

Cryptanalysis of the F-FCSR Stream Cipher Family

Éliane Jaulmes and Frédéric Muller

DCSSI Crypto Lab
51, boulevard de La Tour-Maubourg
75700 PARIS-07 SP
{Eliane.Jaulmes, Frederic.Muller}@sgdn.pm.gouv.fr

Abstract. This paper focuses on F-FCSR, a new family of stream ciphers proposed by Arnault and Berger at FSE 2005. It uses a non-linear primitive called the Feedback with Carry Shift Register (FCSR) as a building block. Its security relies on some properties of the 2-adic numbers. The F-FCSR family contains several stream ciphers, each of them proposing different features.

First, we show a resynchronization attack that breaks algorithms in the family that support initialization vectors. The attack requires at most 2^{16} chosen IV's and a little offline processing to recover the full secret key. We have implemented it with success on a standard PC.

Secondly, we show a time/memory/data trade-off attack which breaks several algorithms in the F-FCSR family, even when initialization vectors are not supported. Its complexity ranges from 2^{64} to 2^{80} operations (depending on which algorithm in the family we consider), while the internal state has size 196 bits at least. Therefore this attack is better than generic attacks.

Keywords. FCSR, Time/memory/data trade-off, stream cipher, resynchronization attack

1 Introduction

Stream ciphers are a special class of secret key cryptosystems. Their principle is to generate a long pseudo-random sequence which is then XORed bitwise to the message in order to produce the ciphertext. Thus, compared to a block cipher, a secure stream cipher is expected to be much faster.

Yet the design of secure stream ciphers also seems to be more difficult. Indeed, over the years, few proposals have withstood cryptanalysis. Many of the attacks staged over stream ciphers exploit the mathematical structure of Linear Feedback Shift Registers (LFSR) which are often used as building blocks. To avoid these pitfalls, alternative constructions have been proposed recently. For instance, it has been suggested to use irregularly-clocked registers or efficient non-linear mapping. One of these suggestions is to replace LFSR by Feedback with Carry Shift Registers (FCSR). A FCSR is a binary register, similar to a LFSR, except that it performs operations with carries. This brings non-linearity,

which is an interesting property to thwart algebraic attacks [8] or correlation attacks [19, 20]. The idea of using FCSR in cryptography was originally proposed by Klapper *et al.* in 1994 [16], but it was shown that they are not secure when used alone [17]. FCSR came back into flavor with recent works by Arnault and Berger [2, 3]. Some of their proposals were broken in 2004 [21]. Later, at FSE 2005, they proposed a concrete family of stream ciphers based on FCSR. It is referred to as the **F-FCSR family** and several variants are suggested. In this paper, we investigate the security of this new family of stream ciphers.

The paper is constructed as follows: in a first section, we recall the principle of the FCSR primitive and describe the proposals of [3].

Secondly, we describe resynchronization attacks against all variants that support Initialization Vectors (IV). We describe how to learn some information about the secret key, by comparing the keystream generated with two related IV's. We manage to recover the full secret key with about 2^{15} pairs of chosen IV's and little offline processing. These attacks have been implemented and take time ranging from a few seconds to a few hours on a standard PC.

Finally, we describe a time/memory/data trade-off against some algorithms in the family. Since the internal state has a size of $n = 196$ bits, the best generic attack is expected to cost about $2^{n/2} = 2^{98}$ computation steps. However we show that the real entropy is only of 128 bits, due to the cycle structure of the state-update function. Using this property, we attack some of the algorithms with time, memory and data of the order of 2^{64} .

2 Stream Ciphers Based on FCSR

2.1 The FCSR Primitive

Feedback with Carry Shift Registers were introduced by Goresky and Klapper [16]. The underlying theory is related to the 2-adic fractions and more details can be found in [3]. Here, we simply recall the main characteristics.

Let q be a negative integer such that $-q$ is prime and let $0 \leq p < -q$. Let $q = 1 - 2d$. We write $d = \sum_{i=0}^{k-1} d_i 2^i$.

The FCSR generator with feedback prime q and initial value p produces the 2-adic expression of the fraction p/q . This corresponds to the infinite sequence of bits $a_i \in \{0, 1\}$, where $p = q \cdot \sum_{i=0}^{\infty} a_i 2^i$.

This sequence has good statistical properties (relative to its period in particular), provided q is prime and 2 is of order $|q| - 1$ modulo q . The sequence of a_i can also be computed in an iterative way through the sequence of integers p_i , defined as:

$$p_0 = p, \tag{1}$$

$$a_i = p_i \bmod 2 \quad \text{and} \quad p_{i+1} = \frac{p_i - q a_i}{2} \equiv \frac{p_i}{2} \bmod q. \tag{2}$$

It is easy to verify that $\forall i, 0 \leq p_i < -q$. Also, the following relation holds $\forall i \geq 0$, $\frac{p_i}{q} = \sum_{j \geq i} a_j 2^{j-i}$.

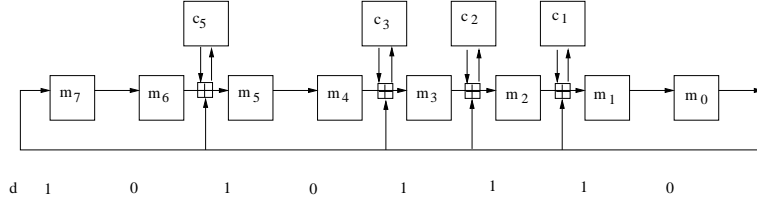


Fig. 1. Example of FCSR.

The sequence of integers a_i can be obtained in a register-oriented fashion. Consider:

- a main register M with k binary memory cells,
- a carry register C with $\ell - 1$ binary memory cells, where ℓ is the Hamming weight of d . The set $I_d = \{i | 0 \leq i \leq k - 2 \text{ and } d_i = 1\}$ denotes the positions of these cells.

The main register M is said to **contain** the integer $m = \sum_{i=0}^{k-1} m_i 2^i$ when the values $(m_0 \dots m_{k-1})$ appear in the k cells of the register. Similarly, the carry register is said to **contain** the integer $c = \sum_{i=0}^{\ell-1} c_i 2^i$ if, for $i \notin I_d$, $c_i = 0$ and, for $i \in I_d$, c_i appears in the corresponding carry cell. We say that, at time t , the FCSR is in state $(m(t), c(t))$ if the main register contains the value $m(t)$ and the carry register contains $c(t)$. The state $(m(t+1), c(t+1))$ is computed from the state $(m(t), c(t))$ according to the following equations:

- 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 $i = k - 1$
 $m_i(t+1) = m_0(t)$

A small example of this register representation with $q = -347$ appears in the Figure 1.

Let the initial state be given by $m(0) = p$ and $c(0) = 0$. The next state $(m(t+1), c(t+1))$ is computed from $(m(t), c(t))$ by looking at the least significant bit $m_0(t)$ of M . The main register is always right-shifted, which corresponds to a division by 2. The content of C is added to the result, as well as the number d when $m_0(t) = 1$. Thus,

$$\begin{aligned} m(t+1) + 2c(t+1) &= \frac{m(t) - m_0(t)}{2} + c(t) + dm_0(t) \\ &= \frac{m(t) + 2c(t) - qm_0(t)}{2} \end{aligned}$$

Hence the following relations are always satisfied:

$$a_t = m_0(t) \quad \text{and} \quad p_t = m(t) + 2c(t).$$

So the register-oriented representation produces indeed the 2-adic representation of the fraction p/q through the bit $m_0(t)$ - generally called the **feedback bit**.

2.2 The F-FCSR Family

Stream ciphers based on FCSR were already proposed in the past [2, 16]. However it was shown that, despite the non-linearity, it is insecure to output directly bits from the FCSR sequence. Efficient algorithms [17] have been proposed to retrieve the initial state: when q is of size 128 bits, knowing 128 output bits is sufficient.

Arnault and Berger suggested [3] to apply a filtering function to the FCSR state, in order to produce output bits. They argued that a linear filter is sufficient, since the FCSR primitive is already non-linear by itself. At FSE 2005, they presented a new family of stream ciphers based on this general idea. The following parameters are chosen for all algorithms in the family: the main register has a size $k = 128$ bits and the feedback prime is

$$-q = 493877400643443608888382048200783943827.$$

This choice guarantees the maximal period, because the order of 2 modulo q is equal to $|q| - 1$. The Hamming weight of $d = \frac{1-q}{2}$ is $\ell = 69$ so the carry register has $\ell - 1 = 68$ memory cells. The secret key of the algorithm (of size 128 bits) is the initial value p introduced in the main register.

Differences between the four proposed algorithms lie in the nature of the output function. Two distinctions are made depending on

- the secrecy of the output taps. Two proposals use a fixed linear combination of the FCSR state. The other two use a dynamic output function (*i.e.* the taps are generated from the secret key, and are therefore unknown to an attacker).
- the number of output bits per advance. The basic construction produces 1 bit at each advance of the FCSR. To improve the encryption speed, it is possible to use 8 filtering functions in order to produce 8 output bits per advance.

2.3 Description of the Proposals

The first proposal is called **F-FCSR-SF1**. SF stands for "Static Filter". The stream cipher is based on the FCSR described above. It produces one single output bit at each advance of the FCSR. This output bit is computed through a known linear filter function, called F , of the form:

$$f(m_0, \dots, m_{k-1}) = \bigoplus_{i=0}^{k-1} f_i m_i,$$

where $f_i \in \{0, 1\}$. We call f the number $\sum_{i=0}^{k-1} f_i 2^i$. In their paper, the authors suggested to use the filter $f = d$. With this choice, the output bit of the stream cipher is the XOR of all the main register cells located just after a carry cell.

The second proposal is called **F-FCSR-SF8**. The same FCSR is used but eight output bits are produced at each advance. This is done quite naturally by using eight different filters $(F_i)_{1 \leq i \leq 8}$. The filters must be

linearly independent and respect other conditions detailed in [3]. In order to have a simple extraction function for the 8 output bits, the authors suggest to use 8 filters with disjoint supports. More precisely, they recommend filters F_i such that

$$\forall i \in [1, 8], \text{Supp}(F_i) \subset \{j | j \equiv i \pmod{8}\}. \quad (3)$$

The filters $(F_i)_{1 \leq i \leq 8}$ are public and part of the stream design.

The third proposal is called **F-FCSR-DF1**. DF stands for "Dynamic Filter". It works exactly as **F-FCSR-SF1**, except that the output filter F is kept secret and derived from the key through an invertible function g . This artificially increases the size of the cipher state from $196 = 128 + 68$ to 324 bits.

The fourth proposal is called **F-FCSR-DF8**. It works exactly as **F-FCSR-SF8**, except that the eight filters are kept secret and derived from the key. Here also the filters verify the condition 3. Since the filters have disjoint supports, a single invertible function g suffices to define the eight filters.

2.4 Initialization Vectors

Since stream ciphers produce bit sequences independently of the messages they encrypt, it is customary to add an initialization vector (IV), that allows the sequence to change from one encryption to the next. The IV is usually a public value, transmitted alongside the ciphertext.

Support for IV is often impossible to avoid: applications deal with relatively small messages (frames or data packets), which makes it highly inefficient to rekey the cipher for each message. Using one long keystream sequence raises important problems of synchronization. Therefore all new stream ciphers are expected to support IV's. For example, this is a requirement in the call for stream ciphers published recently by the european project ECRYPT [10].

In the F-FCSR family, the authors propose to **use the initial content of the carry register as the initialization vector**. Six advances are made before the encryption starts, to guarantee the diffusion of the IV in the initial state.

There is a slight problem of dimension, not solved in the FSE paper: the carry register has length 68 bits which is not a convenient IV size (64 bits would be better for instance). In the later, we assume that the IV has length 68 bits. We claim that variants of our attack could be envisaged even if another IV dimension (64 or 128 bits) was used and it was somehow mapped to the carry register state.

2.5 Resynchronization Attacks

Building a good initialization mechanism for a stream cipher is not an easy task. Indeed, the IV is a public value introduced inside the secret state of the cipher. By looking at the link between the IV and the first keystream bits, some secret information may be leaked. This family of

attacks has first been called "resynchronization attacks" [9], but its spectrum of applications has broadened [1, 13]. Practical applications have been shown for the 802.11 standard [18] or for the GSM standard [6].

Most attacks require only **known IV** (which is always the case since the IV is transmitted in clear). However, there are situations where we can envisage **chosen IV**. Firstly, an active attacker may modify the IV while it is transmitted on the communication channel. Secondly, the IV is often generated by an unspecified random generator, which might be partially controlled by the attacker.

Our attacks require pairs of related IV's which typically differ by only one bit. Since many implementations use a counter, such low-weight differences are likely to appear rapidly. Such chosen IV attacks often turn up to be more practical than expected (see the example of RC4 in the 802.11 standard [18]).

3 Resynchronization Attack with Static Filter

In this section, we describe an attack against the two proposals which use a static filter. We focus on **F-FCSR-SF1** but clearly the same attack applies to **F-FCSR-SF8**.

3.1 Principle of the Attack

Our basic observation is that the 6 initial advances are not sufficient for all cells of the main register to depend on the whole IV. Suppose we flip only one bit of the IV (*i.e.* one bit in the initial state of the carry register), then after 6 advances, only a few cells in the main register may be affected. Our idea is to predict the difference on the first keystream bit. To do that, only a small number of key bits need to be guessed.

The initial state of the main register is just the key $m(0) = K$. Similarly, the initial state of the carry register is denoted by $c(0)$ and is just equal to $c(0) = IV$. After 6 advances, the state of the carry register is $c(6)$ and the state of the main register is $m(6)$. The first keystream bit $z(0)$ is given by $z(0) = \bigoplus_{i=0}^{127} f_i m_i(6)$, where the output filter $f = \sum_i f_i 2^i$ is known.

Let $i \in I_d$ be a position where a carry cell is present. Suppose we initially replace $c_i(0)$ by $c_i(0) \oplus 1$. We are interested in how the first keystream bit $z(0)$ is affected by this modification¹. Note that this difference propagates through the main register up to the end, one cell at a time, and will not disappear. Thus, after $n \leq i + 1$ advances of the FCSR, there will be a difference in the cell $i - n + 1$. Due to the carries, the difference may also linger on previous cells, but the probability of a difference staying for a long time is low.

If $i \geq 5$, then, only the bits $m_i(6), \dots, m_{i-5}(6)$ of the main register may be affected, after the 6 initial advances, by the initial flip. Bit $m_{i-5}(6)$ is always flipped, and for the other bits, it depends on the initial state. This propagation is illustrated in Figure 2. If $i < 5$, then the feedback

The difference on the first keystream bit is therefore always 1, for all values of the 17 key bits. In this case, no information can be obtained by randomizing the IV.

Of course, this is the least favorable case. In most cases, we observed that many candidates for the 17 key bits can be discarded using the previous differential conditions. However, a small number of candidates may remain.

3.3 Key-Recovery Algorithm

To overcome these difficulties, we propose a "sliding window" algorithm. After considering a position i , a small number of candidates for 17 bits of the key remains. Next, we examine position $i - 1$, and guess a few extra key bits in order to repeat the same attack². From pairs of chosen IV's, we obtain new conditions which allow us to eliminate more candidates, and so on. Our goal is to keep the number of candidates as low as possible alongside the execution of the attack.

The resulting algorithm may be described as follows:

- Guess the 6 rightmost bits ($m_0(0), \dots, m_5(0)$).
- Guess the 6 leftmost bits ($m_{122}(0), \dots, m_{127}(0)$).
- For i from 120 down to 5 do:
 - Guess bit $m_{i+1}(0)$.
 - If $(i+5) \in I_d$ (a carry cell is present at the current position) do:
 - * Flip the corresponding IV bit, and do as much experiments as possible (depending on the number of carry bits that are able to influence the output).
 - * Discard guesses that are not compatible with the observed difference on the first keystream bit.
- Output the remaining correct guesses and test each of them to find the secret key.

When no carry bit is present at position i (case where $i \notin I_d$), we cannot eliminate any candidate, so the number of guesses to examine grows by a factor of 2. This can be quite inconvenient, so we propose a simple improvement. We flip the bit $c_{i+6}(0)$ (instead of $c_{i+5}(0)$) and look at the second keystream bit $z(1)$ (instead of $z(0)$). This is roughly the same idea as before with 7 advances instead of 6. Some technical details need to be fixed (for instance, we need to predict one more feedback bit), but the idea remains the same. And so on with more advances.

With this improvement, we can obtain conditions at every position (instead of only the positions $i \in I_d$) for little extra cost. This keeps the number of candidates low and makes the attack feasible.

3.4 Efficiency and Results

We implemented this attack on a standard PC, and observed the behavior of the previous algorithm on several randomly chosen keys. At each stage,

² Because of the overlapping between the two sets, only one new bit needs to be guessed.

the number of possible solutions never climbed above the starting point of 2^{13} . The set of solutions must be explored for each value of index i . Thus, the time complexity of the algorithm is about $128 \times 2^{13} \simeq 2^{20}$. This represents a few seconds on a PC.

With the selected value of q , there is an average of 2^8 experiments for each position, which means the number of possible experiments is about $128 \times 2^8 \simeq 2^{15}$. This means we need to process about 2^{15} pairs of chosen IVs in order to recover the full secret key. For each pair, we are only interested in learning the differential on one keystream bit. Thus the attack is very efficient and does not require a powerful adversary.

We also mention that some trade-offs are possible: if less than 2^{15} pairs of IV are available, we can stop the previous algorithm at any time, when only n bits from the key have been guessed. Then we can go through all the remaining correct guesses in the algorithm (at most 2^{13}) and also guess the remaining $128 - n$ key bits.

3.5 Adapting the Attack to F-FCSR-SF8

The eight filters chosen to produce the eight output bits in the **F-FCSR-SF8** version of the stream cipher have disjoint support. If we XOR the eight output bits, we obtain exactly the one bit of output we had in the previous attack. Thus, at worst, **F-FCSR-SF8** can be attacked with the same complexity as **F-FCSR-SF1**.

Moreover, it is also possible to use the extra information : 8 bits of information about the internal state are outputted at a time. This allows us to reject more candidates at each stage. We expect to discard everything but the correct guess at each step, which would decrease the time complexity of the attack to about 2^{16} as the total number of IV's to process.

4 Resynchronization Attacks with Dynamic Filter

In this section, we describe an attack against the two proposals using a dynamic filter. We will focus on **F-FCSR-DF1** for our description of the attack but it applies similarly on **F-FCSR-DF8**. As in Section 3, the attack is a chosen IV attack and recovers the secret key with only 2^{16} IV's.

4.1 Principle of the Attack

In the **F-FCSR-DF1** version of the stream cipher, the filter function is unknown of the adversary. It is derived from the secret key through an invertible function g . The function g , however, is public. This means that if the adversary is able to reconstruct the filter, he can immediately recover the corresponding secret key. Thus, our attack focuses on recovering the output filter.

As in Section 3, the principle of the attack is a differential cryptanalysis on the initial carry vector. We observe that when we introduce a difference on the bit i of the carry vector, after one clock of the register, this difference will be on the bit i of the main register, after two clocks, it will be on the bit $i - 1$ of the main register, and so on. The Figure 2 shows the cells that may contain a difference after 6 advances of the FCSR.

Since we no longer know the output taps, it is pointless to try to predict the state difference after 6 advances. However, we can try to predict **whether or not an output tap is present at each position**. As observed previously, if we flip the i -th carry bit, it is guaranteed to propagate to position $i - 5$ of the main register after 6 advances. For positions $i - 4 \leq j \leq i$, the difference may subsist depending on the carries, but these events are quite unlikely.

Thus, after six advances the output will be flipped with probability greater than $1/2$ if there is an output tap at position $i - 5$, and not flipped with probability greater than $1/2$ otherwise. This observation opens a way of attack.

4.2 Details of the Attack

In order to predict the output tap number $i - 5$, we flip the carry bit number i . Then we observe the difference on the first keystream bit $z(0)$. For positions $i \notin I_d$ where there is no carry cell, our trick consists in targeting the first $j \in I_d$ such that $j > i$ and observing the keystream bit $z(j - i)$.

With good probability, the presence of a difference in the output indicates that $f_i = 1$. Since we are interested in measuring a probability, we need several such experiments. Like we did in Section 3, we vary the experiments by changing the values of the carry bits in the influence zone. We obtain a ratio δ_i of the number of differences observed by the number of total experiments for the bit i .

We first perform a gross prediction of the filter F , quite simply by saying that $f_i = 1$ if $\delta_i > 0.5$, and $f_i = 0$ otherwise. From the different random keys we tried, we observed that this prediction already gives a quite accurate value of F . Indeed, only 20 to 30 errors remain among the 128 filter bits. Since we do not know the positions of these errors, it would be too long to enumerate all candidates at this point.

In order to decrease the number of errors, we propose an iterative algorithm to converge towards the correct filter. We modify locally some bits on the predicted filter and test whether the result fits better to the reality. This algorithm may be described as follows:

- Collect data on the stream cipher by doing the set of experiments described above. Store the values of the δ_i in an array Δ .
- Based on the data, obtain a gross prediction F' for the filter and deduce the corresponding key K' by inverting g .
- Reproduce the same set of experiments "offline", on a stream cipher using the current candidates for the key and the filter. Obtain an array Δ' .
- While $\Delta \neq \Delta'$, do:

- Pick one byte from the filter and enumerate the 2^8 possible values. Keep the current values of F' for the other bytes. Do the set of experiments for each new guessed value. Keep the value that minimizes $d(\Delta, \Delta')$, where d is some distance function (for instance, the euclidian function).
 - If stuck on a wrong filter value, change the distance d .
- Output the filter and the corresponding key.

The underlying idea is that the **errors on the candidate filters are generally local**. So it is interesting to modify just one byte of the filter and test whether we obtain a better prediction. Random jumps are used in case we get stuck at some point.

We observed in practice that this algorithm was quite efficient and converged reasonably quickly towards the correct key and the correct filter.

4.3 Efficiency and Results

The first part of the algorithm consists in collecting data from the real stream cipher. This requires 2^{15} pairs of IV's. As in Section 3, we are only interested in learning the differential on one output bit. The rest of the attack is performed offline.

The time complexity of the algorithm is trickier to evaluate since it depends on the converging speed. Each step of the loop requires 2^{24} operations. Our experiments with random secret keys suggest that an average of 2^8 passes of the loop are needed to find the correct secret key. Thus the time complexity observed in practice is around 2^{32} . This attack runs in a few hours on a single PC.

4.4 Adapting the Attack to F-FCSR-SD8

The 8 filters are constructed as for **F-FCSR-SF8**. They verify equation 3. For a filter F_i , we know that two bits equal to 1 are separated at least by eight bits. Since the range of our experiments often affects only six or seven bits, we are able to directly guess the correct value of the filter F_i for most of the bits (those that correspond to an experiment with length less than 8). The remaining bits will also have a better prediction than in the previous case. Thus, the attack becomes more powerful and recovers the filter and the secret key much quicker.

5 Time/Memory/Data Trade-off Attacks

5.1 Trade-off Attacks Against Stream Ciphers

Since the internal state of a stream cipher changes during the encryption, a brute-force attack has several possible targets. Knowing any value of the internal state is often sufficient for an attacker. This is the main idea behind trade-off attacks: only a small portion of the possible states is enumerated, and we hope, using the birthday paradox, to find a match between the states encountered during the encryption and the states enumerated offline.

The first attack proposed in the literature is generally referred to as Babbage-Golic trade-off [5, 12]. Roughly, a stream cipher with internal state of n bits can be attacked with time, memory and data of the order of $2^{n/2}$. An alternative is to apply the famous Hellman time-memory trade-off [14] to the case of stream ciphers [7]. This second attack allows many trade-offs, but its complexity always remains above $2^{n/2}$ regarding the time complexity.

Because of these attacks, it is recommended for stream ciphers to use an internal state, which is larger than the expected strength (usually twice the size of the key). Accordingly, Arnault and Berger [3] argued that F-FCSR has a state of $n = 196$ bits at least (it is even more when the output taps are secret), which brings the attack to a complexity larger or equal to 2^{98} , which is not practical. In this Section, we show that a better trade-off attack is possible against the F-FCSR family, due to the mathematical properties of the state-update function.

5.2 Real Entropy of the F-FCSR State

Let $s(t) = (m(t), c(t))$ denote the state of the cipher at time t . There are $2^{k+\ell-1} = 2^{196}$ possible states, and as explained in Section 2.1, each state is characterized by some integer $p_t < (-q)$ such that $p_t = m(t) + 2c(t)$. A crucial observation is that **the state-update function of a FCSR is not invertible**. Take the example of Figure 1. Suppose that $m_7(t) = 0$ and $m_5(t) = c_5(t) = 1$. Then the previous state must satisfy two incompatible constraints:

$$\begin{aligned} 0 &= m_0(t-1) \\ 1 &= m_0(t-1) = m_6(t-1) = c_5(t-1) \end{aligned}$$

Therefore, as the encryption proceeds, **the state loses entropy**. It is known that, provided the order of 2 modulo q is $|q| - 1$, the graph of the state-update function had a unique cycle of length $|q| - 1$. Moreover, each state $s = (m, c)$ in the cycle corresponds uniquely to some integer $x < (-q)$ such that $m + 2c = x$. Alongside the unique cycle, the integer x takes successively all values between 1 and $|q| - 1$.

The question is how fast does the state reach this unique cycle. Consider any pair of states (s, s') , that correspond to the same value x :

$$x = m + 2c = m' + 2c'.$$

In section 5.3, we show that these two states converge to the same internal state with very high probability, after 128 advances of the FCSR in average.

Now suppose that s was a state of the unique cycle. Since s' becomes synchronized with s in 128 advances in average, we know that **any state is mapped to a state of the unique cycle after 128 advances in average**. Provided we discard the first keystream bits, we can therefore assume that all internal states we encounter are part of the unique cycle. So, the real entropy of the internal state is only of $\log_2(q-1) \simeq 128$ bits.

5.3 Resynchronization in Probabilistic Time

We consider two states which satisfy $x = m + 2c = m' + 2c'$. We say that a cell in the main register is **synchronized** when its value is the same for these two states. We want to determine how many advances are necessary before all cells in the two states become synchronized.

First, we observe that the value of x remains identical for both states after any number of iterations. This follows immediately from the definition of a FCSR. Consequently, the feedback bit, *i.e.* the rightmost bit in the main register is always synchronized. Indeed, $x \bmod 2 = m \bmod 2 = m' \bmod 2$.

In addition, suppose that the i leftmost cells of the main register are synchronized at time t . Then, it is clear that they remain synchronized for all $t' > t$ (the only propagation from right to left in the FCSR comes from the feedback bit, which is synchronized). Moreover, if we are lucky the cell m_{127-i} may become synchronized at time $t + 1$.

If $(127 - i) \notin I_d$, there is no carry cell and $m_{127-i}(t + 1) = m_{127-i+1}(t)$. Therefore m_{127-i} **becomes synchronized at time** $t + 1$.

Otherwise, if $(127 - i) \in I_d$, then there is a carry cell and

$$m_{127-i}(t + 1) = m_{127-i+1}(t) \oplus c_{127-i}(t) \oplus m_0(t).$$

Remember that both bits m_0 and $m_{127-i+1}$ are already synchronized. Roughly, m_{127-i} **becomes synchronized with probability** 0.5 depending on the behavior of the carry register.

Therefore, the whole register is likely to be eventually synchronized. The average time for this to happen is

$$T = \sum_{i \notin I_d} 1 + \sum_{i \in I_d} 2 = 68 + 60 \times 2 = 188$$

advances. Actually, a similar property holds if we consider the rightmost cells, so the synchronization goes in both directions. Generally, it takes less than 128 advances. The phenomenon has been confirmed by our practical simulations.

5.4 Trade-off Attack with Static Filters

Suppose that the output taps are known. Then the internal state has a "real" entropy of 128 bits only. This allows to apply trade-off attacks on this reduced space instead of the full space of internal states. We propose to apply a simple variant of Babbage-Golic's attack [5, 12]. It proceeds in two steps:

– **The online phase**

Observe $D = 2^{64}$ keystream bits produced from an unknown secret key K . Discard the first 128 bits (hence it is very likely that we are not in the unique cycle of the state-update function, according to section 5.3), and store in a table each window of 128 keystream bits. There are $D - 128 - 127$ such windows. Each one should uniquely correspond to some value of the internal state³.

³ A chance remains that from two different states, the same window of 128 keystream bits is produced. It is customary to take an extra margin by considering windows slightly larger than 128 bits.

– **The offline phase**

We want to enumerate at random about 2^{64} internal states located on the unique cycle. For each state, we generate 128 keystream bits, and search for a match with the $D - 255$ windows of the online phase. As soon as a match is found, the attack stops.

There are two possibilities to make this enumeration. First, we can just pick some internal states at random and apply 128 advances of the FCSR. Then it is very likely that we select states of the unique cycle. The complexity of the enumeration is 128×2^{64} .

An improved solution is to pick internal states which are located at regular interval on the unique cycle :

- First, pick a state s which is guaranteed to be on this cycle (for instance, $x = 1$ is good since it has a unique representation, $m = 1$ and $c = 0$).
- Next, compute the 2^{64} -th iterate of the state-update function, starting from s . We set the new value of s to this result. Computing such iterates can be done efficiently.
- Generate 128 keystream bits from the actual state s , and search for a match with the $D - 255$ windows of the online phase. If a match is found, we stop. Otherwise, we repeat the previous step.

Hence we need to compute about 2^{64} times some 2^{64} -th iterates of the state-update. Since iterates can be computed efficiently, the complexity of this enumeration is about $T = 2^{64}$. Moreover it is guaranteed that all enumerated states are on the unique cycle.

When a match is found, we learn the value of the state at some time t : $s(t) = (m(t), c(t))$. The state-update is not invertible, however, we can compute $p_t = m(t) + 2c(t)$ and backtrack to the values of $p_{t'}$ for $t' < t$ using equation 2. Since $p_0 = m(0) + 2c(0) = K + 2IV$, it is easy to deduce the value of the secret key.

Other choices of D are possible. They result in different trade-offs between time, data and memory complexity. We propose to choose $D = 2^{64}$, therefore all complexities (time, memory and data) are of the order of 2^{64} . This attack is applicable to **F-FCSR-SF1** and **F-FCSR-SF8**.

5.5 Trade-off Attack with Dynamic Filters

Now suppose that the output taps are unknown. We can no longer apply the previous attack, since the size of the internal state is artificially increased by 128 bits. We would need to guess the output taps in order to apply the trade-off attack.

In the case of **F-FCSR-DF8**, each keystream bit depends on only 16 output taps. We discard the keystream bits such that $i \neq 0 \pmod{8}$. We need to guess only 16 taps, in order to apply the previous attack. The extra cost is a factor 8 in data and memory and 2^{16} in time. Therefore the resulting time complexity is about 2^{80} , while the data and memory complexity become about 2^{67} . Other trade-offs could be envisaged with more time and less data/memory.

6 Conclusion

We demonstrated several attacks against the new F-FCSR family of stream ciphers. The first result is a resynchronization attack which breaks all algorithms in the family which support IV's. The complexity corresponds to about 2^{15} pairs of chosen IV's.

The second result concerns trade-off attacks which breaks several algorithms in the family with time, memory and data of the order of 2^{64} . These attacks are faster than generic attacks and demonstrate some undesirable properties of the underlying primitive, the Feedback with Carry Shift Register (FCSR).

While these attacks do not totally discard the use of FCSR in stream ciphers, they illustrate some important weaknesses in the F-FCSR family. The European ECRYPT project [10] recently launched a call for primitives in the field of stream ciphers. As a result, 34 new algorithms were proposed and are currently under analysis [11]. Arnault and Berger proposed two algorithms called F-FCSR-8 and F-FCSR-H, which are new variants of the F-FCSR family [4]. In a paper posted on the ECRYPT server, we observed that these algorithms are vulnerable to similar attacks than those described here, and we also pointed out new weaknesses [15].

References

1. F. Armknecht, J. Lano, and B. Preneel. Extending the Resynchronization Attack. In H. Handschuh and A. Hasan, editors, *Selected Areas in Cryptography – 2004*, volume 3357 of *Lectures Notes in Computer Science*, pages 19–38. Springer, 2005.
2. F. Arnault and T. Berger. A new class of stream ciphers combining LFSR and FCSR architectures. In A. Menezes and P. Sarkar, editors, *Progress in Cryptology – INDOCRYPT'02*, volume 2551 of *Lectures Notes in Computer Science*, pages 22–33. Springer, 2002.
3. F. Arnault and T. Berger. F-FCSR: design of a new class of stream ciphers. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption – 2005*, volume 3557 of *Lectures Notes in Computer Science*, pages 83–97. Springer, 2005.
4. F. Arnault, T. Berger, and C. Lauradoux. Description of F-FCSR-8 and F-FCSR-H stream Ciphers. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/008, 2005. <http://www.ecrypt.eu.org/stream>.
5. S. Babbage. A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers. In *European Convention on Security and Detection*, volume 408. IEE Conference Publication, may 1995.
6. E. Barkan, E. Biham, and N. Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. In D. Boneh, editor, *Advances in Cryptology – Crypto'03*, volume 2729 of *Lectures Notes in Computer Science*, pages 600–616. Springer, 2003.
7. A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In T. Okamoto, editor, *Advances in*

- Cryptology – Asiacrypt'00*, volume 1976 of *Lectures Notes in Computer Science*, pages 1–13. Springer, 2000.
8. N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In E. Biham, editor, *Advances in Cryptology – Eurocrypt'03*, volume 2656 of *Lectures Notes in Computer Science*, pages 345–359. Springer, 2003.
 9. J. Daemen, R. Govaerts, and J. Vandewalle. Resynchronization Weaknesses in Synchronous Stream Ciphers. In T. Helleseht, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lectures Notes in Computer Science*, pages 159–167. Springer, 1994.
 10. ECRYPT Network of Excellence in Cryptology
<http://www.ecrypt.eu.org/index.html>.
 11. eSTREAM - The ECRYPT Stream Cipher Project
<http://www.ecrypt.eu.org/stream/>.
 12. J. Golić. Cryptanalysis of Alleged A5 Stream Cipher. In W. Fumy, editor, *Advances in Cryptology – Eurocrypt'97*, volume 1233 of *Lectures Notes in Computer Science*, pages 239–255. Springer, 1997.
 13. J. Golic and G. Morgari. On the Resynchronization Attack. In T. Johansson, editor, *Fast Software Encryption – 2003*, volume 2887 of *Lectures Notes in Computer Science*, pages 100–110. Springer, 2003.
 14. M. Hellman. A Cryptanalytic Time-Memory Tradeoff. *IEEE Transactions on Information Theory*, 26(4):401–406, July 1980.
 15. E. Jaulmes and F. Muller. Cryptanalysis of ECRYPT Candidates F-FCSR-8 and F-FCSR-H. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/046, 2005. <http://www.ecrypt.eu.org/stream>.
 16. A. Klapper and M. Goresky. 2-adic shift registers. In R. Anderson, editor, *Fast Software Encryption – 2005*, volume 809 of *Lectures Notes in Computer Science*, pages 174–178. Springer, 1994.
 17. A. Klapper and M. Goresky. Cryptanalysis based on 2-adic rational approximation. In D. Coppersmith, editor, *Advances in Cryptology – Crypto'95*, volume 963 of *Lectures Notes in Computer Science*, pages 262–274. Springer, 1995.
 18. I. Mantin and A. Shamir. A Practical Attack on Broadcast RC4. In M. Matsui, editor, *Fast Software Encryption – 2001*, volume 2355 of *Lectures Notes in Computer Science*, pages 152–164. Springer, 2002.
 19. W. Meier and O. Staffelbach. Fast Correlations Attacks on Certain Stream Ciphers. In *Journal of Cryptology*, pages 159–176. Springer, 1989.
 20. T. Siegenthaler. Correlation-immunity of Nonlinear Combining Functions for Cryptographic Applications. In *IEEE Transactions on Information Theory*, volume 30, pages 776–780, 1984.
 21. B. Zhang, H. Wu, D. Feng, and F. Bao. Chosen Ciphertext Attack on a New Class of Self-Synchronizing Stream Ciphers. In A. Canteaut and K. Viswanathan, editors, *Progress in Cryptology – INDOCRYPT'04*, volume 3348 of *Lectures Notes in Computer Science*, pages 73–83. Springer, 2004.