)$ instead of bits $m_i(t), c_i(t)$ more suitable for software implementations.

If $\oplus$ denotes bitwise addition without carries, $\otimes$ denotes bitwise AND, $shift_+$ the shift of one bit on the right, i.e. $shift_+(m) = \lfloor m(t)/2 \rfloor$ and par is the parity of a number m (1 if m is odd, 0 if it is even):

$$\begin{aligned} m(t+1) &:= shift_+(m(t)) \oplus c(t) \oplus par(m)d \\ c(t+1) &:= shift_+(m(t)) \otimes c(t) \oplus c(t) \otimes par(m)d \oplus par(m)d \otimes shift_+(m(t)) \end{aligned}$$

And the pseudoalgorithm could be written as:

```
$b := par(m)$        (boolean)
$a := shift_+(m)$
$m := a \oplus c$
```

```
$c := a \otimes c$
if $b = 1$ then
    $c := c \oplus (m \otimes d)$
    $m := m \oplus d$
end if
```

The number of cycles needed to implement the FCSR in software seems to be twice greater than the one required for an LFSR but as we will see in the following section, due to the very simplicity of our filtering function, the general speed in software of our Filtering FCSR might be more efficient than usual LFSR based generators.

2.2.3 Parameters of the FCSR automaton for designing the stream ciphers

For a cryptographic use with a security greater than 2^{128}, we recommend the use of a negative retroaction prime $-q$, corresponding to $k = 128$. This implies that $2^{128} < |q| < 2^{129} - 1$.

In order to maximize the period of the generated sequence, the order of 2 modulo q must be maximal i.e. equals to $|q| - 1$. Moreover, to avoid some potential particular cases, we propose to choose a prime q such that $(|q| - 1)/2$ is also a prime.

The FCSR retroaction prime must be public. We propose

$$\begin{aligned} -q &= 493877400643443608888382048200783943827 \qquad (1) \\ &= \mathrm{0x1738D53D56FC4BFAD3D0C6D3430ADD893} \end{aligned}$$

The binary expansion of $d = (|q| + 1)/2$ is
10111001 11000110 10101001 11101010 10110111 11100010 01011111 11010110
10011110 10000110 00110110 10011010 00011000 01010110 11101100 01001010.
Its Hamming weight is 69 and then there are $\ell = 68$ carry cells (the Hamming weight of $d^* = d - 2^{128}$) and $k = 128$ cells in the main register.

3 Design of F-FCSR : Filtered FCSR automaton with a static filter

As for the LFSRs, a binary sequence generated by a single FCSR can not be used directly to produce a pseudorandom sequence (even if the output bits have good statistical properties and high linear complexity), since the initial state and the 2-adic structure can be recovered using a variant of the Berlekamp-Massey algorithm [8, 2]. So, we propose in this section to filter the output of an FCSR with two appropriate static functions and we prove the efficiency and the resistance against known attacks of those two constructions.

3.1 The F-FCSR-SF1 : one output bit

How to filter an FCSR automaton?

For the LFSR case many tools have been developed to mask the structure of the generator, by using boolean functions with suitable properties (see for example [12, 4]) to combine several LFSRs, by using combiners with memory or by shrinking the sequence produced by an LFSR.

It is possible to use similar methods with an FCSR generator, but with a very important difference: since an FCSR generator looks like a random generator for the non linear properties, it is not necessary to use a filter function with high non linearity.

Then the best functions for filtering an FCSR generator are linear functions:

$$f : GF(2)^n \to GF(2), \qquad f(x_1, \ldots, x_n) = \bigoplus_{i=1}^n f_i x_i,\ f_i \in GF(2).$$

As studied previously, the sequence M_i observed on the i-th dividend register is a 2-adic fraction, with known period, good statistical properties and looks like a random sequence except from the point of view of 2-adic complexity.

The sequences C_i (with $i \in I_d^*$) produced by the carry register are not so good from a statistical point of view: these sequences are probably balanced, however, if a carry register is in the state 1 (resp. 0), it remains in the same state 1 (resp. 0) with a probability 3/4 since each of the two other entries of the corresponding addition box corresponds to 2-adic fractions and produces a 1 with a probability approximatively 1/2. It is sufficient to have only one more 1 to produce a 1 in the carry register.

These remarks lead to filter only on the k cells $m_i(t)$ of the main register, not on the cells of the carry register.

To modelize our linear filter, we consider a binary vector $F = (f_0, \dots, f_{k-1})$ of length k.

The output sequence of our filtered FCSR is then

$$S = (s(t))_{t \in \mathbb{N}}, \quad \text{where } s(t) = \bigoplus_{i=1}^{k} f_i \cdot m_i(t).$$

The extraction of the output from the content of the main register M and the filter F can be done using the following algorithm:

```
$S := M \otimes F$
for $i := 6$ to 0 do
    $S := S \oplus shift_{+2^i}(S)$
Output: $par(S)$
```

It needs 7 shifts, 7 Xor and 1 And on 128-bit integers. So, the proposed F-FCSR is very efficient in hardware.

Design of the static filter for the F-FCSR-SF1 stream cipher

Let k_F be the integer such that $2^{k_F} \leq F < 2^{k_F+1}$. We will see in Paragraph 3.2.1 that it is possible to develop an attack on the initial key which needs 4^{k_F} trials.

If F is a power of 2, the output is a 2-adic sequence and is not resistant to 2-adic attacks. Moreover, if F is known, and its binary expansion contains few 1, the first equations of the algebraic attack are simpler, even if it is not possible to develop such an attack (cf. Paragraph 3.2.3).

A first natural solution would be to choose $F = 2^{128} - 1$, that is to xor all the cells of the main register. In this case, suppose that the output is $S = (s(t))_{t \in \mathbb{N}}$. It is easy to check that the sequence $S' = (s(t) + s(t+1))_{t \in \mathbb{N}}$ is the same that the one that would be obtained by xoring all the carry cells. Even if we do not know how to use this fact to develop a cryptanalysis, we prefer to use another filter for this reason.

In our application, we propose to choose $F = d = (|q| + 1)/2$. With this filter, the output is the XOR of all cells of the main register which are just at the right of a carry cell. For the prime q proposed above in (1) the value of k_F is 128 and the Hamming weight of the filter is 69.

We propose a very simple initialization of the F-FCSR-SF1 generator: we choose a key K with 128 bits. The key K is used directly to initialize the main register M. The carry register is initialized at 0.

Statistical properties of the filtered output

When two or more sequences are xored, the resulting sequence has good statistical properties as soon as one of the sequences is good, under the restriction that this sequence is not correlated with the other.

In our generator, each sequence is a 2-adic fraction with denominator q and has good statistical properties. The only problem is the fact that these sequences are not independent, since they are obtained by distinct shifts of the same periodic sequence. Note that the period of the sequence is very large ($T \geq 2^{127}$), and that a priori the 69 distinct shifts looks like random shifts. So the output sequence will have good statistical properties.

This hypothesis is comforted by the fact that our generator passes the NIST statistical test suite, as we checked.

3.2 Cryptanalysis of F-FCSR-SF1

3.2.1 2-adic cryptanalysis of F-FCSR-SF1

2-adic complexity of the XOR of two or more 2-adic integers

A priori, the XOR is not related with 2-adic operations (i.e. operations with carries), and then the sequence obtained by XORing two 2-adic fractions looks like a random sequence from the point of view of 2-adic complexity. Experiments support this assumption.

Moreover, due to the choice of q, in particular to the fact that $(|q|-1)/2$ is prime, the probability to have a high 2-adic complexity is greater than in the general case.

Let q be a negative prime such that 2 is of order $|q|-1$ modulo q. Consider the XOR $(p_1/q) \oplus (p_2/q)$ of the 2-adic expansions of two fractions with q as denominator and $0 < p_1, p_2 < |q|$. By Theorem 2, both summands are a sequence of period $|q|-1$ so the XOR is also a sequence of period $|q|-1$ (or dividing it). Can this latter sequence written also as a fraction p/q? (with $0 \leq p \leq q$ and possibly non reduced). Surely, the answer is yes in some cases (e.g. if $p_1 = p_2$). But in very most cases, the answer is no. Here is an heuristic argument to show this under the assumption that such an XOR gives a random sequence of period dividing $|q|-1$. The number of such sequences is $2^{|q|-1}$ and the number of sequences of the form $(p_1/q) \oplus (p_2/q)$ is at most $(|q|-1)^2/2$. So we can expect that the probability that the XOR can be written p/q is about $|q|^2/2^{|q|}$ which is very small. This remark extends to the XOR of $O(\ln |q|)$ summands.

A 2-adic attack

Theorem 2 *Assume that the filter F is known by the attacker and let k_F be an integer such that $F < 2^{k_F+1}$ (that is all cells selected by the filter belong to the rightmost $k_F + 1$ cells of the main register). Then the attacker can discover the key of the generator at a cost $O(k^2 2^{k_F})$.*

We first state a lemma.

Lemma 1 *Assume that the attacker knows the initial values $m_i(0)$ for $0 \leq i < k_F$ (he also knows the initial values $c_i(0)$ for $0 \leq i < k_F$ which were assumed to be 0). Then he can compute the T first bits $m_{k_F}(t)$ (for $0 \leq t < T$) of the sequence M_{k_F} by observing the sequence outputted by the generator, in time $O(Tk_F)$.*

Proof : The attacker observes first $S(0) = \bigoplus_{i=0}^{k_F} f_i m_i(0)$. In this equality, the only unknown value is $m_{k_F}(0)$ so the attacker can compute it in time $O(k_F)$. For subsequent bits the method generalizes as follows.

Assume that the attacker has computed bits $m_{k_F}(t)$ for $0 \leq t < \tau$ and knows $m_i(t)$ and $c_i(t)$ for $0 \leq t < \tau$ and $0 \leq i < k_F$. Observing the bit $S(\tau)$ he gets

$$S(\tau) = \bigoplus_{i=0}^{k_F} f_i m_i(\tau)$$

and the only unknown value here is $m_{k_F}(\tau)$. So the attacker obtains it, also in time $O(k_F)$. He can also compute $m_i(\tau+1)$ and $c_i(\tau+1)$ for $0 \leq i < k_F$, using the transition function. The time needed to compute these $2k_F$ bits is also $O(k_F)$. We obtain the result by induction. □

The attack whose existence is asserted in Theorem 2 works following six steps.

- Choose an arbitrary new set of values for the bits $m_i(0)$ for $0 \leq i < k_F$ and put $c_i(0) = 0$ for $0 \leq i < k_F$.
- Assuming that these bits correspond to the chosen values, compute the first k bits of the sequence M_{k_F}.
- Using the transition function, compute the first $k + k_F$ bits of the sequence M_0 from the assumed values for the bits $m_i(0)$ with $0 \leq i < k_F$ and the k bits obtained in the previous step.
- Multiply the integer $\sum_{t=0}^{k-1} m_0(t)2^t$ by q modulo 2^k to obtain a candidate p_0 for the key.
- Run a simulation of the generator with the key p_0. Stop it after generating $k + k_F$ bits. Compare the last k_F bits obtained to the ones computed in Step 3. If they don't agree, the candidate found is not the true key. Return to first step until all possibilities are exhausted.
- After all possibilities in Step 1 are exhausted, use some more bits of the generator to determine which key is the true key, if more than one good candidate remains.

Now the proof of the theorem:

Proof : From Lemma 1, the cost of Step 2 is in $O(kk_F) \leq O(k^2)$. Step 3 has also a cost of $O(kk_F)$. The cost of Step 4 is $O(k^2)$ and those of Step 5 is $O(k(k+k_F)) \leq O(k^2)$. The loop defined by Step 1 has to be iterated $O(2^{k_F})$ times. Multiplying the number of iterations by the inner cost gives the cost of the whole attack. □

With our parameters $k = 128$ and $k_F = 127$, this attack is more expensive than the exhaustive attack on the key.

Moreover, if the carries are not initialized to 0, there are 196 unknowns in the system instead of 128.

3.2.2 Linear complexity of F-FCSR-SF1 generator: XOR of two or more 2-adic integers

Arguments for the linear complexity are similar to those yet presented for the 2-adic complexity: since each 2-adic fraction looks like a random sequence from the point of view of linear complexity, the XOR of these sequences have a high linear complexity (cf. [17]). Experiments also support this assumption.

As for the 2-adic case, the particular value chosen for the period T helps for the 2-adic complexity to be high. Let q be a negative prime such that 2 is of order $|q|-1$ modulo q. Consider the XOR $(p_1/q) \oplus (p_2/q)$ of the 2-adic expansion of two fractions with q as denominator, and numerators such that $0 < p_1, p_2 < |q|$. Similar arguments as those above about the 2-adic behavior of this XOR applies to its linear behavior.

If this XOR corresponds to the expansion of a series $P(X)/Q(X)$ (written as a fraction in reduced form), then the order of the polynomial Q must be a divisor of $T = |q|-1$. With the value of q proposed in (1), the order of Q must be 1, 2, T, or $T/2$. The only polynomials of order 1 or 2 are the powers of $(X+1)$. Polynomials of order T or $T/2$ must have an irreducible factor Q_1 of order T or $T/2$. But this order must be a divisor of $2^{\deg(Q_1)} - 1$, so $\deg(Q_1)$ is a multiple of the order of 2 modulo q. In the case of the above value of q, this order is $T/2$, a number of bitsize 127. Hence polynomials Q with a divisor of such a degree are not so frequent.

3.2.3 Algebraic cryptanalysis of F-FCSR-SF1

The algebraic cryptanalysis of a pseudorandom generator is a tool developed recently (cf.[5]).

The principle is simple: we consider the bits of the initial state $m = (m_0, \dots, m_{k-1}) = (m_0(0) \dots, m_{k-1}(0))$ as the set of unknowns (suppose first that the initial value of the carry register is 0) and, using the transition function, we compute the successive outputs of the generator as functions of these unknowns $f_i(m_0, \dots, m_{k-1})$. If the attacker knows the first output bits of the generator, he gets a system of (non linear) equations in k variables. We can add to this system the equations $m_i^2 = m_i$ as the unknowns are Booleans. If the system obtained is not too complicated, it can be solved using for example the Gröbner basis methods [6].

The transition function of an FCSR automaton is quadratic: the first equation is linear on 128 variables (or 196 variables if the carries are not initialized to 0), the second one is quadratic, the third is of degree 3, and so on. For example, the eleventh equation is of degree 11 in 128 variables, its size is about 2^{50} monomials and is not computable. To solve the algebraic system, we need at least 128 equations.

Note that the fact we use a known filter does not increase the difficulty of this attack. The filter is just a firewall against a 2-adic cryptanalysis.

3.2.4 The time/memory/data tradeoff attack

There exists a recent attack on stream ciphers with inner states: the time/memory/data tradeoff attack [3]. The cost of this attack is $O(2^{n/2})$, where n is the number of inner states of the stream cipher. This cost reflects not only the time needed for the attack, but also the use of memory and the amount of data required. For the F-FCSR-SF1, the number of inner states is $n = k + \ell = 128 + 68 = 196$. Even if this attack remains impracticable, it is faster than the exhaustive one. This is why we recommend to use the dynamic filter method.

3.3 Design of F-FCSR-SF8: a static filter and an 8-bit output

In order to increase the speed of the generator, we propose to use several filters to get several bits at each transition of the FCSR automaton. For example, using 8 distinct filters, it is possible to obtain an 8-bit output at each transition. However, the design of several filters may be difficult.

A first cryptanalysis on multiple filters

Suppose that we use 8 filters F_1, …, F_8 on the same state of main register M. Obviously, each of these filters must be resistant to the 2-adic attack. These 8 filters must be linearly independent to avoid a linear dependency on the 8 outputs. Moreover, by linear combinations of the 8 filters, it is possible to obtain 2^8 filters, each of them must also be resistant to the 2-adic attack.

Let $\mathcal{C}$ be the binary linear code generated by F_1, …, F_8.

- The condition on the independence of the 8 filters is the fact that the dimension of $\mathcal{C}$ is exactly 8.
- For $F \in \mathcal{C}$, let k_F be the least integer such that $2^{k_F} > F$ (here F is viewed as an integer). The minimum over $\mathcal{C}$ of the values of k_F must be as larger as possible. Note that $min_{F \in \mathcal{C}, F \neq 0}\{k_F\} \leq k - 8 = 120$. If we choose $\mathcal{C}$ such that $min_{F \in \mathcal{C}, F \neq 0}\{k_F\} = 120$, the

cost of the 2-adic attack is $O(120 \times 2^{120})$ which is approximatively the cost of the exhaustive attack.

Note that it is easy to construct a code $\mathcal{C}$ satisfying this condition.

- We recommend to avoid the use of a code $\mathcal{C}$ with a small minimum distance d. Indeed, from a codeword of weight d, it is possible to construct a filter on d cells of the main register M. Even if we do not know how to design such an attack for $d \geq 2$, we suggest to choose $\mathcal{C}$ satisfying $d \geq 6$.

A simple way to construct 8 simultaneous filters

In order to construct good filters with a very efficient method to extract the 8-bit output, we recommend the following method:

The filters are chosen with supports included in distinct sets. More precisely, for $i = 0$ to 7, $Supp(F_i) \subset \{j \mid j \equiv i \pmod 8\}$.

This construction ensures $dim(\mathcal{C}) = 8$, $min_{F \in \mathcal{C}, F \neq 0}\{k_F\} = min_i\{k_{F_i}\}$ and $d = min_i(w(F_i))$, where $w(F)$ is the Hamming weight of F. Moreover the extraction procedure becomes very simple: First, set $F = \bigoplus_{i=0}^{7} F_i$. The extraction of the 8-bit output from the content of the main register M and the filter F can be done using the following algorithm:

```
$S := M \otimes F$
for $i := 6$ to 3 do
        $S := S \oplus shift_{+2^i}(S)$
Output: $S \otimes 255$ (the 8 lower weight bits of $S$)
```

This needs 4 shifts, 4 Xor and 2 And on 128-bit integers. This extraction is faster than the extraction of a single bit.

Note that conversely, from a 128-bit filter F, we obtain a family of 8-bit filters. As an example, for the value $F = d$ proposed for the F-FCSR-SF1 generator, we obtain a code $\mathcal{C}$ with $dim(\mathcal{C}) = 8$, $mink_F = 113$ and $d = 4$. For this choice of filter, it will be possible to design a 2-adic attack slightly more efficient than the exhaustive one.

A possible attack

Let $S(t) = (S_0(t), \ldots, S_7(t))$ be the 8-bit output at time t. Some entries selected by the filter on which depend $S_0(t+7)$, $S_1(t+6)$,..., $S_7(t)$ may be related. And the relations involved might be partially explicited when the state of the automaton is partially known.

So, even if we do not know how to design such an attack, we do not advice to use the 8-bit output generator with a static filter. The dynamic filter method presented in the next section will resist to such attack and will be preferred. We also propose to use an IV mode with the F-FCSR designs in order to have a high confidence on the security against be sure to resist to the different attacks.

4 Design of F-FCSR-DF1 and F-FCSR-DF8: dynamic filtered FCSR stream ciphers

Due to the fact that the filter is very simple and its quality is easy to check, it is possible to use a dynamic filter: the filter can be constructed as a function of the key K, and then, is not known by the attacker. As soon as the filter is not trivial ($F \neq 0$ and $F \neq 2^i$), it is not possible to use the algebraic attack, nor the attack exploiting the small binary size of F.

The construction of this dynamic filtered FCSR generator (DF-FCSR generator) is very simple: let g be a bijective map of $GF(2)^{128}$ onto itself. For a 128-bit key K, we construct the filter $F = g(K)$ and also we use the key to initialize the main register. The carry register is initialized at 0, since the attacker cannot find the equations for the algebraic attack.

The main interest of the use of a dynamic filter is the fact that the number n of inner state is increased of the size of the filter, i.e. $n = 2k\ell = 324$. The cost of the time/memory/data tradeoffs attack becomes higher than those of the exhaustive one.

4.1 Design of F-FCSR-DF1

This stream cipher is identical to F-FCSR-SF1 except the fact that the filter is dynamic.

We propose to use for g the Rijndael S-box (cf. [14, 15]). This S-box operates on bytes, and using it for each 16 bytes of a 128-bit key, we get a suitable function g.

It is suitable to add a quality test for the filter, for example by testing the binary size k_F of F and its Hamming weight $w(F)$. For example, if $k_F < 100$ or $w(F) < 40$, then we iterate g to obtain another filter with good quality.

The computation of this dynamic filter is very simple. The main advantages are to thwart completely the 2-adic attack (§3.2.1), the algebraic attack (§3.2.3) and to avoid the time/memory/data tradeoff attack.

However, until now, we do not find any attack faster than the exhaustive search against the static filter generator.

4.2 Design of F-FCSR-DF8

For the 8-bit output version, the use of a dynamic filter has also other justification: it avoids all possible attacks on the filter described in Paragraph 3.3.

For a practical use we recommend the following key loading procedure:

• Construction of the filter F from the 128-bit secret key K by applying the Rijndael S-box.
• Test the quality of the 8 subfilters extracted from F. Each of them must have an Hamming weight at least 6, and a binary size at least 100.
• Go to the first step until the test succeed.
• Use the key K to initialize the main register M. The carry register is initialized to 0.

The filter procedure is those of F-FCSR-SF8 (§3.3).

4.3 An initial vector mode for F-FCSR stream ciphers

The IV mode

There are several possibilities to add some initial vector IV to our generators. A first one will be to use it as filter F, where the main register is initialized with the key K and the carry register is initialized to 0.

In that case, we are in the situation of multiple known filters on the same initialization of the automaton. This method will be dangerous.

In fact, the good solution is to use always the same filter from a fixed key K with a static filter for 1 bit output and dynamic filter for 8-bit output. The IV is used to initialize the carry registers.

With our automaton, there are 68 bits in the carry register. It is easy to use them for IV of size 64. In order to avoid some problems related to the use of the same key K for the main register, we recommend to wait 6 cycles of the automaton before using an input after a change of IV. After these 6 cycles, every cell of the main register contains a value depending not only of K but also of IV.

We recommend to use the following protocol either with the F-FCSR-DF1 stream cipher, or with the F-FCSR-DF8 stream cipher:

Pseudocode:

```
1. $F := g(K)$                                    (dynamic construction of the filter).
2. $M := K$;    $M := K$.
3. Clock 6 times the FCSR and discard the output.
4. Clock and filter the FCSR until the next change of $IV$.
5. If change of $IV$, return to step 2.
```

A variant of our generator with a key of size 96 and initial vector of size 64

For some purposes where the security is important only during a limited amount of time, it can be useful to define a variant with a smaller key-size (but with same IV-size). For that we propose to use the retroaction prime

$$q = -145992282562012510535118773123 = -\mathrm{0x1D7B9FC57FE19AFEFEF7C5B83}$$

This prime has been selected according the following criteria. Its bit size is 97, so that d has bitsize 96. Also $(|q| - 1)/2$ is prime. The order of 2 modulo $|q| - 1$ is exactly $|q| - 1$. And $d = \mathrm{0xEBDCFE2BFF0CD7F7F7BE2DC2}$ has weight 65 so that there are 64 useful cells in the carries register.

Conclusion

We proposed a very fast pseudorandom generator, easy to implement especially in hardware (but also in software). It has good statistical properties and it is resistant to all known attacks. Its design can be compared to older generators (such as the summation generator [16]) for whose the heart has a linear structure, and is broken by a 2-adic device. Instead, our generator has a heart with a 2-adic structure which is destroyed by a linear filter. It might be of similar interest of these older generators (the summation generator is one of the best generator known) while being even easier to implement due to the simplicity of the filter.

Acknowledgments: Both authors would like to thank Anne Canteaut and Marine Minier for helpful comments and suggestions.

References

[1] F. Arnault, T. Berger, and A. Necer. A new class of stream ciphers combining LFSR and FCSR architectures. In *Advances in Cryptology - INDOCRYPT 2002*, number 2551 in Lecture Notes in Computer Science, pp 22–33. Springer-Verlag, 2002.

[2] F. Arnault, T.P. Berger, A. Necer. Feedback with Carry Shift Registers synthesis with the Euclidean Algorithm. *IEEE Trans. Inform. Theory*, Vol 50, n. 5, may 04, pp. 910–917

[3] A. Biryukov and A. Shamir *Cryptanalytic time/memory/data tradeoffs for stream ciphers* LNCS 1976 (Asiacrypt 2000), pp 1–13, Springer, 2000.

[4] D. Coppersmith, H Krawczyk, Y. Mansour. *The Shrinking Generator*, Lecture notes in computer science (**773**), Advances Cryptology, CRYPTO'93. Springer Verlag 1994, 22-39

[5] N. Courtois, W. Meier *Algebraic attack on stream ciphers with linear feedback* LNCS 2656 (Eurocrypt'03), Springer, pp 345–359

[6] J.C. Faugère *A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5)* Proceedings of International Symposium on Symbolic and Algebraic Computation, ISSAC'02, Villeneuve d'Ascq, pp. 75–83

[7] A. Klapper and M. Goresky. 2-adic shift registers, fast software encryption. In *Proc. of 1993 Cambridge Security Workshop*, volume 809 of *Lecture Notes in Computer Science*, pages 174–178, Cambridge, UK, 1994. Springer-Verlag.

[8] A. Klapper and M. Goresky. Cryptanalysis based on 2-adic rational approximation. In *Advances in Cryptology, CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 262–274. Springer-Verlag, 1995.

[9] A. Klapper and M. Goresky. Feedback shift registers, 2-adic span, and combiners with memory. *Journal of Cryptology*, 10:11–147, 1997.

[10] A. Klapper and M. Goresky. Fibonacci and Galois representation of feedback with carry shift registers. *IEEE Trans. Inform. Theory*, 48:2826–2836, 2002.

[11] N. Koblitz. *p-adic Numbers, p-adic analysis and Zeta-Functions.* Springer-Verlag, 1997.

[12] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone *Handbook of Applied Cryptography*, CRC Press, 1996.

[13] *"A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications"*, `http://csrc.nist.gov/rng/`

[14] J. Daemen, V. Rijmen *The Block Cipher Rijndael*, Smart Card Research and Applications, LNCS 1820, J.-J. Quisquater and B. Schneier, Eds., Springer-Verlag, 2000, pp. 288-296.

[15] `http://csrc.nist.gov/CryptoToolkit/aes/`

[16] R.A. Rueppel, *Correlation immunity and the summation generator*, Lecture Notes in Computer Science (**218**), Advances in Cryptology, CRYPTO'85, Springer-Verlag 1985, 260-272.

[17] R.A. Rueppel, *Linear complexity of random sequences*, Lecture Notes in Computer Science (**219**, Proc. of Eurocrypt'85, 167–188)