Algebraic Attacks on Summation Generators

Dong Hoon Lee, Jaeheon Kim, Jin Hong,
Jae Woo Han, and Dukjae Moon

National Security Research Institute
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Korea
{dlee,jaeheon,jinhong,jwhan,djmoon}@etri.re.kr

Abstract. We apply the algebraic attacks on stream ciphers with memories to the summation generator. For a summation generator that uses n LFSRs, an algebraic equation relating the key stream bits and LFSR output bits can be made to be of degree less than or equal to $2^{\lceil \log_2 n \rceil}$, using $\lceil \log_2 n \rceil + 1$ consecutive key stream bits. This is much lower than the upper bound given by previous general results. We also show that the techniques of [6, 2] can be applied to summation generators using 2^k LFSRs to reduce the effective degree of the algebraic equation.

Keywords: stream ciphers, algebraic attacks, summation generators

1 Introduction

Among recent developments on stream ciphers, the algebraic attack has gathered much attention. In this attack, an algebraic equation relating the initial key bits and the output key stream bits is set up and solved through linearization techniques.

Algebraic attack was first applied to block ciphers and public key cryptosystems [8, 9, 3]. And its first successful application to stream cipher was done on Toyocrypt [4]. As the method was soon extended to LILI-128 [7], it gathered much attention. Stream ciphers that utilize memory were first thought to be much more resistant to these attacks, but soon it was shown that even these cases were subject to algebraic attacks [1, 5].

The summation generator proposed by Ruepel [12] is a nonlinear combiner with memory. It is known that the generator produces sequences whose period and correlation immunity are maximum, and whose linear complexity is conjectured to be close to the period. Hence it serves as a good building block for stream ciphers.

However, a correlation attack on the summation generator that uses two linear feedback shift registers (LFSR) was presented in [11], even though it is also stated in [11] that this attack is not plausible if there are more than two LFSRs in use. Another well known attack on summation generator is given by [10] and points in the opposite direction. It uses feedback carry shift registers (FCSR) to simulate the summation generator and indicates that for a fixed initial key size, breaking them into too many LFSRs will add to its weakness.

In this work, we study the summation generator with the algebraic attack in mind. We show that for a summation generator that uses n LFSRs, an algebraic equation relating the key stream bits and LFSR output bits can be made to be of degree less than or equal to $2^{\lceil \log_2 n \rceil}$, using $\lceil \log_2 n \rceil + 1$ consecutive key stream bits. This is much lower than the upper bound on the degree of algebraic equations that is guaranteed by the general works [1, 5]. We also show that the techniques of [6, 2] can be applied to summation generators using 2^k LFSRs to reduce the effective degree of the equation further.

The reader may confer to Tables 3 and 4, appearing in later sections, for a quick look on the improvements we have given to understanding the actual strength of algebraic attacks on summation generators. Actually, Table 3 should be a good reference in view of algebraic attacks for anyone considering the use of a summation generator.

The rest of this paper is organized as follows: We recall the definition of a summation generator and introduce elementary symmetric boolean functions in Section 2. We derive an algebraic equation that is satisfied by the summation generator on 4 LFSRs in Section 4. Induction is used in extending this to the summation generator on 2^k LFSRs in the following section. Section 5 deals with the general n case. We show that the degree of the equation can be reduced further in the case $n = 2^k$ using the technique of [6] in Section 6, and present our conclusion in Section 7.

2 Preliminaries

2.1 Summation Generators

The summation generator proposed by Rueppel [12] is a nonlinear combiner with memory. We consider a summation generator that uses n binary LFSRs (Figure 1). The output of the j-th LFSR at time t is denoted by $x_j^t \in \{0, 1\}$. Since we are dealing with binary values, we will not be using powers of these terms, hence the superscript t should not cause any confusion.

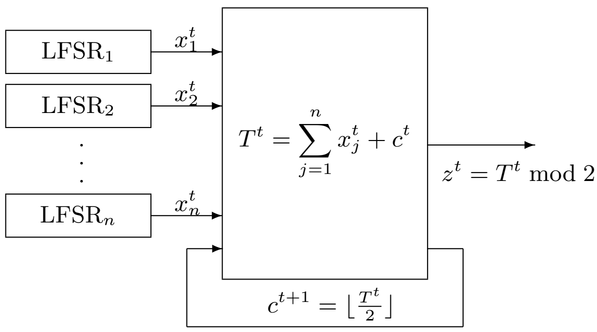

Fig. 1. Structure of summation generators

The current *carry* value from the previous stage is denoted by c^t. Note that the carry can be expressed in $k = \lceil \log_2 n \rceil$ bits. We shall let

$$c^t = (c^t_{k-1}, \ldots, c^t_1, c^t_0)$$

be the binary expression of the carry value.

The binary output z^t and the carry value for the next stage from the summation generator is given by

$$z^t = x^t_1 \oplus x^t_2 \oplus \cdots \oplus x^t_n \oplus c^t_0, \tag{1}$$
$$c^{t+1} = \lfloor (x^t_1 + x^t_2 + \cdots + x^t_n + c^t)/2 \rfloor. \tag{2}$$

2.2 Symmetric polynomials

Let us denote by S^t_i, the i-th elementary symmetric polynomial in the variables $\{x^t_1, \ldots, x^t_n\}$. We view them as boolean functions rather than as polynomials. Explicitly, they are

$$\begin{aligned}
S^t_0 &= 1, \\
S^t_1 &= \oplus^n_{j=1} x^t_j, \\
S^t_2 &= \oplus_{1 \le j_1 < j_2 \le n} x^t_{j_1} x^t_{j_2}, \\
&\vdots \\
S^t_n &= x^t_1 x^t_2 \cdots x^t_n.
\end{aligned}$$

We shall also call these by the name *elementary symmetric boolean functions.*

For any fixed $0 \le b \le n$, consider the condition $\sum_j x^t_j = b$. This condition states that b of the n variable x^t_j are equal to 1 and that the others are equal to 0. Since S^t_a is symmetric for any $0 \le a \le n$, it makes sense to evaluate S^t_a under this condition. Let $m_{a,b}$ denote this value. It is also clear that the values $(m_{a,b})^n_{b=0}$ completely determines the value of S^t_a at an arbitrary input.

To calculate $m_{a,b}$, one has only to count how many of the monomials contained in S^t_a is nonzero. Hence we have

$$m_{a,b} \equiv \binom{b}{a} \pmod 2. \tag{3}$$

Now, consider the $(n+1) \times (n+1)$ matrix $M = (m_{a,b})$. The $n = 4$ case is given in Table 1, as an example. Denote by M', the $n \times n$ matrix obtained by removing the last row and column from M. When we want to make explicit the number of variable used in defining M and M', we shall write $M(n)$ and $M'(n)$.

Lemma 1. *For powers of* 2*, the matrix* M' *satisfies*

$$M'(2^{k+1}) = \begin{pmatrix} M'(2^k) & M'(2^k) \\ 0 & M'(2^k) \end{pmatrix}.$$

$\sum x_j^t$	0	1	2	3	4
S_0^t	1	1	1	1	1
S_1^t	0	1	0	1	0
S_2^t	0	0	1	1	0
S_3^t	0	0	0	1	0
S_4^t	0	0	0	0	1

Table 1. The matrix M for $n = 4$

Proof. Let us first divide M' into four parts and write

$$M' = \begin{pmatrix} \mathrm{II} & \mathrm{I} \\ \mathrm{III} & \mathrm{IV} \end{pmatrix}.$$

It is clear from (3) that the second quadrant, part II, is $M'(2^k)$, as claimed. The equation also shows that the lower triangular part of M' is zero. Hence the third quadrant is filled with zero, as claimed.

We now show that I, II, and IV are identical. Once more, referring to (3), it suffices to show

$$\binom{b}{a} \equiv \binom{2^k + b}{a} \equiv \binom{2^k + b}{2^k + a} \pmod 2 \tag{4}$$

for all $0 \leq a, b < 2^k$. To this end, we may easily check that

$$\begin{aligned} (1+x)^{2^k+b} &= (1+x)^{2^k}(1+x)^b \\ &\equiv (1+x^{2^k})(1+x)^b \pmod 2 \\ &= (1+x)^b + x^{2^k}(1+x)^b. \end{aligned}$$

The coefficient of x^a that would appear in the expansion of the left hand side $(1+x)^{2^k+b}$ is the middle term of (4). Since $a < 2^k$, the x^a term in the right hand side may appear only in $(1+x)^b$ and is equal to the first term of (4). This shows the first equality. Similarly, comparison of the coefficients of x^{2^k+a} shows the equality between the first and the last terms of (4). This completes the proof.

This lemma allows one to write the matrix $M'(2^k)$ explicitly for any given k. Then, owing to (3), the matrix $M(n)$ for any n may be obtained as a submatrix of a big enough $M'(2^k)$. For example, Table 1 is a submatrix of $M'(8)$.

The basic theory on symmetric polynomials tells us that the set consisting of products of elementary symmetric *polynomials* forms a basis for the space of symmetric polynomials. In case of symmetric *boolean functions*, the following lemma can easily be seen to be true.

Lemma 2. *Any boolean function that is symmetric in its variables, may be written as a linear combination of the elementary symmetric boolean functions.*

3 Summation generator on 4 LFSRs

We fix $n = 4$ throughout this section and derive an algebraic equation of degree 4 that is satisfied by the summation generator. The variables of the final equation will consist of the LFSR output bits and the key stream bits but will not contain any carry bits.

Since we are dealing with $4 = 2^2$ LFSRs, we have $k = 2$, and it suffices to use 2 bits in expressing the carry value. It is clear that the carry value should be symmetric with respect to the order of the LFSRs in use. Recalling Lemma 2, this implies that, when the current carry value c^t is fixed, the carry bits c_0^{t+1} and c_1^{t+1} may be expressed as linear combinations of the elementary symmetric boolean functions S_i^t. In the general case when the current carry bits are not fixed, they will be linear combinations of the S_i^t with boolean functions of the carry bits c_0^t and c_1^t used as coefficients.

For each value of c^t and LFSR inputs possible, we explicitly calculated c_0^{t+1} and c_1^{t+1}. We then used Table 1 to expressed them using the elementary symmetric boolean functions. The result is given in Table 2.

$c^t = (c_1^t, c_0^t)$	(0,0)	(0,1)	(1,0)	(1,1)
c_0^{t+1}	S_2^t	$S_1^t \oplus S_2^t$	$S_0^t \oplus S_2^t$	$S_0^t \oplus S_1^t \oplus S_2^t$
c_1^{t+1}	S_4^t	$S_3^t \oplus S_4^t$	$S_2^t \oplus S_4^t$	$S_1^t \oplus S_2^t \oplus S_3^t \oplus S_4^t$

Table 2. The next carry bits in relation to the current carry value

Lemma 3. *For a summation generator on* 4 *LFSRs, the following expresses the next stage carry bits as functions of the current LFSR output bits and current carry bits.*

$$c_0^{t+1} = S_2^t \oplus c_0^t S_1^t \oplus c_1^t \quad (5)$$

$$c_1^{t+1} = S_4^t \oplus c_0^t S_3^t \oplus c_1^t S_2^t \oplus c_0^t c_1^t S_1^t. \quad (6)$$

Proof. Using Table 2, we may write

$$\begin{aligned} c_0^{t+1} = &\{(1 \oplus c_0^t)(1 \oplus c_1^t) S_2^t\} \oplus \{c_0^t (1 \oplus c_1^t)(S_1^t \oplus S_2^t)\} \\ &\oplus \{(1 \oplus c_0^t) c_1^t (S_0^t \oplus S_2^t)\} \oplus \{c_0^t c_1^t (S_0^t \oplus S_1^t \oplus S_2^t)\} \end{aligned}$$

and

$$\begin{aligned} c_1^{t+1} = &\{(1 \oplus c_0^t)(1 \oplus c_1^t) S_4^t\} \oplus \{c_0^t (1 \oplus c_1^t)(S_3^t \oplus S_4^t)\} \\ &\oplus \{(1 \oplus c_0^t) c_1^t (S_2^t \oplus S_4^t)\} \oplus \{c_0^t c_1^t (S_1^t \oplus S_2^t \oplus S_3^t \oplus S_4^t)\}. \end{aligned}$$

Simplification of these equations gives the claimed statements.

Lemma 4. *For $n = 4$, the following expresses the carry bits of the summation generator as polynomials in the LFSR output bits and the key stream bits.*

$$c_0^t = S_1^t \oplus z^t. \tag{7}$$
$$c_1^t = S_2^t \oplus (1 \oplus z^t)S_1^t \oplus S_1^{t+1} \oplus z^{t+1}. \tag{8}$$

The first carry bit c_0^t is linear and the second carry bit c_1^t is of degree 2 in the LFSR output bits.

Proof. The first equation follows immediately from (1). It is a linear function on the variables x_j^t. Substituting this and its shift c_0^{t+1} into (5) gives

$$c_1^t = S_2^t \oplus (S_1^t \oplus z^t)S_1^t \oplus S_1^{t+1} \oplus z^{t+1}.$$

Now, since we are dealing with boolean functions, we have $(S_1^t)^2 = S_1^t$ and the second equation follows.

Finally, with this lemma, we may remove all occurrence of the carry bits from (6) to obtain the following proposition.

Proposition 1. *The following algebraic equation holds true for a summation generator on 4 LFSRs.*

$$0 = \begin{aligned} & S_4^t \oplus (1 \oplus z^t)S_3^t \oplus S_2^t S_1^{t+1} \\ & \oplus (1 \oplus z^{t+1})S_2^t \oplus S_2^{t+1} \oplus (1 \oplus z^t)S_1^t S_1^{t+1} \\ & \oplus (1 \oplus z^t)(1 \oplus z^{t+1})S_1^t \oplus (1 \oplus z^{t+1})S_1^{t+1} \oplus S_1^{t+2} \\ & \oplus z^{t+2}. \end{aligned}$$

It is of degree 4 *in the output bits of the LFSRs and uses* 3 *consecutive key stream bits.*

While simplifying the substitution of (7), (8), and shift of (8) into (6), we have used the equalities

$$S_1^t S_2^t = S_3^t \quad \text{and} \quad S_1^t S_3^t = S_3^t$$

of boolean functions.

4 Summation generator on $n = 2^k$ LFSRs

Let us denote by $F_i^{t+1}(n)$, the (symmetric) polynomial that expresses the next stage carry bit c_i^{t+1} in terms of LFSR outputs x_j^t and current carry bits c_j^t. Since we shall be dealing with polynomials on different number of variables, we shall write $S_j^t(n)$ to denote the elementary symmetric boolean function on n variables. As an example, we saw in the previous section that

$$F_0^{t+1}(2^2) = S_2^t(2^2) \oplus c_0^t S_1^t(2^2) \oplus c_1^t, \tag{9}$$
$$F_1^{t+1}(2^2) = S_4^t(2^2) \oplus c_0^t S_3^t(2^2) \oplus c_1^t S_2^t(2^2) \oplus c_0^t c_1^t S_1^t(2^2). \tag{10}$$

Let us suppose that for some $n = 2^k$, the polynomials $F_i^{t+1}(2^k)$ are given by

$$F_i^{t+1}(2^k) = \oplus_j f_{i,j} S_j^t(2^k). \tag{11}$$

Here, each coefficient $f_{i,j}$ is a boolean function defined on the current carry bits $c_0^t, c_1^t, \ldots, c_{k-1}^t$. For example, we see from (10) that

$$f_{1,4} = 1, \quad f_{1,3} = c_0^t, \quad f_{1,2} = c_1^t, \quad f_{1,1} = c_0^t c_1^t, \quad f_{1,0} = 0,$$

for $k = 2$. The following proposition will allow us to inductively calculate all $F_i^{t+1}(2^k)$ for any i and k.

Proposition 2. *Suppose equation* (11) *holds for some k. Then we have*

$$F_i^{t+1}(2^{k+1}) = \oplus_j f_{i,j} S_j^t(2^{k+1}), \quad \textit{for } i < k-1, \tag{12}$$

$$F_{k-1}^{t+1}(2^{k+1}) = \big(\oplus_j \, f_{k-1,j} S_j^t(2^{k+1}) \big) \oplus c_k^t, \tag{13}$$

$$F_k^{t+1}(2^{k+1}) = \big(\oplus_j \, f_{k-1,j} S_{j+2^k}^t(2^{k+1}) \big) \oplus c_k^t \big(\oplus_j \, f_{k-1,j} S_j^t(2^{k+1}) \big). \tag{14}$$

Proof. See the appendices.

As an immediate application of this proposition to (9) and (10), we may write

$$F_0^{t+1}(2^3) = S_2^t(2^3) \oplus c_0^t S_1^t(2^3) \oplus c_1^t, \tag{15}$$

$$F_1^{t+1}(2^3) = S_4^t(2^3) \oplus c_0^t S_3^t(2^3) \oplus c_1^t S_2^t(2^3) \oplus c_0^t c_1^t S_1^t(2^3) \oplus c_2^t, \tag{16}$$

$$\begin{aligned} F_2^{t+1}(2^3) = {} & S_8^t(2^3) \oplus c_0^t S_7^t(2^3) \oplus c_1^t S_6^t(2^3) \oplus c_0^t c_1^t S_5^t(2^3) \\ & \oplus c_2^t S_4^t(2^3) \oplus c_0^t c_2^t S_3^t(2^3) \oplus c_1^t c_2^t S_2^t(2^3) \oplus c_0^t c_1^t c_2^t S_1^t(2^3). \end{aligned} \tag{17}$$

Now, let us briefly recall the process we went through in Section 3 in obtaining the degree 4 equation of Proposition 1. We started out with three equations.

$$z^t = S_1^t \oplus c_0^t. \tag{18}$$

$$c_0^{t+1} = S_2^t \oplus c_0^t S_1^t \oplus c_1^t. \tag{19}$$

$$c_1^{t+1} = S_4^t \oplus c_0^t S_3^t \oplus c_1^t (S_2^t \oplus c_0^t S_1^t). \tag{20}$$

We used (18) to write c_0^t as a degree 1 equation that involves just the key stream bit and the LFSR outputs. This was substituted in the next equation (19) to write c_1^t as a degree 2 equation of the same kind. Finally, these expressions for c_0^t and c_1^t were substituted in the last equation to obtain the degree 4 equation that involves only key stream bits and LFSR output bits.

What would happen if we wanted to do the same for $n = 2^3$. We would start with the following set of equations.

$$z^t = S_1^t \oplus c_0^t. \tag{21}$$

$$c_0^{t+1} = S_2^t \oplus c_0^t S_1^t \oplus c_1^t. \tag{22}$$

$$c_1^{t+1} = S_4^t \oplus c_0^t S_3^t \oplus c_1^t S_2^t \oplus c_0^t c_1^t S_1^t \oplus c_2^t. \tag{23}$$

$$\begin{aligned} c_2^{t+1} = {} & S_8^t \oplus c_0^t S_7^t \oplus c_1^t S_6^t \oplus c_0^t c_1^t S_5^t \\ & \oplus c_2^t (S_4^t \oplus c_0^t S_3^t \oplus c_1^t S_2^t \oplus c_0^t c_1^t S_1^t). \end{aligned} \tag{24}$$

Notice that the first two equations here are identical to (18) and (19), as stated by (12) of Proposition 2. Hence, as before, c_0^t and c_1^t will be written as degree 1 and 2 polynomials. The main part of (23) is identical to (20), as stated by (13). They only differ in that c_2^t appears at the end of (23). Since (20) is of degree 4, equation (23) gives a degree 4 expression for c_2^t. Now, as given by (14), the right hand side of (24) may be broken into two big terms of degree (less than or equal to) 8. The first term is a degree 4 equation shifted by degree 4 and the second term is a product of two degree 4 equations. Finally, the left hand side of (24) is of degree 4. Hence, substitution of c_0^t, c_1^t and c_2^t into (24) gives a degree 8 polynomial connecting various LFSR output bits and key stream bits.

One can easily see that the above argument is general enough to be seen as the induction step needed in proving the following theorem.

Theorem 1. *Consider a summation generator on $n = 2^k$ LFSRs. There exists an algebraic equation connecting LFSR output bits and $k+1$ consecutive key stream bits in such a way that it is of degree 2^k in the LFSR output bits.*

5 The general case

The following is an easy corollary to Theorem 1.

Theorem 2. *Consider a summation generator of n LFSRs. We shall let $k = \lceil \log_2 n \rceil$. There exists an algebraic equation connecting LFSR output bits and $k+1$ consecutive key stream bits in such a way that it is of degree less than or equal to 2^k in the LFSR output bits.*

Proof. We may model a summation generator on n LFSRs as a summation generator on 2^k LFSRs with $(2^k - n)$ of the LFSRs set to zero. Hence, our claim follows from Theorem 1.

For small n's, we explicitly calculated the algebraic equations. Table 3 compares the upper bounds on the degree of the algebraic equation claimed by various methods.

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
[1, 5]	2	5	6	10	12	14	16	23	25	28	30	33	35	38	40
Thm 2	2	4	4	8	8	8	8	16	16	16	16	16	16	16	16
explicit calc.	2	3	4	6	6	7	8	12	12	13	14	14	14	15	16

Table 3. Degree bounds on algebraic equations for summation generators

We believe this table is big enough to cover any practically usable summation generator and should serve as a good reference for anyone implementing a summation generator and considering its immunity to algebraic attacks.

6 Reducing the degree further for the $n = 2^k$ case

For the case when $n = 2^k$, we may reduce the degree of the algebraic equation a little bit further, assuming that we have access to consecutive key stream bits. We will apply the fast algebraic attack, which is introduced by Courtois [6] and is improved by Armknecht [2].

Following the notation of [3, 6], we classify multivariate equations that relate key bits k_i and output bits z_j into types given by their degrees of k_i and z_j. We say that a polynomial is of type $k^d z^f$ if all of its monomial terms are of the form $k_{i_1} \cdots k_{i_d} z_{j_1} \cdots z_{j_f}$. Capital letters will be used in a similar manner to denote types of equations that may contain lower degree monomials also. For example, $K^2 = k^2 \cup k \cup 1$ and $KZ = kz \cup k \cup z \cup 1$.

In [6], *double-decker equations* (DDE) of degree (d, e, f) are defined to be any multivariate equation of type $K^d \cup K^e Z^f$. Cases where $d > e$ are of interest from the attacker's point of view. The following proposition states that the equations describing the summation generator on 2^k input LFSRs, given by Theorem 1, are DDEs.

Theorem 3. *A double-decker equation of degree* $(2^k, 2^k - 1, 2^{k-1})$ *that relates initial key bits and output stream bits exists for the summation generator on* 2^k *LFSRs.*

Proof. Let us, once more, recall the $n = 2^2$ case. The following is a very simple illustration of the process we went through in obtaining the degree 4 equation.

$$
\begin{aligned}
(18) \quad &\Rightarrow \quad Z^1 = K^1 \oplus c_0^t \\
&\Rightarrow \quad c_0^t \text{ is of type } K^1 \cup Z^1 && (25) \\
(19),\ (25) \quad &\Rightarrow \quad K^1 \cup Z^1 = K^2 \cup (K^1 \cup Z^1)K^1 \oplus c_1^t \\
&\Rightarrow \quad c_1^t \text{ is of type } K^2 \cup K^1 Z^1 && (26) \\
(20),\ (26) \quad &\Rightarrow \\
K^2 \cup K^1 Z^1 &= (K^{2+2} \cup K^{1+2} Z^1) \cup (K^2 \cup K^1 Z^1)(K^2 \cup K^1 Z^1) \\
&\Rightarrow \quad \text{relation of type } K^4 \cup K^3 Z^2 && (27)
\end{aligned}
$$

Let us next consider the $n = 2^3$ case also. Since changing the number of variables, i.e., replacing $S_j^t(2^2)$ with $S_j^t(2^3)$ does not change the degree of these equations, the degree 2^3 equation is obtained as follows.

$$
\begin{aligned}
(18) = (21),\ (25) \quad &\Rightarrow \quad c_0^t \text{ is of type } K^1 \cup Z^1 && (28) \\
(19) = (22),\ (26) \quad &\Rightarrow \quad c_1^t \text{ is of type } K^2 \cup K^1 Z^1 && (29) \\
(20) \sim (23),\ (27) \quad &\Rightarrow \quad c_2^t \text{ is of type } K^4 \cup K^3 Z^2 && (30) \\
(24),\ (30) \quad &\Rightarrow \\
K^4 \cup K^3 Z^2 &= (K^{4+4} \cup K^{3+4} Z^2) \cup (K^4 \cup K^3 Z^2)(K^4 \cup K^3 Z^2) \\
&\Rightarrow \quad \text{relation of type } K^8 \cup K^7 Z^4 && (31)
\end{aligned}
$$

This shows that the general case may be proved by induction. To prove the induction step, it suffices to show that

$$K^{2^k} \cup K^{2^k-1} Z^{2^{k-1}} = (K^{2^k+2^k} \cup K^{2^k-1+2^k} Z^{2^{k-1}}) \\ \cup (K^{2^k} \cup K^{2^k-1} Z^{2^{k-1}})(K^{2^k} \cup K^{2^k-1} Z^{2^{k-1}})$$

gives a DDE of type $K^{2^{k+1}} \cup K^{2^{k+1}-1} Z^{2^k}$. Checking the validity of this statement is trivial. And this completes the proof.

We may assume that all periods of the LFSRs used in the summation generator are relatively prime. Then the summation generator satisfies the requirement of the attack described in [6], if we have access to consecutive key stream bits. The following steps may be taken to reduce the complexity of the algebraic attack.

1. Compute an algebraic equation explicitly using the formulae given in Proposition 2.
2. Write the resulting DDE in the following form.

$$L^t(k) = R^t(k, z).$$

   Here, $L^t(k)$ is the sum of all monomials of type K^d appearing in the equation and $R^t(k, z)$ is the sum of all other monomials of type $K^e Z^f$.
3. Fix an arbitrary nontrivial initial key k' and compute the value $L^t(k')$ for a sequence of length $2\binom{m}{d}$.
4. Using the Berlekamp-Massey algorithm, find a linear relation $\alpha = (\alpha_t)_t$ such that

$$\sum_t \alpha_t L^t(k') = 0.$$

   We note that steps 1,2,3,4 are independent of the initial key k, hence we can pre-compute the relation α.
5. We have obtained an algebraic equation of degree e. It is given by

$$\sum_t \alpha_t R^t(k, z) = 0.$$

6. Apply the general algebraic attack given in [7] to the above equation.

Example 1. The DDE for $n = 2^2$ is given as follows:

$$\begin{pmatrix} S_4^t \oplus S_3^t \oplus S_1^{t+1} S_2^t \\ \oplus S_2^{t+1} \oplus S_2^t \oplus S_1^t S_1^{t+1} \\ \oplus S_1^{t+2} \oplus S_1^{t+1} \oplus S_1^t \end{pmatrix} = \begin{pmatrix} z^t S_3^t \oplus z^{t+1} S_2^t \oplus z^t S_1^t S_1^{t+1} \\ \oplus z^{t+1} S_1^{t+1} \oplus z^t z^{t+1} S_1^t \\ \oplus z^{t+1} S_1^t \oplus z^t S_1^t \oplus z^{t+2} \end{pmatrix}.$$

We take 4 LFSRs defined by the following characteristic polynomials.

$$L_1 : x^3 + x + 1$$
$$L_2 : x^5 + x^2 + 1$$
$$L_3 : x^7 + x + 1$$
$$L_4 : x^{11} + x^2 + 1$$

Since $\binom{26}{4} \simeq 15,000$, we compute 30,000 bits from the left hand side of the above equation for some arbitrary nontrivial key bits k'. After applying the Berlekamp-Massey algorithm, we found that the sequence has a linear relation of length $3892 < \binom{26}{4}$. With a consecutive key stream of length of the order $6520 = 3892 + (3-1) + (\binom{26}{3} - 1)$, one will be able to find the 26 bit initial key k.

Table 4 gives a simple comparison of the data and computational complexities needed for attacks on summation generators. Let w be the Gaussian elimination exponent. We shall use $w = \log_2 7$, as given by the Strassen algorithm. Let the summation generator with 2^k input LFSRs use an m-bit initial key.

generator size		$(m, 2^k)$	$(128, 2^2)$	$(256, 2^2)$	$(256, 2^3)$
[1, 5]	data	$\binom{m}{2^{k-1}(k+1)}$	$2^{32.3}$	$2^{38.4}$	$2^{83.1}$
	computation	$\binom{m}{2^{k-1}(k+1)}^w$	$2^{90.8}$	$2^{107.9}$	$2^{233.2}$
[10]	data	$T = 2^{\frac{m}{2^k}+k+1}$	2^{35}	2^{67}	2^{36}
	computation	$T^2 \log_2 T \log_2 \log_2 T$	$2^{77.5}$	$2^{142.7}$	$2^{79.5}$
Thm 1	data	$\binom{m}{2^k}$	$2^{23.3}$	$2^{27.4}$	$2^{48.5}$
	computation	$\binom{m}{2^k}^w$	$2^{65.5}$	$2^{76.9}$	$2^{136.3}$
Thm 3	data	$\binom{m}{2^k-1}$	$2^{18.4}$	$2^{21.4}$	$2^{43.6}$
	computation	$\binom{m}{2^k-1}^w$	$2^{51.6}$	$2^{60.1}$	$2^{122.3}$

Table 4. Complexity comparison of attacks on summation generators

We remark that for [10] and Theorem 3, the key stream needs to be consecutive. For [1, 5] and Theorem 1, the key stream need only be partially consecutive, i.e., we need groups of $k+1$ consecutive bits, but these groups may be far apart from each other. Hence, a straightforward comparison of data complexity might not be fair. Also, the values for [10] have been calculated assuming that the LFSRs in use have been chosen well, so that their 2-adic span is maximal.

7 Conclusion

We have applied the general results of [1, 5] and [6] on stream ciphers with memories to the summation generator. Our results show that the degree of algebraic equation obtainable and the complexity of the attack applicable are much lower than given by the general results.

For a summation generator that uses n LFSRs, the algebraic equation relating the key stream bits and LFSR output bits can be made to be of degree less than or equal to $2^{\lceil \log_2 n \rceil}$, using $\lceil \log_2 n \rceil + 1$ consecutive key stream bits. Under certain conditions, for the $n = 2^k$ case, the effective degree may further be reduced by

1. And for small n's we have summarized the degrees of the explicit equations in Table 3. The table should be taken into account by anyone using a summation generator.

References

1. F. Armknecht and M. Krause, Algebraic attacks on combiners with memory, *Advances in Cryptology - Crypto 2003*, LNCS 2729, Springer-Verlag, pp. 162–175, 2003.
2. F. Armknecht, Improving Fast Algegraic Attacks, this proceeding, 2004.
3. N. Courtois, The security of Hidden Field Equations (HFE), *CT-RSA 2001*, LNCS 2020, Springer-Verlag, pp. 266–281, 2001.
4. N. Courtois, Higher order correlation attacks, XL algorithm and Cryptanalysis of Toyocrypt, *ICISC 2002*, LNCS 2587, Springer-Verlag, pp. 182–199, 2002.
5. N. Courtois, Algebraic attacks on combiners with memory and several outputs, E-print archive, 2003/125.
6. N. Courtois, Fast algebraic attack on stream ciphers with linear feedback, *Advances in Cryptology - Crypto 2003*, LNCS 2729, Springer-Verlag, pp. 176–194, 2003.
7. N. Courtois and W. Meier, Algebraic attacks on stream ciphers with linear feedback, *Advances in Cryptology - Eurocrypt 2003*, LNCS 2656, Springer-Verlag, pp. 345–359, 2003.
8. N. Courtois and J. Pieprzyk, Cryptanalysis of block ciphers with overdefined systems of equations, *Asiacrypt 2002*, LNCS 2501, Springer-Verlag, pp. 267–287, 2002.
9. A. Kipnis and A. Shamir, Cryptanalysis of the HFE public key cryptosystem by relinearization, *Advances in Cryptoloy - Crypto'99*, LNCS 1666, Springer-Verlag, pp. 19–30, 1999.
10. A. Klapper and M. Goresky, Cryptanalysis based on 2-adic rational approximation, *Advances in Cryptology - Crypto '95*, LNCS 963, Springer-Verlag, pp. 262–273, 1995.
11. W. Meier and O. Staffelbach, Correlation Properties of Combiners with Memory in Stream Cipher, *Journal of Cryptology*, vol.5, pp. 67–86, 1992.
12. R.A. Rueppel, Correlation immunity and the summation generator, *Advances in Cryptology - Crypto'85*, LNCS 219, Springer-Verlag, pp. 260–272, 1985.

A Proof of Proposition 2, equation (12)

To prove (12), we shall evaluate its right hand side at some arbitrary input value and show that it equals the next carry bit c_i^{t+1}. But before we do this, let us make some observations.

Note that we may take

$$c_i^{t+1} \equiv \lfloor (\sum_j x_j^t + c^t)/2^{i+1} \rfloor \qquad (\text{mod } 2) \tag{32}$$

as the definition of the carry bits. We may evaluate the right hand side of (11) at $\sum_j x_j^t = 0$ and equate it with the evaluation of (32) at the same point to shows $c_{i+1}^t = f_{i,0}$. Now, from the discussions in Section 2 on the matrix M, we know that at $\sum_j x_j^t = 2^k$, we have $S_0^t(2^k) = S_{2^k}^t(2^k) = 1$ and all other $S_j^t(2^k) = 0$.

Once more, evaluating (11) and (32) at $\sum_j x_j^t = 2^k$ with this in mind shows $c_{i+1}^t = f_{i,0} \oplus f_{i,2^k}$. Since we already know $c_{i+1}^t = f_{i,0}$, this implies $f_{i,2^k} = 0$, or equivalently, that the term $S_{2^k}^t$ does not appear in the linear sum (11).

We shall now evaluate the right hand side of (12) at some fixed LFSR output values $(x_1^t, \ldots, x_{2^{k+1}}^t)$ and carry value c^t. Set

$$\begin{aligned} x &= \textstyle\sum_j x_j^t, \\ x' &= \text{remainder of } x \text{ divided by } 2^k, \\ c' &= \text{remainder of } c^t \text{ divided by } 2^k. \end{aligned}$$

From the above observation, we know that the right hand side of (12) contains some of the terms $S_0^t(2^{k+1}), \ldots, S_{2^k-1}^t(2^{k+1})$, but does not contain any of the terms $S_j(2^{k+1})$ for $j \geq 2^k$. We also know from discussions of Section 2 that the evaluation of each $S_j^t(2^{k+1})$ at x for $j < 2^k$ is equal to its evaluation at x'. Note also that the term $c_{2^k}^t$ does not appear as input to any of the coefficients $f_{i,j}$ in the right hand side of (12). Hence,

$$\begin{aligned} &\text{RHS of (12) at } x \text{ and } c^t \\ &\qquad = \text{RHS of (12) at } x' \text{ and } c' \\ &\qquad = \text{RHS of (11) at } x' \text{ and } c' \\ &\qquad = \lfloor (x' + c')/2^{i+1} \rfloor \pmod 2 \\ &\qquad = \lfloor (x + c^t)/2^{i+1} \rfloor \pmod 2 \\ &\qquad = c_i^{t+1} \text{ at } x \text{ and } c^t. \end{aligned}$$

The condition $i \leq k-2$ has been used in the fourth equality. And the last equality is just (32). We have completed the proof that (12) is a valid expression for the next carry bits.

B Proof of Proposition 2, equation (13)

The proof for (13) is very similar to that of (12) and hence we shall be very brief. We ask the readers to read Appendix A before reading this section. The carry bit is given by

$$c_{k-1}^{t+1} \equiv \lfloor (\textstyle\sum_j x_j^t + c^t)/2^k \rfloor \pmod 2.$$

Evaluation of (11) at $\sum_j x_j^t = 0$ and 2^k shows that $f_{k-1,0} = 0$ and $f_{k-1,2^k} = 1$, hence we now always have $S_{2^k}^t$ as a linear term, with coefficient equal to 1, in the sums (11) and (13). The temporary values x, x', and c' may be defined as before. From the discussions of Section 2, one may write

$$\big(S_{2^k}^t(2^{k+1}) \text{ at } x\big) = \lfloor x/2^k \rfloor = \big(S_{2^k}^t(2^{k+1}) \text{ at } x'\big) \oplus \lfloor x/2^k \rfloor \pmod 2$$

and

$$\big(S_j^t(2^{k+1}) \text{ at } x\big) = \big(S_j^t(2^{k+1}) \text{ at } x'\big)$$

for $j < 2^k$. Also note that none of the terms $S_j(2^{k+1})$, for $j > 2^k$, appears in the sum (13) and that the term c_k^t visible in (13) is its only use in (13). Hence,

$$
\begin{aligned}
&\text{RHS of (13) at } x \text{ and } c^t\\
&\quad = \big(\text{RHS of (13) at } x \text{ and } c'\big) \oplus c_k^t\\
&\quad = \big(\text{RHS of (13) at } x' \text{ and } c'\big) \oplus c_k^t \oplus \lfloor x/2^k \rfloor \qquad \pmod 2\\
&\quad = \big(\text{RHS of (11) at } x' \text{ and } c'\big) \oplus c_k^t \oplus \lfloor x/2^k \rfloor \qquad \pmod 2\\
&\quad = \lfloor (x'+c')/2^k \rfloor \oplus c_k^t \oplus \lfloor x/2^k \rfloor \qquad \pmod 2\\
&\quad = \lfloor (x+c^t)/2^k \rfloor \qquad \pmod 2\\
&\quad = c_{k-1}^{t+1} \text{ at } x \text{ and } c^t.
\end{aligned}
$$

This completes the proof.

C Proof of Proposition 2, equation (14)

Define $x = \sum_j x_j^t$. Careful reading of Appendices A and B shows that the first term of (14) simplifies to

$$
\oplus_j f_{k-1,j} S_{j+2^k}^t(2^{k+1}) = \begin{cases} 0 & \text{for } 0 \le x \le 2^k, \\ \lfloor (x - 2^k + c^t)/2^k \rfloor \oplus c_k^t & \text{for } 2^k < x \le 2^{k+1}, \end{cases} \tag{33}
$$

and that the second term satisfies

$$
c_k^t\big(\oplus_j f_{k-1,j} S_j^t(2^{k+1})\big) = c_k^t\big(\lfloor (x+c^t)/2^k \rfloor \oplus c_k^t\big). \tag{34}
$$

It now suffices to compare the sum of these two values with

$$
c_k^{t+1} \equiv \lfloor (x+c^t)/2^{k+1} \rfloor \qquad \pmod 2. \tag{35}
$$

Define $x' = x - 2^k$ when $x > 2^k$ and define $c' = c^t - 2^k$ when $c_k^t = 1$.
Case 1) $x \le 2^k$, $c_k^t = 0$.
This is the most easy case.

$$
(33) \oplus (34) = 0 = \lfloor (x+c^t)/2^{k+1} \rfloor.
$$

Case 2) $x \le 2^k$, $c_k^t = 1$.

$$
\begin{aligned}
(33) \oplus (34) &= \lfloor (x+c^t)/2^k \rfloor \oplus 1\\
&= \lfloor (x+c')/2^k \rfloor\\
&= \lfloor \{(x+c') + 2^k\}/2^{k+1} \rfloor\\
&= \lfloor (x+c)/2^{k+1} \rfloor.
\end{aligned}
$$

Case 3) $x > 2^k$, $c_k^t = 0$.

$$\begin{aligned}(33) \oplus (34) &= \lfloor (x' + c^t)/2^k \rfloor \\ &= \lfloor \{(x' + c^t) + 2^k\}/2^{k+1} \rfloor \\ &= \lfloor (x + c^t)/2^{k+1} \rfloor.\end{aligned}$$

Case 4) $x > 2^k$, $c_k^t = 1$.

$$\begin{aligned}(33) \oplus (34) &= \lfloor (x' + c^t)/2^k \rfloor \oplus \lfloor (x + c^t)/2^k \rfloor \\ &= \lfloor (x' + c^t)/2^k \rfloor \oplus (\lfloor (x' + c^t)/2^k \rfloor \oplus 1) \\ &= 1 \\ &= \lfloor (x + c^t)/2^{k+1} \rfloor.\end{aligned}$$

This completes the proof.