The Software-Oriented Stream Cipher SSC2

Abstract SSC2 is a fast software stream cipher designed for wireless handsets with limited computational capabilities. It supports various private key sizes from 4 bytes to 16 bytes. All operations in SSC2 are word-oriented, no complex operations such as multiplication, division, and exponentiation are involved. SSC2 has a very compact structure that makes it easy to implement on 8-,16-, and 32-bit processors. Theoretical analysis demonstrates that the keystream sequences generated by SSC2 have long period, large linear complexity, and good statistical distribution.

1. Introduction

For several reasons, encryption algorithms have been constrained in cellular and personal communications. First, the lack of computing power in mobile stations limits the use of computationally intensive encryption algorithms such as public key cryptography. Second, due to the high bit error rate of wireless channels, encryption algorithms which produce error propagation deteriorate the quality of data transmission, and hence are not well suited to applications where high bit error rates are common place. Third, the shortage of bandwidth at uplink channels (from mobile station to base station) makes encryption algorithms at low encryption (or decryption) rates unacceptable, and random delays in encryption or decryption algorithms are not desirable either. To handle these issues, the European Group Special Mobile (GSM) adopted a hardware implemented stream cipher known as alleged A5 [13]. This stream cipher has two main variants: the stronger A5/1 version and the weaker A5/2 version. Recent analysis by Biryukov and Shamir [16] has shown that the A5/1 version can be broken in less than one second on a single PC. Except the weakness, the hardware implementation of the alleged A5 also incurs additional cost. Especially, the cost to modifying the encryption algorithm in every handset would be exorbitant. For this reason, a software implemented stream cipher which is fast and secure would be preferable.

To this end, we designed SSC2, a software-oriented stream cipher which is easy to implement on 8-, 16-, and 32-bit processors. SSC2 belongs to the stream cipher family of combination generators. It combines a filtered linear feedback shift register (LFSR) and a lagged-Fibonacci generator. All operations involved in SSC2 are word-oriented, where a word consists of 4 bytes. The word sequence generated by SSC2 is added modulo-2 to the words of data frames in the manner of a Vernam cipher. SSC2 supports various private key sizes from 4 bytes to 16 bytes. It has a key scheduling scheme that stretches a private key to a master key of 21 words. The master key is loaded as the initial states of the LFSR and the lagged-Fibonacci generator. To cope with the synchronization problem, SSC2 also supplies an efficient frame key generation scheme that generates an individual key for each data frame. Theoretical analysis indicates that the keystream sequences generated by SSC2 have long period, large linear complexity, and good statistical distribution

2. Specification of SSC2

The keystream generator of SSC2, as is depicted by Figure 1, consists of a filter generator and a lagged-Fibonacci generator. In the filter generator, the LFSR is a word-oriented linear feedback shift register. The word-oriented LFSR has 4 stages with each stage containing a word. It generates a new word and shifts out an old word at every clock. The nonlinear filter compresses the 4-word content of the LFSR to a word. 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.

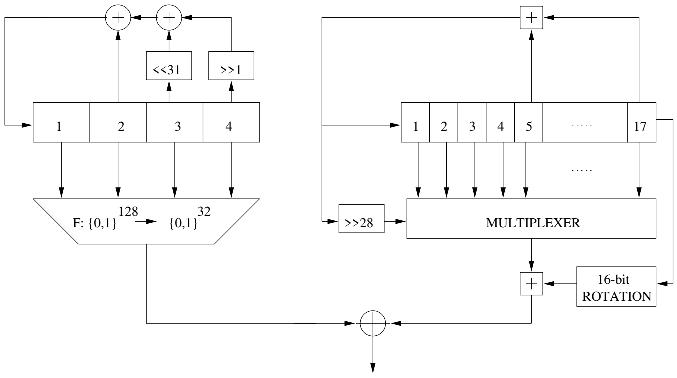

Figure 1: The keystream generator of SSC2

2..1 The word-oriented linear feedback shift register

For software implementation, there are two major problems for LFSR-based keystream generators. First, the speed of a software implemented LFSR is much slower than that of a hardware implemented one. To update the state of a LFSR, a byte-oriented or word-oriented processor needs to spend many clock cycles to perform the bit-shifting and bit-extraction operations. Second, LFSR-based keystream generators usually produce one bit at every clock, which again makes the software implementation inefficient. To make the software implementation efficient, we designed a word-oriented LFSR in SSC2 by exploiting the fact that each word of a linear feedback shift register sequence can be represented as a linear transformation of the previous words of the sequence.

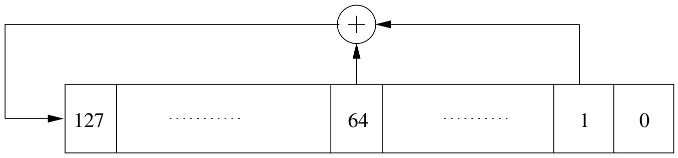

Figure 2: The LFSR with characteristic polynomial $p(x) = x(x^{127} + x^{63} + 1)$

The LFSR used in SSC2, as is depicted by Figure 4.2, has the characteristic polynomial

$$p(x) = x(x^{127} + x^{63} + 1),$$

where the factor $x^{127}+x^{63}+1$ of $p(x)$ is a primitive polynomial over $GF(2)$. After discarding s_0, the LFSR sequence, $s_1, s_2, \ldots$, is periodic and has the least period $2^{127}-1$. The state $S_n = (s_{n+127}, s_{n+126}, \ldots, s_n)$ at time n can be divided into 4 blocks with each block being a word, that is,

$$S_n = (x_{n+3}, x_{n+2}, x_{n+1}, x_n).$$

After running the LFSR 32 times, the LFSR has the state

$$S_{n+32} = (x_{n+4}, x_{n+3}, x_{n+2}, x_{n+1}).$$

It can be shown that

$$x_{n+4} = x_{n+2} \oplus (s_{n+32}, 0, 0, \ldots, 0) \oplus (0, s_{n+31}, s_{n+30}, \ldots, s_{n+1}). \tag{1}$$

Let $\ll$ denote the zero-fill left-shift operation. By $x \ll_j$, it means that the word x is shifted left j bits and a zero is filled to the right-most bit every time when x is shifted left 1 bit. Similarly, let $\gg$ denote the zero-fill right-shift operation. With these notations, we can rewrite equation (1) as follows

$$x_{n+4} = x_{n+2} \oplus x_{n+1} \ll_{31} \oplus x_n \gg_1, \tag{2}$$

which describes the operation of the word-oriented LFSR in Figure 1. It is interesting to notice that the feedback connections of the word-oriented LFSR are not sparse even though the bit-oriented LFSR described by $p(x)$ has very sparse feedback connections.

In the bit-oriented LFSR as described by Figure 2, the stage 0 is not involved in the computation of the feedback and hence is redundant. It is left there just to make the length of the LFSR to be a multiple of 32. For this reason, the content of stage 0 will be excluded from the state of the LFSR. Let S'_n denote the bit-oriented LFSR state consisting of the contents of stage 1 through stage 127, namely,

$$S'_n = (s_{n+127}, s_{n+126}, \ldots, s_{n+1}).$$

Correspondingly, let S''_n denote the state of the word-oriented LFSR, where S''_n is made up of the contents of stage 1 through stage 127 of the word-oriented LFSR at time n. It is clear that

$$S''_n = S'_{32n}, \quad n \geq 0. \tag{3}$$

Proposition 1 Assume that the initial state S''_0 of the word-oriented LFSR described by (2) is not zero. Then the state sequence $S''_0, S''_1, \ldots$ is periodic and has the least period $2^{127}-1$. Furthermore, for any $0 \leq i < j < 2^{127}-1$, $S''_i \neq S''_j$.

Proof. Since $x^{127}+x^{63}+1$ is a primitive polynomial oner $GF(2)$, and $S'_0 = S''_0 \neq 0$, the state sequence $S'_0, S'_1, \ldots$ of the bit-oriented LFSR is periodic and has the least period $2^{127}-1$. Thus, by (3), $S''_{n+2^{127}-1} = S'_{32(n+2^{127}-1)} = S''_n$. Hence, the sequence $S''_0, S''_1, \ldots$ is periodic and has a period of $2^{127}-1$. The least period of the sequence should be a divisor of $2^{127}-1$. On the other hand, $2^{127}-1$ is a prime number (divided by 1 and itself). So the least period of $S''_0, S''_1, \ldots$ is $2^{127}-1$.

Next, assume that $S''_i = S''_j$ for i and j with $0 \leq i < j < 2^{127}-1$. Then $S'_{32i} = S'_{32j}$, which implies that $32(j-i)$ is a multiple of $2^{127}-1$. Since $2^{127}-1$ is prime to 32, $j-i$ is a multiple of $2^{127}-1$, which contradicts to the assumption. $\square$

2..2 The nonlinear filter

The nonlinear filter is a memoryless function, that is, its output at time n only depends on the content of the word-oriented LFSR at time n. Let $(x_{n+3}, x_{n+2}, x_{n+1}, x_n)$ denote the content of the word-oriented LFSR at time n. The output at time n, denoted z'_n, is described by the following pseudo-code:

Nonlinear-Function $F(x_{n+3}, x_{n+2}, x_{n+1}, x_n)$
1 $A \leftarrow x_{n+3} + (x_n \vee 1) \bmod 2^{32}$
2 $c \leftarrow$ carry
3 cyclic shift A left 16 bits
4 **if** $(c = 0)$ **then**
5 $\quad A \leftarrow x_{n+2} + A \bmod 2^{32}$
6 **else**
7 $\quad A \leftarrow (x_{n+2} \oplus (x_n \vee 1)) + A \bmod 2^{32}$
8 $c \leftarrow$ carry
9 **return** $(x_{n+1} \oplus x_{n+2}) + A + c \bmod 2^{32}$

Let c_1 and c_2 denote the first (line 2) and second (line 8) carry bits in the pseudo-code. For a 32-bit integer A, let $\langle A \rangle_{16}$ denote the result of cyclicly shifting A left 16 bits. Then the function F has the following compact form:

$$z'_n = \langle x_{n+3} + (x_n \vee 1) \rangle_{16} + x_{n+2} \oplus c_1(x_n \vee 1) + x_{n+1} \oplus x_{n+2} + c_2 \bmod 2^{32}, \tag{4}$$

where $\vee$ denotes bitwise "OR" operation. The priority of $\oplus$ is assumed to be higher than that of $+$. The least significant bit of x_n is always masked by 1. The reason is to get rid of the effect of stage 0 of the LFSR in Figure 2.

2..3 The lagged-Fibonacci generator

Lagged-Fibonacci generators, also called additive generators, have been widely used as random number generators in Monte Carlo simulation [4,6]. Mathematically, a lagged Fibonacci generator can be characterized by the following recursion:

$$y_n = y_{n-s} + y_{n-r} \bmod M, \quad n \geq r. \tag{5}$$

The generator is defined by the modulus M, the register length r, and the lag s, where $r > s$. When M is prime, periods as large as $M^r - 1$ are possible for the generated sequences. However it is more common to use lagged-Fibonacci generators with $M = 2^m$, $m \geq 1$. These generators with power-of-two moduli are much easier to implement than prime moduli. The following lemma was proved by Brent [1].

Lemma 1 Assume that $M = 2^m, m \geq 1, r > 2$, and the polynomial $x^r + x^r + 1$ is primitive over $GF(2)$. Then the sequence $y_0, y_1, \ldots$ of the lagged-Fibonacci generator described by ( ) has the least period $2^{m-1}(2^r - 1)$ if $y_0, y_1, \ldots, y_{r-1}$ are not all even.

In SSC2, the lagged-Fibonacci generator with $s = 5$, $r = 17$, and $m = 2^{32}$ was adopted. We implemented this generator with a 17-stage circular buffer, B, and two pointers, s, and, r. Initially $\mathsf{B}[17], \mathsf{B}[16], \ldots, \mathsf{B}[1]$ are loaded with $y_0, y_1, \ldots, y_{16}$, and s and r are set to 5 and

17, respectively. At every clock, a new word is produced by taking the sum of $\mathsf{B}[\mathsf{r}]$ and $\mathsf{B}[\mathsf{s}]$ mod 2^{32}, the word $\mathsf{B}[\mathsf{r}]$ is then replaced by the new word, and the pointers s and r are decreased by 1. The output word is computed from the replaced word and another word selected from the buffer B. The selection is based on the most significant 4 bits of the newly produced word. The output word at time n, denoted z''_n, is given by

$$z''_n = \langle y_n \rangle_{16} + \mathsf{B}[1 + ((y_{n+17} \gg_{28}) + s_{n+1} \bmod 16)] \bmod 2^{32}. \tag{6}$$

The pseudo-code for z''_n is listed as follows:

$A \leftarrow \mathsf{B}[\mathsf{r}]$
$D \leftarrow \mathsf{B}[\mathsf{s}] + \mathsf{B}[\mathsf{r}] \bmod 2^{32}$
$\mathsf{B}[\mathsf{r}] \leftarrow D$
$\mathsf{r} \leftarrow \mathsf{r} - 1$
$\mathsf{s} \leftarrow \mathsf{s} - 1$
if $(\mathsf{r} = 0)$ **then** $\mathsf{r} \leftarrow 17$
if $(\mathsf{s} = 0)$ **then** $\mathsf{s} \leftarrow 17$
cyclicly shift A left 16 bits
output $A + \mathsf{B}[1 + (\mathsf{s} + D \gg_{28} \bmod 16)] \bmod 2^{32}$

Proposition 2 Assume that the initial state $y_0, y_1, \ldots, y_{16}$ of the lagged-Fibonacci generator are not all even. Then the sequence $\tilde{z}'' = z''_0, z''_1, \ldots$ is periodic and its least period is a divisor of $17(2^{17} - 1)2^{31}$.

Proof. Let r_n and s_n denote the values of r and s at time n. Then the sequences $r_0 r_1 \ldots$ and $s_0 s_1 \ldots$ are periodic and have the least period 17. Let i_n denote the index at time n, that is, $i_n = 1 + ((y_{n+17} \gg_{28}) + s_{n+1} \bmod 16)$. If $i_n \geq r_n$, then

$$\mathsf{B}[i_n] = y_{n+17-(i_n - r_n)}, \tag{7}$$

since r_n points to y_{n+17} and the time difference between r_n and i_n is $i_n - r_n$. If $i_n < r_n$, then the time difference between r_n and i_n is $17 - r_n + i_n$, thus,

$$\mathsf{B}[i_n] = y_{n+17-(17-r_n+i_n)}. \tag{8}$$

Since $x^{17} + x^5 + 1$ is primitive over $GF(2)$ and $y_0, y_1, \cdots, y_{16}$ are not all even, by Lemma 1, the sequence $y_0, y_1 \ldots$ has the least period $(2^{17} - 1)2^{31}$. From (7) and (8), it follows that

$$\mathsf{B}[i_{n+17(2^{17}-1)2^{31}}] = \mathsf{B}[i_n].$$

Hence, $z''_{n+17(2^{17}-1)2^{31}} = z''_n$, which proves the theorem. □

3. Cryptographic Properties of SSC2

Period, linear complexity, and statistical distribution are three fundamental measures that should be clear for keystream generators. Unfortunately, these measures are not very clear for most of the proposed keystream generators. In this section, an assessment of the strength of SSC2 will be carried out with respect to these fundamental measures.

In the following, we will use $\tilde{z} = z_0, z_1, \ldots$ to denote the keystream sequence generated by SSC2. It is the sum of the two sequences $\tilde{z}' = z'_0, z'_1, \ldots$ of the filter generator, and $\tilde{z}'' = z''_0, z''_1, \ldots$ of the lagged-Fibonacci generator.

Theorem 1 Assume that the initial state S_0'' of the word-oriented LFSR is not zero, and the the initial state $y_0, y_1, \ldots, y_{16}$ of the lagged-Fibonacci generator are not all even. Then the least period of the keystream sequence generated by SSC2 is greater than or equal to $2^{128} - 2$.

Proof. Let $T_{\tilde{z}}, T_{\tilde{z}'}$, and $T_{\tilde{z}''}$ denote the least periods of $\tilde{z}, \tilde{z}'$, and $\tilde{z}''$, respectively. Then, by Proposition 1 and Proposition 2, $T_{\tilde{z}'} = 2^{127} - 1$, and $T_{\tilde{z}''}$ is a factor of $17(2^{17}-1)2^{31}$. Since the two numbers, $2^{127} - 1$, and, $17(2^{17}-1)2^{31}$, are relatively prime, $T_{\tilde{z}'}$ and $T_{\tilde{z}''}$ are also relatively prime. Hence $T_{\tilde{z}} = T_{\tilde{z}'} T_{\tilde{z}''}$. Therefore $T_{\tilde{z}} \geq 2T_{\tilde{z}'} = 2^{128} - 2$. □

Let $\Lambda(\tilde{z}')$ and $\Lambda(\tilde{z}'')$ denote the linear complexity of $\tilde{z}'$ and $\tilde{z}''$. According to [11], the linear complexity of $\tilde{z} = \tilde{z}' \oplus \tilde{z}''$ is bounded by

$$\Lambda(\tilde{z}') + \Lambda(\tilde{z}'') - 2\gcd(T_{\tilde{z}'}, T_{\tilde{z}''}) \leq \Lambda(\tilde{z}) \leq \Lambda(\tilde{z}') + \Lambda(\tilde{z}''). \tag{9}$$

Thus, if we have lower bounds on the linear complexity of either $\tilde{z}'$ or $\tilde{z}''$, we can achieve lower bounds on the linear complexity of $\tilde{z}$. In the following, we will analyze the linear complexity of $\tilde{z}'$.

We can treat the sequence $\tilde{z}' = z_0', z_1', \ldots$ in three different forms. First, it is a sequence of words; second, it is a sequence of bits; and third, it can be considered as a collection of 32 component sequences, $\tilde{z}_i' = z_{i,0}', z_{i,1}', \ldots, 0 \leq i \leq 31$, where $z_n' = (z_{31,n}', z_{30,n}', \ldots, z_{0,n}')$. For any $0 \leq i \leq 31$, $z_{i,n}'$ can be described by

$$z_{i,n}' = f_i(s_{127,n}'', s_{126,n}'', \ldots, s_{1,n}''), \quad n \geq 0, \tag{10}$$

where $(s_{127,n}'', s_{126,n}'', \ldots, s_{1,n}'')$ is the state of the word-oriented LFSR at time n, and f_i is the i-th component of the nonlinear filter F. Assume that the nonlinear order, $ord(f_i)$, of f_i is ℓ_i. According to [5], the linear complexity of $\tilde{z}_i'$ is bounded by

$$\Lambda(\tilde{z}_i') \leq L_{\ell_i} = \sum_{j=1}^{\ell_i} \binom{127}{j}. \tag{11}$$

The upper bound L_{ℓ_i} is usually satisfied with equality. But, there are also few exceptions that the actual linear complexity is deviated from the expected value given by the upper bound. Rueppel [12] proved that, for a LFSR with primitive connection polynomial of prime degree L, the fraction of Boolean functions of nonlinear order ℓ which produce sequences of linear complexity L_ℓ is

$$P_d \approx \exp(-L_\ell/(L \cdot 2^L)) > e^{-1/L}. \tag{12}$$

For the filter generator of SSC2, we have $P_d > e^{-\frac{1}{127}}$. Hence, the linear complexity of $\tilde{z}_i'$ is virtually certain to be L_{ℓ_i}.To determine the nonlinear order of f_i, let's first review the principle of integer addition.

Let $x = (x_{31}, x_{30}, \ldots, x_0)$ and $y = (y_{31}, y_{30}, \ldots, y_0)$ denote the binary representation of two 32-bit integers, where x_0 and y_0 are the least significant bits of x and y. The sum, $x+y \bmod 2^{32}$, defines a new 32-bit integer $z = (z_{31}, z_{30}, \ldots, z_0)$. The binary digits $z_i, 0 \leq i \leq 31$, are recursively computed by

$$z_i = x_i \oplus y_i \oplus c_{i-1}, \tag{13}$$

$$c_i = x_i y_i \oplus (x_i \oplus y_i) c_{i-1}, \tag{14}$$

where c_{i-1} denotes the carry bit, $c_{-1} = 0$. The 31-th carry bit c_{31} is also called the carry bit of $x + y$. In (13) and (14), z_i and c_i are Boolean functions of x_i and y_i, $0 \le i \le 31$. In the following, we will use $ord(z_i)$ and $ord(c_i)$ to denote the nonlinear order of the Boolean functions represented by z_i and c_i.

Lemma 2 Assume that $x = (x_{31}, x_{30}, \ldots, x_0)$ and $y = (y_{31}, y_{30}, \ldots y_0)$ be two 32-bit integers, $x_0 = 1$. Let $z = (z_{31}, z_{30}, \ldots, z_0)$ denote the sum of $x + y$, and $c = (c_{31}, c_{30}, \ldots, c_0)$ denote the carry bits produced by the summation. Then $ord(c_i) = i + 1, 0 \le i \le 31$. Furthermore, $ord((x_i \oplus y_i)c_{31}) = 32$, $1 \le i \le 31$, and $ord(c_{15}c_{31}) = 33$.

Proof. By (14), $c_0 = y_0$, and $c_1 = x_1y_1 \oplus (x_1 \oplus y_1)y_0$. So $ord(c_0) = 1$ and $ord(c_1) = 2$. Assume that $ord(c_i) = i+1, i \ge 2$. Since x_{i+1} and y_{i+1} do not appear in the Boolean function represented by c_i, we have

$$\begin{aligned} ord(c_{i+1}) &= ord(x_{i+1}y_{i+1} \oplus (x_{i+1} \oplus y_{i+1})c_i) \\ &= ord((x_{i+1} \oplus y_{i+1})c_i) \\ &= i + 2 \end{aligned}$$

By induction, $ord(c_i) = i+1, 0 \le i \le 31$. Similarly, it can be proved that $ord((x_i \oplus y_i)c_{31}) = 32$ and $ord(c_{15}c_{31}) = 33$. □

Lemma 3 Let $z = F(x_3, x_2, x_1, x_0)$ denote the nonlinear function described by (4), where $z = (z_{31}, z_{30}, \ldots, z_0)$, and $x_i = (x_{i,31}, x_{i,30}, \ldots, x_{i,0}), 0 \le i \le 3$. For any $0 \le i \le 31$, let $z_i = f_i(x_3, x_2, x_1, x_0)$, then $ord(f_i) \ge 64 + i$.

Proof. The nonlinear function F is a mapping of $GF(2)^{128}$ to $GF(2)^{32}$, which has the following form:

$$z = \langle x_3 + (x_0 \vee 1)\rangle_{16} + x_2 \oplus c_1(x_0 \vee 1) + x_1 \oplus x_2 + c_2 \bmod 2^{32},$$

where c_1 and c_2 are the carry bits produced by the first two additions.

Let $Z_1 = (z_{1,31}, z_{1,30}, \ldots, z_{1,0})$ denote the sum of x_3 and $x_0 \vee 1$. The carry bits produced by the summation is denoted by $c_{1,31}, c_{1,30}, \ldots, c_{1,0}$. It is clear that $c_1 = c_{1,31}$. By Lemma 2, $ord(c_1) = 32$, and $ord(z_{1,i}) = i, 1 \le i \le 31$.

Let $Z'_1 = (z'_{1,31}, z'_{1,30}, \ldots, z'_{1,0})$ denote $\langle Z_1\rangle_{16}$, that is, $z'_{1,i} = z_{1,16+i}$, for $0 \le i \le 15$, and $z'_{1,i} = z_{1,i-16}$, for $16 \le i \le 31$. Let $Z_2 = (z_{2,31}, z_{2,30}, \ldots, z_{2,0})$ denote the sum of Z'_1 and $x_2 \oplus c_1(x_0 \vee 1)$, and $c_{2,31}, c_{2,30}, \ldots, c_{2,0}$ denote the carry bits produced by the summation. By (13) and (14),

$$z_{2,i} = z'_{1,i} \oplus x_{2,i} \oplus c_1x_{0,i} \oplus c_{2,i-1} \tag{15}$$

$$c_{2,i} = z'_{1,i}(x_{2,i} \oplus c_1x_{0,i}) \oplus (z'_{1,i} \oplus x_{2,i} \oplus c_1x_{0,i})c_{2,i-1}, \tag{16}$$

where $c_{2,-1} = 0$ and $x_{0,0} = 1$. Rewriting (16), we have

$$c_{2,i} = (z'_{1,i} \oplus c_{2,i-1})x_{2,i} \oplus z'_{1,i}c_1x_{0,i} \oplus (z'_{1,i} \oplus c_1x_{0,i})c_{2,i-1}.$$

Since $x_{2,i}$ does not appear in $z'_{1,i}c_1x_{0,i} \oplus (z'_{1,i} \oplus c_1x_{0,i})c_{2,i-1}$, we have

$$ord(c_{2,i}) \ge ord((z'_{1,i} \oplus c_{2,i-1})x_{2,i}). \tag{17}$$

The carry bit $c_{2,0}$ has the following expression,

$$\begin{aligned}
c_{2,0} &= z'_{1,0}(x_{2,0} \oplus c_1 x_{0,0}) \\
&= z_{1,16}(x_{2,0} \oplus c_{1,31}) \\
&= (x_{3,16} \oplus x_{0,16} \oplus c_{1,15})x_{2,0} \oplus (x_{3,16} \oplus x_{0,16})c_{1,31} \oplus c_{1,15}c_{1,31}
\end{aligned}$$

By Lemma 2, $ord((x_{3,16} \oplus x_{0,16})c_{1,31}) = 32$, and $ord(c_{1,15}c_{1,31}) = 33$. The order of $(x_{3,16} \oplus x_{0,16} \oplus c_{1,15})x_{2,0}$ is equal to 17. So the order of $c_{2,0}$ equals 33. On the other hand, $ord(z_{1,i}) \leq 32, 0 \leq i \leq 31$. Hence, $ord(c_{2,0}) > ord(z'_{1,1})$. By (17), $ord(c_{2,1}) \geq ord(c_{2,0}x_{2,0}) > 33$. By induction, it can be proved that $ord(c_{2,i}) \geq 33, 0 \leq i \leq 31$. Thus $ord(c_{2,i}) \geq ord(c_{2,i-1}) + 1$. Hence $ord(c_{2,31}) \geq 64$.

Let $Z_3 = (z_{3,31}, z_{3,30}, \ldots, z_{3,0})$ denote the sum of $Z_2 + x_1 \oplus x_2 + c_2$. The carry bits produced by the summation is denoted by $c_{3,31}, c_{3,30}, \ldots, c_{3,0}$. It is obvious that $c_2 = c_{2,31}$, and $z = Z_3$. By (13) and (14), we have the following expressions for z_i and $c_{3,i}$,

$$z_i = z_{2,i} \oplus x_{1,i} \oplus x_{2,i} \oplus c_{3,i-1} \tag{18}$$

$$c_{3,i} = z_{2,i}(x_{1,i} \oplus x_{2,i}) \oplus (z_{2,i} \oplus x_{1,i} \oplus x_{2,i})c_{3,i-1}, \tag{19}$$

where $c_{3,-1} = c_{2,31}$. By (15), $ord(z_{2,0}) = ord(c_1) = 32$. Thus, $ord(z_0) = ord(c_{3,-1}) \geq 64$. Rewriting (19), we have the following expression for $c_{3,i}$,

$$c_{3,i} = (z_{2,i} \oplus c_{3,i-1})x_{1,i} \oplus z_{2,i}x_{2,i} \oplus (z_{2,i} \oplus x_{2,i})c_{3,i-1}. \tag{20}$$

Substituting (20) into (18),

$$z_i = z_{2,i} \oplus x_{1,i} \oplus x_{2,i} \oplus (z_{2,i-1} \oplus c_{3,i-2})x_{1,i-1} \oplus z_{2,i-1}x_{2,i-1} \oplus (z_{2,i-1} \oplus x_{2,i-1})c_{3,i-2}. \tag{21}$$

Since $x_{1,i-1}$ only appears in $(z_{2,i-1} \oplus c_{3,i-2})x_{1,i-1}$, it is clear that

$$ord(z_i) \geq ord(z_{2,i-1} \oplus c_{3,i-2}) + 1. \tag{22}$$

By (20), $z_{2,i} \oplus c_{3,i-1}$ is described by

$$z_{2,i} \oplus c_{3,i-1} = z_{2,i} \oplus (z_{2,i-1} \oplus c_{3,i-2})x_{1,i-1} \oplus z_{2,i-1}x_{2.i-1} \oplus (z_{2,i-1} \oplus x_{2,i-1})c_{3,i-2}. \tag{23}$$

Since $x_{1,i-1}$ only appears in $(z_{2,i-1} \oplus c_{3,i-2})x_{1,i-1}$, $ord(z_{2,i} \oplus c_{3,i-1}) \geq ord(z_{2,i-1} \oplus c_{3,i-2}) + 1$. Let $d_i = ord(z_{2,i} \oplus c_{3,i-1})$, $0 \leq i \leq 31$. Then $d_i \geq d_{i-1} + 1$. On the other hand,

$$\begin{aligned}
d_0 &= ord(z_{2,0} \oplus c_{3,-1}) \\
&= ord(z_{2,0} \oplus c_{2,31}) \\
&\geq 64.
\end{aligned}$$

Hence, $d_i \geq 64 + i$, $1 \leq i \leq 31$. By (22), $ord(z_i) \geq 64 + i$, which proves the lemma. □

Theorem 2 Let $\tilde{z} = z_0, z_1, \ldots$ denote the word sequence generated by SSC2. For any $n \geq 0$, let $z_n = (z_{31,n}, z_{30,n}, \ldots, z_{0,n})$. Let S''_0 be the initial state of the word-oriented LFSR and $y_0, y_1, \ldots, y_{16}$ be the initial state of the lagged-Fibonacci generator. Assume that S''_0 is not zero and $y_0, y_1, \ldots, y_{16}$ are not all even. Then, with a probability greater than $e^{-\frac{1}{127}}$, the binary sequences $\tilde{z}_i = z_{i,0}z_{i,1}\ldots, 0 \leq i \leq 31$, have linear complexity

$$\Lambda(\tilde{z}_i) \geq \sum_{j=1}^{64+i} \binom{127}{j} - 64 \geq 2^{126}.$$

Proof. Similar to the decomposition of $\tilde{z} = z_0, z_1, \ldots$, we can also decompose the word sequence $\tilde{z}' = z'_0, z'_1, \ldots$ generated by the filter generator into 32 component sequences, $\tilde{z}'_i = z'_{i,0}, z'_{i,1}, \ldots, 0 \leq i \leq 31$. Also let $\tilde{z}''_i = z''_{i,0}, z''_{i,1}, \ldots, 0 \leq i \leq 31$ denote the component sequences of $\tilde{z}'' = z''_0, z''_1, \ldots$ generated by the lagged-Fibonacci generator. Then $\tilde{z}_i = \tilde{z}'_i \oplus z''_i$. Let $T_{\tilde{z}_i}$,$T_{\tilde{z}'_i}$, and $T_{\tilde{z}''_i}$ denote the least period of $\tilde{z}_i$, $\tilde{z}'_i$, and $\tilde{z}''_i$. By Proposition 1, the word sequence $\tilde{z}' = z'_0, z'_1, \ldots$ has the least period $2^{127} - 1$. Hence, the component sequence $\tilde{z}'_i$ has a period of $32(2^{127} - 1)$. By Proposition 2, it is easy to verify that the sequence $\tilde{z}''_i$ has a period of $17(2^{17} - 1)2^{36}$. Therefore, $\gcd(T_{\tilde{z}'_i}, T_{\tilde{z}''_i}) \leq 32$. By (9), we have

$$\Lambda(\tilde{z}_i) \geq \Lambda(\tilde{z}'_i) - 64$$

By Lemma 3, with a probability no less than $e^{-\frac{1}{127}}$, the linear complexity of $\tilde{z}'_i$ is greater than or equal to L_{64+i}, which proves the theorem. □

Theorem 2 implies that the linear complexity of the component sequences $\tilde{z}_i, 0 \leq i \leq 31$, is exponential to the length of the LFSR.

After having established the period and linear complexity of the keystream sequences generated SSC2, we might now ask a normal question: how closely do the keystream sequences generated by SSC2 resemble a truly random sequence? Mathematically, a truly random sequence can be modeled by a sequence of independent and uniformly distributed random variables. To measure the randomness of the keystream sequences generated by SSC2, let's consider the distribution of every 32-bit word in a period.

Proposition 3 Assume that X_3, X_2, X_1, and X_0 are independent and uniformly random variables over $GF(2)^{32}$, $Z = F(X_3, X_2, X_1, X_0)$. Then Z is uniformly distributed over $GF(2)^{32}$.

Proof. Let $Z' = \langle X_3 + (X_0 \vee 1) \rangle_{16} + X_2 \oplus c_1(X_0 \vee 1) + c_2 \bmod 2^{32}$. Then $Z = X_1 \oplus X_2 + Z' \bmod 2^{32}$. By the chain rule [2], we can express the joint entropy $H(Z, Z', X_1 \oplus X_2)$ as follows

$$\begin{aligned} H(Z, Z', X_1 \oplus X_2) &= H(Z') + H(Z|Z') + H(X_1 \oplus X_2|Z, Z') \\ &= H(Z') + H(X_1 \oplus X_2|Z') + H(Z|X_1 \oplus X_2, Z'). \end{aligned}$$

Since $X_1 \oplus X_2$ is uniquely determined by Z and Z', $H(X_1 \oplus X_2|Z, Z') = 0$. For the similar reason, $H(Z|X_1 \oplus X_2, Z')$. Hence, $H(Z|Z') = H(X_1 \oplus X_2|Z')$.

Since X_1 does not appears in the expression represented by Z', X_1 and Z' are statistically independent. For any $a, b \in GF(2)^{32}$,

$$\begin{aligned} p(X_1 \oplus X_2 = a|_{Z'=b}) &= \frac{p(X_1 \oplus X_2 = a, Z' = b)}{p(Z' = b)} \\ &= \frac{\sum_{c \in GF(2)^{32}} p(X_1 \oplus X_2 = a|_{Z'=b, X_2=c}) p(Z' = b, X_2 = c)}{p(Z' = b)} \\ &= \frac{\sum_{c \in GF(2)^{32}} p(X_1 = a \oplus c|_{Z'=b, X_2=c}) p(Z' = b, X_2 = c)}{p(Z' = b)}. \end{aligned}$$

Since X_1 and (Z', X_2) are independent, $p(X_1 = c \oplus a|_{Z'=b, X_2=c}) = 2^{-32}$. Thus,

$$p(X_1 \oplus X_2 = a|_{Z'=b}) = \frac{2^{-32} \sum_{c \in GF(2)^{32}} p(Z' = b, X_2 = c)}{p(Z' = b)} = 2^{-32}.$$

Hence, $H(Z|Z') = H(X_1 \oplus X_2|Z') = 32$. Therefore, $H(Z) \geq H(Z|Z') = 32$, which implies that $H(Z) = 32$, or equivalently, Z is uniformly distributed over $GF(2)^{32}$. $\square$

Recall that the state sequence $S''_0, S''_1, \ldots$ of the word-oriented LFSR has the least period $2^{127}-1$ and any two states are not equal in the period if the initial state S''_0 is not zero. Hence every non-zero state appears exactly once in the least period. For this reason, we model the state S''_n as a uniformly distributed random variable over $GF(2)^{127}$. Proposition 3 indicates that we can model the filter generator sequence as a sequence of uniformly distributed random variables when the initial state of the filter generator is not zero.

Theorem 3 Let S''_0 and $Y_0 = (y_0, y_1, \ldots, y_{16})$ be the initial states of the LFSR and the lagged-Fibonacci generator in Figure 1. Assume that S''_0 and Y_0 are random variables, and $S''_0 \neq 0$. Let Z_n denote the word generated by SSC2 at time n. Then

$$32 - I(S''_0; Y_0) \leq H(Z_n) \leq 32.$$

Proof. Since Z_n is a random variable over $GF(2)^{32}$, it is obvious that $H(Z_n) \leq 32$. Let Z'_n and Z''_n denote the outputs of the filter generator and the lagged-Fibonacci generator at time n. Then $Z_n = Z'_n + Z''_n \bmod 2^{32}$. Moreover, $H(Z_n|Z''_n) = H(Z'_n|Z''_n)$. According to the data processing inequality [2],

$$I(Z'_n; Z''_n) \leq I(Z'_n; Y_0) \leq I(X''_0; Y_0).$$

Hence,

$$\begin{aligned} H(Z_n|Z''_n) &= H(Z'_n|Z''_n) \\ &= H(Z'_n) - I(Z'_n; Z''_n). \end{aligned}$$

Since $S''_0 \neq 0$, $H(Z'_n) \approx 32$. Therefore, $H(Z_n) \geq H(Z_n|Z''_n) \geq 32 - I(X''_0; Y_0)$. $\square$

By Theorem 3, we can conclude that the keystream sequence of SSC2 is a sequence of uniformly distributed random variables if the initial states of the word-oriented LFSR is non-zero and statistically independent of the initial state of the lagged-Fibonacci generator. However, We do not know whether the keystream sequence of SSC2 is a sequence of independent random variables or not. We leave this problem for the future research.

4. Correlation Analysis of SSC2

SSC2 is a very complex mathematical system in which several different types of operations, such as exclusive-or, integer addition, shift, and multiplexing, are applied to data iteratively. If we analyze the keystream generator in Figure 1 as a whole, it would be difficult to get information about the internal states of the word-oriented LFSR and the lagged-Fibonacci generator. However, if the keystream sequence leaks information about the filter generator sequence or the lagged-Fibonacci sequence, this information might be exploited to attack the filter generator or the lagged-Fibonacci generator separately. This kind of attack is called divide-and-conquer correlation attack which has been successfully applied to over a dozen keystream generators [3,8,10,11,14,15]. For the moment, let's assume that the key of SSC2 consists of the initial states S_0'' and Y_0 of the word-oriented LFSR and the lagged-Fibonacci generator.

Theorem 4 Assume that the initial states S_0'' and Y_0 of the filter generator and the lagged-Fibonacci generator of SSC2 are random variables, $S_0'' \neq 0$. Then the outputs Z_n' and Z_n'' of the filter generator and the lagged-Fibonacci generator at time n are also random variables. Let Z_n denote the output of SSC2 at time n. Then

$$\begin{aligned} I(Z_n;Y_0) &\leq I(S_0'';Y_0), \\ I(Z_n;S_0'') &\leq 32 - H(Z_n'') + I(S_0'';Y_0). \end{aligned}$$

Proof. By the chain rule [2], the joint entropy $H(Z_n', Y_0, Z_n)$ can be represented as follows:

$$\begin{aligned} H(Z_n', Y_0, Z_n) &= H(Y_0) + H(Z_n|Y_0) + H(Z_n'|Z_n, Y_0) \\ &= H(Y_0) + H(Z_n'|Y_0) + H(Z_n|Z_n', Y_0). \end{aligned}$$

Since Z_n' can be uniquely determined by Z_n and Y_0, $H(Z_n'|Z_n, Y_0) = 0$. For the similar reason, $H(Z_n|Z_n', Y_0) = 0$. Thus, $H(Z_n|Y_0) = H(Z_n'|Y_0)$. Therefore,

$$\begin{aligned} I(Z_n;Y_0) &= H(Z_n) - H(Z_n|Y_0) \\ &= H(Z_n) - H(Z_n'|Y_0) \\ &= H(Z_n) - H(Z_n') + I(Z_n';Y_0) \end{aligned}$$

According to the data processing inequality [2], $I(Z_n';Y_0) \leq I(S_0'';Y_0)$. Hence, it follows that

$$I(Z_n;Y_0) \leq H(Z_n) - H(Z_n') + I(S_0'';Y_0).$$

On the other hand, $H(Z_n) \leq 32$, while $H(Z_n') \approx 32$. Thus,

$$I(Z_n;Y_0) \leq I(S_0'';Y_0).$$

Similarly, it can be proved that

$$I(Z_n;S_0'') \leq 32 - H(Z_n'') + I(S_0'';Y_0).$$

□

According to the empirical test in [7], let's assume that the sequence $\tilde{y} = y_0, y_1, \ldots$ of the lagged-Fibonacci generator is a sequence of pairwise independent and uniformly distributed random variables. By (6), $Z_n'' = \langle y_n \rangle_{16} + y_n' \bmod 2^{32}$, where y_n' is uniformly selected from $y_{n+1}, y_{n+2}, \ldots, y_{n+17}$. If $y_n' \neq y_{n+17}$, it is obvious that Z_n'' is uniformly distributed. If $y_n' = y_{n+17}$, then $Z_n'' = \langle y_n \rangle_{16} + y_n + y_{n+12} \bmod 2^{32}$, which is also uniformly distributed. Thus, for any $n \geq 0$, Z_n is not correlated to either S_0'' or Y_0 if S_0'' and Y_0 are statistically independent. So we can not get any information about S_0'' or Y_0 from each Z_n. However, this does not mean that we can not get information about S_0'' and Y_0 from a segment $Z_0, Z_1, \ldots, Z_m$ of the keystream sequence. The question is how to get information about S_0'' and Y_0 from a segment of the keystream sequence, which is a problem that deserves further research.

5. Key Scheduling Scheme

SSC2 supports private keys of various sizes, from 4 bytes to 16 bytes. To stretch a private key less than or equal to 4 words to 21 words, a key scheduling scheme is required. By Theorem 3 and Theorem 4, the initial states of the word-oriented LFSR and the lagged-Fibonacci generator had better be independent. With a hash function such as SHA-1[9], it is not difficult to generate such 21 words. When a good hash function is not available, we designed the following scheme which generates 21 words (called the master key) from the private key K.

Master-Key-Generation $K_{master}(K)$

1 load K into the LFSR S, repeat K when necessary
2 **for** $i \leftarrow 0$ **to** 127 **do**
3   run the linear feedback shift register once
4     $S[1] \leftarrow S[1] + F(S) \bmod 2^{32}$
5     $i \leftarrow i + 1$
6 **for** $i \leftarrow 1$ **to** 17 **do**
7   run the linear feedback shift register once
8     $\mathsf{B}[i] \leftarrow S[4]$
9     $i \leftarrow i + 1$
10 $A \leftarrow S[1]$
11 **for** $i \leftarrow 1$ **to** 34 **do**
12   run the linear feedback shift register once
13   run the lagged Fibonacci generator once
14     $index \leftarrow 1 + A \gg 28$
15     $A \leftarrow \mathsf{B}[index]$
16     $\mathsf{B}[index] \leftarrow A \oplus S[1]$
17     $S[1] \leftarrow A + S[1] \bmod 2^{32}$
18     $i \leftarrow i + 1$
19 $\mathsf{B}[17] \leftarrow \mathsf{B}[17] \vee 1$
20 **return** S and B

In the above pseudo-code, the LFSR is denoted by S, which is an array of 4 words. Every time when the LFSR runs, the word in $S[4]$ moves out, the array shifts right one word, and the newly computed word moves in $S[1]$. The lagged-Fibonacci generator is denoted by B as usual. The master key generation experiences 5 stages. In stage 1, the private key is loaded into the LFSR. In stage 2 (line 2 - line 5) , the private key is processed. We run the LFSR (actually the filter generator) 128 times in order that approximately half of the bits in S will be 1 even though that is only one 1 in K. In stage 3 (line 6 - line 9), 17 words are generated for the lagged-Fibonacci generator. In stage 4 (line 10 - line 18) , the LFSR and the lagged-Fibonacci generator are interact with each other for 34 times. A major goal for the interaction is to make it difficult to gain information about the state of the LFSR from the state of the lagged-Fibonacci generator and vice versa. For this purpose, an index register A is introduced. Initially, A has the content equal to $S[1]$. Every time after the LFSR and the lagged-Fibonacci generator run, a pointer $index$ is computed according to the most significant 4 bits of A, and then A is updated by the word $\mathsf{B}[index]$. Following the update of A, $\mathsf{B}[index]$ is updated by $A \oplus S[1]$ and $S[1]$ is updated by $A + S[1] \bmod 2^{32}$. Through the register A, the states of the LFSR and the lagged-Fibonacci generator are not only related to each other but are also related to their previous states. For example, assume that the state of the LFSR is known at the end of stage 4. To obtain the previous state of the LFSR, we have to know the content of A, which in derived from the previous state of the lagged-Fibonacci generator. In stage 5, the least significant of $\mathsf{B}[17]$ is set to 1 in order to make sure that the 17 words of B are not all even. Afterwards the states of the LFSR and the lagged-Fibonacci generator are output as the master key.

In addition to the master key generation, SSC2 supplies an optional service. It can generate a key for every frame. The key for a frame is used to re-load the LFSR and the lagged-Fibonacci generator when the frame is encrypted. The purpose of frame-key generation is to cope with the synchronization problem. In wireless communications, there is a high probability that packets may be lost due to noise, or synchronization between the mobile station and the base station may be lost due to signal reflection, or a call might be handed off to a different base station as the mobile station roams. When frames are encrypted with their individual keys, the event that a frame was lost will not affect the decryption of other frames.

Assume that each frame is labeled by a 32-bit frame number that is not encrypted. Let K_n denote the frame key of the n-th frame. The frame key generation should satisfy two fundamental requirements: (1) it is fast to generate a frame key; and (2) it is difficult to gain information about K_i from K_j when $i \neq j$. Taking into consideration of the two requirements, we designed a scheme that generates K_n from the master key K_{master} and the frame number n. To generate different keys for different frames, we divide the 32-bit frame number into 8 consecutive blocks and have each block involve in the frame key generation. Let $n_0, n_1, \ldots, n_7$ denote the 8 blocks of n, where n_0 is the least significant 4 bits of n and n_7 is the most significant 4 bits of n. The frame key generation is demonstrated by the following pseudo-code:

Frame-Key-Generation $K_n(K_{master})$
1 load K_{master} into S and B
2 **for** $j \leftarrow 0$ **to** 3 **do**
3   **for** $i \leftarrow 0$ **to** 7 **do**
4         $S[1] \leftarrow S[1] + \mathsf{B}[1 + (i + n_i \bmod 16)] \bmod 2^{32}$
5         $S[2] \leftarrow S[2] + \mathsf{B}[1 + (8 + i + n_i \bmod 16)] \bmod 2^{32}$
6   run the linear feedback shift register one time
7         $\mathsf{B}[17 - (i + 8j \bmod 16)] \leftarrow S[1] \oplus \mathsf{B}[17 - (i + 8j \bmod 16)]$
8         $i \leftarrow i + 1$
9     $j \leftarrow j + 1$
10 $\mathsf{B}[17] \leftarrow \mathsf{B}[17] \vee 1$
11 **return** S and B

The frame key generation consists of two loops. Corresponding to each $n_i, 0 \leq i \leq 7$, the inner loop (line 4 - line 8) selects two words from the buffer B to update the contents of $S[1]$ and $S[2]$. Then the LFSR runs and the newly computed word of the LFSR is used to update one word of B. The outer loop executes the inner loop 4 times. Assume that n and n' are two different frame numbers. In the generation of K_n and $K_{n'}$, the content of S will be different after the first execution of the inner loop. The following 3 executions of the inner loop make the contents of S and B different, and hence generates different frame keys.

6. Performance

We have run SSC2 on various platforms. Table 4.1 illustrates the experiment results derived by running the ANSI C code listed in Appendix 1. Key setup times are not included in Table 1. On a 16-bit processor (233MHz cpu), the time for the master key generation is approximately equal to the encryption time for one CDMA packet (384 bits in 20 ms duration), and the time for the frame key generation is about one-twentieth of the encryption time for one CDMA packet. Suppose that the CDMA phones have the 16-bit Pentium processor and the average conversion takes 3 minutes. Then the total time for frame key generations is about 2 ms. Hence, the overhead induced by key setup is nearly negligible.

Table 1: Throughput of SSC2

Machine	Size	Clock rate (MHz)	Memory (Mbyte)	OS	Compiler	Throughput (Mbits/s)
Sun SPARC2	32	40	30	Sun OS	gcc -O3	22
Sun Ultra 1	32	143	126	Sun Solaris	gcc -O3	143
PC	16	233	96	Linux	gcc -O3	118

7. Conclusion

SSC2 is a fast software stream cipher portable on 8-, 16-, and 32-bit processors. All operations in SSC2 are word-oriented, no complex operations such as multiplication, division, and exponentiation are involved. SSC2 has a very compact structure, it can be easily remembered. SSC2 does not use any look-up tables and does not need any pre-computations. Its software implementation requires very small memory usage. SSC2 supports varies private key sizes, it has an efficient key scheduling scheme and an optional frame key scheduling scheme. Its keystream sequence has large period, large linear complexity and small correlation to the component sequences. SSC2 is one of the few software stream ciphers whose major cryptographic properties have been established.

References

[1] R. P. Brent,"On the periods of generalized Fibonacci recurrences", *Mathematics of Computation*, vol. 63, pp. 389-401, 1994.

[2] T. M. Cover and Y. A. Thomas, *Elements of Information Theory*, John Wiley, 1991.

[3] R. Forre, "A fast correlation attack on nonlinearly feedforward filtered shift register sequences", *Advance in Cryptology-Proceedings of EUROCRYPT'89 (LNCS 434)*, 586-595, 1990.

[4] F. James, "A review of pseudo-random number generators", *Computer Physics Communications*, vol. 60, pp. 329-344, 1990.

[5] E. L. Key, "An analysis of the structure and complexity of nonlinear binary sequence generators", *IEEE Transactions on Infor. Theory*, vol. 22, Nov. 1976.

[6] D. E. Knuth, *The Art of Computer programming. Volume 2: Seminumerical Algorithms*, 3rd Edition, Addison-Wesley, 1997.

[7] G. Marsaglia, "A current view of random number generators", *Computer Science and Statistics: Proc. 16th Symposium on the Interface*, Elsvier Science Publishers B. V. (North-Holland), 1985.

[8] W. Meier and O. Staffelbach, "Fast correlation attacks on stream ciphers", *Journal of Cryptology*, vol. 1, no. 3, 159-176, 1989.

[9] FIPS 180, "Secure hash standard", *Federal Information Processing Standard Publication* 180, April, 1995.

[10] M.J.B. Robshow, "Stream ciphers", Technical Report TR-701 (version 2.0), RSA Laboratories, 1995.

[11] R. Rueppel, "Stream Ciphers" in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 65-134.

[12] R. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, Berlin, 1986.

[13] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, New York, 2nd edition, 1996.

[14] T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only", *IEEE Transactions on Computing*, vol. 34, 1985, 81-85.

[15] T. Siegenthaler, "Cryptanalyst's representation of nonlinearity filtered ml-sequences", *Advance in Cryptology-Proceedings of Eurocrypt'85 (LNCS)*, Springer-Verlag, 1986.

[16] A. Biryukov and A. Shamir, "Real time cryptanalysis of the alleged A5/1 on a PC", unpublished draft, http://cryptome.org/a5.ps, 1999.

Appendix 1

The following is the ANSI C code for the keystream generator of SSC2. The code for key setup is not included.

```
unsigned long int R1, R2, R3, R4, B[18], output, temp1, temp2;
int c, s=5, r=17;

temp1 = R2 ^ (R3<<31) ^ (R4>>1);
R4 = R3;
R3 = R2;
R2 = R1;
R1 = temp1;

temp1 = B[r];
temp2 = B[s] + temp1;
B[r] = temp2;
if (--r == 0) r = 17;
if (--s == 0) s = 17;
output = ((temp1>>16) ^ (temp1<<16)) + B[(((temp2>>28) + s) & 0xf)+1] ;

temp1 = (R4 | 0x1) + R1;
c = (temp1 < R1);
temp2 = (temp1<<16) ^ (temp1>>16);
if (c) {
   temp1 = (R2  ^ (R4 | 0x1)) + temp2;
} else {
   temp1 = R2 + temp2; }
c = (temp1 < temp2);
output = (c + (R3 ^ R2) + temp1) ^ output;
```