Low Data Complexity Inversion Attacks on Stream Ciphers via Truncated Compressed Preimage Sets

Xiao Zhong[1,2], Mingsheng Wang[3], Bin Zhang[1,4], and Shengbao Wu[1,2]

[1] Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] Graduate School of Chinese Academy of Sciences, Beijing, China
[3] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[4] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
zhongxiao456@163.com, mingsheng_wang@aliyun.com, {zhangbin,wushengbao}@tca.iscas.ac.cn

Abstract. This paper focuses on the analysis of LFSR-based stream ciphers with low data complexity. We introduce a novel parameter called the k-th truncated compressed preimage set (TCP set), and propose a low data complexity attack to recover the initial LFSR state via the TCP sets. Our method costs very few keystream bits and less time than the brute force under some condition. We apply our method to a 90-stage LFSR-based keystream generator with filter Boolean function which can resist the algebraic attack and inversion attack given by Golić to the greatest extent. It needs only 10-bit keystream to recover the 90-bit initial state, costing less time and data than the algebraic attack. The time complexity is also less than that of the inversion attack. Moreover, we recover the 128-bit initial state of the stream cipher LILI-128 with our method. The data cost is just 9 keystream bits along with a memory cost of $O(2^{8.5})$, which is the minimum data cost to theoretically break LILI-128 so far as we know. The time complexity is $O(2^{122.4})$, better than the brute force. We also define a new security parameter called T_{comp} and suggest a design criterion for the LFSR-based stream ciphers.

Keywords: LFSR-based stream ciphers, k-th truncated compressed preimage set, algebraic attack, inversion attack, LILI-128.

1 Introduction

Last decades have witnessed the fast development of stream ciphers. As a key component of many stream ciphers, LFSR-based keystream generator is often fused with nonlinear filter generator for better performance. There are many stream ciphers which adopt the LFSR-based nonlinear filter generator, such as Grain v1 [8], SNOW 3G [5], WG-7 [9] and LILI-128 [4].

W. Susilo and Y. Mu (Eds.): ACISP 2014, LNCS 8544, pp. 131–147, 2014.

There are many classical analytical methods on LFSR-based stream ciphers, such as algebraic attack [2,1] and inversion attack [6,7]. For LFSR-based generators with nonlinear filter Boolean function, the algebraic immunity [10] of the Boolean function should be large enough to resist the algebraic attack. To resist the inversion attack, the memory size of the stream cipher should be close or equal to the length of the LFSR. We need to note that what is called "memory" has nothing to do with filters or combiners with memory and refers to a specific inversion attack [6,7] in which the attacker guesses as many consecutive bits of the LFSR as spanned by the taps of the filter function. What is called "memory" in these attacks is the span of the filter function. Designers often choose keystream generators filtered by Boolean functions of optimum algebraic immunity along with large memory size.

Analysts value attacks on stream ciphers which cost less time than the brute force or the declared security level. To sufficiently understand the security of the analyzed stream cipher, we should pay attention to the fact that sometimes the amount of the data available to the adversary is extremely small due to the practical restrictions. Then it is necessary to pursue the research of attacks costing small amount of data, along with a time complexity less than the brute force or the declared security level.

In this paper, we propose a low data complexity attack on the LFSR-based keystream generators with nonlinear filter. Our method can recover the initial LFSR state with very few keystream bits faster than the brute force under some condition. It also shows that although the filter Boolean function is of optimum algebraic immunity and the memory size is equal to the length of the LFSR, our method may recover the initial state in less time and data than that of the algebraic attack or inversion attack given by Golić, J.D. et al.

For the model of LFSR-based keystream generator with nonlinear filter Boolean function $f \in B_n$, where B_n is the ring of Boolean functions in n variables, we introduce two parameters called the k-th compressed preimage set (CP set) and k-th truncated compressed preimage set (TCP set). We propose a low data complexity attack to recover the initial LFSR state via the k-th TCP sets. Our method costs very few keystream bits to recover the initial state when the number of the k-th TCP sets for the filter Boolean function is large enough. When the algebraic immunity of the filter function is optimum, people can try our method to see whether they can recover the initial state with less time and data than that of the algebraic attack.

Our method can recover the initial LFSR state with time complexity less than the exhaustive search on condition that at least one k-th appropriate TCP set (ATCP set) exists. We define a new security parameter called T_{comp} when there exists at least one k-th ATCP set. To resist our attack, we suggest that T_{comp} should be larger than 2^{l-1}, where l is the length of the LFSR, which is another design criterion for the LFSR-based stream ciphers.

Furthermore, we apply our method to a 90-stage LFSR-based keystream generator with a 9-variable Carlet-Feng Boolean function as its filter, and its memory size is 90, which indicates that it can resist the algebraic attack given in [2] and

inversion attack [6,7] to the greatest extent. The time complexity of our method to recover the 90-bit initial state is $T_{comp} = O(2^{75.1})$, and the data complexity is $D_{comp} = 10$ bits. The time complexity of the algebraic attack is $T_{AA} = O(2^{76.2})$ with a data complexity of $D_{AA} = O(2^{25.4})$. Moreover, the time complexity of the inversion attack [6,7] is close to $O(2^{90})$, which is larger than that of our method. We also recover the 128-bit initial state of the stream cipher LILI-128 with our method. The data cost is just 9 keystream bits along with a memory cost of $O(2^{8.5})$, which is the minimum data cost to theoretically break LILI-128 so far as we know. It highlights the advantage of the low data cost for our method. The time complexity is $O(2^{122.4})$, better than the brute force.

This paper is organized as follows: Section 2 introduces some preliminaries related to our work. In Section 3, we introduce two novel parameters called the k-th compressed preimage set and k-th truncated compressed preimage set and give an algorithm to compute the k-th ATCP sets. In Section 4, for LFSR-based keystream generators with nonlinear filter Boolean function, we propose a low data complexity attack to recover the initial state via the k-th TCP sets. An example is given in Section 5, along with the analysis of the time and data complexity. We also apply our method to the stream cipher LILI-128 in Section 6. Section 7 concludes this paper.

2 Preliminaries

2.1 Brief Description of the LFSR-Based Keystream Generator with Nonlinear Filter

Denote the ring of Boolean functions in n variables as B_n. Let f be any Boolean function in B_n, denote $S_1(f) = \{x \in F_2^n | f(x) = 1\}$, $S_0(f) = \{x \in F_2^n | f(x) = 0\}$.

In this paper, we focus on the model of LFSR-based keystream generator with nonlinear filter Boolean function, which is a common component of the stream ciphers. Figure 1 shows the general model.

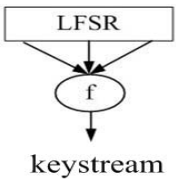

Fig. 1. LFSR-based keystream generator with nonlinear filter

First, we give a brief description for this model. Let the length of the linear feedback shift register be l. L is the "connection function" of the LFSR, and it is linear. The LFSR generator polynomial is a primitive polynomial $p(x) = p_0 + p_1x + ... + p_{l-1}x^{l-1} + x^l$. Let the initial state of the LFSR be $s^0 = (s_0, s_1, ..., s_{l-1})$,

and it generates a m-sequence $s_0, s_1, s_2, ...$ For sake of narrative convenience, we call this m-sequence as LFSR sequence. The state of the LFSR at time t is

$$s^t = (s_t, s_{t+1}, ..., s_{t+l-1}) = L^t(s_0, s_1, ..., s_{l-1}),$$

which is filtered by a balanced nonlinear Boolean function $f \in B_n$ and outputs one bit c_t at time t. For any c_t, there are 2^{n-1} possible preimage tuples $(s_t^1, s_t^2, ..., s_t^n)$. Define the corresponding preimage set as

$$S_{c_t} = \{s \in F_2^n | f(s) = c_t\}.$$

Our goal is to recover the l initial state bits of the LFSR. Suppose we observe $m = \lceil \frac{l}{n} \rceil$ keystream bits $c_{t_1}, c_{t_2}, ..., , c_{t_m}$ at time t_1, t_2..., t_m, then we can build an equation system.

$$c_{t_i} = f(s^{t_i}) = f(L^{t_i}(s_0, ..., s_{l-1})) = \sum_{j=0}^{l-1} a_{i,j} s_j, i = 1, 2, ..., m. \tag{1}$$

Notice that the "connection function" of the LFSR is linear, so the coefficient $a_{i,j}$ can be derived from the "connection function" L. Moreover, if the coefficient matrix of the equation system (1) is full rank, then its solution is unique, resulted to the initial state bits of the LFSR.

2.2 Algebraic Attack and Inversion Attack

In this section, we would like to review two classical methods: algebraic attack [2] and inversion attack [6,7], which are efficient analytical methods on LFSR-based keystream generators.

Algebraic Attack
With the same notation in Section 2.1, for each c_t, we can construct an equation involving some key bits and initial value as its variables. Denote the output of the filter generator by $c_0, c_1, c_2, ...$, where $c_i \in F_2$, then we can get the following equation system:

$$\begin{cases} c_0 = f \quad (s_0, s_1, ..., s_{l-1}) \\ c_1 = f(L\,(s_0, s_1, ..., s_{l-1})) \\ c_2 = f(L^2(s_0, s_1, ..., s_{l-1})) \\ \vdots \end{cases} \tag{2}$$

Then the problem of recovering the l initial state bits of the LFSR is reduced to solving the equation system (2).

The main idea of the algebraic attack proposed in [2] is to decrease the degree of the equation system (2) by using the annihilators of f or $f + 1$.

Algebraic attack motivated the research of the annihilators and algebraic immunity for Boolean functions.

Definition 1. *[10] For $f \in B_n$, define $AN(f) = \{g \in B_n | fg = 0\}$. Any function $g \in AN(f)$ is called an annihilator of f. The algebraic immunity of f, denoted by $AI(f)$, is the minimum degree of all the nonzero annihilators for f or $f+1$.*

By Courtois and Meier's theorem [2], $AI(f) \leq \lceil \frac{n}{2} \rceil$. In general $AI(f)$ should be as large as possible in order to resist the algebraic attack.

Table 1 shows the complexity of the algebraic attack on the LFSR-based keystream generator in Figure 1, where $N = \binom{l}{AI(f)}$, ω is the parameter of the Gaussian elimination and in theory $\omega \leq 2.376$ [3].

Table 1. Complexity of AA for the Model in Figure 1

Time	Data	Memory
N^{ω}	N	N^2

While as the authors of [2] declare, the (neglected) constant factor in that algorithm is expected to be very big and they regard Strassen's algorithm [12] as the fastest practical algorithm. Then they evaluate the complexity of the Gaussian reduction to be $7 \cdot N^{log_2^7}/64$ CPU clocks. Many scholars adopt $\omega = 3$ when they use Table 1 to evaluate the time and data complexity of the algebraic attack. In this paper, we also adopt $\omega = 3$ in Table 1 to estimate the complexity of the algebraic attack.

Inversion Attack

The main idea of the inversion attack is proposed in [6,7]. With the above notations, let $\gamma = (\gamma_i)_{i=1}^n$ denote the tapping sequence specifying the inputs to the filter Boolean function f, and let $M = \gamma_n - \gamma_1$ denote the input memory size of the nonlinear filter generator regarded as the finite input memory combiner with one input and one output.

The inversion attack in [6] targets to the case when the filter function is linear in the first or the last input variable, and runs forwards or backwards accordingly. The attack guesses M-bit unknown initial LFSR bits first and then recover the initial LFSR state by taking advantage of the property of the filter function and the recursion of the LFSR.

It takes 2^{M-1} trials on average to find a correct initial memory state. One may as well examine all 2^M initial memory states.

Golić, J.D. et al. generalized the inversion attack in [7]. Unlike the inversion attack which requires that the filter function be linear in the first or the last input variable, the attack in [7] can be applied for any filter function. The time complexity remains close to 2^M.

Remark 1. In fact, since algebraic attack and inversion attack are powerful tools for the LFSR-based stream ciphers with nonlinear filter generators, designers often adopt Boolean functions of optimum algebraic immunity, with a memory size close or equal to the length of the LFSR.

3 k-th Truncated Compressed Preimage Sets

In this section, we propose two novel parameters called the k-th compressed preimage set (CP set) and k-th truncated compressed preimage set (TCP set), which helps to recover the l-bit initial LFSR state. To begin with, we give the following definition.

Definition 2. *For a balanced Boolean function $f(x_1, x_2, ..., x_n) \in B_n$, we can get the preimage set $S_u(f)$ for $f(x) = u$, $u \in \{0, 1\}$. For a fixed $k \in [1, n]$, for some fixed set of indexes $I = \{i_1, i_2, ..., i_k\} \subseteq \{1, 2, ..., n\}$ and a certain k-dimensional vector $b = (b_1, b_2, ..., b_k) \in F_2^k$, define the k-th compressed preimage sets of $S_u(f)$ as:*

$$e_{k,b} = \{a \in S_u(f) | a_{i_j} = b_j \text{ for } j = 1, 2, ..., k\}.$$

Denote

$$N_{k,b} = |e_{k,b}|,$$

here $|.|$ denotes the number of the elements in a set.
Define the k-th truncated compressed preimage set $E_{k,b}$ corresponding to $N_{k,b}$ as

$$E_{k,b} = \{b\} = \{(b_1, b_2, ..., b_k)\}.$$

Then we can get that for $f(x_1, x_2, ..., x_n) = u$, the probability that $p(x_{i_1} = b_1, x_{i_2} = b_2, ..., x_{i_k} = b_k)$ is

$$p_k = \frac{N_{k,b}}{2^{n-1}}.$$

Notice that there may exist another k-dimensional vector $b' = (b_1', b_2', ..., b_k') \in F_2^k$ such that $|e_{k,b'}| = N_{k,b}$.

For $f(x) = u$, given a k-th TCP set of $S_u(f)$, $E_{k,b} = \{(b_1, b_2, ..., b_k)\}$, we can get that $p(f(x_1, x_2, ..., x_n) = u | x_{i_1} = b_1, x_{i_2} = b_2, ..., x_{i_k} = b_k) = \frac{N_{k,b}}{2^{n-1}}$. We use the method called "guess and determine" to solve this nonlinear equation at an expected cost of $\frac{2^{n-k}+2^{n-k-1}}{2} = 2^{n-k-1} + 2^{n-k-2}$, for the worst complexity is the exhaustive search of the 2^{n-k} possible bit-strings for the left $n-k$ unknown bits, and the best case is that one of the left $n-k$ unknown bits can be uniquely determined by guessing the other $n-k-1$ bits. The probability that the solution is the right one is $p = \frac{N_{k,b}}{2^{n-1}}$.

Then we are expected to do the above operation $\frac{1}{p}$ times to get the right solution, we can make it by choosing $\frac{1}{p}$ keystream bits.

The time complexity that we recover the right solution is

$$T = \frac{1}{p} \cdot (2^{n-k-1} + 2^{n-k-2}) = \frac{2^{2n-k-2} + 2^{2n-k-3}}{N_{k,b}}.$$

The data complexity is

$$D = \frac{1}{p}.$$

We can derive that when $2^{n-k-1} + 2^{n-k-2} < N_{k,b} < 2^{n-k}$, then $T < 2^{n-1}$, which means that it is less than the complexity of exhaustive search. We call the k-th TCP sets which satisfy the condition $2^{n-k-1} + 2^{n-k-2} < N_{k,b} < 2^{n-k}$ as the k-th appropriate TCP sets (ATCP sets). The following example shows that we can make the complexity strictly less than the exhaustive search with our idea.

Example 1. Given a 5-variable Carlet-Feng Boolean function $f = x_1x_2x_3x_5 + x_1x_2x_5 + x_1x_2 + x_1x_3x_4x_5 + x_1x_3x_4 + x_1x_3x_5 + x_1x_4x_5 + x_1x_4 + x_2x_3 + x_2x_4x_5 + x_2x_5 + x_3x_4 + x_4x_5 + 1$. $|S_0(f)| = |S_1(f)| = 16$. Table 2 shows some k-th ATCP sets of $S_0(f)$ and $S_1(f)$. Here we choose to compute the sets for $k = 2$.

Table 2. Compute the k-th appropriate TCP sets of $S_0(f)$ and $S_1(f)$

(a) ATCP sets of $S_0(f)$

k	Indexes	$N_{k,b}$	b
2	$\{4, 5\}$	7	$\{1, 1\}$
2	$\{1, 4\}$	7	$\{1, 1\}$

(b) ATCP sets of $S_1(f)$

k	Indexes	$N_{k,b}$	b
2	$\{2, 4\}$	7	$\{0, 0\}$

From Table 2, for $f(x) = 0$, the time complexity to recover the right solution is

$$T_0 = \frac{2^{2n-k-2} + 2^{2n-k-3}}{N_k} = \frac{2^6 + 2^5}{7} = 2^{3.77} < 2^4.$$

For $f(x) = 1$, the complexity to recover the right solution is

$$T_1 = \frac{2^{2n-k-2} + 2^{2n-k-3}}{N_k} = \frac{2^6 + 2^5}{7} = 2^{3.77} < 2^4.$$

We give an algorithm to compute the k-th appropriate TCP sets which satisfy $2^{n-k-1} + 2^{n-k-2} < N_{k,b} < 2^{n-k}$.

Algorithm 1. Compute the k-th ATCP sets of $S_u(f)$ (**ATCP Algorithm**)

Input: Boolean function f, $u \in \{0, 1\}$.
Set $E = \emptyset$, $E_0 = S_u(f)$, $k = 1$.
while $k \leq n$ **do**
  for $\{i_1, i_2, ..., i_k\} \subseteq \{1, 2, ..., n\}$, $b = (b_1, b_2, ..., b_k) \in F_2^k$ **do**
    Compute $E_{k,b}$ defined in Definition 2 and the corresponding $N_{k,b}$;
    if $2^{n-k-1} + 2^{n-k-2} < N_{k,b} < 2^{n-k}$ **then**
      $E = E \bigcup \{((i_1, i_2, ..., i_k), N_{k,b}, E_{k,b})\}$;
  $k = k + 1$;
Output E.

4 Low Data Complexity Inversion Attack to Recover the Initial LFSR State via the k-th ATCP Sets

According to Section 2, we can reduce the problem of recovering the initial state of LFSR to solving an equation system whose coefficient matrix is full rank.

With the same model introduced in Figure 1, let the length of the LFSR be l. The LFSR sequence is $s_0, s_1, s_2, ...$ The nonlinear filter Boolean function is $f \in B_n$, which is balanced. The keystream bits generated by the LFSR-based nonlinear filter generator are $c_0, c_1, c_2, ...$

In this section, we give a method to recover the initial LFSR state via the k-th ATCP sets. We divide the process into two parts. One is the precomputation phase, the other is the online phase.

Precomputation Phase: For Boolean function $f \in B_n$, for a fixed $k \in [1, n]$, compute the k-th ATCP sets of $S_0(f)$ and $S_1(f)$ respectively, and denote them as group G_0 and group G_1. Choose one set from each group and denote them as E_0 and E_1 respectively. Compute the corresponding probability $p_0 = \frac{N_{k,b}}{2^{n-1}}$ and $p_1 = \frac{N_{k,b'}}{2^{n-1}}$, where $N_{k,b}$ and $N_{k,b'}$ can be derived from the output of the ATCP algorithm.

Online Phase: Denote $m = \lceil \frac{l}{n} \rceil$.

Step 1: According to the specific tap positions of the filter Boolean function f, choose m-bit keystream $c_{t_1}, c_{t_2}..., c_{t_m}$ (continuous or not) which satisfy the following condition:
(1)Denote the set of the tap positions corresponding to c_{t_i} as $A_{t_i} = \{s^1_{t_i}, s^2_{t_i}, ..., s^n_{t_i}\}$. Require that A_{t_i}, $i = 1, 2, ..., m$ are pairwise disjoint.
(2)The coefficient matrix of the corresponding equation system $c_{t_i} = f(s^{t_i}) = f(L^{t_i}(s_0, ..., s_{l-1})) = \sum_{j=0}^{l-1} a_{i,j} s_j$, $i = 1, 2, ..., m$ should be full rank.

Step 2: For each c_{t_i}, we can get the k-th ATCP sets of $S_{c_{t_i}}(f)$ from the precomputation phase directly. Choose one set and denote it as E_{t_i}, and then we can get a nonlinear equation with probability of $p_{c_{t_i}}$. Solve this nonlinear equation with "guess and determine" method, we can get a candidate solution $\hat{E}_{t_i}$ for $f(x) = c_{t_i}$ with an expected cost of $2^{n-k-1} + 2^{n-k-2}$. Then we can get a candidate vector $E = \hat{E}_{t_1} || \hat{E}_{t_2} || \cdots || \hat{E}_{t_m}$ for l bits of the LFSR sequence, where "||" denotes a concatenation of two vectors. Because A_{t_i}, $i = 1, 2, ..., m$ are pairwise disjoint and the coefficient matrix of the corresponding linear equation system is full rank, the probability that E is the right solution for the l-bit LFSR sequence is $P = p_{c_{t_1}} \cdot p_{c_{t_2}} \cdot \cdots \cdot p_{c_{t_m}}$.

Step 3: Test the candidate vector E and check that if it is the right one. If it is, then we can derive the initial LFSR state bits, otherwise back to Step 2.
We can also choose the other sets in group G_0 and group G_1 to do the operation.

With the similar analysis in Section 3, the time complexity of the online phase is

$$T = (2^{n-k-1} + 2^{n-k-2})^m \cdot \frac{1}{P}. \tag{3}$$

According to Algorithm 1, we know that $\frac{1}{P} < (\frac{2^{n-1}}{2^{n-k-1}+2^{n-k-2}})^m$, then $T < 2^{l-1}$.

In the precomputation phase, compute all the k-th ATCP sets of $S_0(f)$: E_0^1, E_0^2,..., and denote the number of them as l_0. Also, compute all the k-th TCP sets of $S_1(f)$: E_1^1, E_1^2,..., and denote the number of them as l_1.

For the keystream bits chosen in Step 1: $c_{t_1}, c_{t_2}, ..., c_{t_m}$, denote

$$n_0 = |\{c_{t_i} | c_{t_i} = 0, i = 1, ..., m\}|, n_1 = |\{c_{t_i} | c_{t_i} = 1, i = 1, ..., m\}|.$$

Then for each m-bit keystream chosen in Step 1, the number of the candidate vectors for the l LFSR sequence bits in Step 2 is

$$l_0^{n_0} \cdot l_1^{n_1}.$$

Then the data complexity of our method is

$$D = m \cdot \frac{\frac{1}{P}}{l_0^{n_0} \cdot l_1^{n_1}}. \tag{4}$$

When the parameters l_0 and l_1 are large enough such that

$$\frac{\frac{1}{P}}{l_0^{n_0} \cdot l_1^{n_1}} \leq 1, \tag{5}$$

then the data complexity of our method would become very small, that is, we need only m keystream bits to recover the initial state of the LFSR. In fact, the values of l_0 and l_1 can satisfy the condition (5) in most cases.

For a fixed $k \in [1, n]$, when there exists at least one k-th ATCP set, we give the following definition.

Definition 3. *For a fixed $k \in [1, n]$, denote the time complexity and data complexity to recover the initial LFSR state via the k-th ATCP sets as T_k and D_k respectively, define*

$$T_{comp} = min\{T_k | k \in [1, n]\}.$$

Denote the data complexity corresponding to T_{comp} as D_{comp}.

Remark 2. Our method suggests a new design criterion for the LFSR-based stream ciphers with nonlinear filter. Suppose the time complexity of our method to recover the l-bit initial LFSR state is T_{comp} given in Definition 3, and the corresponding data complexity D_{comp} is acceptable, then the stream cipher should satisfy the following condition to resist our attack:

$$2^{l-1} < T_{comp}. \tag{6}$$

In the next section, we would like to give an example to show how to apply our method to the LFSR-based nonlinear filter keystream generators.

5 Analysis on a Keystream Generator with a Filter Boolean Function of Optimum Algebraic Immunity

In this section, we choose a model of keystream generator with nonlinear filter Boolean function which can resist the algebraic attack [2] and the inversion attack [6,7] to the greatest extent. Let the length of the LFSR be 90. The filter Boolean function f is a 9-variable Carlet-Feng Boolean function which is listed in Appendix A. The input memory size of the filter function is 90, which is the length of the LFSR. We can see that the stream cipher possesses two advantages: optimum algebraic immunity and large input memory size. The keystream generator outputs one bit each clock. In the following, we apply our method to the above keystream generator.

First of all, we compute the k-th ATCP sets of $S_0(f)$ and $S_1(f)$ using the ATCP algorithm. Practically, we usually choose the k-th ATCP sets whose $N_{k,b}$ is large, which helps to decrease the time and data complexity. Table 3 shows some k-th ATCP sets of $S_0(f)$ and $S_1(f)$.

Table 3. Compute the k-th appropriate TCP sets of $S_0(f)$ and $S_1(f)$

(a) ATCP sets of $S_0(f)$

k	Indexes	$N_{k,b}$	b	k	Indexes	$N_{k,b}$	b
5	$\{1,2,6,7,8\}$	13	[0, 0, 1, 1, 1]	5	$\{1,5,6,7,9\}$	13	[0, 1, 1, 1, 0]
5	$\{2,3,5,7,9\}$	13	[1, 1, 0, 0, 0]	5	$\{1,2,3,5,6\}$	13	[1, 1, 1, 0, 1]
5	$\{2,3,4,6,7\}$	13	[1, 0, 0, 0, 1]	5	$\{3,4,5,6,8\}$	13	[0, 1, 1, 1, 0]
5	$\{4,5,6,8,9\}$	13	[1, 1, 1, 0, 0]	5	$\{1,3,5,7,9\}$	13	[1, 0, 1, 0, 1]
5	$\{1,3,5,6,8\}$	13	[0, 0, 1, 1, 0]	5	$\{3,4,5,7,8\}$	13	[1, 0, 0, 0, 1]
5	$\{2,4,5,7,9\}$	13	[0, 1, 1, 0, 0]	5	$\{1,3,4,8,9\}$	13	[1, 0, 0, 1, 1]
5	$\{2,3,7,8,9\}$	13	[0, 0, 1, 1, 1]	5	$\{1,3,4,6,8\}$	13	[0, 1, 1, 0, 0]
5	$\{1,2,4,6,8\}$	13	[1, 1, 0, 0, 0]	5	$\{1,2,4,5,9\}$	13	[1, 1, 0, 0, 1]
5	$\{2,4,6,8,9\}$	13	[0, 1, 0, 1, 1]	6	159 groups of indexes	7	many

(b) ATCP sets of $S_1(f)$

k	Indexes	$N_{k,b}$	b	k	Indexes	$N_{k,b}$	b
5	$\{1,3,4,5,6\}$	14	[0, 0, 1, 1, 0]	5	$\{1,2,3,7,9\}$	14	[1, 1, 0, 0, 0]
5	$\{1,2,3,4,8\}$	14	[0, 1, 1, 0, 0]	5	$\{4,6,7,8,9\}$	14	[1, 0, 1, 1, 0]
5	$\{1,2,6,8,9\}$	14	[1, 0, 0, 0, 1]	5	$\{2,3,4,5,9\}$	14	[0, 1, 1, 0, 0]
5	$\{1,5,7,8,9\}$	14	[0, 1, 0, 1, 1]	6	130 groups of indexes	7	many

We choose $\lceil \frac{90}{9} \rceil = 10$ keystream bits which obey the two conditions in Step 1 given in Section 4, and denote them as $c_{t_1}, c_{t_2}, , ..., c_{t_{10}}$. Then we follow Step 2, here we choose $k = 6$. Denote $n_0 = |\{c_{t_i} | c_{t_i} = 0, i = 0, 1, ..., 10\}|$ and $n_1 = |\{c_{t_i} | c_{t_i} = 1, i = 0, 1, ..., 10\}|$. Then the time and data complexity of recovering the 90-bit initial LFSR state by the low data complexity attack are

$$T_{LDA} = (2^{9-6-1} + 2^{9-6-2})^{10} \cdot (\frac{256}{7})^{10} = 2^{75.1}.$$

$$D_{LDA} = 10 \cdot \frac{(\frac{256}{7})^{10}}{159^{n_0} \cdot 130^{n_1}}.$$

Because $\frac{(\frac{256}{7})^{10}}{159^{n_0} \cdot 130^{n_1}} < 1$, we just need 10 bits to recover the 90-bit initial LFSR state, then

$$D_{LDA} = 10.$$

The successful probability that we can recover the right 90-bit initial state is

$$P = 1 - (1 - (\frac{7}{256})^{10})^{(\frac{256}{7})^{10}} \approx 1 - e^{-1} \approx 0.63.$$

According to Table 1, the time and data complexity of the algebraic attack on this model are

$$T_{AA} = \binom{90}{AI(f)}^3 = \binom{90}{5}^3 = 2^{76.2}, D_{AA} = \binom{90}{AI(f)} = \binom{90}{5} = 2^{25.4}.$$

If we adopt inversion attack [6,7] to analyze this model, the time complexity T_{IA} is close to 2^{90}.

Table 4 shows the comparison among our method (LDA), algebraic attack (AA) and inversion attack (IA) on the above model.

Table 4. Comparison among LDA, AA and IA

T_{LDA}	D_{LDA}	T_{AA}	D_{AA}	T_{IA}
$O(2^{75.1})$	10	$O(2^{76.2})$	$O(2^{25.4})$	near $O(2^{90})$

We can see that our method costs less time and data than the algebraic attack [2] in this case. The time complexity is also less than that of the inversion attack [6,7].

Remark 3. For the LFSR-based keystream generator model given in Section 2, when the filter Boolean function is of optimum algebraic immunity, people can try our method to see whether the cost of time and data can be less than that of the algebraic attack.

6 Low Data Complexity Attack on LILI-128 via the k-th TCP Sets

In this section, we apply our method to the stream cipher LILI-128 [4] to show the advantage of the low data complexity for our method. The structure of the LILI-128 generator is illustrated in Figure 2. It contains two subsystems: clock control and data generation.

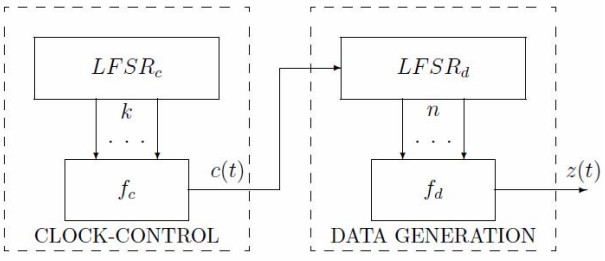

Fig. 2. Structure of LILI-128 Keystream Generator

The clock-control subsystem of LILI-128 adopts a pseudorandom binary sequence produced by a regularly clocked LFSR, $LFSR_c$, of length 39 and a function, f_c, operating on the contents of $k = 2$ stages of $LFSR_c$ to produce a pseudorandom integer sequence, $c = c(t)_{t=1}^{\infty}$. The feedback polynomial of $LFSR_c$ is chosen to be the primitive polynomial

$$x^{39} + x^{35} + x^{33} + x^{31} + x^{17} + x^{15} + x^{14} + x^2 + 1.$$

The data-generation subsystem of LILI-128 uses the integer sequence c produced by the clock subsystem to control the clocking of a binary LFSR, $LFSR_d$, of length $L_d = 89$. The contents of a fixed set of $n = 10$ stages of $LFSR_d$ are input to a specially chosen Boolean function, f_d. The binary output of f_d is the keystream bit $z(t)$. After $z(t)$ is produced, the two LFSRs are clocked and the process repeated to generate the keystream $z = z(t)_{t=1}^{\infty}$.
The feedback polynomial of $LFSR_d$ is chosen to be the primitive polynomial

$$x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1.$$

The initial state of $LFSR_d$ is never the all zero state. Let the stages of $LFSR_d$ be labeled $\alpha[0], \alpha[1], ..., \alpha[88]$ from left to right. Let the LFSR shift left. Then at time t, we have the following formula to calculate the feedback bit:

$$\alpha[89+t] = \alpha[88+t] \oplus \alpha[50+t] \oplus \alpha[47+t] \oplus \alpha[36+t] \oplus \alpha[34+t] \oplus \alpha[9+t] \oplus \alpha[6+t] \oplus \alpha[t],$$

where $\oplus$ indicates the exclusive-or operation on bits(equivalent to addition modulo 2).

The 10 inputs to f_d are taken from $LFSR_d$ according to this full positive difference set: (0,1,3,7,12,20,30,44,65,80). The following is the expression of f_d:

$$f_d = x_4x_6x_7x_8x_9x_{10} + x_5x_6x_7x_8x_9x_{10} + x_4x_6x_7x_9x_{10} + x_5x_6x_7x_9x_{10} + x_3x_7x_8x_9x_{10} + x_4x_7x_8x_9x_{10} + x_4x_6x_7x_8x_9 + x_5x_6x_7x_8x_9 + x_4x_8x_9x_{10} + x_6x_8x_9x_{10} + x_4x_6x_7x_9 + x_5x_6x_7x_9 + x_2x_7x_8x_9 + x_4x_7x_8x_9 + x_3x_7x_8x_{10} + x_5x_7x_8x_{10} + x_2x_7x_9x_{10} + x_4x_7x_9x_{10} + x_6x_7x_9x_{10} + x_1x_8x_9x_{10} + x_3x_8x_9x_{10} + x_6x_7x_{10} + x_3x_8x_{10} + x_4x_8x_{10} + x_2x_9x_{10} + x_3x_9x_{10} + x_4x_9x_{10} + x_5x_9x_{10} + x_3x_7x_9 + x_6x_7x_9 + x_3x_8x_9 + x_6x_8x_9 + x_4x_7x_{10} + x_5x_7x_{10} + x_6x_7 + x_1x_8 + x_2x_8 + x_1x_9 + x_3x_9 + x_4x_{10} + x_6x_{10} + x_2 + x_3 + x_4 + x_5.$$

To begin with, we first guess the 39-bit internal state of $LFSR_c$ and attack the second component $LFSR_d$ alone. The total time complexity should be multiplied by 2^{39}. In the following, we recover the internal state of $LFSR_d$ by using our method given in Section 4.

In the case of LILI-128, for $k = 5$, even the parameter $N_{k,b}$ defined in Definition 2 is less than $2^{n-k-1} + 2^{n-k-2}$, the time complexity of recovering the initial state is better than the brute force, which highlights the power of the TCP sets. To comprehensively consider the requirements of less time complexity than the brute force and low data complexity, we choose the 5-th TCP sets whose $N_{k,b}$ satisfy the condition of $20 \leq N_{k,b} < 32$. Table 5 shows the 5-th TCP sets which would be adopted.

We choose $\lceil \frac{89}{10} \rceil = 9$ keystream bits which obey the two conditions in Step 1 given in Section 4, and denote them as $c_{t_1}, c_{t_2}, , ..., c_{t_9}$. Denote $n_0 = |\{c_{t_i} | c_{t_i} = 0, i = 0, 1, ..., 9\}|$ and $n_1 = |\{c_{t_i} | c_{t_i} = 1, i = 0, 1, ..., 9\}|$.

Then the time and data complexity of recovering the 128-bit internal state of LILI-128 are about

$$T_{LDA} = 2^{39} \cdot (2^{10-5-1} + 2^{10-5-2})^9 \cdot (\frac{512}{20})^{n_0} \cdot (\frac{512}{20})^{n_1} = 2^{39} \cdot 24^9 \cdot (\frac{512}{20})^9 = 2^{122.4}.$$

$$D_{LDA} = 9 \cdot \frac{(\frac{512}{20})^{n_0} \cdot (\frac{512}{20})^{n_1}}{36^{n_0} \cdot 35^{n_1}}.$$

Notice that $\frac{(\frac{512}{20})^{n_0} \cdot (\frac{512}{20})^{n_1}}{36^{n_0} \cdot 35^{n_1}} < 1$, then the data complexity is

$$D_{LDA} = 9.$$

We need to store the k-th TCP sets of $S_0(f)$ and $S_1(f)$ shown in Table 5. Then the required memory is

$$M_{LDA} = (36 + 35) \cdot 5 = 2^{8.5}.$$

The successful probability that we can recover the right 89-bit $LFSR_c$ internal state is

$$P = 1 - (1 - (\frac{20}{512})^9)^{(\frac{512}{20})^9} \approx 1 - e^{-1} \approx 0.63.$$

If we apply algebraic attack to LILI-128, the time and data complexity are about

$$T_{AA} = 2^{39} \cdot \begin{pmatrix} 89 \\ AI(f) \end{pmatrix}^3 = 2^{39} \cdot \begin{pmatrix} 89 \\ 4 \end{pmatrix}^3 = 2^{102.7}, D_{AA} = \begin{pmatrix} 89 \\ AI(f) \end{pmatrix} = \begin{pmatrix} 89 \\ 4 \end{pmatrix} = 2^{21.2}.$$

The required memory is about

$$M_{AA} = \begin{pmatrix} 89 \\ AI(f) \end{pmatrix}^2 = \begin{pmatrix} 89 \\ 4 \end{pmatrix}^2 = 2^{42.4}.$$

As related research we note that Tsunoo, Y. et al. proposed an attack which recovers the internal state of LILI-128 by using 2^7 keystream bits and $2^{99.1}$ computations, along with $2^{28.6}$-bit memory [11].

Table 6 shows the comparison among our method (LDA), algebraic attack (AA) and the method in [11] on LILI-128.

Table 5. Compute the 5-th TCP sets of $S_0(f)$ and $S_1(f)$

(a) TCP sets of $S_0(f)$

k	Indexes	$N_{k,b}$	b	k	Indexes	$N_{k,b}$	b
5	$\{1,3,4,5,6\}$	22	[0, 0, 0, 0, 0]	5	$\{2,3,4,5,6\}$	22	[0, 0, 0, 0, 0]
5	$\{1,2,4,5,6\}$	21	[0, 0, 0, 0, 0]	5	$\{1,2,4,5,6\}$	21	[1, 0, 1, 0, 0]
5	$\{1,2,3,4,6\}$	21	[0, 0, 0, 1, 1]	5	$\{1,2,3,4,5\}$	20	[0, 0, 0, 1, 1]
5	$\{1,2,3,4,5\}$	20	[0, 0, 0, 0, 0]	5	$\{2,5,6,7,9\}$	20	[0, 1, 1, 1, 1]
5	$\{2,5,6,7,10\}$	20	[0, 1, 1, 1, 0]	5	$\{2,5,6,9,10\}$	20	[0, 0, 0, 1, 0]
5	$\{2,5,6,8,9\}$	20	[0, 1, 1, 1, 1]	5	$\{2,5,6,8,10\}$	20	[0, 1, 1, 1, 0]
5	$\{2,5,6,8,9\}$	20	[0, 0, 0, 1, 1]	5	$\{2,5,6,9,10\}$	20	[1, 1, 0, 1, 0]
5	$\{2,3,4,5,6\}$	20	[0, 1, 1, 0, 0]	5	$\{2,5,6,8,9\}$	20	[1, 1, 0, 1, 1]
5	$\{2,5,6,9,10\}$	20	[1, 0, 1, 1, 0]	5	$\{1,3,4,5,6\}$	20	[1, 1, 1, 1, 1]
5	$\{2,5,6,7,9\}$	20	[0, 0, 0, 1, 1]	5	$\{2,5,6,7,8\}$	20	[0, 1, 1, 1, 1]
5	$\{2,5,6,8,10\}$	20	[1, 0, 1, 1, 0]	5	$\{2,5,6,7,10\}$	20	[1, 0, 1, 1, 0]
5	$\{2,5,6,7,10\}$	20	[0, 0, 0, 1, 0]	5	$\{2,5,6,8,9\}$	20	[1, 0, 1, 1, 1]
5	$\{1,2,3,4,6\}$	20	[0, 0, 0, 0, 0]	5	$\{2,5,6,8,10\}$	20	[0, 0, 0, 1, 0]
5	$\{1,2,3,5,6\}$	20	[0, 0, 0, 1, 1]	5	$\{2,5,6,8,10\}$	20	[1, 1, 0, 1, 0]
5	$\{2,5,6,7,10\}$	20	[1, 1, 0, 1, 0]	5	$\{2,5,6,7,8\}$	20	[1, 1, 0, 1, 1]
5	$\{2,5,6,7,8\}$	20	[0, 0, 0, 1, 1]	5	$\{1,2,3,5,6\}$	20	[1, 1, 0, 0, 1]
5	$\{2,5,6,7,8\}$	20	[1, 0, 1, 1, 1]	5	$\{2,5,6,9,10\}$	20	[0, 1, 1, 1, 0]
5	$\{2,5,6,7,9\}$	20	[1, 0, 1, 1, 1]	5	$\{2,5,6,7,9\}$	20	[1, 1, 0, 1, 1]

(b) TCP sets of $S_1(f)$

k	Indexes	$N_{k,b}$	b	k	Indexes	$N_{k,b}$	b
5	$\{1,3,4,5,6\}$	20	[0, 0, 0, 1, 0]	5	$\{1,3,4,5,6\}$	20	[0, 0, 0, 0, 1]
5	$\{1,3,4,5,6\}$	20	[0, 0, 1, 0, 0]	5	$\{1,3,4,5,6\}$	20	[0, 0, 0, 1, 0]
5	$\{2,5,6,7,9\}$	20	[1, 1, 1, 1, 1]	5	$\{2,5,6,7,8\}$	20	[1, 0, 0, 1, 1]
5	$\{2,5,6,8,9\}$	20	[0, 0, 1, 1, 1]	5	$\{2,5,6,7,10\}$	20	[0, 1, 0, 1, 0]
5	$\{2,5,6,7,8\}$	20	[0, 0, 1, 1, 1]	5	$\{2,5,6,8,10\}$	20	[1, 1, 1, 1, 0]
5	$\{2,5,6,9,10\}$	20	[1, 0, 0, 1, 0]	5	$\{2,5,6,7,8\}$	20	[0, 1, 0, 1, 1]
5	$\{2,5,6,7,10\}$	20	[1, 1, 1, 1, 0]	5	$\{2,5,6,8,10\}$	20	[0, 1, 0, 1, 0]
5	$\{2,5,6,8,10\}$	20	[1, 0, 0, 1, 0]	5	$\{2,5,6,7,9\}$	20	[0, 0, 1, 1, 1]
5	$\{2,5,6,9,10\}$	20	[0, 1, 0, 1, 0]	5	$\{2,3,4,5,6\}$	20	[0, 0, 0, 0, 1]
5	$\{1,3,4,5,6\}$	20	[0, 1, 0, 0, 0]	5	$\{1,3,4,5,6\}$	20	[1, 0, 0, 0, 0]
5	$\{2,5,6,9,10\}$	20	[0, 0, 1, 1, 0]	5	$\{2,5,6,7,9\}$	20	[0, 1, 0, 1, 1]
5	$\{2,5,6,7,9\}$	20	[1, 0, 0, 1, 1]	5	$\{2,5,6,7,8\}$	20	[1, 1, 1, 1, 1]
5	$\{2,5,6,8,9\}$	20	[1, 0, 0, 1, 1]	5	$\{2,3,4,5,6\}$	20	[1, 0, 1, 1, 1]
5	$\{2,5,6,7,10\}$	20	[1, 0, 0, 1, 0]	5	$\{2,5,6,8,9\}$	20	[0, 1, 0, 1, 1]
5	$\{2,3,4,5,6\}$	20	[1, 0, 0, 0, 0]	5	$\{2,5,6,7,10\}$	20	[0, 0, 1, 1, 0]
5	$\{2,5,6,8,10\}$	20	[0, 0, 1, 1, 0]	5	$\{2,3,4,5,6\}$	20	[0, 1, 0, 0, 0]
5	$\{2,5,6,9,10\}$	20	[1, 1, 1, 1, 0]	5	$\{1,2,3,4,6\}$	20	[0, 0, 0, 1, 0]
5	$\{2,5,6,8,9\}$	20	[1, 1, 1, 1, 1]				

Table 6. Comparison among LDA, AA and method in [11]

	T	D	M
Our method	$O(2^{122.4})$	9	$O(2^{8.5})$
Algebraic attack	$O(2^{102.7})$	$O(2^{21.2})$	$O(2^{42.4})$
Method in [11]	$O(2^{99.1})$	$O(2^{7})$	$O(2^{28.6})$

7 Conclusion

This paper introduces two novel parameters for Boolean functions called the k-th compressed preimage set (CP set) and k-th truncated compressed preimage set (TCP set). We give an algorithm to compute the k-th appropriate TCP sets and propose a low data complexity attack to recover the initial LFSR state via the k-th TCP sets. Our method costs very few keystream bits to recover the initial state when the number of the k-th TCP sets is large enough. We apply our method to a 90-stage LFSR-based keystream generator with a 9-variable filter Boolean function of optimum algebraic immunity. The time complexity and data complexity are both less than that of the algebraic attack [2]. The time complexity is also less than that of the inversion attack [6,7]. Moreover, we recover the 128-bit initial state of the stream cipher LILI-128 by using our method. The data cost is just 9 keystream bits along with a memory cost of $O(2^{8.5})$, which is the minimum data cost to theoretically break LILI-128 so far as we know. It highlights the advantage of the low data cost for our method. The time complexity is $O(2^{122.4})$, better than the brute force. Our method also suggests a new design criterion for the LFSR-based stream ciphers with nonlinear filter: with an acceptable data cost, the parameter T_{comp} should be larger than 2^{l-1}, where l is the length of the LFSR.

Acknowledgements. We are grateful to the anonymous reviewers for their valuable comments on this paper. This work was supported by the National Basic Research Program of China (Grant No. 2013CB834203, Grant No. 2013CB338002) and the National Natural Science Foundation of China (Grant No. 61379142, Grant No. 11171323, Grant No. 60833008, Grant No. 60603018, Grant No. 61173134, Grant No. 91118006, Grant No. 61272476), the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDA06010701), IIEs Research Project on Cryptography (Grant No. Y3Z0016102).

References

1. Armknecht, F., Krause, M.: Algebraic attacks on Combiners with Memory. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 162–175. Springer, Heidelberg (2003)
2. Courtois, N.T., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003)

3. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symbolic Computation 9, 251–280 (1990)
4. Dawson, E., Clark, A., Golic, J., Millan, W., Penna, L., Simpson, L.: The LILI-128 Keystream Generator, NESSIE submission. In: The Proceedings of the First Open NESSIE Workshop (2000)
5. ETSI/SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 2: SNOW3G Specification, version 1.1 (2006), http://www.3gpp.org/ftp/
6. Golić, J.D.: On the security of nonlinear filter generators. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 173–188. Springer, Heidelberg (1996)
7. Golić, J.D., Clark, A., Dawson, E.: Inversion Attack and Branching. In: Pieprzyk, J., Safavi-Naini, R., Seberry, J. (eds.) ACISP 1999. LNCS, vol. 1587, pp. 88–102. Springer, Heidelberg (1999)
8. Hell, M., Johansson, T., Meier, W.: Grain-A Stream Cipher for Constrained Environments. eStream Project, http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain-p3.pdf
9. Luo, Y., Chai, Q., Gong, G., Lai, X.: A lightweight stream cipher wg-7 for RFID encryption and authentication. In: GLOBECOM, pp. 1–6 (2010)
10. Meier, W., Pasalic, E., Carlet, C.: Algebraic attacks and decomposition of boolean functions. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 474–491. Springer, Heidelberg (2004)
11. Tsunoo, Y., Saito, T., Shigeri, M., Kubo, H., Minematsu, K.: Shorter bit sequnence is enough to break stream cipher LILI-128. Trans. Inf. Theory 51(12), 4312–4319 (2008)
12. Strassen, V.: Gaussian Elimination is Not Optimal. Numerische Mathematik 13, 354–356 (1969)

A Appendix: 9-variable Carlet-Feng Boolean Function

$$f = x_1x_2x_3x_4x_5x_6x_7x_8 + x_1x_2x_3x_4x_5x_6x_8 + x_1x_2x_3x_4x_5x_6x_9 + x_1x_2x_3x_4x_5x_6 + x_1x_2x_3x_4x_5x_7x_8x_9 + x_1x_2x_3x_4x_5x_8 + x_1x_2x_3x_4x_6x_7 + x_1x_2x_3x_4x_6x_8x_9 + x_1x_2x_3x_4x_7x_8x_9 + x_1x_2x_3x_4x_7x_9 + x_1x_2x_3x_4x_8x_9 + x_1x_2x_3x_5x_6x_7x_9 + x_1x_2x_3x_5x_6x_8 + x_1x_2x_3x_5x_7x_8 + x_1x_2x_3x_5x_7x_9 + x_1x_2x_3x_5x_7 + x_1x_2x_3x_5x_8x_9 + x_1x_2x_3x_5x_8 + x_1x_2x_3x_5x_9 + x_1x_2x_3x_6x_7x_8x_9 + x_1x_2x_3x_6x_7x_8 + x_1x_2x_3x_6x_7x_9 + x_1x_2x_3x_6x_7 + x_1x_2x_3x_6x_8 + x_1x_2x_3x_7x_8 + x_1x_2x_3x_8 + x_1x_2x_3x_9 + x_1x_2x_4x_5x_6x_7x_8x_9 + x_1x_2x_4x_5x_6x_7x_9 + x_1x_2x_4x_5x_6x_9 + x_1x_2x_4x_5x_7x_8x_9 + x_1x_2x_4x_5x_8x_9 + x_1x_2x_4x_5x_9 + x_1x_2x_4x_6x_7x_8x_9 + x_1x_2x_4x_6x_7x_8 + x_1x_2x_4x_6x_7 + x_1x_2x_4x_6x_8x_9 + x_1x_2x_4x_6x_8 + x_1x_2x_4x_6 + x_1x_2x_4x_7x_8 + x_1x_2x_4x_7x_9 + x_1x_2x_4x_8 + x_1x_2x_4x_9 + x_1x_2x_5x_6x_7x_8 + x_1x_2x_5x_6x_7 + x_1x_2x_5x_6x_8 + x_1x_2x_5x_7x_8x_9 + x_1x_2x_5x_7x_8 + x_1x_2x_5x_8x_9 + x_1x_2x_5x_9 + x_1x_2x_6x_7x_8x_9 + x_1x_2x_6x_8x_9 + x_1x_2x_6x_8 + x_1x_2x_6 + x_1x_2x_7x_8 + x_1x_2x_7x_9 + x_1x_2x_7 + x_1x_2x_8 + x_1x_2x_9 + x_1x_3x_4x_5x_6x_7x_8x_9 + x_1x_3x_4x_5x_6x_7x_8 + x_1x_3x_4x_5x_6x_7x_9 + x_1x_3x_4x_5x_6x_8x_9 + x_1x_3x_4x_5x_6x_8 + x_1x_3x_4x_5x_6x_9 + x_1x_3x_4x_5x_6 + x_1x_3x_4x_5x_7x_8x_9 + x_1x_3x_4x_5x_7x_8 + x_1x_3x_4x_5x_7x_9 + x_1x_3x_4x_5x_8 + x_1x_3x_4x_5x_9 + x_1x_3x_4x_5 + x_1x_3x_4x_6x_7x_8x_9 + x_1x_3x_4x_6x_7x_8 + x_1x_3x_4x_6x_7x_9 + x_1x_3x_4x_6x_7 + x_1x_3x_4x_7x_8x_9 + x_1x_3x_4x_7x_8 + x_1x_3x_4x_7x_9 + x_1x_3x_4x_8 + x_1x_3x_5x_6x_7x_8x_9 + x_1x_3x_5x_6x_7x_8 + x_1x_3x_5x_6x_8 + x_1x_3x_5x_7x_8 + x_1x_3x_5x_7 + x_1x_3x_5x_8x_9 + x_1x_3x_5x_8 + x_1x_3x_6x_7x_8 + x_1x_3x_6x_7x_9 + x_1x_3x_6x_8 + x_1x_3x_6 + x_1x_3x_7x_8x_9 + x_1x_3x_7x_8 + x_1x_3x_7 + x_1x_3x_8x_9 + x_1x_3x_8 + x_1x_3 + x_1x_4x_5x_6x_7x_8x_9 + x_1x_4x_5x_6x_7x_8 + x_1x_4x_5x_6x_7 + x_1x_4x_5x_6x_8x_9 + x_1x_4x_5x_6 + x_1x_4x_5x_7x_8x_9 + x_1x_4x_5x_8x_9 + x_1x_4x_5x_8 + x_1x_4x_5x_9 + x_1x_4x_5 + x_1x_4x_6x_7x_8x_9 + x_1x_4x_6x_7x_8 + x_1x_4x_6x_7 + x_1x_4x_6x_8x_9 + x_1x_4x_6x_9 + x_1x_4x_7x_8 + x_1x_4x_7 + x_1x_4x_8x_9 + x_1x_4 + x_1x_5x_6x_7x_8 + x_1x_5x_6x_7x_9 + x_1x_5x_6x_7 + x_1x_5x_6x_8x_9 + x_1x_5x_6x_9 + x_1x_5x_7 + x_1x_5x_8x_9 + x_1x_5x_8 + x_1x_6x_7x_8x_9 + x_1x_6x_7x_8 + x_1x_6x_7x_9 + x_1x_6x_7 + x_1x_6x_8 + x_1x_6 + x_1x_7 + x_1x_8x_9 + x_2x_3x_4x_5x_6x_7 + x_2x_3x_4x_5x_6x_9 + x_2x_3x_4x_5x_7x_8 + x_2x_3x_4x_6x_7x_8x_9 + x_2x_3x_4x_6x_8x_9 + x_2x_3x_4x_6x_8 + x_2x_3x_4x_6x_9 + x_2x_3x_4x_7x_8x_9 + x_2x_3x_4x_7x_8 + x_2x_3x_4x_7x_9 + x_2x_3x_4x_8x_9 + x_2x_3x_4x_9 + x_2x_3x_5x_7x_8x_9 + x_2x_3x_5x_7x_8 + x_2x_3x_5x_7x_9 + x_2x_3x_5x_7 + x_2x_3x_5x_8x_9 + x_2x_3x_5x_9 + x_2x_3x_6x_7x_8x_9 + x_2x_3x_6x_7x_8 + x_2x_3x_6x_7x_9 + x_2x_3x_6x_8x_9 + x_2x_3x_7x_9 + x_2x_3x_7 + x_2x_3x_8x_9 + x_2x_3x_8 + x_2x_3x_9 + x_2x_4x_5x_6x_7 + x_2x_4x_5x_6x_8x_9 + x_2x_4x_5x_6x_9 + x_2x_4x_5x_6 + x_2x_4x_5x_7x_8x_9 + x_2x_4x_5x_7x_8 + x_2x_4x_5x_8x_9 + x_2x_4x_5x_9 + x_2x_4x_6x_8x_9 + x_2x_4x_6x_8 + x_2x_4x_6x_9 + x_2x_4x_7x_8x_9 + x_2x_4x_7x_9 + x_2x_4x_7 + x_2x_4x_8x_9 + x_2x_4x_8 + x_2x_4x_9 + x_2x_4 + x_2x_5x_6x_7x_8x_9 + x_2x_5x_6x_7x_8 + x_2x_5x_6x_7 + x_2x_5x_6x_9 + x_2x_5x_6 + x_2x_5x_7x_8x_9 + x_2x_5x_7x_8 + x_2x_5x_8x_9 + x_2x_5x_8 + x_2x_5 + x_2x_6x_7x_8x_9 + x_2x_6x_7x_8 + x_2x_6x_8 + x_2x_6x_9 + x_2x_7x_8x_9 + x_2x_7x_8 + x_2x_7x_9 + x_2x_7 + x_2x_8 + x_3x_4x_5x_6x_7x_8 + x_3x_4x_5x_6x_8x_9 + x_3x_4x_5x_7x_9 + x_3x_4x_5x_8x_9 + x_3x_4x_6x_8x_9 + x_3x_4x_6x_8 + x_3x_4x_7x_8x_9 + x_3x_4x_8 + x_3x_4x_9 + x_3x_5x_6x_7x_8 + x_3x_5x_6x_7 + x_3x_5x_6x_8x_9 + x_3x_5x_7x_9 + x_3x_5x_8 + x_3x_5x_9 + x_3x_5 + x_3x_6x_7x_8x_9 + x_3x_6x_7x_8 + x_3x_6x_7 + x_3x_6x_8x_9 + x_3x_6x_9 + x_3x_6 + x_3x_7x_8x_9 + x_3x_7x_9 + x_3x_8x_9 + x_3x_8 + x_3x_9 + x_4x_5x_6x_7x_8x_9 + x_4x_5x_7x_9 + x_4x_5x_9 + x_4x_6x_7x_8x_9 + x_4x_6x_7x_8 + x_4x_6x_9 + x_4x_6 + x_4x_7x_8x_9 + x_4x_7x_8 + x_4x_7 + x_4x_9 + x_5x_7x_8x_9 + x_5x_7 + x_5x_8x_9 + x_5x_8 + x_6x_8 + x_6x_9 + x_7x_9 + 1.$$