An Improved Time-Memory-Data Trade-Off Attack against Irregularly Clocked and Filtered Keystream Generators

Lin Jiao[1,2], Mingsheng Wang[3], Bin Zhang[3], and Yongqiang Li[3]

[1] Institute of Software, Chinese Academy of Sciences, Beijing 100190, P.R. China
[2] Graduate University of Chinese Academy of Sciences, Beijing 100049, P.R. China
[3] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, P.R. China
jiaolin@is.iscas.ac.cn

Abstract. In this paper, we propose a new key recovery attack against irregularly clocked keystream generators, using the approach of time-memory-data trade-offs. The main idea behind our attack is creating several look-up tables and finally recovering the initial states of $LFSR_d$ and $LFSR_c$ synchronously, by alternatively deriving the initial states of $LFSR_d$ and $LFSR_c$ along the chains. We show that our attack is more efficient, and improves the previous attacks on the cipher model. Especially, we prove that our attack almost always needs less complexity than that of the normal time-memory-data trade-off attack [3] on the cipher model. We test our attack on LILI-128, and find out that it can successfully break the cipher with $2^{56.6}$ bit-comparison operations, 2^{49} pairs of 89-bit words memory and 2^{59} keystream bits. This result is better than those in [15,6], which possess the complexity of 2^{62} parity checks and 2^{63} bit operations respectively. Moreover, our attack can be divided and computed in parallel, and the actual runtime of the attack can be reduced depending on the number of computers we access.

Keywords: Time-Memory-Data Trade-Off Attack, Stream Cipher, Irregularly Clocked Shift Registers, LILI-128.

1 Introduction

In this paper, we present a new key recovery time-memory-data trade-off attack against ciphers based on an irregularly clocked linear feedback shift register (LFSR) filtered by a Boolean function. The cipher model we attack is composed of two components, the clock control generator and the data generator, which is shown in Fig 1.

1. The data generator subsystem consists of $LFSR_d$ of length l_d and a nonlinear filter function f_d, which outputs the bit stream v.
2. The clock control subsystem consists of $LFSR_c$ of length l_c and a clock function f_c, whose output is the clock control sequence of integers c.

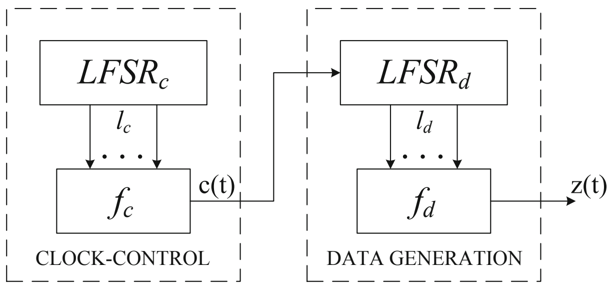

Fig. 1. The general cipher model we attack in this paper

The effect of the irregularly clocking is that v is irregularly decimated and the positions of the bits in the stream are altered. The result from this decimation is the keystream z. The secret key in this cipher is the $(l_c + l_d)$ initialization bits for $LFSR_c$ and $LFSR_d$ $(I_c,\ I_d)$. If the matching states of $LFSR_c$ and $LFSR_d$ at certain clock are obtained, it is equivalent that the key of the initialization is recovered.

The previous effective algorithms are not specially designed to attack irregularly clocked and filtered generators. In 1980, Hellman introduced a technique of time-memory trade-offs for breaking block ciphers [8]. An analogous technique for stream ciphers was proposed by Babbage in 1995 [2]. More recently, Biryukov and Shamir combined these approaches in work when the adversary has more data to deal with, which can lessen the memory and the precomputation complexity [3]. Khoo et. al. presented a time-memory-data trade-off attack against stream ciphers whose filter or combiner generators are based on Maiorana-McFarland functions [11].

Although nearly all the attacks against irregularly clocked and filtered keystream generators are correlation attacks in the past [10,14,15], we propose a time-memory-data trade-off attack on the cipher model in this paper. Our attack is not designed to attack only the data generator or the control subsystem, but especially aims at clock controlled keystream generators as one system. In most irregularly clocked cryptosystems, the length of $LFSR_c$ is usually much smaller than that of $LFSR_d$. Then by means of comparing the keystream and the output sequence which is regularly clocked by $LFSR_d$, the state of $LFSR_c$ can be determined within a complexity not too large. The attack consists of two stages. In the offline stage, several look-up tables are obtained. First, a fixed string is chosen to be a segment of keystream whose length is big enough to derive the initial state of $LFSR_c$ by comparison. Next, we create several matrices of chains with the points which are l_d-bit words formed by the possible initial states of $LFSR_d$. They are generated by alternatively determining the initial states of $LFSR_d$ and $LFSR_c$, in the way of guessing the startpoint of the chain to be the initial state of $LFSR_d$ and deriving the initial state of $LFSR_c$ by comparing the fixed string and the regularly clocked output sequence of $LFSR_d$, then running the whole system to obtain the following l_d bits in the keystream which form the next point in the chain and updating the initial state of $LFSR_d$ with this point, repeatedly. We store the startpoints and endpoints of these chains in several tables corresponding to the matrices. In the online stage, we search the tables and recover the key. We observe the keystream to find the fixed string, and the following l_d bits in the keystream indeed form a point in the chains. Then we search for a match among the endpoints in the way beforehand along the chains and determine the initial states synchronously. Actually, we propose a general framework to attack the cipher model, and all the effective attacks against the subsystems [4,13,9,5,16,17] can be applied depending on the specific ciphers.

The complexity of our attack is analyzed. We prove that the runtime of our attack is less than that of the original time-memory-data trade-off attack proposed by Biryukov and Shamir on the cipher model in case of the same memory requirement. Especially, to attack the LILI-128 cipher, our method needs $2^{56.6}$ bit-comparison operations, with 2^{49} pairs of 89-bit words memory and 2^{59} keystream bits, whose success probability is more than 98%. There is also a time-memory trade-off attack against LILI-128 proposed by Saarinen [18] using the period of $LFSR_c$, which needs approximately 2^{45} 89-bit words of computer memory and 2^{46} keystream bits, with the success probability of 90%. The runtime complexity is claimed to be 2^{48} DES operations, which is not easy to be compared with the general runtime complexity. The algebraic attack proposed by Courtois and Meier [6] aims at the data generator subsystem of LILI-128, and calls for 2^{63} bit operations, with 2^{42} bit memory and 2^{57} keystream bits to break the whole cipher. The success probability of algebraic attacks is hard and scarcely analyzed in literatures so far. A correlation attack proposed by Molland and Helleseth was presented in [15], which just determines the initial state of $LFSR_c$. The attack is in the runtime of 2^{62} parity checks, with 2^{46} bit memory and 2^{29} keystream bits. In fact, the efficiency of our attack depends on the memory size. Besides, if there are 2^{10} or more(at most 2^{20}) computers to parallel compute the algorithm, the runtime of the attack can be reduced below $2^{46.6}$(accordingly $2^{36.6}$) bit-comparison operations.

The rest of this paper is organized as follows. In Section 2, we give the general model we attack. In Section 3, we present the attack in details. In Section 4, we apply the attack to LILI-128. Conclusion is given in Section 5.

2 General Model

Here we present some details of the general model for irregularly clocked and filtered stream ciphers we attack in this paper.

Let $g_d(x)$ and $g_c(x)$ be the feedback polynomials for the shift registers $LFSR_d$ of length l_d and $LFSR_c$ of length l_c. The initialization states (I_c, I_d) define the secret key for the given cipher system. From $g_c(x)$ we can calculate a clock control sequence c in the following way. Let $c(t) = f_c(L_c^t(I_c)) \in \{a_1, a_2, \ldots, a_A\}, a_j \geq 0$, be a function where the input $L_c^t(I_c)$ is the inner state of $LFSR_c$ after t feedback shifts and A is the number of values that $c(t)$ can take. $LFSR_d$ produces the stream which is filtered by f_d. The output from f_d is $v_k = f_d(L_d^t(I_d))$. The clock

$c(t)$ decides how many times $LFSR_d$ is clocked before the output bit v_k is taken as keystream bit $z(t)$. Thus the keystream $z(t)$ is produced by $z(t) = v_{k(t)}$, where $k(t)$ is the total sum of the clock at time t, that is $k(t) \leftarrow k(t-1) + c(t)$.

The f_c function described above can be those in the shrinking generator, the step-1/step-2 generator, the stop and go generator and so on, in this model.

3 Attack

3.1 Cryptanalytic Time-Memory-Data Trade-Offs for Stream Ciphers

At first, we give an introduction to the approach of time-memory-data trade-offs by briefly citing contents of [3]. There are five key parameters:

- N represents the size of the search space.
- P represents the time required by the preprocessing phase of the attack.
- M represents the amount of random access memory available to the attacker.
- T represents the time required by the realtime phase of the attack.
- D represents the amount of realtime data available to the attacker.

The origin of the attack against stream ciphers is Hellman's time-memory trade-off attack against block ciphers, which considers the random function f that maps the key x to the ciphertext block y for some fixed chosen plaintext, where f is easy to evaluate but hard to invert. Hellman uses a preprocessing stage which tries to cover all the N points of the preimage space with an $m \times t$ matrix whose rows are the chains obtained by iterating the function f t times on m randomly chosen startpoints. The pairs of startpoints and endpoints are stored. During the actual attack, we are given a value y and asked to find its predecessor x under f. Since x is covered by one of the precomputed chains, the algorithm repeatedly applies f to y until it reaches the stored endpoint, then jumps to its associated startpoint, and repeatedly applies f to the startpoint until it reaches y again. The previous point it visits is the desired x. The matrix in Hellman's attack is shown in Fig 2.

A single matrix cannot efficiently cover all the N points, thus we add more rows to the matrix. Assume that the first m chains are all disjoint, and the additional path contains t distinct points, where t is less than $\sqrt{N}$. By the birthday paradox, the two sets are likely to be disjoint as long as $t \times mt \leq N$, and thus we choose m and t that satisfy the relationship $mt^2 = N$, which we call the matrix stopping rule. Thus, the waste of repetitive coverage can be reduced. A single $m \times t$ matrix covers only a fraction of $mt/N = 1/t$ of the space according to the matrix stopping rule, and thus we need at least t unrelated matrices to cover the whole space. Hellman's method is using variants f_is of the original f defined by $f_i(x) = h_i(f(x))$, where h_i is some simple output modification. The total precomputation requires $P \approx N$ time, since we have to cover the space with the precomputed chains. The total memory requires to store mt pairs of startpoints and endpoints of the chains in the t matrices. We

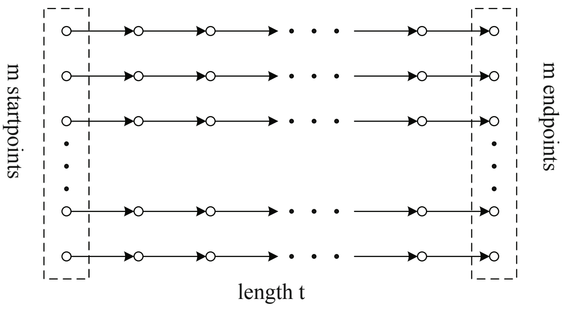

Fig. 2. Hellman's Matrix

have to perform t inversion attempts according to every matrix, each requiring at most t evaluations of some f_i to recover x. Thus the total time complexity of the attack is t^2.

Differently, in the stream ciphers all the given output vectors can be inverted with respect to the same function by using the same precomputed table. The attack is successful if any of the D given output vectors is found, since under such condition the generator can be run forward to find the initial state. Thus, Biryukov and Shamir use a large value of D to speed up the attack. Same basic approaches are used in this attack as those in Hellman's. We reduce the total number of points to be covered from N to N/D, and still get a collision between the stored and actual states. The attack reduces the number of matrices r from t to t/D in order to decrease the total coverage by a factor of D and keep each matrix as large as possible, which is allowed by the matrix stopping rule $mt^2 = N$. Then the total memory is reduced from $M = mt$ to $M = mt/D$. The total preprocessing time is similarly reduced from $P = N$ to $P = N/D$. The attack time T is the product of the number of matrices, the length of each chain, and the number of available data points. This product is $T = (t/D) \cdot t \cdot D = t^2$, which is the same as in Hellman's original attack. We can use the matrix stopping rule to find the time-memory-data trade-off in this attack, which satisfies the invariant relationship:

$$TM^2D^2 = t^2 \cdot (mt/D)^2 \cdot D^2 = (mt^2)^2 = N^2.$$

This relationship is valid for any $t \geq D$, i.e., $D^2 \leq T \leq N$.

3.2 Our Attack against Irregularly Clocked and Filtered Keystream Generators

Our time-memory-data trade-off attack is based on the aforementioned cryptanalysis, but specially designed to analyze irregularly clocked and filtered keystream generators described in Section 2. We usually see $l_c < l_d$ in most clock controlled cryptosystems. The initial state of $LFSR_c$ can be determined within a short runtime by comparing the keystream z and the regular output sequence

v. The attack has two phases: During the preprocessing phase, we explore the general structure of the cryptosystem, and summarize the findings in a large table, which are not tied to particular keys. During the realtime phase, we are given actual data produced from a particular unknown key, and our goal is to use the precomputed table to find the key as quickly as possible.

Procedures in details are as follows. The representations are defined the same as in the previous subsection.

- Preprocessing Phase:
  1. Choose a fixed string $s \in GF(2)^l$ (as a segment of keystream for determining the initial state of $LFSR_c$).
  2. Randomly seed an l_d-bit maximum LFSR and generate a sequence of distinct l_d-bit vectors X_1,X_2,...,X_r, which is similar to that described in (Khoo et. al.,2007) for reducing the cross points of the chains as h_is in Hellman's method.
  3. Form r number of $m \times t$ matrices that try to cover $1/d$ of the whole search space which is composed of all the possible initial states of $LFSR_d$ as follows. For matrix i, $i = 1, 2, \ldots, r$:

     (a) Randomly choose m startpoints of the chains, each point formed by a vector of length of l_d which is to be an injection into $LFSR_d$ as initialization $\widehat{I_d}$.

     (b) Regularly clock $LFSR_d$ and obtain the output string v. Compare v with s, through the method of matching a fixed bit of s with consecutive A bits of v, to filter out the impossible values of the control sequence c and determine the correspondingly initial state of $LFSR_c$. By further cutting down the possible initial states of $LFSR_c$ through the check of several more keystream bits in s, the unique solution of the initial state of $LFSR_c$ $\widehat{I_c}$ can be derived. Thus, the overall system is obtained. Operate the system and generate the following l_d keystream bits from this moment on. Exclusive-OR the l_d-bit vector with X_i and make it the next point in the chain. Update the initial state of $LFSR_d$ with this point. Suppose that the runtime of this step is about t_0.

     (c) Iterate Step (b) t times on each startpoint respectively.

     (d) Store the pairs of startpoints and endpoints (SP_j,EP_j), $j = 1, \ldots, m$ in table i.

     The relationship among the integers t, m, d, r is $mtr \geq 2^{l_d}/d$, and $mt^2 = 2^{l_d}$ according to the matrix stopping rule.
- Realtime Phase:
  1. Observe the keystream and find d number of l-bit strings matching with string s. For one such string, let the following l_d bits in the keystream be y.
  2. For each y, search among the endpoints in overall tables in this way. For table i, check if there is EP_j, $j = 1,\ldots,m$ matching with $y \oplus X_i$ first. If not, iterate Step (b) w times on $y \oplus X_i$ until it matches with one

of EP_j, $j = 1,\ldots,m$, where $w = 1,\ldots,t$. When there is a match, jump to the corresponding startpoint, and repeatedly apply Step (b) to the startpoint until the l_d-bit keystream vector reaches y again. Then the previous point visited is the initial state of $LFSR_d$ I_d, and the state $\widehat{I_c}$ just figured out in Step (b) is the initial state of $LFSR_c$ I_c.

Firstly, we consider the complexity of the attack. Analogously to the last subsection, the whole search space is composed of all the possible initial states of $LFSR_d$, whose cardinality is 2^{l_d}. Our tables only need to cover $1/d$ of the space because we just need to break the cipher for one out of the d strings in the keystream. The memory is $M = m \cdot r$, since we only store the startpoints and endpoints of all the chains in the matrices. For each of the d data, we need to compute for r tables and calculate Step (b) at most t times for each table. The time taken is $T = t \cdot r \cdot d \cdot t_0$. Moreover, we need to sample $D = d \cdot 2^l$ consecutive keystream bits to collect the required d strings, because the string s of length l occurs on average once in 2^l keystream bits . The preprocessing time for building the tables is $P = m \cdot t \cdot r \cdot t_0$.

Secondly, we compare our attack with the normal time-memory-data trade-off attack proposed by Biryukov and Shamir, which is introduced in the last subsection. Based on the same irregularly clocked and filtered keystream generator, the whole search space of the normal attack is made up of all the combinations of the possible initial states of $LFSR_d$ and $LFSR_c$. We prove that our attack outperforms the normal one on the cipher model at runtime, with regard to the same success probability and available memory which is in consideration of feasibility.

Theorem 1. *Let the representations for variables be the same as stated above, and mark the normal attack and ours with subscript 1 and subscript 2 respectively. We have* $m_1t_1r_1 = 2^{l_d+l_c}/d_1$, $M_1 = m_1r_1$, $T_1 = t_1r_1d_1$, *and* $m_2t_2r_2 = 2^{l_d}/d_2$, $M_2 = m_2r_2$, $T_2 = t_2r_2d_2t_0$. *Then we derive that*

$$T_2 < T_1 \text{ if and only if } t_0 < 2^{l_c}$$

when $M_1 = M_2$ *and* $r_1 = r_2$.

Proof. We easily have that

$$\frac{t_1}{t_2} = 2^{l_c}\frac{d_2}{d_1}$$

from the conditions. Then

$$\frac{T_1}{T_2} = \frac{t_1d_1}{t_2d_2}\frac{r_1}{r_2t_0} = 2^{l_c}\frac{r_1}{r_2t_0}.$$

We have $\frac{T_1}{T_2} > 1$ if and only if

$$t_0 < 2^{l_c}\frac{r_1}{r_2}.$$

When $r_1 = r_2$, it is equivalent to $t_0 < 2^{l_c}$, which means that our attack takes less runtime if and only if Step (b) is better than exhaustive search. □

Moreover, we have $D_1 = d_1$, $D_2 = d_2 d_0$ where d_0 denotes the extra data for matching with string s, and $m_1 t_1^2 = 2^{l_d + l_c}$, $m_2 t_2^2 = 2^{l_d}$ according to the matrix stopping rule. Then we derive

$$\frac{m_1}{m_2} \frac{t_1}{t_2}^2 = 2^{l_c}.$$

Substitute $\frac{t_1}{t_2} = 2^{l_c} \frac{d_2}{d_1}$ and $\frac{m_1}{m_2} = \frac{r_2}{r_1}$ into the equation, and have

$$d_2 = 2^{-l_c/2} d_1 \sqrt{\frac{r_1}{r_2}}.$$

We may as well let $d_0 = 2^{l_c}$ like the example of LILI-128, and derive

$$D_2 = D1 \cdot 2^{l_c/2} \sqrt{\frac{r_1}{r_2}},$$

which is more than D_1. It seems that we use data to exchange for time, whereas the runtime of the normal attack is always too long to make the attack practical. We calculate

$$\triangle T = \frac{T_1}{T_2} = \frac{2^{l_c}}{t_0} \text{ and } \triangle D = \frac{D_2}{D_1} = 2^{l_c/2}$$

when $r_1 = r_2$, and find out $\triangle T > \triangle D$ if $t_0 < 2^{l_c/2}$. In our attack against LILI-128, we see $\triangle T$ is much more than $\triangle D$. In general, our attack no more treats the clock controlled cipher as a black box, and creates the chains only consisting the possible states of the data LFSR as the corresponding states of the clock control LFSR are implicitly tackled based on the internal structure of the cipher. Moreover, the choice of d is more flexible, since there is not restriction of $d < t$ like that in the normal attack.

Thirdly, we compare our attack with other attacks. Unlike those attacks which just can be computed as a whole and cannot be divided and conquered, such as correlation attacks and algebraic attacks, our attack can be parallel computed by at most d computers, each using one of the d strings. Then the actual runtime can be reduced to $t \cdot r \cdot t_0 \cdot d/u$, where u is the number of computers we can use. Actually, this is a general framework to analyze irregularly clocked and filtered keystream generators, and all the previous effective attacks against the subsystems can be applied, such as in Step (b), depending on the specific ciphers. Moreover, for certain ciphers conforming to the model, if there is an attack against the data generator subsystem with a short runtime and a small data requirement, the whole search space can change into the set of all the possible initial states of $LFSR_c$ instead and our attack can be still valid. That is to say, our attack may improve the previous attacks on the cipher model.

Finally, we analyze the success probability of our attack. The fundamental problem is that points can appear within more than one chain. Therefore, estimating the probability of success is equivalent to estimating the probability of such duplication. We use the classic occupancy problem to estimate the success probability, which is described nicely by Feller [7]. We apply the model of throwing mtr balls into $2^{l_d}/d$ urns, where balls and urns correspond to the points in the matrices and the keys to be covered in the search space respectively. The ratio of the expected number of urns that have at least one ball and the number of the urns is the success probability. Then, approximately,

$$Pr(success) = 1 - e^{-\frac{mtrd}{2^{l_d}}}.$$

The probabilities for various choices of $mtrd$ are given in Table 1. Although the attack is only probabilistic, the probability of success is high.

Table 1. Approximate success probability

$mtrd$	0	2^{l_d-5}	2^{l_d-4}	2^{l_d-3}	2^{l_d-2}	2^{l_d-1}	2^{l_d}	2^{l_d+1}	2^{l_d+2}	2^{l_d+3}	∞
$Pr(success)$	0	0.03	0.06	0.12	0.22	0.39	0.63	0.86	0.98	0.99	1.00

4 Application

The LILI-128 cipher [19] is based on the general model we attack in this paper. To explain our new attack, we have exemplified it on this cipher.

4.1 The LILI-128 Cipher

LILI-128 is a stream cipher proposed by Simpson et al. for NESSIE. It is comprised of a 39-bit linear feedback shift register $LFSR_c$ responsible for clock control and an 89-bit $LFSR_d$ to generate the keystream. The 128-bit secret key is substituted into $LFSR_c$ and $LFSR_d$ respectively as the initial states. The feedback polynomial of $LFSR_c$ which is a primitive polynomial is

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

After clocking $LFSR_c$ once, 2 bits from $LFSR_c$ are input to

$$f_c(x_{12}, x_{20}) = 2x_{12} + x_{20} + 1$$

to output $c(t) \in \{1, 2, 3, 4\}$. The feedback polynomial of $LFSR_d$ which is a primitive polynomial is

$$g_d(x) = x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1.$$

After clocking $LFSR_d$ $c(t)$ times, 10 bits from $LFSR_d$ are input to
$f_d(x_0, x_1, x_3, x_7, x_{12}, x_{20}, x_{30}, x_{44}, x_{65}, x_{80}) = x_{13}+x_8+x_4+x_2+x_{81}x_{21}+x_{81}x_8+ x_{66}x_4 + x_{66}x_1 + x_{45}x_2 + x_{45}x_1 + x_{31}x_{21} + x_{81}x_{66}x_{13} + x_{81}x_{66}x_8 + x_{81}x_{66}x_4 + x_{81}x_{66}x_2+x_{81}x_{45}x_8+x_{81}x_{45}x_4+x_{81}x_{31}x_{21}+x_{81}x_{31}x_{13}+x_{81}x_{31}x_8+x_{66}x_{45}x_{21}+ x_{66}x_{45}x_4 + x_{66}x_{31}x_{21} + x_{66}x_{31}x_8 + x_{66}x_{31}x_4 + x_{81}x_{66}x_{45}x_{21} + x_{81}x_{66}x_{45}x_8 +$

$x_{81}x_{66}x_{45}x_4+x_{81}x_{66}x_{45}x_1+x_{81}x_{66}x_{31}x_{21}+x_{81}x_{66}x_{31}x_8+x_{81}x_{66}x_{31}x_2+x_{81}x_{45}x_{31}$ $x_{13}+x_{81}x_{45}x_{31}x_4+x_{66}x_{45}x_{31}x_8+x_{66}x_{45}x_{31}x_2+x_{66}x_{31}x_{21}x_{13}+x_{66}x_{31}x_{21}x_8+$ $x_{81}x_{66}x_{45}x_{31}x_8+x_{81}x_{66}x_{45}x_{31}x_4+x_{81}x_{66}x_{31}x_{21}x_{13}+x_{81}x_{66}x_{31}x_{21}x_8+x_{66}x_{45}x_{31}$ $x_{21}x_{13}+x_{66}x_{45}x_{31}x_{21}x_8+x_{81}x_{66}x_{45}x_{31}x_{21}x_{13}+x_{81}x_{66}x_{45}x_{31}x_{21}x_8$,
which is balanced and with the nonlinearity of 480, to output binary data $z(t)$ used as the keystream.

4.2 Our Attack against LILI-128

4.2.1 Step (b) In Our Attack against LILI-128

At first, we present the detailed process in Step (b) in our attack against LILI-128. Assume that we have obtained 39 keystream bits $z(t), t = 1, \dots, 39$ and the initial state of $LFSR_d$ right now. Our goal is to derive the unique solution of the initial state of $LFSR_c$. The method is the same as that mentioned in [20].

(b.1) Narrow down the output sequence that $c(t), t = 1, \dots, 8$ can take, by comparing the keystream $z(t), t = 1, \dots, 8$ with the output sequence $v(t)$ regularly clocked by $LFSR_d$. For example, let $z(t) = \{1, 0, 0, 1, 1, 0, 1, 0, \dots\}$ and $v(t) = \{1, 0, 1, 0, 0, 0, 1, 1, \dots\}$. First, we check all the four values of $c(1)$ to see if $z(1) = v(c(1))$ holds, and find

$$\begin{aligned} z(1) &= v(1), \text{ when } c(1) = 1 \\ z(1) &\neq v(2), \text{ when } c(1) = 2 \\ z(1) &= v(3), \text{ when } c(1) = 3 \\ z(1) &\neq v(4), \text{ when } c(1) = 4. \end{aligned}$$

Thus, 1 or 3 is the possible value of $c(1)$. Next, we check that what value $c(2)$ takes can make $z(2) = v(c(1) + c(2))$ tenable for either of these two possible values of $c(1)$, 1 or 3. Repeat until $t = 8$. Compute 16 bits of $(x_{12}(t), x_{20}(t))$ in the state of $LFSR_c$ where $t = 1, \dots, 8$, using the bijective function f_c, i.e,

$$\begin{aligned} (x_{12}(1), x_{20}(1)) &= (0, 0), \text{ when } c(1) = 1 \\ (x_{12}(1), x_{20}(1)) &= (0, 1), \text{ when } c(1) = 2 \\ (x_{12}(1), x_{20}(1)) &= (1, 0), \text{ when } c(1) = 3 \\ (x_{12}(1), x_{20}(1)) &= (1, 1), \text{ when } c(1) = 4 \end{aligned}$$

and the sequences of $c(t), t = 1, \dots, 8$ narrowed down above. The sketch is given in Fig 3.

(b.2) Narrow down the output sequence candidates of $c(t), t = 9, \dots, 31$ to compute the remaining indeterminate 23 bits in the state of $LFSR_c$. It is done in the same way as that described in the preceding step. Since $x_{12}(t), t = 9, \dots, 31$ is already known for $x_{12}(t) = x_{20}(t-8), t = 9, \dots, 31$, only one bit $x_{20}(t)$ of $(x_{12}(t), x_{20}(t))$ can be determined per clock, for

$t = 9, \ldots, 31$. Repeat this series of computation until t becomes 31, then consecutive 39 state bits of $LFSR_c$ are determined, and the initial state of $LFSR_c$ is obtained by running forward.

(b.3) Compare the output sequence of the whole system based on those 39-bit candidates for the initial state of $LFSR_c$ narrowed down in Step (b.2) with the keystream $z(t), t = 31, \ldots, 39$ to see if the candidates are correct. Whenever a comparison of 1 bit is made, those candidates are reduced by one half in number . When $t = 39$, only one candidate remains, and it is just the unique solution of the initial state of $LFSR_c$. (Specific illustrates are given in the following analysis of the complexity of Step (b)).

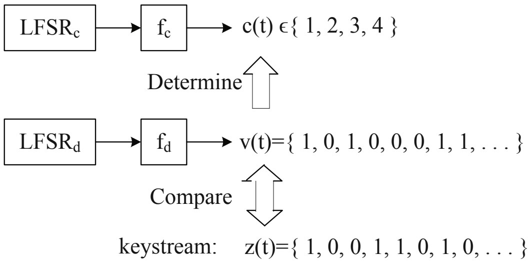
LFSRc
fc
c(t) ϵ{ 1, 2, 3, 4 }
Determine
LFSRd
fd
v(t)={ 1, 0, 1, 0, 0, 0, 1, 1, . . . }
Compare
keystream:
z(t)={ 1, 0, 0, 1, 1, 0, 1, 0, . . . }

Fig. 3. Sketch for comparing and determining

Next, we discuss the amount of computation needed for Step (b). The basic operation is bit comparison. Assume that the keystream behaves as random numbers, since the keystream is filtered by a balanced Boolean function from an LFSR based on a primitive polynomial, and the values of 0 and 1 are equally output in the sequence of $LFSR_d$ and the keystream of the whole system.

It is thought that comparing one bit of the obtained keystream with the output sequence regularly clocked by $LFSR_d$ allows narrowing down the possible values of $c(t)$ by an average of one half. Concretely, when $t = 1$, we check the four candidate values of $c(1)$ and narrow down them to two. The number of trails is four, and the number of remaining candidates of $c(1)$ is two. When $t = 2$, we check the four candidate values of $c(2)$, for either of these two possible values of $c(1)$. The number of trials totals 2×4, and the number of remaining candidates for $(c(1), c(2))$ is 2^2. The number of trials and the number of remaining candidates for $(c(1), c(2), \ldots, c(t))$ are analyzed in this way until $t = 8$ in Step (b.1). The amount of computation at t-th clock is $4\times$ (the number of remaining candidates of $(c(1), c(2), \ldots, c(t-1))$ at $(t-1)$-th clock), and the number of remaining candidates of $(c(1), c(2), \ldots, c(t))$ is one half of the computation above, for $t = 1, \ldots, 8$. Consequently, the total number of trials for $c(t), t = 1, \ldots, 8$ in Step (b.1) is obtained as follows.

$$4 + 2 \times 4 + 2^2 \times 4 + \cdots + 2^7 \times 4 \approx 2^{10.0}$$

And the number of remaining candidates of $(c(1), c(2), \ldots, c(8))$ at 8th clock is about 2^8.

In Step (b.2), when $t = 9$, only the value of $x_{20}(9)$ remains indeterminate since $x_{12}(9)$ is fixed. For f_c is invertible, there are two possible values of $c(9)$. They are narrowed down by one half through one bit comparison of $z(9)$ and $v(c(1) + ... + c(9))$. The number of trials made in this fragment is $2^8 \times 2$, and the number of possible values of $(c(1), c(2), \ldots, c(9))$ stays 2^8. The number of trials and the number of remaining candidates for $(c(1), c(2), \ldots, c(t))$ are analyzed in the same way until $t = 31$. The amount of computation at t-th clock keeps $2^8 \times 2$, and the number of remaining candidates for $(c(1), c(2), \ldots, c(t))$ is 2^8 all along for $t = 9, \ldots, 31$. Thus, the total number of trials for $c(t), t = 9, \ldots, 31$ in Step (b.2) is as follows.

$$2^8 \times 2 \times (31 - 9 + 1) \approx 2^{13.6}$$

And the number of remaining candidate sequences of $(c(1), c(2), \ldots, c(31))$ at 31st clock stays about 2^8.

Since there is a one-to-one correspondence between $c(t)$ and $(x_{12}(t), x_{20}(t))$, by the analysis of Step (b.1) and (b.2), there are 2^8 candidates for the initial state of $LFSR_c$ narrowed down. We check to find a match of 8 bits in the following output sequences from the overall system based on those candidates with the remaining 8 bits of the obtained keystream. As far as one of the 8 bits is concerned, the candidates are narrowed down by one half. Thus, the total number of trials made in Step (b.3) is

$$2^8 + 2^7 + \cdots + 2 \approx 2^{9.0}.$$

At 39-th clock, only one candidate stays. Thus the overall system is obtained. Operate the cipher and generate the following 89 keystream bits. Exclusive-OR them with X_i and let the sum form the next point in the chain.

Consequently, the total amount of computation in Step (b) becomes

$$t_0 = 2^{10.0} + 2^{13.6} + 2^{9.0} \approx 2^{13.8}.$$

4.2.2 The Complete Attack against LILI-128

Given Step (b) stated above, we present our complete attack against LILI-128. In the preprocessing phase, we choose a fixed string $s \in GF(2)^{39}$ as the obtained keystream segment, and generate a sequence of distinct 89-bit vectors $X_1, X_2, \ldots, X_d$ as stated in Step 2. Next, we form r number of $m \times t$ matrices to cover the whole search space of $\mathbb{F}_2^{89}$. For every matrix, we randomly choose m startpoints, each formed by a possible initial state of $LFSR_d$ $\widehat{I_d}$. Then regularly clock the $LFSR_d$ and execute Step (b) in part 4.2.1. Hereto, only one solution of the initial state of $LFSR_c$ $\widehat{I_c}$ remains. Thus the overall system is obtained. We generate 89 more keystream bits and X-or them with X_i corresponding to the matrix to form the next point in the chain. It is iterated t times on each startpoint. Store the pairs of startpoints and endpoints (SP_j, EP_j). The integers t, m, d, r satisfy the relationship $mtr \geq 2^{89}/d$, and $mt^2 = 2^{89}$ according to the matrix stopping rule,.

In the realtime phase, we observe the keystream to find d number of 39-bit strings matching with s, and let the following 89 bits in the keystream be y. Check a match of $y \bigoplus X_i$ and a match of the result of impacting Step (b) on $y \bigoplus X_i$ at most t times, with the endpoints in the tables. When there is a match, we jump to the associated startpoint, and repeatedly apply Step (b) to the startpoint until it reaches $y \bigoplus X_i$ again. Then the previous point it has visited is the initial state I_d of $LFSR_d$, and the corresponding initial state of $LFSR_c$ just figured out in Step (b) is I_c. Thereby, the key of LILI-128 is recovered.

4.2.3 Techniques to Improve Our Attack against LILI-128

Many techniques can be employed to enhance our attack, since what we provide is a general framework on clock controlled keystream generators. For example, the skills to analyze the filter generator which is a subsystem of the cipher model can be used to improve our attack according to specific cryptosystems. Besides, our attack is a method based on time-memory-data trade-offs. Then the skills to reduce the cross points can be used to increase the success probability of our attack as well. There are a great many literatures for reference, such as "false alarms" provided in [1], "rainbow chains" provided in [12], and "special point" provided in [3]. In addition, we can exploit hash functions on the pairs of startpoints and endpoints in order to reduce the memory complexity.

Especially, for our attack against LILI-128, we still use the method of time-memory trade-offs to reduce the amount of computation in Step (b). The basic computation in Step (b) is the comparison of $v(t)$ and $z(t)$ at every clock which is for narrowing down the candidates for $LFSR_c$ data. If the steps for such comparison are computed and stored as a table, we can look it up to determine the candidates for $LFSR_c$ data directly in the online stage (The specific way is mentioned in [20] which is similar to the discussion on the amount of computation given in part 4.2.1). Table 2 shows the frequency of clocks versus the memory size required to hold the computed tables and the frequency of table-lookups that is needed in Step (b.1) and Step (b.2). The frequency of table-lookups is nothing but the runtime in Step (b.1) and Step (b.2). However, the amount of computation in Step (b.3) cannot be reduced by means of providing tables to look up. The complexity of our attack that can be reduced depends on the size of extra memory M'. In this paper, we estimate the amount of computation in Step (b) when the memory size is 2^{42} bits or less. We will see that the extra memory can hardly affect the order of the memory size of the whole attack against LILI-128 in the next part. Thus Step (b) takes

$$t_0 = 2^{4.6} + 2^{10.0} + 2^{9.0} \approx 2^{10.6}.$$

4.2.4 Complexity and Success Probability of Our Attack against LILI-128

In this part, the complexity of our attack is considered. We discuss the relationship among the memory size, the amount of computation and the data required for our attack. Different parameters for the complexity and success probability

Table 2. Diminished computation of Step (b.1) and Step (b.2)

Frequency of clocks	Memory	Step (b.1)
-	0	$2^{10.0}$
1	2^{7}	$2^{8.0}$
2	2^{14}	$2^{6.4}$
4	2^{27}	$2^{4.6}$
8	2^{52}	1

Frequency of clocks	Memory	Step (b.2)
-	0	$2^{13.6}$
1	2^{7}	$2^{12.6}$
2	2^{14}	$2^{11.6}$
3	2^{21}	$2^{11.0}$
4	2^{28}	$2^{10.6}$
6	2^{42}	$2^{10.0}$
8	2^{56}	$2^{9.6}$

are presented. We also give a comparison of our time-memory-data trade-off attack and the normal one proposed by Biryukov and Shamir, and a comparison of our attack and the previous attacks against LILI-128.

The whole search space of our attack is composed of all the possible initial states of $LFSR_d$, whose cardinality is 2^{89}. Our tables need to cover $1/d$ of the space. The memory is $M = mr + M'$. For each of the d data, we need to look up r tables and compute Step (b) at most t times. The time taken is $T = trdt_0$, where t_0 is $2^{10.6}$. The number of consecutive keystream bits needed to collect the required d strings is $D = d \cdot 2^{39}$. The preprocessing time for building the tables is $P = mtrt_0$. As stated above, we know the success probability of our attack which is approximately

$$Pr(success) = 1 - e^{-\frac{mtrd}{2^{89}}}.$$

As shown in Table 3, we have calculated some versions of the parameters of m, t, r, d versus the success probability, which satisfy the relationship

$$mt^2 = 2^{89},\ mtr \geq 2^{89}/d.$$

From the table, we see that the extra memory M' required as stated in part 4.2.3 can hardly affect the memory size of the whole attack.

We compare our attack with the normal time-memory-data trade-off attack against LILI-128. For clarification, we only list the case with the same success probability. Moreover, for the sake of hardware requirement, we set the memory of these two attacks in the same size. For the normal attack against LILI-128, the whole search space is formed by all the 2^{128} combinations of the possible initial states of $LFSR_d$ and $LFSR_c$. For the success probability of 63%, the time, memory and data complexity of the normal attack should satisfy the relationship below

$$TM^2D^2 = 2^{2 \cdot 128} \text{ for } D^2 \leq T \leq N$$

Table 3. Different versions of parameters of our attack with the corresponding complexity and success probability

t	r	m	d	T	M	D	P	$Pr(success)$
2^{20}	1	2^{49}	2^{20}	$2^{50.6}$	2^{49}	2^{59}	$2^{79.6}$	0.63
2^{22}	2^3	2^{45}	2^{20}	$2^{55.6}$	2^{48}	2^{59}	$2^{80.6}$	0.86
2^{22}	2^4	2^{45}	2^{20}	$2^{56.6}$	2^{49}	2^{59}	$2^{81.6}$	0.98
2^{23}	2^6	2^{43}	2^{20}	$2^{59.6}$	2^{49}	2^{59}	$2^{82.6}$	0.99

according to the matrix stopping rule. A comparison of parameters of these two attacks are presented in Table 4, where the parameters in the normal attack is calculated as follows

$$M = mt/D, T = t^2, P = N/D, t \geq D.$$

From the table, we find out that the ratio $\Delta D = \frac{D_2}{D_1} = 2^{19}$ is much smaller than $\Delta T = \frac{T_1}{T_2} = 2^{27.4}$, which conforms to the statement in the last section. We also see that the runtime of the normal attack is too long for feasibility.

Table 4. Complexity comparison with the normal time-memory-data trade-off attack against LILI-128

	t	r	m	d	T	M	D	P
normal	2^{39}	1	2^{50}	–	2^{78}	2^{50}	2^{39}	2^{89}
improved	2^{20}	2	2^{49}	2^{19}	$2^{50.6}$	2^{50}	2^{58}	$2^{80.6}$

We also compare our attack with the previous attacks against LILI-128. The amount of memory, computation, keystream needed, and the according success probability are shown in Table 5. The total amount of computation in our attack against LILI-128 is

$$T = 2^{22} \times 2^4 \times 2^{20} \times 2^{10.6} = 2^{56.6}.$$

The memory calls for 2^{49} pairs of 89-bit words. The data requires 2^{59} keystream bits. The success probability of our attack is more than 0.98. Our attack can easily be parallelized and distributed among processors, since there is no need for communication between the processors, unlike those attacks which just can be computed as a whole.

Table 5. Complexity comparison with the previous attacks against LILI-128

	Time (T)	Data (D)	Memory (M)	Success probability (Pr(success))
Our method	$2^{56.6}$ bit-comparisons	2^{59} bits	2^{49} pairs of 89-bit words	0.98
Algebraic Attack [6]	2^{63} bit operations	2^{57} bits	2^{42} bits	hard to estimate
TMTO [18]	2^{48} DES operations	2^{46} bits	2^{45} 89-bit words	0.90
Correlation Attack [15]	2^{62} bit parity checks	2^{29} bits	2^{46} bits	0.99

5 Conclusion

A new key recovery time-memory-data trade-off attack against irregularly clocked keystream generators is proposed in this paper. It is a method of low runtime, which is feasible to compute in parallel. Moreover, we attack LILI-128 for example to illustrate our attack. Since the attack is a frame, it can be combined with other efficient approaches to analyze this kind of ciphers, such as Grain, and we believe that it can improve the previous attacks on the cipher model.

Acknowledgements. We would like to thank anonymous referees for their helpful comments and suggestions. The research presented in this paper is supported by the National Natural Science Foundation of China (Grant No. 60833008, No.60603018, No.60970134), the "Strategic Priority Research Program" of the Chinese Academy of Sciences (Grant No. XDA06010701) and the IIE's Cryptography Research Project (Grant No. Y2Z0011102).

References

1. Avoine, G., Junod, P., Oechslin, P.: Time-memory trade-offs: False alarm detection using checkpoints. In: Maitra, S., Veni Madhavan, C.E., Venkatesan, R. (eds.) INDOCRYPT 2005. LNCS, vol. 3797, pp. 183–196. Springer, Heidelberg (2005)
2. Babbage, S.: Improved exhaustive search attacks on stream ciphers. In: European Convention on Security and Detection 1995, IEE Conference Publication, pp. 161–166 (1995)
3. Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)
4. Chepyzhov, V.V., Johansson, T., Smeets, B.: A simple algorithm for fast correlation attacks on stream ciphers. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 181–195. Springer, Heidelberg (2001)

5. Courtois, N.T.: Fast algebraic attacks on stream ciphers with linear feedback. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 176–194. Springer, Heidelberg (2003)
6. 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)
7. Grubbs, F.E.: An introduction to probability theory and its applications. Technometrics 9(2), 342 (1967)
8. Hellman, M.: A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory 26, 401–406 (1980)
9. Johansson, T., Jonsson, F.: Theoretical analysis of a correlation attack based on convolutional codes. IEEE Transactions on Information Theory 48, 2173–2181 (2002)
10. Jonsson, F., Johansson, T.: A fast correlation attack on LILI-128. Information Processing Letters 81(3), 127–132 (2002)
11. Khoo, K., Chew, G., Gong, G., Lee, H.K.: Time-memory-data trade-off attack on stream ciphers based on Maiorana-McFarland functions. IEICE Transactions 92A(1), 11–21 (2009)
12. Khoo, K., Gong, G., Lee, H.-K.: The rainbow attack on stream ciphers based on Maiorana-McFarland functions. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 194–209. Springer, Heidelberg (2006)
13. Meier, W., Staffelbach, O.: Fast correlation attacks on stream ciphers. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 301–314. Springer, Heidelberg (1988)
14. Molland, H.: Improved linear consistency attack on irregular clocked keystream generators. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 109–126. Springer, Heidelberg (2004)
15. Molland, H., Helleseth, T.: An improved correlation attack against irregular clocked and filtered keystream generators. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 373–389. Springer, Heidelberg (2004)
16. Oechslin, P.: Making a faster cryptanalytic time-memory trade-off. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 617–630. Springer, Heidelberg (2003)
17. Pasalic, E.: On guess and determine cryptanalysis of LFSR-based stream ciphers. IEEE Transactions on Information Theory 55, 3398–3406 (2009)
18. Saarinen, M.-J.O.: A time-memory tradeoff attack against LILI-128. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 231–236. Springer, Heidelberg (2002)
19. Simpson, L.R., Dawson, E., Golić, J.D., Millan, W.L.: LILI keystream generator. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 248–261. Springer, Heidelberg (2001)
20. Tsunoo, Y., Saito, T., Shigeri, M., Kubo, H., Minematsu, K.: Shorter bit sequence is enough to break stream cipher LILI-128. IEEE Transactions on Information Theory 51, 4312–4319 (2005)