Efficient Software Implementation of LFSR and Boolean Function and Its Application in Nonlinear Combiner Model

Sandeepan Chowdhury and Subhamoy Maitra

Applied Statistics Unit, Indian Statistical Institute
203 B T Road, Kolkata, Pin 700 108, INDIA
Email : sandeepan@consultant.com, subho@isical.ac.in

.

Abstract. Here we present an efficient implementation strategy and some general design criteria for the standard nonlinear combiner model. This model combines the output sequences of several independent Linear Feedback Shift Registers (LFSRs) using a Boolean function to produce the running key sequence. The model is well studied and a standard target for many cryptanalytic attacks. The naive bitwise software implementation of the LFSRs is not efficient. In this paper we explore an efficient block oriented software implementation technique to make it competitive with the recently proposed fast stream ciphers. Our proposed specifications on this model can resist the fast correlation attacks. To evaluate our design criteria and implementation techniques, we carry out the security and performance analysis considering a specific scheme based on this model.

Keywords: Linear Feedback Shift Register, Block Oriented Software Implementation, Boolean Function, Resiliency, Nonlinearity, Algebraic Degree.

1 Introduction

Linear Feedback Shift Registers (LFSRs) are the main building block of most of the stream cipher systems. The slow software realization of the bit oriented LFSRs (i.e., LFSRs over GF(2)) reduces the efficiency of LFSR-based stream ciphers in software. To improve software implementation, recently proposed fast stream ciphers like SNOW [8, 9], t-classes of SOBER [18], TURING [19] have opted for word-oriented LFSRs which are actually LFSRs over GF(2^b). The value of b is taken as 8/16/32 depending on different word sizes of the processor. However, these newly proposed fast stream ciphers are not yet time-tested and some of the newly proposed stream ciphers, like SSC2 [24], SNOW (Version-1.0), t-classes of SOBER [18], despite being very fast in software, found to have certain weaknesses in their design. In this context we like to draw attention to the well-known LFSR-based nonlinear combiner model of stream cipher. In the standard nonlinear combiner model (presented in Figure 1) the output sequences of several independent Linear Feedback Shift Registers (LFSRs) are combined using the nonlinear Boolean function to produce the running key stream K. The n LFSRs and the Boolean function are presented as $S_1, \ldots, S_n$ and f (see Figure 1) respectively. This key stream K is bitwise XORed with the message bit stream M to produce the cipher C. The decryption machinery is identical to the encryption machinery. The initial conditions of the LFSRs constitute the secret key of the system. There are several variants of this model where the

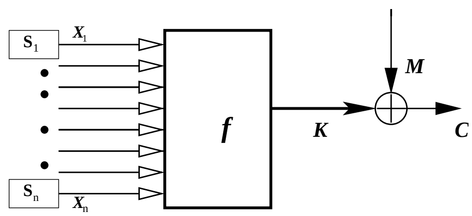

Fig. 1. Nonlinear Combiner Model.

LFSRs may be clock-controlled or the Boolean function may contain a few bits of memory. The basic nonlinear combiner model is a standard target for most of the correlation attacks [22, 15, 7, 1, 2, 11, 16]. As the correlation attacks on this model appear to be stabilized, it is now possible to fix the proper design criteria for the LFSRs [15, 1, 14] as well as that of the Boolean function [20, 17, 23]. So considering the present scenario, we re-examine this model in detail. At this point, let us highlight the salient features of this initiative.

1. The slow software realization of the bit oriented LFSRs reduces the efficiency of this model in software. So, for the realization of LFSRs over GF(2), we propose a block oriented technique to get considerable speed up in software. Further, we show that the algebraic normal form representation of Boolean functions can be efficiently used for block oriented strategy.
2. We start the design procedure of the basic nonlinear combiner model to resist the fast correlation attack presented by Chepyzhov, Johansson and Smeets [2]. Later, it has been checked that our design methodology provides robustness against the attack proposed by Canteaut and Trabbia [1]. We suitably incorporate recent theoretical developments in design of Boolean function and LFSRs to choose parameters of each component of the system.
3. On the basis of our analysis and implementation techniques, we present a concrete scheme, specifying the exact LFSR connection polynomials and the Boolean function. Analysing this scheme we show that it is robust against existing fast correlation attacks which performs better than the exhaustive key search. We also show that following the proposed block oriented technique, software implementation of the scheme is quite competitive with most of the other recently proposed fast software stream ciphers. However, low linear complexity of the generated scheme still remains a problem. Some modifica-

tion on the basic nonlinear combiner model may remove this weakness and make the model suitable for practical purpose.

The organization of the paper is as follows. Section 2 presents some preliminary notions. In Section 3, techniques for efficient software implementation of LFSRs and the Boolean function have been discussed. In Section 4 we develop the design approach on the basis of the existing attacks. On the basis of our design strategy, we analyse a specific scheme in Section 5.

2 Preliminaries

The basic components of the nonlinear combiner model are some LFSRs and a Boolean function.

An LFSR of length d generates a binary pseudorandom sequence following a recurrence relation over GF(2). The addition operator over $GF(2)$ is denoted by $\oplus$. It takes a small seed of length d and expands it to a much larger binary pseudorandom sequence with high periodicity. We consider a degree d primitive polynomial over GF(2) of the form $x^d \oplus \bigoplus_{i=0}^{d-1} a_i x^i$. The corresponding binary recurrence relation is $s_{j+d} = \bigoplus_{i=0}^{d-1} a_i s_{j+i}$, $j \geq 0$. Note that one uses a primitive polynomial to get maximum possible periodicity, i.e., $2^d - 1$. This polynomial is called the "connection polynomial" of the LFSR. The weight of this connection polynomial is the number of places where the coefficient has the value 1, i.e., $1 + \#\{a_i = 1\}$. The taps of the LFSR come from the positions where $a_i = 1$ and there is no tap at the positions with $a_i = 0$.

By a τ-nomial multiple of the connection polynomial, we mean a multiple of the form $x^{i_1} + x^{i_2} + \ldots + x^{i_{\tau-1}} + 1$ with degree less than $(2^d - 1)$.

An n-variable Boolean function f is defined as $f : GF(2^n) \rightarrow GF(2)$. Now we present some definitions relevant to Boolean functions.

Definition 1. *For binary strings S_1, S_2 of same length λ, we denote by $\#(S_1 = S_2)$ (respectively $\#(S_1 \neq S_2)$), the number of places where S_1 and S_2 are equal (respectively unequal). Hamming distance between S_1, S_2 is denoted by $d(S_1, S_2)$, i.e., $d(S_1, S_2) = \#(S_1 \neq S_2)$. Also the Hamming weight or simply the weight of a binary string S is the number of ones in S. This is denoted by $wt(S)$. An n-variable function f is said to be balanced if its output column in the truth table contains an equal number of 0's and 1's (i.e., $wt(f) = 2^{n-1}$).*

Definition 2. *An n-variable Boolean function $f(X_1, \ldots, X_n)$ can be considered to be a multivariate polynomial over $GF(2)$. This polynomial can be expressed as a sum of products representation of all distinct r-th order products $(0 \leq r \leq n)$ of the variables. More precisely, $f(X_1, \ldots, X_n)$ can be written as $a_0 \oplus (\bigoplus_{i=1}^{i=n} a_i X_i) \oplus (\bigoplus_{1 \leq i \neq j \leq n} a_{ij} X_i X_j) \oplus \ldots \oplus a_{12\ldots n} X_1 X_2 \ldots X_n$ where the coefficients $a_0, a_i, a_{ij}, \ldots, a_{12\ldots n} \in \{0, 1\}$. This representation of f is called the algebraic normal form (ANF) of f. The number of variables in highest order product term with nonzero coefficient is called the algebraic degree, or simply degree of f.*

We will later show that using ANF helps faster implementation of Boolean functions when realized in block oriented manner.

Definition 3. *Functions of degree at most one are called affine functions. An affine function with constant term equal to zero is called a linear function. The set of all n-variable affine (respectively linear) functions is denoted by $A(n)$ (respectively $L(n)$). The nonlinearity of an n variable function f is $nl(f) = min_{g \in A(n)}(d(f,g))$, i.e., the distance from the set of all n-variable affine functions.*

Definition 4. *Let $\overline{X} = (X_1, \ldots, X_n)$ and $\overline{\omega} = (\omega_1, \ldots, \omega_n)$ both belong to $\{0,1\}^n$ and $\overline{X} \cdot \overline{\omega} = X_1\omega_1 \oplus \ldots \oplus X_n\omega_n$. Let $f(\overline{X})$ be a Boolean function on n variables. Then the Walsh transform of $f(\overline{X})$ is a real valued function over $\{0,1\}^n$ that can be defined as $W_f(\overline{\omega}) = \sum_{\overline{X} \in \{0,1\}^n} (-1)^{f(\overline{X}) \oplus \overline{X} \cdot \overline{\omega}}$.*

Definition 5. [10] *A function $f(X_1, \ldots, X_n)$ is m-th order correlation immune (CI) iff its Walsh transform W_f satisfies $W_f(\overline{\omega}) = 0$, for $1 \leq wt(\overline{\omega}) \leq m$. Also f is balanced iff $W_f(\overline{0}) = 0$. Balanced m-th order correlation immune functions are called m-resilient functions. Thus, a function $f(X_1, \ldots, X_n)$ is m-resilient iff its Walsh transform W_f satisfies $W_f(\overline{\omega}) = 0$, for $0 \leq wt(\overline{\omega}) \leq m$.*

By an (n, m, u, x) function we mean an n-variable, m-resilient function with degree u and nonlinearity x. It is known that for $m > \frac{n}{2} - 2$, the maximum possible nonlinearity can be $2^{n-1} - 2^{m+1}$ and such functions have three valued Walsh spectra [20]. The maximum possible algebraic degree of such functions is $n-m-1$ [21]. Construction of such functions has been demonstrated in [23, 17].

3 Software Implementation

In this section we discuss the block oriented software implementation of LFSRs and the Boolean function to enhance the processing speed.

3.1 Software Implementation of LFSRs

Hardware implementation of an LFSR generates a single bit per clock. We denote the output bit sequence (of N bits) as $\{s_j\}$, $j \leq 0 < N$. In naive software implementation, more than one (say v) logical operations (bitwise XOR, AND, and bit shifting) are required to generate a single output bit. The motivation behind the fast software implementation of an LFSR is to generate a block of b output bits in $< bv$ logical operations. For this purpose we discuss the block oriented implementation, where the LFSR outputs one block at a time. We denote this output of b bit vectors as a "block", i.e., $\mathbf{w}_j = \{s_{jb+(b-1)}, \ldots, s_{jb+1}, s_{jb}\}$. Thus a sequence of N output bits is now obtained as N' blocks $\{\mathbf{w}_j\}, 0 \leq j < N'$, where $N = N'b$. For convenience of storage and operations in processors, it is natural to take $b = 8, 16, 32, 64$ etc.

First we discuss a block oriented implementation of an LFSR by matrix operation. The standard matrix representation of an LFSR can be seen as $\mathbf{x}_b = \mathcal{A}^b \mathbf{x}_0$, where $\mathcal{A}$ is the $d \times d$ state transition matrix and $\mathbf{x}_0$ is the d-bit initial state vector of the LFSR [12]. Note that, to get one b length block we need a binary matrix multiplication. For LFSRs of high length (say > 100), even precomputation and efficient exponentiation (for calculating $\mathcal{A}^b$) will render this technique slow.

Another approach is to extend the bitwise recurrence relation of an LFSR $s_{j+d} = \bigoplus_{i=0}^{d-1} a_i s_{j+i}$, to that over blocks $\mathbf{w}_{j+d} = \bigoplus_{i=0}^{d-1} a_i \mathbf{w}_{j+i}$. For implementing this block oriented relation, we consider an LFSR consists of d blocks corresponding to the original LFSR of d bits. We denote the LFSR blocks by $(S[d-1], \ldots, S[1], S[0])$ from left to right. The initial state is represented as $S[d-1] = \mathbf{w}_{d-1}, \ldots, S[1] = \mathbf{w}_1, S[0] = \mathbf{w}_0$. So one needs the first bd bits of the LFSR bit sequence s_j to initialize these d blocks. Once initialized, the block oriented recurrence relation can generate the output sequence of blocks i.e., $\{\mathbf{w}_j\}_{j \geq 0}$. Only $\frac{t}{b}$ logical operations are required per output bit, where t is the total number of taps. Considering a primitive trinomial (i.e., $t = 2$) and block size $b = 64$, we get $\frac{t}{b} = \frac{1}{32}$. Precomputation involving the generation of bd bits for initialization of the d blocks makes the process slow and memory dependent (≈ 3 KByte generation and storage for $d \approx 300$). Further, for synchronized operation, every reinitialization of initial d blocks is a time consuming affair. For fast implementation in software, by hard coding the LFSR involves using distinct variables for the d LFSR blocks. It is not practical when $d \approx 300$. In that case, using arrays for storing the LFSR needs indirect addressing at machine level and the implementation becomes inefficient.

We extend the idea of [24, 3]. According to this approach, the LFSR (length d) consists of y number of blocks, each of length b, i.e., $d = yb$. Here we denote the blocks $(S[y-1], \ldots, S[1], S[0])$ from left to right. Initially $S[y-1] = \mathbf{w}_{y-1}, \ldots, S[0] = \mathbf{w}_0$. The block oriented recurrence relation [3] for the LFSR is

$$\mathbf{w}_{j+y} = \bigoplus_{i'=0}^{t-1} \left(\left(\mathbf{w}_{j+q_{i'}+1} << (b - r_{i'}) \right) \oplus \left(\mathbf{w}_{j+q_{i'}} >> r_{i'} \right) \right), \text{ for } j \geq 0. \quad (1)$$

Here $bq_{i'} + r_{i'} = p_{i'}$, and $p_0, p_1, \ldots, p_{t-1}$ are the tap positions. It may be noted that taps are only between the positions 0 and $d - b + 1$. Number of logical operations required for each output bit is $\frac{t_1 + 4t_2}{b}$. Here, t_1 is the number of boundary taps, i.e., at any of the positions $0, b, 2b, \ldots, d(b-1)$, and t_2 is the number of non-boundary taps. Total number of taps $t = t_1 + t_2$. Considering primitive trinomials, $t_1 = 1$, $t_2 = 1$. Thus for $b = 64$, the number of logical operation per bit is $\frac{5}{64}$. Also the memory requirement is only $y = \frac{d}{b}$ blocks instead of d blocks as mentioned earlier. Further the initialization can be done directly without any precomputation. Unfortunately, the method presented in [24, 3] works for LFSRs of size $d = yb$, i.e., the degree has to be multiple of block size. Here we extend this technique for LFSR of any length $d = yb + a, 0 < a < b$, where d may not be a multiple of the block size b. For the sake of faster implementation, we consider that there is no tap between the positions $d - 1$ and

$(d-b-a+1)$. Here, an LFSR (denote it by M_d) of length $d = yb + a, 0 < a < b$ is assumed to consist of $y+1$ blocks, namely $(S[y], S[y-1], \ldots, S[0])$ from left to right. In order to make all the blocks of same length b, we pad the leftmost $(b-a)$ bits of the block $S[y]$ by zeroes. So at any stage we denote its state by the bit vector $\mathbf{D}_j = \{0, \ldots, 0, s_{j+d-1}, \ldots s_{j+yb}\}, j \geq 0$. Initially, $S[y] = \mathbf{D}_0, S[y-1] = \mathbf{w}_{y-1}, \ldots, S[0] = \mathbf{w}_0$. The y blocks $S[y-1], \ldots, S[0]$ can be viewed as an LFSR $(M_{d'})$ of length $d' = yb$ and the block $S[y]$ is considered separately. Recurrence relation for LFSR $(M_{d'})$ is obtained from Equation (1). The output of the LFSR (M_d) is dependent on both $M_{d'}$ and $S[y]$. Se, we need to define two simultaneous block oriented recurrence relations, for the LFSR M_d. These can be obtained as,

$$\mathbf{w}_{j+y} = (\mathbf{w'}_j << a) \oplus \mathbf{D}_{j-1} \text{ for } j \geq 0, \text{ for } M_{d'} \text{ and}$$

$$\mathbf{D}_j = \mathbf{w'}_j >> (b-a) \text{ for } j \geq 0 \text{ for single block } S[y].$$

Here, $\mathbf{w}'_j$ can be obtained from Equation (1) considering the LFSR $M_{d'}$.

Note that $\frac{t_1+4t_2+3}{b}$ logical operations are required for each output bit. Using a primitive trinomial $(t_1 = 1)$ as connection polynomial, only $\frac{8}{b}$ logical operations are required for each bit on average. The storage requirement is only $y+1$ blocks and no precomputation is required for initialization of the blocks. The implementation can be made faster by hard coding the LFSR in the program code, which involves use of only $y+1$ variables in the memory to represent the LFSRs. The pseudo code for updating the LFSR after generation of each new block will be as follows.

$$newS[0] = S[1],\ newS[1] = S[2],\ \ldots,\ newS[y-2] = S[y-1];$$

$$newS[y-1] = (newBlock << a) \oplus S[y],\ newS[y] = newBlock >> (b-a);$$

Here, $newS[\,]$ corresponds to the new state of a block $S[\,]$ and $newBlock$ corresponds to the new block obtained from LFSR $M_{d'}$. So, considering all the aspects we advocate this method for fast software implementation of an LFSR. Next we present some experimental results in Table 1.

Table 1. LFSR performance for different block size.

t	$b = 64$		$b = 32$	
	cycles/byte	op./byte	cycles/byte	op./byte
2	8.04	1	8.76	2
4	14.12	2	9.29	4
6	16.80	3	9.77	6
8	20.74	4	11.08	8
10	24.14	5	12.15	10
12	28.43	6	12.70	12

We consider an LFSR of length 377 and calculate the number of cycles required per byte of LFSR output in actual "C" language implementation (P-IV

2.4 GHz processor, gcc compiler, LINUX (RedHat Version 8) operating system) of the technique described above. Two different block sizes of 32 and 64 have been considered. The available data type "unsigned long long" in gcc/cc compilers provides this 64 bit storage block. The corresponding theoretically calculated values of required number of logical operations per output byte (denoted as op./byte) have also been provided. For all the cases $t_1 = 1$. Note that the implementation speed is much worse than the theoretical speed and it needs some more effort to reduce the gap.

3.2 Software Implementation of Boolean Function

The truth table of the Boolean function can be implemented in software using a look up table (of 2^n bits) which gives one bit output for n bit input. In order to combine the n output blocks generated by the n LFSRs, the ANF of Boolean function provides an interesting option. First we present the following construction method [23, 17].

Construction 31 *Let f be an (n, m, d, x) function ($m > \frac{n}{2} - 2$), $f = (1 \oplus X_n)f_1 \oplus X_n f_2$, where f_1, f_2 are $(n-1)$-variable m-resilient functions. Let $F = X_{n+2} \oplus X_{n+1} \oplus f$ and $G = (1 \oplus X_{n+2} \oplus X_{n+1})f_1 \oplus (X_{n+2} \oplus X_{n+1})f_2 \oplus X_{n+2} \oplus X_n$. Also $H = (1 \oplus X_{n+3})F \oplus X_{n+3}G$. The function H constructed from f above is an $(n+3, m+2, d+1, 2^{n+1}+4x)$ function. Moreover, F, G are both $(n+2)$-variable, $(m+2)$-resilient functions.*

Table 2. Construction of Boolean function

Algebraic Normal Form	&	$\oplus$	Total
$f = (1 \oplus X_n)f_1 \oplus X_n f_2$	2	2	4
$F = X_{n+2} \oplus X_{n+1} \oplus f$	0	2	2
$G = (1 \oplus X_{n+2} \oplus X_{n+1})f_1 \oplus (X_{n+2} \oplus X_{n+1})f_2 \oplus X_{n+2} \oplus X_n$ $= f_1 \oplus (X_{n+2} \oplus X_{n+1})(f_1 \oplus f_2) \oplus X_{n+2} \oplus X_n$	1	5	6
$H = (1 \oplus X_{n+3})F \oplus X_{n+3}G$	2	2	4
Total logical operations			16

Now consider that the algebraic normal form of f_1, f_2 are available and they need l_1, l_2 number of logical operations (AND, XOR) respectively. Following Construction 31, we require $l_1 + l_2 + 16$ logical operations to derive the function H. Table 2 gives step wise breakup of required logical operations. We consider the input variables $(X_1, \ldots, X_n)$ are n output blocks from the LFSRs. So the output of the Boolean function is one block of b bits after $l_1 + l_2 + 16$ operations. Thus b bits are produced in $l_1 + l_2 + 16$ operations. So on an average, one bit is generated in $\frac{l_1+l_2+16}{b}$ operation(s).

4 The Design Approach

Existing correlation attacks on the nonlinear combiner model exploits the correlation between the cipher text bit sequence and the bit sequence coming out of one or more LFSRs. To be specific, one considers the correlation between the sequence generated by one or more LFSRs and the cipher text C [21, 22] (in turn K). Here we consider an (n, m, d, x) Boolean function f. It is known that there always exists a linear function $\bigoplus_{i=1}^{n} \omega_i X_i$ (with $wt(\omega_1, \ldots, \omega_n) = m + 1$) such that the correlation coefficient between the stream coming out of f and the linear function $\bigoplus_{i=1}^{n} \omega_i X_i$ is not zero.

Definition 6. *The sequence σ generated by XORing the sequences generated by the individual LFSRs, say, $S_{i_1}, \ldots, S_{i_{m+1}}$ is same as that produced by the LFSR S with length $L = \sum_{i=1}^{m+1} d_i$ and connection polynomial $\psi(x) = \prod_{j=1}^{m+1} c_{i_j}(x)$ [7]. We refer to this single LFSR of length L and connection polynomial $\psi(x)$ as the equivalent LFSR.*

The nonlinear combiner model can be attacked if the initial condition of an equivalent LFSR can be obtained. The standard models of correlation attacks recover the initial condition of a target LFSR from a perturbed version of the original output sequence. These attacks fit into the nonlinear combiner model, if we consider the equivalent LFSR as the target LFSR, the sequence σ as the original sequence and the running key sequence K as the perturbed sequence. Note that it is generally not possible to get K and the perturbed sequence is actually C. Naturally the attacker will consider the smallest length equivalent LFSR. So from the designer's viewpoint, fix the design parameters of the nonlinear combiner model to make the smallest length equivalent LFSR robust against correlation attacks.

The fast correlation attack proposed in [15] improved the complexity of this exhaustive search to $2^{\beta L}$ where $\beta < 1$. It has been mentioned in [15, Section 1] that the attack is not feasible if the target LFSR has more than 10 taps. So the weight of the equivalent LFSR S must be > 10. However one also needs to ensure that a high weight connection polynomial does not have a sparse multiple of low degree [15, Section 5]. This criteria has been dealt in more detail later.

The modified fast correlation attacks developed subsequently [2, 11, 1, 16] and they transformed this problem of cryptanalysis to some decoding problem. According to this model of cryptanalysis, the running key sequence K is the perturbed version of the output sequence σ of an LFSR (here the equivalent LFSR), after passing through a Binary Symmetric Channel (BSC) with error probability $p = P[\sigma_i \neq K_i] < \frac{1}{2}$ (see Figure 2). Now we relate this error probability p of the BSC with the parameters of the nonlinear combiner model. In that case, $p = P[\sigma_i \neq K_i] = \frac{1}{2} - \frac{\max_{\omega \in \{0,1\}^n}(|W_f(\overline{\omega})|)}{2^{n+1}}$.

If we correlate the cipher text bits C with σ, then the error probability p increases further [1] and hence the complexity of the attack increases. In order to establish the robustness of our scheme we consider the correlation between σ and K. Depending on the nature of the fast correlation attacks they can be

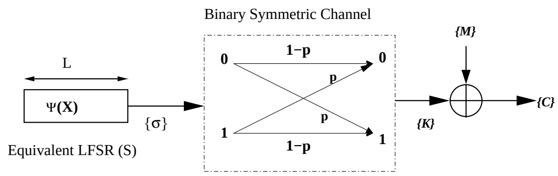

Fig. 2. Coding theory based model of Correlation Attack

divided as iterative algorithm and single pass algorithm (see [4] for details). We first consider the *single pass* algorithm described by Chepyzhov, Johansson and Smeets (CJS) [2]. The basic idea is to correctly guess a small number of output bits of the LFSR output sequence σ from the perturbed sequence C. To reduce the decoding complexity of the $[N, L]$ binary code, it is associated to another code $[n_2, k]$ of lower dimension ($k < L$). This $[n_2, k]$ code is obtained using sets of t ($t \geq 2$) parity check equations of the $[N, L]$ code [2, Algorithm A2]. This helps to recover k initial bits of the target LFSR (of length L). According to [2], the BSC has probability $p = \frac{1}{2} - \epsilon < 0.5$. The amount of cipher text bits (N) required for cryptanalysis is given by [2]

$$N = \frac{1}{4}(2k\,t!\,\ln 2)^{1/t}\epsilon^{-2}2^{\frac{L-k}{t}} \qquad (2)$$

where k is the number of initial bits recovered correctly from the attack. For given values of L and k, cipher text bit requirement N increases as ϵ decreases. Comparing the expression of p in BSC and the Boolean function model, we get $\epsilon = \frac{\max_{\omega \in \{0,1\}^n}(|W_f(\overline{\omega})|)}{2^{n+1}}$.

It is important to note here that the Boolean function needs to have highest possible nonlinearity in order to possess maximum resistance against the best affine approximation (BAA) attack [7]. Moreover, maximum nonlinearity provides the minimum value of ϵ, which in turn increases the complexity of the attack. We generally select a resilient function with three valued Walsh spectra. The (n, m, u, x) functions with best possible nonlinearity must have three valued Walsh spectra for $m > \frac{n}{2} - 2$ [20]. In this case, $\max_{\omega \in \{0,1\}^n} |W_f(\overline{\omega})| = 2^{m+2}$ and hence, $\epsilon = \frac{2^{m+2}}{2^{n+1}} = 2^{m-n+1}$. We start our design from the formulae available from [2].

4.1 Design Criteria for Equivalent LFSR

We denote the key length by q. From designer's viewpoint the time complexity of any attack should be $\geq 2^q$. The precomputation memory and time complexity of [2, Algorithm A2] are $N^{\lfloor \frac{t-1}{2} \rfloor}$ and $N^{\lceil \frac{t-1}{2} \rceil}$ [8, Section 4], where N, L, k, t are as given in Equation 2. The decoding complexity is $2^k\, k\, \frac{2 \ln 2}{(2\epsilon)^{2t}}$.

Fact 41 *The criteria for safe design against CJS attack gives $L = 4\,q$.*

Reason 41 *We refer the Equation 2. From the designer's point of view we can safely approximate (underestimate)* N *as* $2^{\frac{L-k}{t}}$*. Thus,* $N^{\lceil \frac{t-1}{2} \rceil}$ *can be approximated as* $2^{\frac{L-k}{t} \cdot \lceil \frac{t-1}{2} \rceil}$*. If* t *is even,* $\lceil \frac{t-1}{2} \rceil = \frac{t}{2}$*, and* $2^{\frac{L-k}{t} \cdot \lceil \frac{t-1}{2} \rceil} = 2^{\frac{L-k}{2}}$*. If* t *is odd, then* $2^{\frac{L-k}{t} \cdot \lceil \frac{t-1}{2} \rceil} = 2^{\frac{L-k}{t} \cdot \frac{t-1}{2}} = 2^{\frac{L-k}{t} \cdot \frac{t}{2}} 2^{\frac{L-k}{t} \cdot \frac{-1}{2}} = 2^{\frac{L-k}{2}} 2^{-\frac{L-k}{2t}}$*. The term* $2^{-\frac{L-k}{2t}}$ *will be minimized when* t *is the smallest possible odd integer greater than 2, i.e.,* $t = 3$*. Then the minimum value of the term is* $2^{-\frac{L-k}{6}}$*. So, we can underestimate* $N^{\lceil \frac{t-1}{2} \rceil}$ *as* $2^{\frac{L-k}{2}} 2^{-\frac{L-k}{6}} = 2^{\frac{L-k}{3}}$*. Hence for safe design,* $\frac{L-k}{3} \geq q$*, which gives* $L = 3q + k$*. Here, we need to keep in mind that the decoding complexity is* $2^k \cdot k \frac{2 \ln 2}{(2\epsilon)^{2t}}$*. Considering* k *as large as* q*, we get* $2^{\frac{L-k}{3}} = 2^q$ *and also the decoding complexity is greater than* 2^q*. Thus we take* $L = 3q + k = 4q$ *for a safe design.*

However, for a more tight design, we can very well decrease L by some small amount considering the other terms, mainly the terms related to ϵ.

Example 1. Let us consider $q = 256, L = 1000, \epsilon = 0.125$. Note that $4q = 1024$, but considering ϵ, we reduce L a little bit and take $L = 1000$. For this value of L, one can calculate the cipher text bit requirement, preprocessing and decoding complexity for carrying out the CJS attack for different values of t and k. Note that, in Table 3, for any given values of k and t, the corresponding maximum value of time complexity is always at least 2^{256}. Thus for $L = 1000$, the attack of [2] can succeed in no way with complexity less than the exhaustive key search, i.e., 2^{256}.

Table 3. Time complexity for CJS Attack [2] ($L = 1000, \epsilon = 0.125$)

k	t	Bit Requirement (N)	Preprocessing Complexity	Decoding complexity
		$\frac{1}{4}(2k\,t!\,\ln 2)^{1/t} \epsilon^{-2} 2^{\frac{L-k}{t}}$	$N^{\lceil \frac{t-1}{2} \rceil}$	$2^k \cdot k \frac{2 \ln 2}{(2\epsilon)^{2t}}$
32	2	2^{491}	2^{491}	2^{40}
32	3	2^{329}	2^{329}	2^{44}
64	2	2^{476}	2^{476}	2^{72}
64	3	2^{319}	2^{319}	2^{76}
96	2	2^{460}	2^{460}	2^{104}
96	3	2^{309}	2^{309}	2^{108}
128	2	2^{444}	2^{444}	2^{136}
128	3	2^{298}	2^{298}	2^{140}
160	2	2^{428}	2^{428}	2^{168}
160	3	2^{287}	2^{287}	2^{172}
192	2	2^{413}	2^{413}	2^{200}
192	3	2^{277}	2^{277}	2^{204}
224	2	2^{397}	2^{397}	2^{232}
224	3	2^{266}	2^{266}	2^{236}
256	2	2^{381}	2^{381}	2^{264}
256	2	2^{256}	2^{256}	2^{268}

Now we concentrate on the *iterative decoding* algorithm proposed by Canteaut and Trabbia (CT) [1]. The correct value of the LFSR output sequence σ are obtained by iteratively modifying the values of the sequence C using parity check equations of weight 4 and 5. According to their estimation, the cipher text bit requirement is given by $N = 2^{\alpha_\tau(p) + \frac{L}{\tau - 1}}$, where $\alpha_\tau(p) = \frac{1}{\tau-1} log_2[(\tau-1)! \frac{k_\tau}{C_{\tau-2}(p)}]$, $C_{\tau-2}$ is the capacity of the binary symmetric channel with overall error probability $p_{\tau-2}$, i.e., $C_{\tau-2} = p_{\tau-2} \log_2(p_{\tau-2}) + (1 - p_{\tau-2}) \log_2(1 - p_{\tau-2})$ and L is the length of the target LFSR. Here $k_\tau \approx 3$ for $\tau = 3$ and $k_\tau = 1$ for $\tau \geq 4$. The overall error probability of the BSC is given by $p_{\tau-2} = \frac{1}{2}(1-(1-2p)^{\tau-2}$, p is the error probability of the BSC. Number of parity check equations of weight τ is $m(\tau) = \frac{N^{\tau-1}}{(\tau-1)!2^L}$. Complexity of the preprocessing stage is $\frac{N^{\tau-2}}{(\tau-2)!}$. The decoding time complexity is given by $5(\tau-1) \cdot N \cdot m(\tau)$. The total memory requirement is $2N + (\tau - 1) \cdot m(d)$ computer words.

Fact 42 *The condition $L = 4q$ resists CT attack.*

Reason 42 *The time complexity of the preprocessing and decoding stages of CT attack is given by $\frac{N^{\tau-2}}{(\tau-2)!}$ and $5(\tau-1) \cdot N \cdot m(\tau)$ respectively. The memory requirement for the attack is $2N+(\tau-1)\,m(\tau)$. Here τ, (weight of the parity check equation) is 3, 4 or 5. So, clearly the complexity of the attack is proportional to the cipher text bit requirement N. Now N is given as $2^{\alpha_\tau(p)+\frac{L}{\tau-1}}$. We can underestimate N as $N \approx 2^{\frac{L}{\tau-1}}$, $(\alpha_\tau(p) > 0)$. Putting $\tau = 5$, the minimum cipher text bit requirement is $N \approx 2^{\frac{L}{4}}$. Putting $L = 4q$, we get $N \approx 2^q$. Hence, for $L = 4q$, the complexity of of the CT attack is $\approx 2^q$.*

It is of interest to see how the recently published algebraic attacks [6] works on the nonlinear combiner model designed with the criteria described here.

4.2 Fixing Boolean function and LFSR parameters

We have already noted that $\epsilon = \frac{2^{m+2}}{2^{n+1}} = 2^{m-n+1}$. For a given value of ϵ,

1. we first need to decide about n, m and
2. once n, m are fixed, the length of each LFSR is calculated.

Now the important question is deciding n and m. First of all, we are interested about resilient functions with maximum possible nonlinearity and 3-valued Walsh spectra which needs the condition $m > \frac{n}{2} - 2$ [20]. It is known that the algebraic degree of such function is $u = n - m - 1$ [23]. Further from the viewpoint of software implementation, it is better to have lesser values of n, m, i.e., less number of LFSRs which makes the software more efficient (see Subsection 3.1 for details). Thus the requirement is to get n, m satisfying the conditions (i) $m > \frac{n}{2} - 2$, and (ii) $2^{m-n+1} \leq \epsilon$.

High algebraic degree of the Boolean function ensures high linear complexity for the output key stream K obtained from the Boolean function f [7]. For achieving high linear complexity one needs n-variable m-resilient Boolean functions with the maximum possible algebraic degree $n-m-1$ [21]. It is now known that the resilient function with maximum nonlinearity must have the maximum algebraic degree too [20].

From Definition 6, length of an equivalent LFSR $L = \sum_{j=1}^{m+1} d_{i_j}$. As the attacker will target the smallest length equivalent LFSRs, we need the average length of an LFSR (or degree of its connection polynomial) $d_{av} = \frac{L}{m+1}$.

Next we fix the degree of the LFSRs $d_1, d_2, \ldots, d_n$ and the criteria for connection polynomials. The degrees need to be pairwise coprime [12, Page 224] in order to maximize linear complexity of the running key sequence K. Also we need that $d_{i_1} + d_{i_2} + \ldots + d_{i_{m+1}} \geq L$ for any subset of $m+1$ LFSRs. Now we consider the following fact.

Fact 43 *The connection polynomial $\psi(x)$ of equivalent LFSR must have weight > 10, but weight of the connection polynomial $c_i(x), 1 \leq i \leq n$, of the individual LFSRs S_i should be low (< 10).*

Reason 43 *To resist the attacks of Meier and Stafflebach [15], the connection polynomial $\psi(x)$ of equivalent LFSR must have a high (> 10) weight (already explained in Subsection 4). The number of logical operations required for generating the output from an LFSR is dependent on the number of XOR operations (i.e., the number of taps) in its recurrence relation. So, for the sake of fast implementation in software, we need connection polynomial of individual LFSRs of low weight (see Subsection 3.1 for details). These two apparently contradictory requirements are achievable. For example, consider three trinomials of degree d_1, d_2, d_3, say. If properly chosen, the product polynomial may have weight as high as 27. Thus we have to ensure that weight of $\prod_{j=1}^{m'} c_{i_j}(x)$ should be high, for any subset (of LFSRs) having size m' ($m + 1 \leq m' \leq n$).*

Now consider about τ-nomial multiples of $\psi(x) = \prod_{j=1}^{m+1} c_{i_j}(x)$ of degree L. It has been explained in $[1, 14]$ that the expected degree of the least degree τ-nomial multiple of $\psi(x)$ is of the order of $2^{\frac{L}{\tau-1}}$. The most important attack presented using τ-nomial multiples is the CT attack [1]. In that case, the value of τ has been taken as $3, 4, 5$. Note that, even if $\tau = 5$, the minimum degree τ-nomial multiple is of the order $2^{\frac{L}{4}} = 2^q$ and the CT attack seems to be infeasible under this circumstances.

5 A concrete scheme

On the basis of the design criteria discussed so far, we analyse a specific scheme of the nonlinear combiner model with key length $q = 256$. We consider a block size of $b = 64$ bits. As discussed in Section 4, we start with $L = 4q = 1024$. However, considering $\epsilon = 0.125$ (i.e., $p = 0.375$), the actual value of L can be decreased to ≈ 1000. We take a $(6, 2, 3, 24)$ Boolean function. Here $n = 6$, $m = 2$ satisfies thei two conditions (i) $m > \frac{n}{2} - 2$, and (ii) $2^{m-n+1} \leq \epsilon$. We follow the Construction 31 with $f = (1 \oplus X_3)X_1 \oplus X_3X_2$, i.e., $f_1 = X_1, f_2 = X_2$. In this case no operation is required to calculate f_1, f_2 and hence $l_1 = l_2 = 0$. So, 16 logical operations are needed to get an output of 1 block from the Boolean function.

The average degree of the connection polynomials are $d_{av} = \frac{1000}{3} \approx 333$. We choose the LFSRs of length 332, 333, 337, 343, 353, 377 and take the primitive trinomials $x^{332} \oplus x^{123} \oplus 1$, $x^{333} \oplus x^{331} \oplus 1$, $x^{337} \oplus x^{135} \oplus 1$, $x^{343} \oplus x^{205} \oplus 1$, $x^{353} \oplus x^{69} \oplus 1$, $x^{377} \oplus x^{75} \oplus 1$. Note that the degrees are mutually coprime. Considering the three ($m + 1 = 2 + 1 = 3$) least lengths out of these 6 LFSRs, $L = 1002$, which is greater than our initial design assumption (i.e., 1000). Take the initial state vector considering all the LFSRs, i.e., in total 2075 bits. Since we choose the LFSRs of degree mutually coprime, this state will return after $(2^{332}-1)(2^{333}-1)(2^{337}-1)(2^{343}-1)(2^{353}-1)(2^{377}-1) \approx 2^{2075}$ states.

Table 4. Minimum weight of the connection polynomial of composite LFSRs.

No. of LFSRs taken (m')	Minimum weight	Corresponding degree
3	25	1062
4	75	1395
5	209	1698
6	495	2075

Now we look at the possible minimum weights of the connection polynomial ($\psi(x)$) corresponding to the equivalent LFSRs (see Table 4). This is obtained for different subsets (size m') of the 6 LFSRs. Here $3 \leq m' \leq 6$. The least weight of the products of m' connection polynomials is 25 (> 10). Clearly the criteria given in Fact 43 is satisfied and no fast correlation attack is possible in the line of [15]. From [14], the expected degree of the least degree τ-nomial multiple for $L = 1002$ is $2^{\frac{1002}{\tau-1}}$ which is approximately $2^{500}, 2^{334}, 2^{250}$ for $\tau = 3, 4, 5$. Finding exact multiples of such high degrees seems to be very hard. Moreover, we have checked (see Table 5) that the actual complexities of the CT [1] attack for our scheme and it is always $\geq 2^{256}$. We have already discussed the robustness of this scheme against the CJS attack (see Example 1 and Table 3).

Table 5. Time/ Memory complexity for Canteaut and Trabbia Attack [1]

τ	Cipher text bits $N = 2^{\alpha_\tau(p)+\frac{L}{\tau-1}}$	Memory Req. $2N + (\tau-1)\cdot m(d)$	Preprocessing Comp. $\frac{N^{\tau-2}}{(\tau-2)!}$	Decoding Comp. $5(\tau-1)\cdot N \cdot m(\tau)$
3	2^{503}	2^{504}	2^{503}	2^{512}
4	2^{337}	2^{338}	2^{673}	2^{349}
5	2^{254}	2^{255}	2^{760}	2^{271}

As mentioned earlier in Section 1, low linear complexity still remailns a problem with this model. For this specific example, using the ANF of the Boolean function, the linear complexity of the generated key stream is found to be $\approx 2^{28}$. So 2^{29} known plain text bits may be exploited to attack the scheme using Berlekamp-Massey Shift Register synthesis algorithm [13]. The concrete scheme with specific parameters are presented here to show the way of using the criteria fixed in Section 4 to evolve a nonlinear combination generator scheme safe against the existing correlation attack. We have taken trinomials (least possible weight) as connection polynomials of individual LFSRs to maximize the speed of LFSRs in software and still get the desired level of security. Depending on the security requirement (i.e., key length q), similar schemes can be designed in the same way using required number of LFSRs, their connection polynomials and the Boolean function.

5.1 Performance in Software

We study the encryption speed of our specific scheme in software, using a simple "C" language implementation. Here $b = 64$. The theoretical calculation gives that 8 logical operations are needed to get one byte output. However, in actual implementation we get much worse result as available in Table 6 and it needs further effort to optimize it in software. The storage requirement for this implementation is less than $1KB$ for storing the LFSRs. For comparison purpose we also present the encryption speed of other stream cipher schemes in Table 7. It is clear that our scheme is quite fast compared to other schemes except SNOW in Table 7.

Table 6. Key stream Generation Speed on various platforms.

Processor/RAM	Operating System	Compiler	Speed	
			Mbps	cycles/byte
PIV 1.8 GHZ/512 MB	Linux (RH 9.0)	gcc -O3	250	54
PIV 2.4 GHZ/512 MB	Linux (RH 8.0)	gcc -O3	432	42

Table 7. Performance of Different Software stream Ciphers.

Scheme	Processor/ OS	Compiler	*Speed (Mbps)
LILI-II [5]	PIII 300MHz/Win NT	Borland C++ 5.0	6
	SPARCv9/Linux	gcc	4
SSC2 [24]	Sun Ultra 1.143 MHz/Sun Solaris	gcc -O3	143
	PII 233 MHz/Linux	gcc -O3	118
	Sun SPARK2 40 MHz/Sun	gcc-O3	22
SNOW-1.0 [8]	PIII 500 MHz/Linux	gcc -O3	610
	PIII 500 MHz/Win NT	VC++	520
	PII 400 MHz/Linux	gcc -O3	380
SNOW-2.0 [9]	PIV 1.8 GHz/Linux	gcc -O3	3,610
SOBER-t32	Sun Ultra Sparc	-	76
t16 [18]	200 MHz/ Sun Solaris		44

Acknowledgment: This work has been supported partially by the ReX program, a joint activity of USENIX Association and Stichting NLnet. The authors also like to acknowledge Prof. Bimal Roy of Indian Statistical Institute for providing initial idea behind the design.

References

1. A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Advances in Cryptology - EUROCRYPT 2000*, LNCS Volume 1807, pages 573–588. Springer Verlag, 2000.
2. V. V. Chepyzhov, T. Johansson and B. Smeets. A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers. In *Proceedings of FSE 2000*, pages 181–195, LNCS volume 1978. Springer Verlag, 2001.
3. S. Chowdhury and S. Maitra. Efficient software implementation of Linear Feedback Shift Registers. INDOCRYPT 2001, number 2247 in Lecture Notes in Computer Science. Pages 297–307. Springer Verlag, December 2001.
4. P. Chose, A. Joux and M. Mitton. Fast Correlation Attacks: an Algorithmic Point of View. In *EUROCRYPT 2002*, pages 209–221, LNCS volume 2332. Springer Verlag, 2002.
5. A. Clark, E. Dawson, J. Fuller, J. D. Golic, H. -J. Lee, W. Millan, S. -J. Moon, L. Simpson. The LILI-II Keystream Generator. In *Information Security and Privacy - ACISP 2002*, pages 25–39, LNCS volume 2384. Springer Verlag, 2002.

6. N. T. Courtois and W. Meier. Algebraic attack on Stream Ciphers with linear feedback. In *Advances in Cryptology - EUROCRYPT 2003*, LNCS Volume 2656, pages 345–359. Springer Verlag, 2003.
7. C. Ding, G. Xiao, and W. Shan. *The Stability Theory of Stream Ciphers.* LNCS volume 561. Springer-Verlag, 1991.
8. P. Ekdahl and T. Johansson. SNOW - a New Stream Cipher. In *Proceedings of the first open NESSIE Workshop*, Heverlee, Belgium, November 13-14, 2000.
9. P. Ekdahl and T. Johansson. A new version of the stream cipher SNOW. In *Selected Areas in Cryptography, SAC 2002, August 2002*, Pages 47–61, number 2595 in Lecture Notes in Computer Science, Springer Verlag, 2003.
10. X. Guo-Zhen and J. Massey. A spectral characterization of correlation immune combining functions. *IEEE Transactions on Information Theory*, 34(3):569–571, May 1988.
11. T. Johansson and F. Jonsson. Fast correlation attacks through reconstruction of linear polynomials. In *Advances in Cryptology - CRYPTO 2000*, LNCS volume 1880, pages 300–315. Springer Verlag, 2000.
12. R. Lidl and H. Niederreiter. Introduction to finite fields and their applications. Cambridge University Press, 1994.
13. J. L. Massey. Shift-register synthesis and BCH decoding. In *IEEE Transactions on Information Theory*, Vol IT-15, July 1968.
14. S. Maitra, K. C. Gupta and A. Venkateswarlu. Multiples of Primitive Polynomials and Their Products over GF(2). In *Selected Areas in Cryptography, SAC 2002, August 2002*, Pages 214–231, number 2595 in Lecture Notes in Computer Science, Springer Verlag, 2003.
15. W. Meier and O. Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1:159–176, 1989.
16. M. J. Mihaljevic, M. P. C. Fossorier and H. Imai. Fast correlation attack algorithm with list decoding and an application. In *Proceedings of FSE 2001*, pages 196–210, LNCS 2355, Springer-Verlag, 2002.
17. E. Pasalic, S. Maitra, T. Johansson and P. Sarkar. New constructions of resilient and correlation immune Boolean functions achieving upper bounds on nonlinearity. In *Workshop on Coding and Cryptography*, Paris, January 2001. Electronic Notes in Discrete Mathematics, Volume 6, Elsevier Science, 2001.
18. G. Rose and P. Hawkes. The t-Class of SOBER Stream Ciphers. In *Proceedings of the first open NESSIE Workshop*, Heverlee, Belgium, 2000.
19. G. Rose and P. Hawkes. Turing: a fast stream cipher. In *preproceedings of the FSE 2003*, Lund, Sweden, 2003.
20. P. Sarkar and S. Maitra. Nonlinearity bounds and constructions of resilient Boolean functions. In *Advances in Cryptology - CRYPTO 2000*, LNCS volume 1880, pages 515–532. Springer Verlag, 2000.
21. T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, IT-30(5):776–780, September 1984.
22. T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Computers*, C-34(1):81–85, January 1985.
23. Y. V. Tarannikov. On resilient Boolean functions with maximum possible nonlinearity. In *Proceedings of INDOCRYPT 2000*, LNCS volume 1977, 19-30, 2000.
24. M. Zhang, C. Carrol and A. Chan. The software oriented stream cipher SSC2. In *Proceedings of FSE 2000*, pages 31–48, LNCS volume 1978, 2001.