Analysis of SSC2

Daniel Bleichenbacher[1] and Willi Meier[2]

[1] Bell Laboratories, Rm. 2A-366, 700 Mountain Av
Murray Hill, NJ 07974-0636, USA
bleichen@bell-labs.com
[2] FH Aargau
CH-5210 Windisch
meierw@fh-aargau.ch

Abstract. This paper analyses the stream cipher SSC2 [ZCC00]. We describe some weaknesses and attacks exploiting these weaknesses. The strongest attack needs about 2^{52} words of known key stream and has a time complexity of about 2^{75}.

1 Introduction

The stream cipher SSC2 has been proposed in [ZCC00]. It consists of a filter generator and a filtered lagged Fibonacci generator. The outputs of the two generators are XORed to give the key stream.

This paper investigates the strength of the stream cipher SSC2. We describe some weaknesses and some attacks based on those weaknesses. The strongest attack needs about 2^{52} words of known key stream and has a time complexity of about 2^{75}. Note, recently Hawkes, Quick and Rose have found a new attack that is faster than ours [HR01].

The outline of the paper is as follows. Section 3 analyses the linear feedback shift register. In Section 4 properties of the nonlinear filter are described. We compute the distribution of some carry bits, so that we can improve one of the attacks later.

In Section 6 properties of the lagged Fibonacci generator are discussed. Then we describe two different attacks on SSC2, exploiting different weaknesses of SSC2. Our first attack needs about 2^{52} known plaintext and 2^{75} time. This attack is divided into two sections. First, in Section 7 we describe how to find the internal state of the LFSR by exploiting the short period of the lagged Fibonacci generator. Then in Section 8 we cryptanalyse the lagged Fibonacci generator. Section 9 describes our second attack exploiting the bias in the lagged Fibonacci generator. One variant of this attack needs 2^{32} words of known plaintext, but has a time complexity of 2^{123}. In Section 10 we identify a bug in the frame key generation.

2 The Structure of SSC2

The stream cipher SSC2 as shown in figure 1 consists of a filter generator and a lagged Fibonacci generator. The word-oriented LFSR has 4 stages with each

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 219–232, 2002.

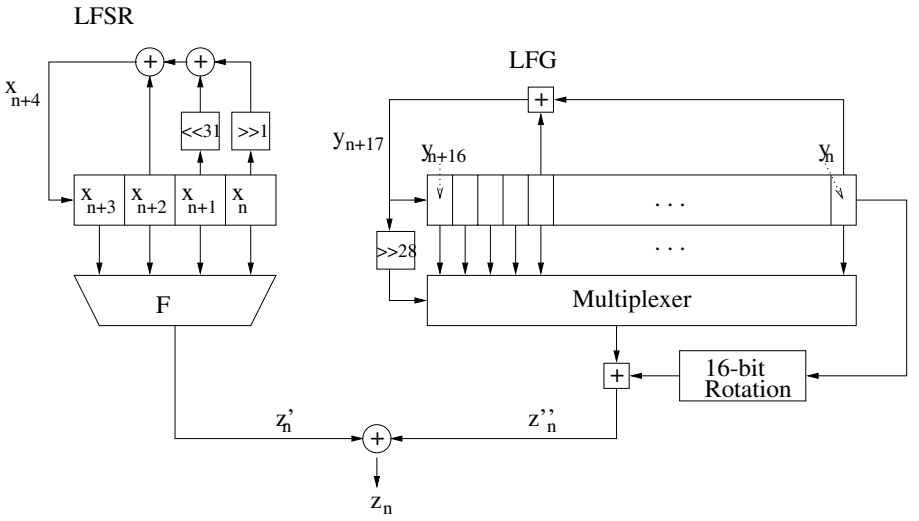

Fig. 1. The key stream generator of SSC2.

stage containing one word. It generates a new word and shifts out an old word at every clock. The nonlinear filter has the 4-word content of the LFSR as its input and a single word as output. The lagged Fibonacci generator has 17 stages and is also word-oriented. The word shifted out by the lagged Fibonacci generator is left-rotated 16 bits and then added to another word selected from the 17 stages. The sum is XOR-ed with the word produced by the filter generator to give a word of the key stream.

3 The Linear Feedback Shift Register

The LFSR in [ZCC00] is described using 4 registers where each of them is 32 bits long. The LFSR is regularly clocked. Clocking the LFSR turns a state $(x_{n+3}, x_{n+2}, x_{n+1}, x_n)$ into the state $(x_{n+4}, x_{n+3}, x_{n+2}, x_{n+1})$, where the 32-bit variables satisfy the linear recurrence relation

$$x_{n+4} = x_{n+2} \oplus (x_{n+1} \ll 31) \oplus (x_n \gg 1). \tag{1}$$

More convenient for this paper is another equivalent description of the LFSR using 8 registers where each of them is 16 bits long as shown in figure 2.

The linear feedback shift register can be described using 8 registers of 16 bit each. Cycling this register twice is equivalent to cycling the 32-bit register once. It can be observed that the least significant bits of the register determine the least significant bits of further states of the register. If we know the t least significant bits of $\mathrm{hi}(x_i), \mathrm{lo}(x_i)$ for all $n+1 \le i \le n+4$ then we can determine, the t least significant bits of x_n (except the bit that is not used in the nonlinear filter). For $v < t$ we can further determine the $t-v$ least significant bits of $\mathrm{hi}(x_i)$ and $\mathrm{lo}(x_i)$ for all $n+4v+1 \le i \le n+4v+4$.

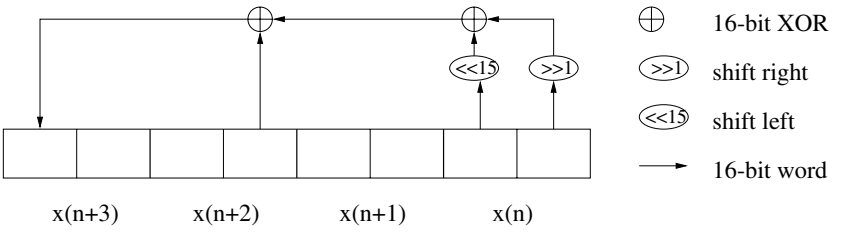

Fig. 2. The linear feedback shift register using 16-bit operations. One cycle in the 32-bit variant corresponds to two cycles in the 16-bit variant. It can be observed that the low significant bits of this register depend on low significant bits of previous states.

4 The Nonlinear Filter

SSC2 derives the sequence z'_n from the LFSR state $x_{n+3}, x_{n+2}, x_{n+1}, x_n$ by using a nonlinear filter F. The filter uses 16-bit rotation and therefore describing F using 16-bit arithmetic rather than 32-bit simplifies the analysis. Let $\mathrm{hi}(x)$ and $\mathrm{lo}(x)$ denote the high order 16 bits rsp. the low order 16 bits of a 32 bit integer. The original description of the nonlinear filter used 16-bit rotations, which we will denote by $(x \ggg 16)$. However, we can get rid of these rotations by describing the nonlinear filter using 16-bit block rather than 32-bit block. The resulting description of the nonlinear filter is given in Figure 3. Additional carry bits $c1, c3$ and $c5$ are necessary to simulate 32-bit additions with 16-bit operations.

Correlation Properties. Ideally, correlations of the output of a nonlinear filter to a linear function of the LFSR should be minimized. The nonlinear filter of SSC2 has some correlations. At the moment we do not know, whether such correlations can be used in an attack.

Property 1. *The nonlinear filter of SSC2 satisfies the following equation, which is a linear approximation.*

$$Prob\left(\mathrm{lsb}(\mathrm{hi}(z'_n)) = \mathrm{lsb}(x_{n+3}) \oplus \mathrm{lsb}(\mathrm{hi}(x_{n+1}))\right) = 7/12$$

The probability $7/12$ was estimated using heuristic assumptions about the independence of the inputs and was verified experimentally.

Philip Hawkes, Frank Quick and Greg Rose have recently presented some other correlation properties in [HR01]. They use these observations for a new attack that is faster than ours.

5 Approximation to the Nonlinear Filter

In the analysis we will use an approximation G to the nonlinear filter F. Essentially G tries to guess the carry bits, during the computation of F. The motivation for approximating F by G is the simplification that we get, because there are less dependencies in G. The disadvantage of using an approximation G,

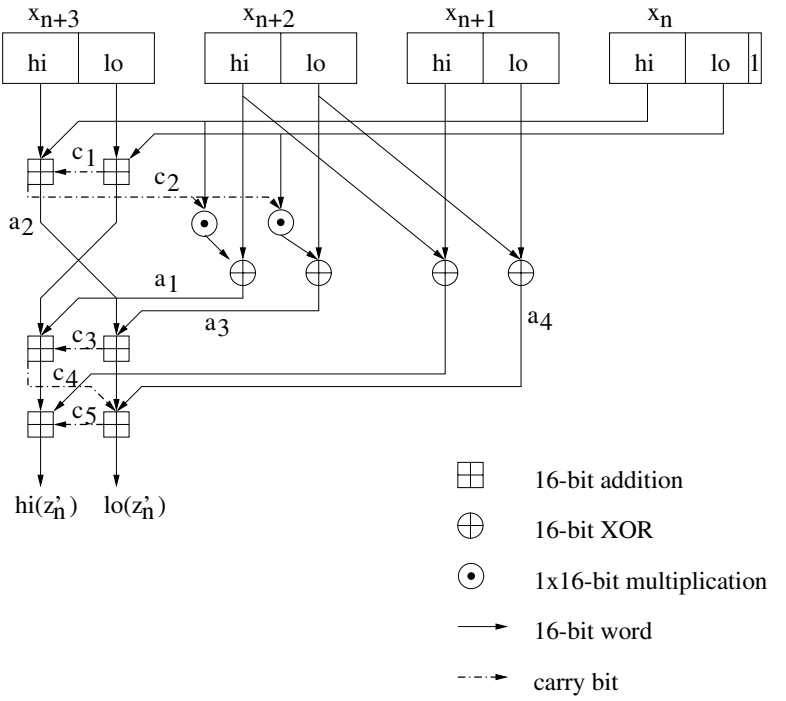

Fig. 3. This figure describes the nonlinear filter F of SSC2. Here, we give an equivalent description of the filter using 16-bit operations rather than the original 32-bit operations. This change is provided to simplify further analysis of SSC2, because in our description does not need any rotations of integers. Note, that this description needs 3 additional carry bits to simulate the 32-bit additions.

rather than F itself is the error that is introduced. In our attack we will choose sequences of key stream, and hope that the differentials between G and F cancel out. Therefore we will have to repeat our attack with different parts of the key stream. In this section we describe the function G and compute the probability distribution of the error $F \oplus G$.

The function G is described in Figure 4. The difference to F is that in G no carry bits are computed. Rather the effect of carry bits in F is simulated in G by the additional inputs $d_1 \in \{0, 1, 2\}, d_2 \in \{0, 1\}$ and $d_3 \in \{0, 1, 2\}$. Otherwise, F and G are identical. In particular, let c_1, c_2, c_3, c_4 and c_5 be the carry bits during the computation of $F(x_{n+3}, x_{n+2}, x_{n+1}, x_n)$ and let $d_1 = c_1 + c_4, d_2 = c_2$ and $d_3 = c_3 + c_5$. Then

$$F(x_{n+3}, x_{n+2}, x_{n+1}, x_n) = G(x_{n+3}, x_{n+2}, x_{n+1}, x_n, d_1, d_2, d_3).$$

There are two possible approaches for our attack. We could either try all possible combinations for the values d_1, d_2, d_3 or we can guess the most likely values and then try our attack on different sequences of the key stream until we

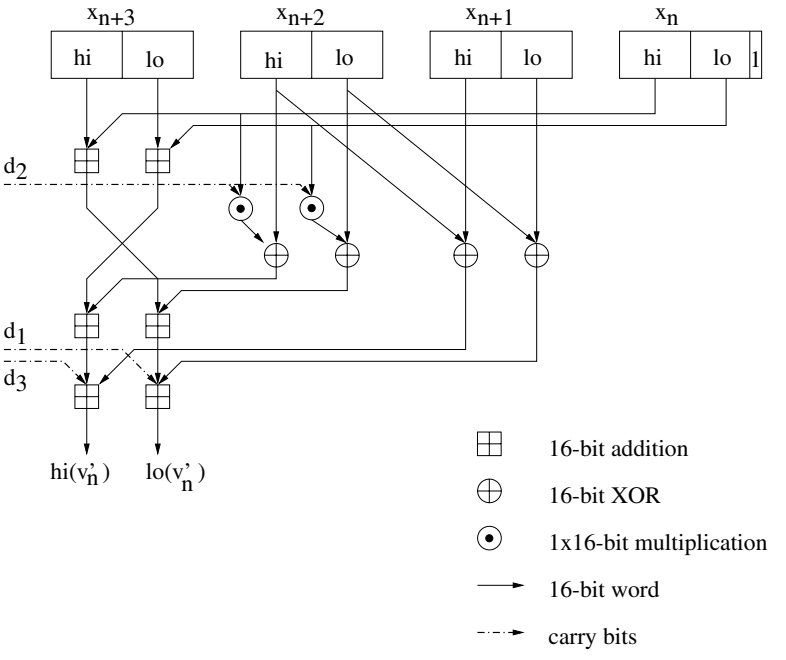

Fig. 4. A picture of the function $G(x_{n+3}, x_{n+2}, x_{n+1}, x_n, d_1, d_2, d_3)$, which is used to approximate F. G and F would be equivalent if the carry bits $d_2 = c_2$, $d_1 = c_1 + c_4$ and $d_3 = c_3 + c_5$ are used. However, we will use $d_1 = 1$ and $d_3 = 1$ and compute the distribution of the differential.

have found a sequence where our guess was correct. We will choose the second approach, because it is slightly more efficient. We will use $d_1 = d_3 = 1$ and only guess d_2. We are therefore interested in the differential

$$\Delta_n = F(x_{n+3}, x_{n+2}, x_{n+1}, x_n) \oplus G(x_{n+3}, x_{n+2}, x_{n+1}, x_n, 1, c_2, 1).$$

To analyse Δ_n we first investigate the distribution of $c_1 + c_4$ and $c_3 + c_5$ in the computation of F. Let

$$\begin{aligned} a_1 &= (c_2 \cdot \mathrm{hi}(x_n)) \oplus \mathrm{hi}(x_{n+2}) \\ a_2 &= \mathrm{hi}(x_{n+3}) + \mathrm{hi}(x_n) \bmod 2^{16} \\ a_3 &= (c_2 \cdot \mathrm{lo}(x_n)) \oplus \mathrm{lo}(x_{n+2}) \\ a_4 &= \mathrm{lo}(x_{n+1}) \oplus \mathrm{lo}(x_{n+2}). \end{aligned}$$

Then it can be observed that

$$c_1 + c_4 = \left\lfloor \frac{\mathrm{lo}(x_{n+3}) + \mathrm{lo}(x_n) + a_1}{2^{16}} \right\rfloor$$

and

$$c_3 + c_5 = \left\lfloor \frac{a_2 + a_3 + a_4 + c_4}{2^{16}} \right\rfloor$$

The expression for c_1+c_4 and c_3+c_5 are basically (ignoring the carry bit c_4) sums of three 16-bit integers. In [SM91, Section 2.2] the carry of sums of three integers has been analysed. Under the assumption that the summands are independent and uniformly distributed it follows that the probability of $c_1 + c_4 = 0$ and $c_1 + c_4 = 2$ is $\approx 1/6$ in each case, and the probability of $c_1 + c_4 = 1$ is $\approx 2/3$. $c_3 + c_5$ is similarly distributed.

We now compute the distribution of $\mathrm{hi}(\Delta_n)$ under the assumption that $d_2 = c_2$. Hereby, we have to distinguish two cases. Firstly, if $c_3+c_5 = 1$, then $\mathrm{hi}(\Delta_n) = 0$. Secondly, if $c_3 + c_5 \neq 1$ then $\mathrm{hi}(\Delta_n)$ is the XOR of two 16-bit integers with difference 1. Hence the distribution of $\mathrm{hi}(\Delta_n)$ is as follows

$\mathrm{hi}(\Delta_n)$	Probability
0	$\approx 2/3$
1	$\approx 1/6$
$2^k - 1$ for $k = 2, \ldots, 15$	$\approx 2^{-k}/3$
$2^{16} - 1$	$\approx 2^{-15}/3$

Similar arguments can be used to show that $\mathrm{lo}(\Delta_n)$ has the same distribution as $\mathrm{hi}(\Delta_n)$.

6 Lagged Fibonacci Generator

From [ZCC00] we briefly recall the description of the component of SSC2 consisting of a (suitably filtered) lagged Fibonacci generator. The generator is word oriented and is characterized by the linear recurrence relation

$$y_{n+17} = y_{n+12} + y_n \pmod{2^{32}}.$$

Since the corresponding feedback polynomial $x^{17} + x^5 + 1$ is irreducible over $\mathbb{Z}/(2)$ it follows that the sequence has a period of $(2^{17} - 1)2^{31}$ (or a divisor of this number and it achieves the maximal period if one of the least significant bits of $f_1, \ldots, f_{17}$ is 1). In SSC2 this generator is implemented with a 17-stage circular buffer B and two pointers s and r. Initially $B[17]$, $B[16]$,..., $B[1]$ are loaded with $y_0, y_1, ...y_{16}$, and s and r are set to 5 and 17, respectively. At every clock, a new word is computed by taking the sum of $B[r]$ and $B[s]$ mod 2^{32}. The word $B[r]$ is then replaced by the new word, and the pointers r and s are decreased by 1. A filter is used to produce the output. The output word z''_n is computed from the replaced word y_n and another word selected from the buffer B. The selection uses a multiplexer based on the most significant 4 bits of the newly generated word y_{n+17}. The output word z''_n is given by

$$z''_n = (y_n \ggg 16) + B[1 + ((y_{n+17} \gg 28) + s_{n+1} \bmod 16)] \bmod 2^{32}, \qquad (2)$$

where s_{n+1} denotes the value of s at time $n+1$ (note that $1 \leq s \leq 17$). Observe that this filter is not memoryless and has indeed a period 17, which has to be considered when computing the period of the z''_n. In [ZCC00] it is shown that

Property 2. *The period of* z''_n *divides* $17(2^{17}-1)2^{31}$.

The multiplexer can select the same elements of the Fibonacci generator several times in a row. The linear properties of the Fibonacci generator are only partially destroyed. It can be observed that the sequence S_n defined by

$$S_n = z''_{n+17} - z''_{n+12} - z''_n \bmod 2^{32}$$

has a nonuniform distribution. This distribution has the following property

Property 3. *Let*

$$\mathcal{A} = \{\texttt{0x0000000}, \texttt{0x00000001}, \texttt{0xffff0000}, \texttt{0xffff0001}\}.$$

Then $S_n \in \mathcal{A}$ *with probability about* 2^{-8}.

Property 3 can be explained as follows. The buffer B contains $y_{n+1}, \ldots y_{n+17}$ during the computation of z''_n. Therefore, there exists an integer $1 \le t_n \le 17$, such that

$$z''_n = y_{n+t_n} + (y_n \ggg 16) \bmod 2^{32},$$

We make the heuristic assumption that the high order bits of y_n are uniformly distributed and have verified this assumption with some experiments. With a probability of about 2^{-8} we have $t_n = t_{n+12} = t_{n+17}$. In this case it follows

$$\begin{aligned} S_n &\equiv z''_{n+17} - z''_{n+12} - z''_n \\ &\equiv y_{n+17+t_n} + (y_{n+17} \ggg 16) - y_{n+12+t_n} - (y_{n+12} \ggg 16) - y_{n+t_n} - (y_n \ggg 16) \\ &\equiv (y_{n+17} \ggg 16) - (y_{n+12} \ggg 16) - (y_n \ggg 16) \pmod{2^{32}}. \end{aligned}$$

This value can be evaluated by considering the high order 16 bits and low order 16 bits of y_i separately. We have

$$\begin{aligned} \mathrm{lo}(y_{n+17}) + c2^{16} &= \mathrm{lo}(y_{n+12}) + \mathrm{lo}(y_n) \\ \mathrm{hi}(y_{n+17}) + c'2^{16} &= \mathrm{hi}(y_{n+12}) + \mathrm{hi}(y_n) + c, \end{aligned}$$

where $c, c' \in \{0,1\}^2$ are two carry bits. Hence,

$$\begin{aligned} S_n &\equiv (\mathrm{lo}(y_{n+17}) - \mathrm{lo}(y_{n+12}) - \mathrm{lo}(y_n))2^{16} + \mathrm{hi}(y_{n+17}) - \mathrm{hi}(y_{n+12}) - \mathrm{hi}(y_n) \\ &\equiv -c'2^{16} + c \pmod{2^{32}} \end{aligned}$$

Inserting the four possible values for the pair of carry bits (c, c') results in the set $\mathcal{A}$. Hence, we have given an explanation for the high probability of $S_n \in \mathcal{A}$.

7 An Attack Based on the Small Period of the Lagged Fibonacci Generator

In this section we describe an attack against SSC2 that exploits the rather small period of the lagged Fibonacci generator. The attack requires about 2^{52} words

of known plaintext and the expected time complexity is about $T_0 2^{72.5}$, where T_0 is a constant denoting the work for the innermost loop of our attack. The attack find the internal state of the LFSR. After the internal state of the LFSR is found, we can apply the algorithm described in Section 9 to complete the attack.

The algorithm can be described as follows. Note that we describe the attack in a breadth first version, i.e., here guessing values means choosing all possible values and performing the remaining operations of the algorithm with all possible values in parallel. An implementation of this algorithm would most likely be depth first, but the breadth first version is easier to describe.

1. Let $w = 17(2^{17} - 1)2^{31}$ be the period of the lagged Fibonacci generator. Then compute
$$W_i = z_i \oplus z_{i+w} = z'_i \oplus z'_{i+w}.$$
2. For all i do
   2.1. Let $\mathcal{M} = \{i, i+1, i+2, i+3, i+w, i+w+1, i+w+2, i+w+3\}$.
   2.2. Let $\mathcal{M}' = \{i, \ldots, i+7, i+w, \ldots, i+w+7\}$.
   2.3. For all $m \in \mathcal{M}'$ guess a carry bit g_m for the computation of z'_m.
   2.4. For j from 2 to 16 do
      2.4.1. For all $m \in \mathcal{M}$ guess or extend previous guesses on $\mathrm{hi}(x_m)$, $\mathrm{lo}(x_m)$ to the j least significant bits.
      2.4.2. Use Equation (1) to compute the $j-1$ least significant bits of $\mathrm{hi}(x_m)$ and $\mathrm{lo}(x_m)$ for $m \in \{i+4, \ldots, i+7, i+w+4, \ldots, i+w+7\}$, and the $j-2$ least significant bits of $\mathrm{hi}(x_m)$ and $\mathrm{lo}(x_m)$ for $m \in \{i+8, i+9, i+10, i+w+8, i+w+9, i+w+10\}$.
      2.4.3. Now compute as many bits as possible of
$$v_m = G(x_{m+3}, x_{m+2}, x_{m+1}, x_m, 1, g_m, 1).$$
      That is compute the j least significant bits of $\mathrm{hi}(v_m)$ and $\mathrm{lo}(v_m)$ for $m \in \{i, i+w\}$,
      compute the $j-1$ least significant bits of $\mathrm{hi}(v_m)$ and $\mathrm{lo}(v_m)$ for $m \in \{i+1, i+2, i+3, i+4, i+w+1, i+w+2, i+w+3, i+w+4\}$,
      and compute the $j-2$ least significant bits of $\mathrm{hi}(v_m)$ and $\mathrm{lo}(v_m)$ for $m \in \{i+5, i+6, i+7, i+w+5, i+w+6, i+w+7\}$.
      2.4.4. Now, compute as many bits as possible of $W'_m = v_m \oplus v_{m+w}$ for $i \le m \le i+7$.
      2.4.5. For all $i \le m \le i+7$ compare the known bits W'_m with the corresponding bits of W_m. If any of these bits are different then reject the current guess.
   2.5. For all $i \le m \le i+7$ compute
$$\begin{aligned} \hat{v}_m &= F(x_{m+3}, x_{m+2}, x_{m+1}, x_m) \\ \hat{v}_{m+w} &= F(x_{m+w+3}, x_{m+w+2}, x_{m+w+1}, x_{m+w}) \\ \hat{W}_m &= \hat{z_m} \oplus \hat{z}_{m+w}. \end{aligned}$$
   2.6. For all $i \le m \le i+7$ compare W_m with $\hat{W}_m$. If any of them are different then reject the guess.

First, we have to analyse the consequence of the assumption $d_1 = d_3 = 1$. That is, we are looking for the probability $W_m = W'_m$, where as described earlier W_m is computed using the filter F, but W'_m is computed using the approximation G. We have

$$W'_m \oplus W_m = \Delta_m \oplus \Delta_{m+w}.$$

Hence we can use the distribution of Δ to guess the probability of $W'_m = W_m$. Under the heuristic assumption that Δ_m and Δ_{m+w} are independently distributed we find

$$Prob(W'_m = W_m) = 0.2318.$$

Hence, the number of i we have to check before being successful is $(0.2318)^8 \approx 2^{-16.9}$. Therefore the expected number of key stream words we have to examine before we find such an event is about $2^{17.5}$. (Note this number is slightly larger than $2^{16.9}$ since the events are not independent.)

In Step 2.3 we guess the carry bits g_m for the computations of z'_m for all $m \in \mathcal{M}'$. Hence, we have to guess 16 bits here.

Now we have to divide the analysis into the two cases $j = 2$ and $j > 2$. If $j = 2$, then we guess 16 bits in Step 2.4.1. Note that we do not need to guess the lsb of $\mathrm{lo}(x_i)$ and $\mathrm{lo}(x_{i+w})$. Hence, we guess 30 bits here. We compute 12 bits in the following Steps 2.4.2 through 2.4.4, which gives 12 single bit equations in Step 2.4.5.

If $j > 2$ we have to guess 16 bits in each step, but we also compute 16 new bits in the following equations, which does balance the number of guesses.

Hence, the complexity is

$$2^{17.5} \cdot 2^{16} \cdot 2^{30} \cdot 2^9 \approx 2^{72.5},$$

where the $2^{17.5}$ come from Step 1, 2^{16} from Step 2.3, 2^{30} from Step 2.4.2 with $j = 2$ and the 2^9 is an estimation of the work necessary for every guess in the loop 2.4. Some optimizations that are not described here are still possible. Furthermore, bit slice techniques [Bih97] can be used to speed up the computations.

8 Cryptanalysis of the Lagged Fibonacci Generator

In this section we describe an attack on the lagged Fibonacci generator (endowed with the filter function). The complexity bounds of this attack will be used in the next section and also yield lower bounds on the complexity to find the whole initial state in SSC2.

Assume first that only slightly more than 17 consecutive output words z''_i, $z''_{i+1}, ...$, of the generator are known, and that we want to determine the initial state $y_i, y_{i+1}, ..., y_{i+16}$. After possible reindexing suppose $i = 0$.
Consider the 17 equations (mod 2^{32}), derived by the recurrence relation of the generator,

1. $y_{17} = y_{12} + y_0$
...
5. $y_{21} = y_{16} + y_4$
6. $y_{22} = y_{12} + y_0 + y_5$
...
17. $y_{33} = y_{13} + y_1 + y_6 + y_{11} + y_{16}$

The output words z''_m are computed using the filter function (2). This function is nonlinear, but it becomes linear if we apply mod n cryptanalysis as introduced in [KSW99]. Thereby $n = 2^{32} - 1$ or a prime factor of $2^{32} - 1$ (the prime factors are 3, 5, 17, 257 and $2^{16} + 1$). Recall from [KSW99] that for a 32-bit word x, $x \lll j \equiv 2^j x \bmod 2^{32} - 1$. This implies that using one of the prime factors of $2^{32} - 1$ as a modulus, the 16-bit rotation in equation (2) becomes just multiplication by 1, except for the modulus $2^{16} + 1$, where 16-bit rotation becomes multiplication by -1. The price to pay is an ambiguity in passing from addition mod 2^{32} to addition mod n, depending on whether there was a carry in integer addition mod 2^{32}:

$$(x + y \bmod 2^{32}) \bmod n = \begin{cases} x + y \bmod n \text{ if there was no carry out} \\ x + y - 1 \bmod n \text{ if there was a carry out} \end{cases}$$

To get this relationship note that $2^{32} \equiv 1 \bmod n$ if $n = 2^{32} - 1$ or one of its prime factors.

As a consequence, equation (2) reduces to the two variants

$$z''_m = \varepsilon y_m + y_{m+t_m} \bmod n \text{ or } z''_m = \varepsilon y_m + y_{m+t_m} \bmod n, \tag{3}$$

depending on the carry. Moreover $\varepsilon = 1$ or -1, depending on n. The value of t_m is determined by the 4 most significant bits (msb) of y_{m+17}, which need to be guessed. Considering the 17 recurrence equations, the 4 msb's of $y_{17}, y_{18}, ..., y_{33}$ are guessed, and the equations also hold modulo n if the right sides are subtracted by the appropriate numbers depending on the carry. For equations 1. to 5. the carry needs also to be guessed. However from equation 6. onwards, the carry is mostly determined by the previous choice of the 4 msb's: Consider an arbitrary equation $C = A + B$ of integers mod 2^{32}. If the integer value determined by the 4 msb's of C are smaller than the 4 msb of A (or B), there is a carry, if this value is greater than the 4 msb of A or B, there is no carry. Thus the uncertainty by the carry bits in the 17 equations is about 6 bits. For every choice of the 4 msb's and the carry bits we combine the 17 recurrence equations with equations (3) to get 17 linear equations mod n for y_0, y_1,..., y_{16}. For every prime factor n of $2^{32} - 1$ one solves the system of equations, and the solution mod $2^{32} - 1$ is got by the Chinese remainder theorem. Then the solution found is checked for contradictions with the 4 msb's and the carry's chosen. (In the rare case of a zero value one has to further check whether this value is indeed zero or $2^{32} - 1$.) This procedure is repeated until no contradiction occurs anymore. The remaining candidates are checked by producing one or two further output words z''_m, (which are assumed to be known). Solving systems of linear equations with

17 unknowns is of the order of 2^{12} operations. Hence the total complexity of this analysis is estimated as

$$2^{68} \cdot 2^6 \cdot 2^{17} \cdot 2^{12} = 2^{103}.$$

This analysis can be improved if more output of the generator is assumed to be known. Suppose that $t_m = t_{m+12} = t_{m+17}$. Then S_m takes one of the values in $\mathcal{A}$. Recall that the probability for this event is about 2^{-8}. Let x_1 x_2, x_3 denote the 4-bit integers given by each of the 4 msb's of y_{m+17},..., determining t_m, t_{m+12}, t_{m+17}. Because of the recursion, $x_3 = x_1 + x_2$ or $x_3 = x_1 + x_2 + 1$, if there is a carry from less significant bits. Furthermore the corresponding pointer variables are related by $r_1 = s_1 + 12 \bmod 17$, $s_2 = s_1 + 5 \bmod 17$ and $s_3 = s_1$. Therefore every such pointer variable is expressible in terms of $s_1 := s$. Note that in an attack s can be assumed to be known. Further recall that $1 \leq r, s \leq 17$. Hence arithmetic modulo 17 in the context of the pointers r and s is modified in the sense that a pointer value is added by the value 17 if it becomes 0 or negative. We derive the following equations:

t_m $= ((x_1 + s) \bmod 16)$ - $(s + 12) \bmod 17$
$t_{m+12} = ((x_2 + ((s + 5) \bmod 17)) \bmod 16)$ - s mod 17
$t_{m+17} = ((x_1 + x_2 + s) \bmod 16)$ -$(s + 12) \bmod 17$

or

$t_{m+17} = ((x_1 + x_2 + 1 + s) \bmod 16)$ -$(s + 12) \bmod 17$

if there is a carry. From $t_m = t_{m+17}$ one thus gets $x_2 = 0$ if there is no carry, and $x_2 = 15$ if there is a carry. Moreover, the first two equations and $t_m = t_{m+12}$ imply that a given x_1 and a known s determine the value of x_2. Hence we get a reduction of uncertainty about x_1, x_2 from 16^2 to 2 bits. Further investigation limits the triple (x_1, x_2, x_3) to the values $(0, 0, 0)$, $(15, 15, 15)$, $(1, 0, 1)$, $(15, 0, 15)$ and $(0, 15, 0)$. Experiments show that the frequency of each of these triples is quite different, as different s-values can lead to the same triple. Hence the sequence S_n is not only biased but in case $S_n \in \mathcal{A}$ allows to derive key material.

Suppose now that S_m, S_{m+1}, S_{m+2} and S_{m+3} are all elements of $\mathcal{A}$. This happens with probability about 2^{-32}. Then the uncertainty of the 4 msb's of 8 of the 17 y_n's to be found drops from 16^8 to 2^4 bits. This is a reduction by a factor 2^{28}.

As a consequence, the complexity of our analysis of the filtered lagged Fibonacci generator reduces from 2^{103} to about 2^{75} operations, if the number of known plaintexts is of the order of 2^{32} output words.

9 An Attack Based on the Bias in the Lagged Fibonacci Generator

In this section, we describe briefly an attack against SSC2 that exploits the bias in the lagged Fibonacci generator. This attack requires on average between 2^{45} and 2^{46} known plaintext words and has an expected time complexity of 2^{109}. Because of the large time complexity we only describe the idea behind the attack and omit some details.

Let z_n denote the n-th word of key stream. Then the idea behind the attack is to scan the key stream and hope that for some m the following happens:

- S_m, S_{m+1}, S_{m+2} and S_{m+3} are all elements in the set $\mathcal{A}$.
- For all $n \in \{m, \ldots, m+3, m+12, \ldots, m+15, m+17, \ldots, m+20\}$ the carry bits that occur during the computation of z'_n satisfy the equations $c_1 + c_4 = 1$ and $c_3 + c_5 = 1$.

The probability that $S_m, S_{m+1}, S_{m+2}, S_{m+3} \in \mathcal{A}^4$ is approximately 2^{-32} and the probability that the sum of the carry bits during a computation of z'_n are both 1 is $(2/3)^2$. It follows that all conditions mentioned above are satisfied with a probability of approximately $2^{-32}(2/3)^{24} \approx 2^{-46}$. Hence, we expect to find such an event in 2^{46} words of known key stream.

Thus for all m, let

$$M = \{m, \ldots, m+3, m+12, \ldots, m+15, m+17, \ldots, m+20\}$$

and do the following:

1. Guess the 5 least significant bits of $\mathrm{hi}(x_j)$ and $\mathrm{lo}(x_j)$ for all $m+1 \le j \le m+4$.
2. Guess the carry bit c_2 for all computations of z_n where $n \in M$.
3. For i from 6 to 16 do:
   - 3.1. Guess the i-th least significant bits of $\mathrm{hi}(x_j)$ and $\mathrm{lo}(x_j)$ for all $m+1 \le i \le m+4$.
   - 3.2. Compute the $i-4$ least significant bits of $\mathrm{hi}(x_n)$ and $\mathrm{lo}(x_n)$ for $m \le n \le m+20$.
   - 3.3. Use the computation in figure 4 to compute the $i-4$ least significant bits of $\mathrm{hi}(z'_n)$ and $\mathrm{lo}(z'_n)$ for $n \in M$.
   - 3.4. Use the known key stream z_n to compute z''_n for all $n \in M$.
   - 3.5. For all $m \le n \le m+3$ compute the $i-4$ least significant bits of $\mathrm{hi}(S_n)$ and $\mathrm{lo}(S_n)$. If these bits do not correspond to one of the values in the set $\mathcal{A}$ then reject the guess, otherwise go on with this guess.
4. *At this point we have a guess for all the bits in the linear feedback shift register.* Now, compute the real value of the carry bits. If they don not correspond to the assumptions then reject this guess.
5. Compute more (say 1000) values of S_n and check whether there are more values that are in the set $\mathcal{A}$. If this is not the case then reject the guess.
6. Now, solve for the internal state of the lagged Fibonacci register, as was done in the previous section.

The time complexity of this approach can be computed as follows: We have to go through 2^{46} values for m on average. In step (1) we guess 40 bits. In step (2) we guess 12 carry bits. Hence, up to here we have to guess 98 bits. Then in step (3.1) we guess 8 bits, increasing the complexity to 2^{106}, but in step (3.5) we compute 8 output bits, and hence decrease the complexity at this point to 2^{98} again. Step (4) 2^{72}. So the time requirement for step (5) is insignificant. Hence, the significant part of the complexity comes from step (3). This loop is performed 11 times for about 2^{106} different guesses, giving a total complexity of about 2^{109}.

9.1 Variants

There is a tradeoff possible between the number of known plaintext words and the complexity of the attack:

Instead of assuming that the carry bits that occur during the computation of z'_n satisfy the equations $c_1 + c_4 = 1$ and $c_3 + c_5 = 1$ one may assume this only for a subset of the 24 equations, or one could even do an exhaustive search over all the possible values on the right side of these equations. In the latter case we would need only about 2^{32} (rather than 2^{46}) words or known key stream.

On the average, 8 equations are incorrect, i.e., the right side is either 0 or 2 instead of 1. A rough estimate shows that the complexity for the search with probability 1/2 is not larger than $2\binom{24}{8}2^8 \approx 2^{28}$. Thus with probability about 1/2 the complexity of this variant of the attack is not larger than $2^{32} \cdot 2^{40} \cdot 2^{28} \cdot 2^{12} \cdot 2^8 \cdot 2^3 = 2^{123}$.

10 Frame Key Generation

For synchronization purposes, SSC2 supplies generation of new keys out of the master key for every frame. Each frame is given by a 32-bit number, which is not encrypted. The frame key generation algorithm should satisfy the property that it is difficult to gain information about a frame key from another frame key ([ZCC00]). This property is not satisfied however. The frame key generation is illustrated in [ZCC00] by a pseudo-code, to which we refer. Then one sees that the word $B[1]$ of the key remains constant, i.e. does not depend on the frame number: In line 7 of the code, $B[17 - (i + 8j \bmod 16)] \leftarrow S[1] \oplus B[17 - (i + 8j \bmod 16)]$, the index 1 is never reached.

11 Countermeasures

Certainly, various countermeasures are possible. We think that most importantly the period of the lagged Fibonacci generator should be increased. Even though we do not know how to use the correlation properties of the LFSR alone for an attack, we would still recommend to remove those properties. We have used that the output of the two registers are combined using a simple XOR. Using a nonlinear function instead might further improve the security of the cipher.

12 Conclusion

We have shown some weaknesses of SSC2 and derived two attacks based on those weaknesses. These attacks may not be a threat in the applications, for which SSC2 has been designed. Nonetheless, the results show that the security margin of SSC2 is smaller than previously believed. It is therefore advisable to remove the weaknesses described in this paper. The recent attacks by Hawkes, Quick and Rose [HR01] give even more weight to our opinion.

Acknowledgement. We are thankful to Muxiang Zhan for reviewing a previous version of this paper and pointing out the period in the filter of the lagged Fibonacci generator. We also thank Greg Rose for pointing out an error in a correlation.

References

[Bih97] Eli Biham. A fast new DES implementation in software. In Eli Biham, editor, *Fast Software Encryption '97*, pages 260–272. Springer Verlag, 1997.

[KSW99] J. Kelsey, B. Schneier, and D. Wagner. Mod n cryptanalysis with applications against RC5P and M6. In L. Knudsen, editor, *Fast Software Encryption '99*, volume 1636 of *Lecture Notes in Computer Science*, pages 139–155. Springer Verlag, 1999.

[HR01] P. Hawkes, F. Quick and G. Rose. A practical cryptanalysis of SSC2. In S. Vaudenay and A.M. Youssef, editor, *Selected Areas in Cryptography' 01*, volume 2259 of *Lecture Notes in Computer Science*, pages 27–37, Springer Verlag, 2001.

[SM91] O. Staffelbach and W. Meier. Cryptographic significance of the carry for ciphers based on integer addition. In A.J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology – CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 601–615. Springer Verlag, 1991.

[ZCC00] Muxiang Zhang, Christopher Carroll, and Agnes Chan. The software-oriented stream cipher SSC2. In *Fast Software Encryption '2000 (preproceedings)*, 2000.