Description of F-FCSR-8 and F-FCSR-H stream Ciphers

F. Arnault, T.P. Berger, C. Lauradoux

Introduction

Our Filtered FCSR stream ciphers are based on a very simple mechanism: the output is obtained by filtering internal states of an FCSR automaton using linear Boolean functions. A full description of the method is given in Section 1. Some more extensive documentation could be find in enclosed references [1, 2, 3].

Our proposal contains two variants of such F-FCSR stream ciphers:

F-FCSR-8: this version satisfies the requirements of Profile 1 (key length: 128 bits, IV length: up to 128 bits). However its most efficient use would be in a mixed software/hardware implementation. More precisely, a part of the setup phase (the selection of the filter) would gain to be software implemented, while the remaining of the primitive (in particular the keystream generation) would gain to be hardware implemented. In this case the efficiency of F-FCSR-8 would be even greater than the purely hardware version we propose below in this same document, because an FCSR of shorter length (128) is used.

In a pure software implementation, the performances of F-FCSR-8 are given in Section 5.

F-FCSR-H: this version satisfies the requirements of Profile 2: key length: 80 bits, IV length: up to 80 bits.

In this version, the filter is fixed and never changed. For that reason, we have chosen an FCSR automaton of length 160 in order the cost of Time/Memory/Data tradeoff attacks be as expensive as an exhaustive keysearch.

A complete hardware and software description of the primitives F-FCSR-8 and F-FCSR-H are given in Section 1.

The security level of the two proposals is expected to be equal to an exhaustive keysearch.

Details about security analysis on Filtered FCSR generators are given in Section 2 and in the enclosed papers [1, 2, 3].

The main advantage of the use of filtered FCSRs is the combination of two fact:
1) Sequences generated by FCSR automata are well-known, with proved properties, which helps to evaluate security.
2) FCSR automata with linear filters are very fast, especially in hardware, but also in software (See Section 3).

Due to the simplicity of these stream ciphers, the design rationale is limited to
- the choice of the FCSR automaton, that is the choice of the size of the register and of the connection integer q.
- the choice of the filter for F-FCSR-H
- the construction of the filter for F-FCSR-8
- the design of the key setup and change of IV procedures.

For details, see Section 4

F-FCSR designs are very suitable for hardware applications since they are very easy to describe and very efficient (cf Section 1.1.2). The number of gates used is small enough to allow integration of F-FCSR-H or F-FCSR-8 designs in embedded system.

Hardware and software performances are given in Section 5.

1 Description of the primitives

The proposed stream ciphers are additive ones : the key K and the IV are used to produce a pseudorandom stream of binary digits of same length as the plaintext. Encryption is done by combining the pseudorandom stream with the plaintext stream using the XOR function. Decryption is done by combining the pseudorandom stream with the ciphertext stream.

In the two proposals, the pseudorandom stream is obtained using a filtered FCSR automaton. A FCSR automaton has two registers: a main register M which stores n bits values, and a carries register C which stores ℓ bits values. For Proposal F-FCSR-8, $n = 128$ and $\ell = 65$. For Proposal F-FCSR-H, $n = 160$ and $l = 82$.

For each of the proposals, we will give a complete description and details about the following procedures:

a. Key setup This procedure takes as input a key K of size k and outputs a value M_{init} of bitsize n used to initialize the main register (this M_{init} will be recalled before each change if IV if any). The procedure outputs also a filter F of bitsize n that will be constant while the key will be unchanged. It is called only once for each new key. For F-FCSR-8, we will have $k = n = 128$.

b. IV setup (or change of IV) This procedure will be used just after Key setup, and also when change of IV occurs. It takes the value M_{init} computed by Key setup and the IV as inputs. It sets the FCSR automaton (both registers M and C) in a state just ready for beginning extraction of pseudorandom stream.

These two procedures will be merged in only one [Key+IV setup] for F-FCSR-H. The resulting procedure will be used at each change of Key and/or IV. In this version, $k = 80$ and $0 \leq v \leq 80$ (for example $v = 64$ or 32).

c. Extraction of the pseudorandom stream This procedure is iterated (after [a] and [b] have been run) while pseudorandom data is needed. It can be described as two steps. First, the automaton is clocked (the transition function is applied). Then a pseudorandom byte is extracted by filtering the contents of the cells of the automaton.

Before the description of each version, we will give a description of a FCSR automaton and the conditions required for the parameter q of this automaton.

1.1 FCSR automaton

Detailed descriptions can be found in [1, 2, 3].

A Feedback with Carry Shift Register (FCSR) is an automaton which computes the binary expansion of a 2-adic number p/q, where p and q are some integers, with q is odd. We will assume that $q < 0 < p < |q|$. The size n of the FCSR is such that $n + 1$ is the bitlength of $|q|$.

In our applications, p depends on the secret key (and the IV), and q is a public parameter. The choice of q induces many properties of the keystream. The most important one is that it completely determines the length of the period of the keystream. The conditions for an optimal choice are:

Conditions 1

- *q is a (negative) prime of bitsize $n + 1$.*
- *The order of 2 modulo q is $|q| - 1$.*
- *$T = (|q| - 1)/2$ is also prime.*
- *Set $d = (1 + |q|)/2$. The Hamming weight $W(d)$ of the binary expansion of d is not too small. Typically, $W(d) > n/2$.*

1.1.1 Software description of the transition function

The FCSR automaton contains two registers (sets of cells): the main register M and the carries register C.

The main register M contains n cells. We denote m_i $(0 \leq i \leq n - 1)$ the binary digits contained in these cells and we call the integer $m = \sum_{i=0}^{n-1} m_i 2^i$ the content (or state) of M.

Let d be the positive integer $d = (1 - q)/2$ and $d = \sum_{i=0}^{n-1} d_i 2^i$ its binary expansion. The carries register contains l cells where $l + 1$ is the number of nonzero d_i digits. More precisely, the carries register contains one cell for each nonzero d_i with $0 \leq i \leq n - 2$. We denote c_i the binary digit contained in this cell. We also put $c_i = 0$ when $d_i = 0$ or when $i = n - 1$. We call the integer $c = \sum_{i=0}^{n-2} c_i 2^i$ the content (or state) of C. The Hamming weight of the binary expansion of c is at most l.

The transition function can be described by

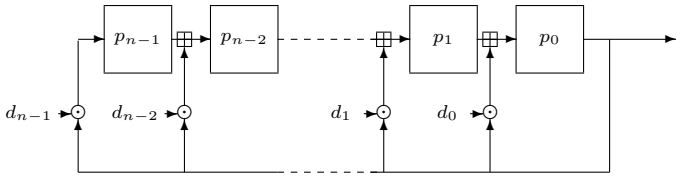
$$\begin{aligned} m(t+1) &:= (m(t) \operatorname{div} 2) \oplus c(t) \oplus m_0(t)d \\ c(t+1) &:= (m(t) \operatorname{div} 2) \otimes c(t) \oplus c(t) \otimes m_0(t)d \oplus m_0(t)d \otimes (m(t) \operatorname{div} 2) \end{aligned}$$

where $\oplus$ denotes bitwise XOR, $\otimes$ denotes bitwise AND, and $\operatorname{div} 2$ is a just a shift to the right.

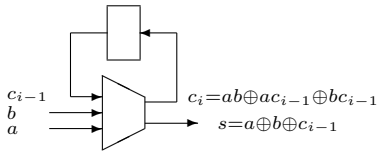
Note that $m_0(t)$ is the least significant bit of $m(t)$. The integers $m(t)$, $c(t)$ and d are integers of bitsize n (or less).

1.1.2 Hardware description of the transition function

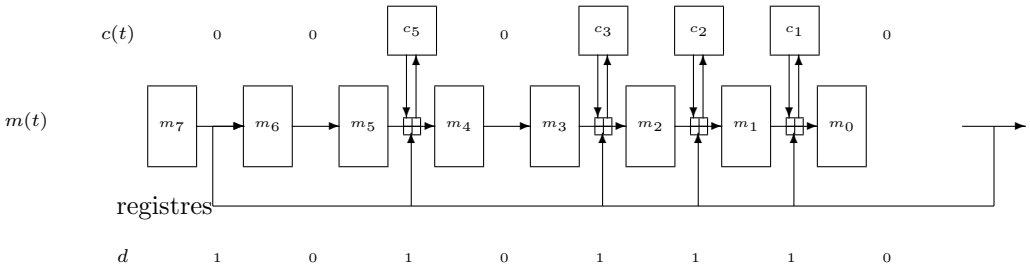
With the same notations, the hardware description of the FCSR generator is



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



As an example, if $q = -347$, so $d = 174 = 0$xAE, $n = 8$ and $l = 4$, we obtain the following diagram:



1.2 Filtering

We extract each pseudorandom bit from the state of the main register of the FCSR automaton using a filter. This filter describes which cells are selected to produce the pseudorandom bit. In order to obtain a byte in output, eight one bit subfilters are used to extract the output byte after each transition of the automaton.

1.2.1 Principle of one bit filtering

The filter F is a bitstring $(f_0, \ldots, f_{n-1})$ of length n (or equivalently the integer $\sum_{i=0}^{n-1} f_i 2^i$). The output bit is obtained by computing the weight parity of the bitwise AND of the state M of the main register and of the filter F:

$$\text{Output bit} := \bigoplus_{i=0}^{n-1} f_i m_i.$$

Or, equivalently:

$S = M \otimes F$

Output bit := parity(S)

1.2.2 Byte filtering

This method is very similar to bit filtering.

The filter F is also a bitstring $(f_0, \ldots, f_{n-1})$ of length n (which is a multiple of 8). It splits into 8 subfilters $F_0, \ldots, F_7$ each defined by

$$F_j = \sum_{i=0}^{n/8-1} f_{8i+j} 2^i.$$

Each subfilter F_j selects some cells m_i in the main register among the ones satisfying $i \equiv j$ modulo 8. The parity of the binary word obtained gives one pseudorandom bit :

$$\text{bit } j \text{ of output byte} := \bigoplus_{i=0}^{n/8-1} f_{8i+j} m_{8i+j}.$$

As there are 8 subfilters, we get 8 bits at each transition of the automaton.

This procedure can be described equivalently as follows. The filter F and the state of M are combined with the AND function. The result is split in $n/8$ bytes. The pseudorandom byte is obtained by XORing these $n/8$ bytes:

$S := M \otimes F$
Define S_i by $S = \sum_{i=0}^{n/8-1} S_i \cdot 256^i$, with $0 \leq S_i \leq n/8 - 1$
Output byte $:= \bigoplus_{i=0}^{n/8-1} S_i$.

Note that it is faster to extract a byte than a single bit.

1.3 F-FCSR-8: Profile 1, output 1 byte per round

This proposal uses keys of length $k = 128$ and an IV of length $v = 128$ or 64 (any length $v \leq 128$ can be used). An IV of value 0 can be used as a default if no value is provided by the application.

According to Conditions 1 we choose for q the following number

$$-q = 49387740064344360888838204820078394382 7$$

as the public parameter of the automaton. The corresponding bitstring $d = (|q| + 1)/2$ which describes the positions of the carries cells is

$$d = (\text{B9C6A9EA B7E25FD6 9E86369A 1856EC4A})_{16}.$$

Its Hamming weight is 69 and there are $\ell = 68$ cells (the Hamming weight of $d^* = d - 2^{127}$) in the carries register and $n = 128$ cells in the main register.

The filter depends on the key. To avoid potentially weak cases, we need a quality test on the filter. This test is provided by the following function which takes as input a bitstring of length 128 and outputs True if it is suitable as a filter for our application. Else it outputs False.

Function GoodFilter (F)
Define the 8 subfilters $F_0, \ldots, F_7$, each of bitlength 16, by

$$F_j = (f_j, f_{8+j}, f_{16+j}, \ldots, f_{8j+j}, \ldots, f_{120+j}).$$

IF any one of the subfilters F_j has an Hamming weight < 3 THEN output False;
Output True (in all other cases).
End Function

a. Initial Setup (Input a key K of 128 bits)

$M := K$ (put the key in the main register)
While NOT GoodFilter(M) Repeat
$C := 0$
Clock the FCSR automaton 6 times.
End while.
$F := M$ (The content of main register will be the filter)
$C := 0$ (Clear the carries)
Clock the FCSR automaton 128 times (Wait for diffusion of the key)
$M_{init} := M$ (Save the content of the main register)

b. Change of IV (Input: an IV of bitsize $v \leq 128$)

$M := M_{init}$
$C := 0$ (Clear the carries)
If $v \leq 64$ Then $\text{IV}_1 := (0^{64-v} \| \text{IV})$ (complete IV with zeroes)
Else ($65 \leq v \leq 128$) Do $(\text{IV}_2 \| \text{IV}_1) := (0^{128-v} \| \text{IV})$ (complete IV with zeroes and split it in two 64 bits strings)
$C := (0^4 \| \text{IV}_1)$ (Put IV_1 in the 64 least significant bits of C)
Apply 64 times the transition function to the FCSR automaton.
If $65 \leq v$ Then
  $C := (0^4 \| \text{IV}_2)$ (Put IV_2 in the 64 least significant bits of C)
  Apply 64 times the transition function to the FCSR automaton.
End If
$M := M_{init}$ (Recall the content of M saved after phase [a])

c. Extraction of the pseudorandom stream We use the one byte filtering method described above, while pseudorandom data is needed. At each clock of the FCSR automaton, the content of the main register M is ANDed with the filter F:
$S = M \otimes F$
S is split in 16 bytes $S = \sum_{i=0}^{15} S_i 16^i$
The pseudorandom byte is the XOR of these bytes: Output byte $:= \bigoplus_{i=0}^{15} S_i$

The Initial Setup step is easier to implement in software (at least the filter quality check). Steps b and c would be extremely fast if implemented in hardware.

1.4 F-FCSR-H: Profile 2, output 1 byte per round

This second proposal uses keys of length 80 and IV of bitsize v with $32 \leq v \leq 80$. An IV of value 0 can be used as a default if no value is provided.

The FCSR length (size of the main register) is $n = 160$. The carries register contains $\ell = 82$ cells. The retroaction prime is

$$q = -199352459131827501532804161134421503646014008796 3$$

so addition boxes and carries cells are present at the positions matching the ones (except of the leading one) in the following 160 bits string (which has Hamming weight 83)

$$d = (1 + |q|)/2 = (\text{AE985DFF 26619FC5 8623DC8A AF46D590 3DD4254E})_{16}.$$

Filtering

To extract one pseudorandom byte, we use the static filter

$$F = d = (\text{AE985DFF 26619FC5 8623DC8A AF46D590 3DD4254E})_{16}$$

The filter F splits in 8 subfilters (subfilter j is obtained by selecting the bit j in each byte of F)

$$\begin{array}{ll} F_0 = (0011\ 0111\ 0100\ 1010\ 1010)_2, & F_4 = (0111\ 0010\ 0010\ 0011\ 1100)_2, \\ F_1 = (1001\ 1010\ 1101\ 1100\ 0001)_2, & F_5 = (1001\ 1100\ 0100\ 1000\ 1010)_2, \\ F_2 = (1011\ 1011\ 1010\ 1110\ 1111)_2, & F_6 = (0011\ 0101\ 0010\ 0110\ 0101)_2, \\ F_3 = (1111\ 0010\ 0011\ 1000\ 1001)_2, & F_7 = (1101\ 0011\ 1011\ 1011\ 0100)_2. \end{array}$$

Recall that the bit b_i (with $0 \leq i \leq 7$) of each extracted byte is expressed by

$$b_i = \bigoplus_{j=0}^{19} f_i^{(j)} m_{8j+i} \qquad \text{where } F_i = \sum_{j=0}^{19} f_i^{(j)} 2^j$$

and where the m_k are the bits contained in the main register.

a+b. Key+IV setup (Inputs a key K of length $k = 80$ and an IV of length $v \leq 80$)

1. The main register M is initialized with the key and the IV:

$$M := K + 2^{80} \cdot IV = (0^{80-v} \| \text{IV} \| K)$$

2. The carries register is initialized to 0 :

$$C := 0 = (0^{82})$$

3. The FCSR is clocked 160 times. (Output is discarded in this step)

c. Extraction of pseudorandom data After setup phase, the pseudorandom stream is produced by repeating the following process as many times as needed

- Clock the FCSR
- Extract one pseudorandom byte using filter F as described above.

2 Security properties

The expected security level of the two proposals are those of the exhaustive keysearch.

More details about security analysis on Filtered FCSR generators are given in the enclosed papers [1, 2, 3].

Resistance to generic attacks

- Statistical properties. There is not any known statistical bias on the pseudorandom sequences output by our filtered FCSR. We have check that they pass the Statistical Test Suite of the NIST.
- Linear complexity. Since 2-adic structure and quadratic automaton are not related with linear structure, we can expect that the linear complexity of our pseudorandom sequences satisfies the same distribution law as for a random sequence of period $|q|-1 > 2^{128}$. Experiments we have done support this assumption.
- 2-adic complexity. The 2-adic structure is broken by the linear properties of the filter function. Hence, we can expect that the 2-adic complexity of our pseudorandom sequences is high, as it is the case for random sequences of period $|q| - 1 > 2^{128}$.
- Algebraic cryptanalysis. The transition function of a FCSR automaton is quadratic and the filter function F_ℓ is linear.

  The algebraic equations are of the form $F_\ell(T_q^i(x)) = s_i$.

  At each iteration the degree of equations is increasing. It becomes computationally infeasible to obtain such equations for $i \geq 12$. To solve this system, we need at least 128 iterations.
- Correlation attack. There are two major obstacles to the adaptation of this attack on a filtered FCSR.

  The first one is the fact that the function used to filter the automaton is linear with l inputs. Such a function is $l-1$ resilient, that is balanced and without correlation between its output and any sum of at most $l-1$ of its inputs. In that situation, the attack is more difficult than the exhaustive one.

  The second one is the fact that the dependencies between the cells of an FCSR automaton are nonlinear, since the transition function is quadratic. It seems difficult to obtain linear dependencies.
- Time-Memory-Data tradeoff attacks. The size of the registers has been chosen in order the stream cipher to be resistant to these attacks.
  - If the filter F is known (as in F-FCSR-H), the number of states belonging to the main cycle of the automaton is $2T$, with $2^n < 2T < 2^{n+1}$, where $n+1$ is the size of q. We have chosen a size n for the FCSR which is twice the size of the key in order to thwart time-memory-data trade-off attacks. Precisely, we have chosen $n = 160$ for F-FCSR-H as the key size is $k = 80$.

– Suppose now that the filter is unknown from attacker. This is the situation for F-FCSR-8. The number of states belonging to the main cycle of the automaton is greater than 2^n for a fixed filter F. But there exists approximatively 2^n possible filters. This gives a total number of states of 2^{2n}. With $n = k$, the stream cipher remains resistant to time-memory-data trade-off attacks. This is why we have chosen $n = k = 128$.

- Distinguishing attacks. Distinguishing attacks can be based on the existence of linear relations between some internal states of the automaton which occur with a biased probability. Due to the existence of carries, we did not find such relations and we think that there are none.

  Moreover, when the filter is unknown, it is not clear that such relations would be useful for an attack.

  Probably more investigation would be interesting in order to confirm that distinguishing attacks on our stream ciphers are not easy, or else to find one.

Dedicated attacks

Some dedicated attacks are designed in [1, 2, 3]:

- 2-adic attack. This attack applies when the filter is known and when it (or its subfilters) is small (that is selects only cells which are near the lower weight end of the FCSR). Precisely, if k_F is the binary size of F, i.e. the least integer such that $F < 2^{k_F}$, the initial value of the main register can be recovered in $\mathcal{O}(2^{k_F} k_F k^2)$ operations. This attack does not apply for F-FCSR-8 as the filter is unknown. For F-FCSR-H, the filter we have chosen has a high value for k_F and this attack would be much more expensive than an exhaustive one.

- Cryptanalysis with multiple filters. If more than one filter are used starting from the same initial state of the automaton, it is possible to compute the filtered output that would be obtained with any filter in the subspace generated by the set of used filters. This is one of the justifications for the use of filters with distinct supports.

  Another remark about the design of our stream ciphers is the fact that it is not possible, by changing the IV, to change the filter without also changing the initial state of the main register. So an attacker cannot mount a multiple filters attack.

Weak keys

F-FCSR-8 Using the null key with a null IV makes the stream generation fail (no filter passing the quality test will be found). Since a quality test is used to select the filter, there are no other weak keys in view of the current cryptanalytic knowledge.

F-FCSR-H Using the null key (0^{80}) with a null IV makes the ciphertext identical to the plaintext. As the period of the FCSR automaton is $|q| - 1$, there are no other weak keys.

3 Strengths and advantages of the primitive

The main advantages in the use of filtered FCSRs is the fact that there are some proved results on the output sequence, and the proposed algorithms are efficient both in software and hardware implementations.

3.1 Advantages in the use of a FCSR automaton

- The period of the sequence is well known and proved.
- Except of the state 0, there is no other degenerated state.
- The outputsequence is non linear. It becomes possible to use a linear filter.
- A FCSR automaton is quadratic: it is intrinsically resistant to algebraic attacks.
- The hardware and software implementations of an FCSR automaton are simple, efficient and of low cost. They use similar techniques as for LFSR registers.

3.2 Advantages in the use of a linear filter

- Linear Boolean functions are the best ones from the point of view of correlation and non-resilience.
- Linear Boolean function are simple to implement both in hardware and software. In particular, it is possible to use Boolean functions with a large number of inputs (typically at least 64). They are cheap, both in time and in circuit size.

Both versions use 8 subfilters to output a whole byte at each round.

3.3 F-FCSR-8

The use of a filter which is key-dependent permits to expect a high level of security with a relatively small FCSR length.

In counterpart, it needs a quality test to choose the filter. This test would be quite expensive in size in an hardware realization. This is why we recommend to implement it in software.

3.4 F-FCSR-H

In this version, the filter is known and not key-dependent. This implies that the size of the main register must be twice the keysize. However, the use of a fixed filter avoids the requirement for a quality test. Hence a pure hardware implementation of the stream cipher remains very cheap.

4 Design rationale

4.1 Choice of the connection integer q of the FCSR automaton

Recall the conditions the integer q must satisfy:

- q is a negative prime of size $n+1$.
- The order of 2 modulo q is $|q|-1$.
- $T=(|q|-1)/2$ is a prime.
- If $q=1-2d$, the Hamming weight of the binary expansion of d is not too small. Typically, $W(d)>n/2$.

The first one is simply to be able to compute the 2-adic integer p/q, with $0<p<-q$, with registers of size n. Under this constraint, the output is strictly periodic without preperiod.

The second condition implies that the output sequence is periodic with a maximal period $2T=|q|-1$.

When we filter the internal states of the automaton, we perform the XOR of some sequences of period $2T$. The condition T is a prime ensures that the period of the filtered output is at least T.

The weight $W(d)$ corresponds to the number of carries memories: it contributes to the quadratic part of the automaton. A large value ensures a good diffusion of the quadratic properties and avoids linear attacks.

The proposed connection integers are of size respectively 129 and 161 bits. They are chosen randomly between the integers satisfying Conditions 1. They could be replaced by any prime satisfying these.

4.2 Choice of the filter

Known filter: F-FCSR-H

In F-FCSR-H, the filter is known. We choose for F the integer d, since each filtered cell of the main register is separated from the other by at least a carry cell. This insures that the 2-adic fractions corresponding to the filtered cells corresponds to different parts of the whole period (typically about 2^{127} bits) of the FCSR (see [1] for more details).

Unknown filter: F-FCSR-8

The dedicated attack described in 3.2.1 of [3] is not possible. The remaining problem is the case of degenerated filters, i.e. $F=0$, $F=2^i$ or $F=2^i+2^{i+j}$ with no carry between i and $i+j$, i.e. for any k, $i\leq k<i+k$, $d_k=0$. The quality test on the filter ensures to avoid such situation.

4.3 Key setup and change of IV procedures

We choose to avoid as far as possible other functions than the FCSR automaton and the filter. It is why we use the automaton to expand the key and diffuse the IV.

Key setup procedure for F-FCSR-8

The key setup procedure is constituted of two distinct parts: the first one is used to obtain a non-degenerated filter F.

The second part is devoted to the construction of the initial state M_{init} of the main register. This value is derived from the key K (and then from the filter F), but theses will not be easy to exploit in algebraic or related attacks.

Instead of using an external mechanism, we use 128 rounds of the FCSR automaton: the output M_{init} is a function of the key K. However each of the 128 coordinate functions are of algebraic degree 127 or 128, with no particularly known properties.

Change of IV procedure for F-FCSR-8

In this procedure, the IV value are put into the carry cells to ensure a rapid diffusion on the main register. This diffusion is made by 64 rounds of the automaton. If the size of the IV is greatest than 64 bits, this procedure is repeated twice.

Key setup and change of IV procedures for F-FCSR-H

For the F-FCSR-H stream cipher, there is no distinct procedure between key setup and change of IV. The initial value of the main register is obtained by concatenation of the IV value (eventually 0) and the 80 bits key. To be sure that each bit of the IV and the key is well diffused, we recommend 128 initialization rounds of the FCSR automaton before outputting the sequence.

5 Computational efficiency in hardware and software

Software efficiency was not the main objective of our proposal, as attested by Figure 2. Any platform that requires to split the register involved in the Galois setup of the FCSR is not suitable for obtaining fast encryption with FCSR. The Fibonacci setup is not possible since it is slower and more complicated than the Galois setup. The only way to achieve high throughput with FCSR is the use of SIMD instructions, like Altivec or SSE2. Clocking the FCSR and the filtering function are quite simple with SIMD instructions (Figure 1 provides the code for the clocking function). But, the drawback is that, in this case, encrypting 128-bit data blocks is suitable in order to avoid unaligned memory access. In addition to our reference implementation, we also provide an Altivec evaluation version of F-FCSR-8. As expected, the Altivec implementation is the most efficient one and it achieves unexpected performance for software implementation (20 cycles per Byte). One other advantage is that it does not use table like Snow 2 then memory parameter are not very important for data encryption. SIMD instructions are only required.

Unfortunately, the IV insertion is still complicated and cannot be improved with SIMD instructions.

FCSR leads to better performance for hardware applications. One major strength of the F-FCSR-H design is its short critical path (virtually only one 1 flip flop and 1 or 2 LUT). E0 or A5/1 have the same advantage but they only output 1 bit per cycle whereas F-FCSR-H outputs 1 byte. This fact is confirmed by our hardware implementation on a low cost FPGA. The VHDL description is very easy to understand and is not difficult to apply to F-FCSR-8 or to any variant (e.g. with a smaller or longer FCSR or with another filtering function). The only drawback of FCSR in hardware conception is the fan-in fan-out problem. The feedback cell of the register has to be sent to 83 online adders. This implies that the feedback cell and its predecessors have to be replicated many times. Our F-FCSR-H design is expected to use 243 Flip-Flop cells; 10 additional Flip-Flop are required for the replication of the last cell of the FCSR.

Note that the speed of the F-FCSR-8 hardware implementation is the same as F-FCSR-H, that is 623 Mb/s.

```
/* vector unsigned char define a 128 bit word decomposed in 16 sub-word of length 8 */
vector unsigned char buffer,feedback;

/* bit expansion for feedback computation */
feedback = vec_splat( shiftRegister , 15 );
feedback = vec_sl( feedback , EXPAND );
feedback = vec_sra( feedback , EXPAND );
feedback = vec_and( feedback , RETROACTION );

/* Shift the register */
shiftRegister = vec_srl( shiftRegister , SHIFT );

/* Compute the next state of the register */
buffer = vec_xor( shiftRegister , carryRegister );
carryRegister = vec_and( shiftRegister , carryRegister );
carryRegister = vec_xor( carryRegister , vec_and( buffer , feedback ) );
shiftRegister = vec_xor( buffer , feedback );
```

Figure 1: Altivec code for one step of FCSR

CISC target	parameters			performance			
	Frequency	L2 Cache Size	Compiler	Speed	Code	IV loading	Key loading
Pentium 3	800 Mhz	256KB	GCC 3.2.2	92 cycles/B	7.5 KB	10700 cycles/IV	10200 cycles/Key
Pentium 4	2.3 Ghz	512KB	GCC 3.2.2	94 cycles/B	9.1 KB	11600 cycles/IV	11300 cycles/Key
Pentium 4	2.6 Ghz	512KB	GCC 3.2.2	106 cycles/B	9,2 KB	13400 cycles/IV	13600 cycles/Key
Pentium 4	3.2 Ghz	1MB	GCC 3.2.2	101 cycles/B	5.7 KB	11400 cycles/IV	11700 cycles/Key

RISC target	parameters			performance			
	Frequency	L2 Cache Size	Compiler	Speed	Code	IV loading	Key loading
PPC 7457	1.2 Ghz	512 KB	GCC 3.3.0	84 cycles/B	17 KB	8300 cycles/IV	17000 cycles/Key
Alpha EV67	1Ghz	64 KB	GCC 3.4.0	46 cycles/B	20 KB	6200 cycles/IV	5400 cycles/Key
PPC 7457 (altivec)	1.2 Ghz	512 KB	GCC 3.3.0	20 cycles/B	13 KB	– cycles/IV	– cycles/Key

Figure 2: F-FCSR-8 32-bit evaluation

Stream cipher	target	performance			
		Flip Flop	LUT	gate count	Speed
E0	Virtex-E V2600E-FG1156	300	-	1637	93 Mb/s
A5/1	Virtex-E HQ800	64	70	932	90 Mb/s
RC4	Virtex-E HQ800	279	-	12952	171 Mb/s
F-FCSR-H	Spartan2E 300e-6pq208	253	205	3254	623 Mb/s

Figure 3: Different speed of stream cipher on FPGA

Data for E0, A5/1 and RC4 are from "Energy, performance, area versus security trade-offs for stream ciphers" L. Batina, J. Lano, N. Mentens, S.B.Örs, B. Preneel, I. Verbauwhede, SASC 2004, Brugge.

References

[1] F. Arnault and T.P. Berger. Design and properties of a new pseudo-random generator based on a filtered FCSR automaton. Submitted.

[2] F. Arnault and T.P. Berger. Design of new pseudo random generators based on a filtered FCSR automaton. In *SASC*, State of the Art of Stream Ciphers Workshop, pages 109–120, Bruges, Belgium, October 2004.

[3] F. Arnault and T.P. Berger. F-FCSR: design of a new class of stream ciphers. To appear in *Fast Software Encryption, FSE'05*, Lecture Notes in Computer Science, Springer-Verlag, 2005.