Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks

Muhammed F. Esgin[1,2] and Orhun Kara[1]

[1] TÜBİTAK BİLGEM UEKAE, Gebze, Kocaeli, Turkey
{muhammed.esgin, orhun.kara}@tubitak.gov.tr
[2] Graduate School of Natural and Applied Sciences, Istanbul Şehir University, Istanbul, Turkey

Abstract. A new lightweight stream cipher, Sprout, has been presented at FSE 2015. The main concern in the design philosophy of the cipher is to decrease the internal state size without compromising the security against Time-Memory-Data (TMD) tradeoff attacks. In this work, we have mounted a TMD tradeoff attack to Sprout using 2^d output bits in $2^{71.7-d}$ encryptions of Sprout along with 2^d table lookups. The memory complexity is 2^{85-d} where $d \leq 40$. In one instance, it is possible to recover the key in faster than 2^{33} encryption time if we have 2^{40} bits of keystream output by using tables of 770 Terabytes in total. The offline phase of preparing the tables consists of solving roughly 2^{42} system of linear equations with 20 unknowns.

1 Introduction

One of the main design principal for stream ciphers is that internal state size should be at least twice as large as the key size. Otherwise it is possible to mount Time-Memory-Data (TMD) tradeoff attacks [4, 12, 7, 6, 17]. This design principal makes lightweight stream cipher design more challenging. Indeed, we do not see so much examples of lightweight stream cipher designs after the most popular lightweight stream cipher designs such as Trivium [10], Mickey [5] or Grain [1, 14, 16, 15] of the eSTREAM project. However, there have been several lightweight block cipher designs appeared in the literature recently such as Present [8], ITUBee [18], Lblock [22], Prince [9], Ktantan [11], Twine [21] and many more.

One contrary example of this common tendency may be considered as the new stream cipher design Sprout presented in FSE 2015 [2, 3] by Armknecht and Mikhalev. The cipher makes use of a variable internal state of only 80 bits and a fixed key of 80 bits as well. The authors show that a cipher design such as Sprout is resistant to classical TMD tradeoff attacks even though its (variable) internal state size is less then twice the key size.

Even though Sprout is a very recent cipher presented at FSE 2015, there have been couple of its analysis instantly appeared on the net. The first attack paper is by Lallemand and Naya-Plasencia and they have a claim that it is possible to recover the key with a time complexity equivalent to exhaustive search of roughly

2^{70} attempts by merging the sets of possible LFSRs and NLFSRs by carefully sieving [19]. Indeed, the actual workload is $2^{74.51}$ steps but since each step costs $2^{-5.64}$ trial in exhaustive search, the overall time complexity is finalized as $2^{69.36}$. The memory complexity for leading the values for the registers is 2^{46}.

In another analysis, Maitra et. al. show that when the whole (variable) internal state is known, the key can be found using a SAT solver [20]. Indeed, only a few number of terms in the system of nonlinear equations with unknown key bits generated from the output appear and hence the system is easy to solve. This is demonstrated on a PC and shown that it is possible to solve the system with roughly 900 keystream bits in less than half second on an ordinary PC. Moreover, the authors show that it is still possible to solve the system even though two third of LFSR bits is also unknown. However, solving the system takes around one minute this time. Hence, guessing the variables of whole NLFSR and one third of the variables of LFSR and then solving all the corresponding 2^{54} systems of equations, it is possible to recover the key. On the other hand, the authors do not compare the time complexity of solving 2^{54} equations with that of exhaustive search. Moreover, they mount a fault attack.

In [13], a related-key chosen-IV distinguisher is shown. However, the designers of the Sprout regard related-key attacks as out of scope since the key is assumed to be fixed.

Our Contribution: We present a new cryptanalysis of full Sprout within the practical bounds where all data, memory and time complexities can be upper-bounded by 2^{45}. We first show that when the internal state is known (excluding the key), it is much easier to obtain the secret key when the cipher is run backwards compared to running the cipher forward. Later, guessing the internal states from the keystream bits is described based on a time-memory-data tradeoff approach. For the key-recovery part, our attack is based on both guess-and-determine and divide-and-conquer approach.

Organization of the paper: Section 2 describes the inner workings of Sprout. In Section 3, we show how to efficiently recover the secret key when the (variable) internal state of Sprout is known. Section 4 introduces a TMD tradeoff approach to deal with filling the internal state and checking its validity. We later provide the complexity analysis of our attack and conclude the paper with some remarks in Section 5.

2 Description of Sprout

Sprout [2, 3] is a lightweight stream cipher inspired from Grain family [1, 14, 16, 15]. The (variable) internal state of Sprout consist of an LFSR and an NLFSR, and there is also a fixed key that is used in the state update function. The sizes of LFSR and NLFSR are 40-bits each and the key length is 80-bits. An IV of size 70-bits is also incorporated during the initialization phase. The feedback functions of NLFSR and LFSR and the nonlinear part of the output function are denoted by g, f and h respectively (See Figure 2). We follow the notations below throughout the paper.

- t - the clock-cycle number
- $\oplus$ - the XOR operation
- $L_t := (l_0^t, l_1^t, \ldots, l_{39}^t)$ - state of the LFSR at clock-cycle t
- $N_t := (n_0^t, n_1^t, \ldots, n_{39}^t)$ - state of the NLFSR at clock-cycle t
- $C_t := (c_0^t, c_1^t, \ldots, c_8^t)$ - state of the counter at clock-cycle t
- $K := (k_0, k_1, \ldots, k_{79})$ - the fixed key
- $IV := (iv_0, iv_1, \ldots, iv_{69})$ - the initialization vector
- k_t^* - the round key bit generated during clock t
- n_t - the output bit of NLFSR during clock t
- l_t - the output bit of LFSR during clock t
- z_t - the keystream bit generated during clock t
- RKF - Round Key Function

A 9-bit counter is used in the algorithm to count the number rounds for the initialization phase (which has 320 rounds). After initialization, its first seven bits are used to determine the index of the key bit selected at the current clock-cycle. Moreover, 4th bit of the counter c_4^t is involved in the NLFSR feedback.

The characteristic polynomial of the LFSR, $P(x)$, is given as

$$P(x) = x^{40} \oplus x^{35} \oplus x^{25} \oplus x^{20} \oplus x^{15} \oplus x^6 \oplus 1$$

Hence, $l_{39}^{t+1} = f(L_t) = l_0^t \oplus l_5^t \oplus l_{15}^t \oplus l_{20}^t \oplus l_{25}^t \oplus l_{34}^t$ and $l_i^{t+1} = l_{i+1}^t$ for $0 \leq i \leq 38$. We would like to note that the designers does not specify f explicitly in [2, 3]. So, $f(L_t) = l_0^t \oplus l_6^t \oplus l_{15}^t \oplus l_{20}^t \oplus l_{25}^t \oplus l_{35}^t$ may also be inferred. However, our attack works analogously in that case as well.

The function g is the nonlinear feedback function for the NLFSR. Its output is XORed with the round key bit k_t^*, the counter bit c_4^t and the output of the LFSR $l_t = l_0^t$. So, the feedback of the NLFSR n_{39}^{t+1} is given as follows:

$$\begin{aligned} n_{39}^{t+1} &= g(N_t) \oplus k_t^* \oplus l_0^t \oplus c_4^t \\ &= k_t^* \oplus l_0^t \oplus c_4^t \oplus n_0^t \oplus n_{13}^t \oplus n_{19}^t \oplus n_{35}^t \oplus n_{39}^t \oplus n_2^t n_{25}^t \\ &\oplus n_3^t n_5^t \oplus n_7^t n_8^t \oplus n_{14}^t n_{21}^t \oplus n_{16}^t n_{18}^t \oplus n_{22}^t n_{24}^t \oplus n_{26}^t n_{32}^t \\ &\oplus n_{33}^t n_{36}^t n_{37}^t n_{38}^t \oplus n_{10}^t n_{11}^t n_{12}^t \oplus n_{27}^t n_{30}^t n_{31}^t \end{aligned}$$

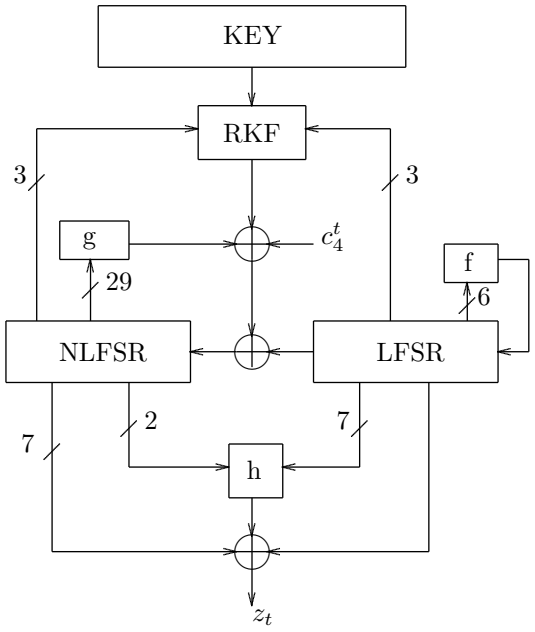
where

$$\begin{aligned} k_t^* &= k_t, \quad 0 \leq t \leq 79 \\ k_t^* &= k_{t \bmod 80} \cdot (l_4^t \oplus l_{21}^t \oplus l_{37}^t \oplus n_9^t \oplus n_{20}^t \oplus n_{29}^t) \end{aligned}$$

We will define $\delta_t := l_4^t \oplus l_{21}^t \oplus l_{37}^t \oplus n_9^t \oplus n_{20}^t \oplus n_{29}^t$.

Once the internal state is determined for clock-cycle t, the keystream bit z_t is generated as follows:

$$\begin{aligned} z_t &= h(x) \oplus l_{30}^t \oplus n_1^t \oplus n_6^t \oplus n_{15}^t \oplus n_{17}^t \oplus n_{23}^t \oplus n_{28}^t \oplus n_{34}^t \\ &= n_4^t l_6^t \oplus l_8^t l_{10}^t \oplus l_{32}^t l_{17}^t \oplus l_{19}^t l_{23}^t \oplus n_4^t l_{32}^t n_{38}^t \\ &\oplus l_{30}^t \oplus n_1^t \oplus n_6^t \oplus n_{15}^t \oplus n_{17}^t \oplus n_{23}^t \oplus n_{28}^t \oplus n_{34}^t \end{aligned}$$



Initialization phase: The feedback registers are initialized as follows:

$$\begin{aligned} n_i &= iv_i, && \text{for } 0 \le i \le 39 \\ l_i &= iv_{40+i}, && \text{for } 0 \le i \le 29 \\ l_i &= 1, && \text{for } 30 \le i \le 38 \\ l_{39} &= 0 \end{aligned}$$

After filling the registers, the cipher is run 320 clocks without producing keystream while the output z_t is fed back into both feedback registers such that $l_{39}^{t+1} = z_t \oplus f(L_t)$ and $n_{39}^{t+1} = z_t \oplus k_t^* \oplus l_0^t \oplus c_4^t \oplus g(N_t)$. The keystream generator starts generating the keystream after the initialization phase.

The designers of Sprout suggest to generate up to 2^{40} keystream bits with one (key, IV) pair.

Workload of exhaustive search: To exhaustively search a key, one has to run the initialization phase first (320 clocks), and then generate 80-bits of keystream

for a unique match. However, since each keystream bit generated matches the correct one with probability $\frac{1}{2}$, 2^{80} keys are tried for 1 clock and roughly half of them are eliminated, 2^{79} for 2 clocks and half of the remaining keys are eliminated, and so on. Hence, the average number of clocks per one trial among 2^{80} keys is

$$\sum_{i=0}^{79} \frac{(i+1)2^{80-i}}{2^{80}} = \sum_{i=0}^{79} (i+1)\frac{1}{2^i} \approx 4$$

As a result, we will assume that clocking the registers once will cost roughly $\frac{1}{324} \approx 2^{-8.34}$ encryptions.

3 A Key Recovery Attack

Observe that Sprout's next state function is invertible. So, once we obtain the whole state (including the key), we can clock the internal states of the cipher forward and backwards, as well. So, it is straightforward to recover the internal state at clock-cycle t from the internal state at clock-cycle $t+1$. We first decrease the counter. Then, for the LFSR feedback, we have

$$\begin{aligned} l_0^t &= l_{39}^{t+1} \oplus l_5^t \oplus l_{15}^t \oplus l_{20}^t \oplus l_{25}^t \oplus l_{34}^t \\ &= l_{39}^{t+1} \oplus l_4^{t+1} \oplus l_{14}^{t+1} \oplus l_{19}^{t+1} \oplus l_{24}^{t+1} \oplus l_{33}^{t+1} \end{aligned} \tag{1}$$

and $l_{i+1}^t = l_i^{t+1}$ for $0 \le i \le 38$. For the NLFSR feedback, we have

$$\begin{aligned} n_0^t &= k_t^* \oplus l_0^t \oplus c_4^t \oplus n_{39}^{t+1} \oplus n_{13}^t \oplus n_{19}^t \oplus n_{35}^t \oplus n_{39}^t \oplus n_2^t n_{25}^t \\ &\quad \oplus n_3^t n_5^t \oplus n_7^t n_8^t \oplus n_{14}^t n_{21}^t \oplus n_{16}^t n_{18}^t \oplus n_{22}^t n_{24}^t \oplus n_{26}^t n_{32}^t \\ &\quad \oplus n_{33}^t n_{36}^t n_{37}^t n_{38}^t \oplus n_{10}^t n_{11}^t n_{12}^t \oplus n_{27}^t n_{30}^t n_{31}^t \\ &= k_t^* \oplus c_4^t \oplus l_{39}^{t+1} \oplus l_4^{t+1} \oplus l_{14}^{t+1} \oplus l_{19}^{t+1} \oplus l_{24}^{t+1} \oplus l_{33}^{t+1} \\ &\quad \oplus n_{39}^{t+1} \oplus n_{12}^{t+1} \oplus n_{18}^{t+1} \oplus n_{34}^{t+1} \oplus n_{38}^{t+1} \oplus n_1^{t+1} n_{24}^{t+1} \\ &\quad \oplus n_2^{t+1} n_4^{t+1} \oplus n_6^{t+1} n_7^{t+1} \oplus n_{13}^{t+1} n_{20}^{t+1} \oplus n_{15}^{t+1} n_{17}^{t+1} \oplus n_{21}^{t+1} n_{23}^{t+1} \oplus n_{25}^{t+1} n_{31}^{t+1} \\ &\quad \oplus n_{32}^{t+1} n_{35}^{t+1} n_{36}^{t+1} n_{37}^{t+1} \oplus n_9^{t+1} n_{10}^{t+1} n_{11}^{t+1} \oplus n_{26}^{t+1} n_{29}^{t+1} n_{30}^{t+1} \end{aligned} \tag{2}$$

where

$$\begin{aligned} k_t^* &= k_t, \quad 0 \le t \le 79 \\ k_t^* &= k_{t \bmod 80} \cdot (l_3^{t+1} \oplus l_{20}^{t+1} \oplus l_{36}^{t+1} \oplus n_8^{t+1} \oplus n_{19}^{t+1} \oplus n_{28}^{t+1}) \end{aligned}$$

and $n_{i+1}^t = n_i^{t+1}$ for $0 \le i \le 38$.

Now, we can generate the keystream z_t while the index t is decreasing.

Maitra et al. has shown in their recent paper that it is possible to recover the key once the (variable) internal state is known by solving a system of nonlinear

equations by a SAT Solver in less than half second on a single PC, using roughly 900 bits of keystream sequence [20]. We have a similar problem indeed: We make a guess for the internal state and then, we just do not want to determine the key from the internal state but also we would like to check if our guess is correct simultaneously without recovering the whole set of the key bits. The simple observation belove gives us a much faster key recovery and internal state checking mechanism. Indeed, we do not need to solve a system of nonlinear equations. The following property suggests that recovering key from the internal state and output is much easier if we trace backwards through the registers.

Proposition 1. *Assume that at time $t+1$, we know the internal states of both registers, but the whole key is unknown and that $\delta_t = 1$. While clocking the registers backwards, when a key bit appears in the keystream for the first time, it will appear as a single unknown inside n_1^{t-1} in the keystream bit z_{t-1}. This happens before the key bit is incorporated into the feedback of NLFSR through the g function.*

The proof of Proposition 1 is straightforward. Assume that while the cipher is run backwards, at some clock-cycle $t+1$, we guess the value of the whole internal state (excluding the key) and $\delta_t = 1$. One clock later, n_0^t becomes a term of the form $k_i \oplus a$ where a is a known value obtained from the NLFSR feedback and the counter. Now, at time t, n_0^t is not incorporated into the NLFSR feedback function. Thus, $k_i \oplus a$ shifts to the position n_1^{t-1} and n_0^{t-1} does not depend on k_i (it may depend on another key bit but that is not important). Now, at time $t-1$, we know all the values except for n_0^{t-1} and $n_1^{t-1} = k_i \oplus a$. But, n_0^{t-1} is not involved in the output function. So, we can easily determine k_i as $k_i = z_{t-1} \oplus a \oplus a'$ where a' is a known value coming from the tap points of the output function.

As a result, when we make a guess for the registers, at each clock, we will either have opportunity to check if the keystream bit we generate matches the corresponding output bit, or a key bit will appear as a single unknown and we will determine the key bit from the output. Hence, if the state candidate does not yield a contradiction, we again end up with registers that are completely known except maybe for the first bit n_0^t of NLFSR. However, if a key bit is involved in that term, it will be determined one clock later before going into the feedback. To sum up, continuing the procedure recursively, either we recover a single key bit or we have a check bit for each clock. Let us illustrate this with a simple example.

Example 1. Assume that at time $t+1$ we know the whole internal state but the secret key and let $\delta_t = \delta_{t-1} = \delta_{t-2} = 1$ and $\delta_{t-3} = 0$. Let k_0, k_1, k_2 and k_3 be the key bits selected in given order. In Table 1, we show how the values of NLFSR bits and keystream bits proceed.

Clock-cycle	δ_i	n_0^i	n_1^i	n_2^i	n_3^i	$\cdots$	n_{39}^i	z_i	D/C
$i = t+1$	-	✓	✓	✓	✓	✓	✓	✓	C
$i = t$	1	$k_0 \oplus$ ✓	✓	✓	✓	✓	✓	✓	C
$i = t-1$	1	$k_1 \oplus$ ✓	$k_0 \oplus$ ✓	✓	✓	✓	✓	$k_0 \oplus$ ✓	$D(k_0)$
$i = t-2$	1	$k_2 \oplus$ ✓	$k_1 \oplus$ ✓	✓	✓	✓	✓	$k_1 \oplus$ ✓	$D(k_1)$
$i = t-3$	0	✓	$k_2 \oplus$ ✓	✓	✓	✓	✓	$k_2 \oplus$ ✓	$D(k_2)$
$i = t-4$	-	✓	✓	✓	✓	✓	✓	✓	C

Table 1. ✓denotes a known value, $D(k_i)$ determining the value of k_i and C is a check if the keystream bit generated matches the actual one.

The probability that a key bit does not appear in the output for p blocks of length 80 bits is 2^{-p}. Hence, after roughly 160 clocks, 60 different key bits will appear in the output and will thus be determined. The time complexity of recovering roughly 60 bits of the key for a correct guess of internal state is almost 160 clocks of Sprout. The remaining 20 bits can be recovered by searching exhaustively. On the other hand, the probability that a guess for an internal state survives for $2r$ clocks is 2^{-r}. On average for each 2 clocks, half of the possible guesses will be eliminated. So, the average number of clocks for each elimination among 2^s possible guesses is

$$\sum_{i=0}^{s} \frac{2(i+1)2^{s-i}}{2^s} = \sum_{i=0}^{s} (i+1)\frac{1}{2^{i-1}} \approx 8 \quad \text{for } 21 \leq s \leq 40.$$

4 A Time-Memory-Data Tradeoff Attack

Recall that to treat the bits of the registers as a sequence, we denote $l_{t+i+j} := l_i^{t+j}$ and $n_{t+i+j} := n_i^{t+j}$. We mount an attack on Sprout by using 2^d data with a time complexity of 2^{80-d} steps where $d \leq 40$. We make use of memory also, having roughly 2^{85-d} entries. The attack scenario is simple. Assume that δ_t is zero for consecutive d clocks. That is, the key bits are not incorporated into the NLFSR during d consecutive clocks: $t-9, t-8, \ldots, t+d-10$. Then we can make a guess to the internal state at time t and then check if the guess is correct and the condition is satisfied since we can produce d bits of outputs without knowing any key bit.

Assuming that $\delta_{t-9} = \delta_{t-8} = \cdots = \delta_{t+d-10} = 0$ we get the following d linear equations of the internal state bits.

$$
\begin{array}{llllllll}
l_{t-5} & \oplus\, l_{t+12} & \oplus\, l_{t+28} & \oplus\, n_t & \oplus\, n_{t+11} & \oplus\, n_{t+20} & = 0 \\
l_{t-4} & \oplus\, l_{t+13} & \oplus\, l_{t+29} & \oplus\, n_{t+1} & \oplus\, n_{t+12} & \oplus\, n_{t+21} & = 0 \\
 & & & \vdots & & & \\
l_{t+4} & \oplus\, l_{t+21} & \oplus\, l_{t+37} & \oplus\, n_{t+9} & \oplus\, n_{t+20} & \oplus\, n_{t+29} & = 0 \\
l_{t+5} & \oplus\, l_{t+22} & \oplus\, l_{t+38} & \oplus\, n_{t+10} & \oplus\, n_{t+21} & \oplus\, n_{t+30} & = 0 \\
 & & & \vdots & & & \\
l_{t+d-6} & \oplus\, l_{t+d+11} & \oplus\, l_{t+d+27} & \oplus\, n_{t+d-1} & \oplus\, n_{t+d+10} & \oplus\, n_{t+d+19} & = 0
\end{array}
$$

If $d \leq 20$ then we have d linear equations with at most 80 unknowns. Note that we can write an LFSR feedback as a linear equation without adding new unknowns. So, we simply exclude both the linear equations and new unknowns coming from LFSR feedback. However, if $d > 20$ then the new unknowns from the feedback of the NLFSR will appear with some nonlinear equations. The new equations coming from the feedback are

$$
\begin{array}{llll}
c_4^t & \oplus\, l_t & \oplus\, n_{t+40} & \oplus\, g(N_t) \quad = 0 \\
c_4^{t+1} & \oplus\, l_{t+1} & \oplus\, n_{t+41} & \oplus\, g(N_{t+1}) \quad = 0 \\
 & & \vdots & \\
c_4^{t+d-21} & \oplus\, l_{t+d-21} & \oplus\, n_{t+d+19} & \oplus\, g(N_{t+d-21}) = 0,
\end{array}
$$

which are adding $d-20$ more equations with $d-20$ new unknowns n_{t+40},..., n_{t+d+19}. We have $2d-20$ equations with $60+d$ unknowns. We see that by guessing most appropriate $80-d$ unknowns, we mostly come up with a $2d-20$ linear equations with $2d-20$ unknowns. Solving the linear system for each counter set, we can determine $2d-20$ unknown bit. That is, we can determine the whole internal state. Then we can produce the output up to $d+3$ bits for all possible counter combinations. See Section 4.1 for solving the system of equations in the most extreme case.

Let us store all the guessed internal states where $\delta_t = 0$ for d consecutive clocks with their outputs up to $d+3$ bits for each counter, sorted according to the outputs. Note that, we can generate keystream bits $z_{t-10}, z_t, \ldots, z_{t+d-8}$ due to the fact that the output is not affected by the most and the least significant taps of the NLFSR.

We need 32 tables having 2^{80-d} rows each to produce a table for each counter. During the on-line phase of the attack, any $d+3$ clock output x at a certain time (so counter is known) is searched in the table with the related counter. There are 2^{77-2d} internal states producing the output x . Check if any of the internal state is correct. Repeat this procedure 2^d times since the probability that $\delta_t = 0$ for d consecutive clocks is 2^{-d}. We expect this event to be occurred once. Then, we can recover the internal state from the tables and then recover the key easily

once the internal state is known. Recovering the key from a known internal state is explained in Section 3.

For each output of $d+3$ bits, we have 2^{77-2d} internal states producing the output. We both check the validity of the internal state and recover the key bits for each candidate. On average, the output bits of 8 clocks more is enough for the checking. Hence, the time complexity is $8 \cdot 2^{77-d} = 2^{80-d}$ clocks which is equivalent to $2^{71.7-d}$ encryptions of Sprout along with 2^d table lookups.

4.1 Detailed workload for $d = 40$

We focus on the extreme case $d = 40$ and give the workloads in detail for this case. The most complicated part is solving the system of equations during the precomputation phase where we list all the possible internal states having $\delta_t = 0$ for 40 consecutive t values.

We need to solve the following systems of equations: a linear system $\mathcal{LS}$ and a non-linear system $\mathcal{NS}$.

$$\mathcal{LS} := \begin{cases} l_{t-5} \oplus l_{t+12} \oplus l_{t+28} \oplus n_t \oplus n_{t+11} \oplus n_{t+20} = 0 \\ l_{t-4} \oplus l_{t+13} \oplus l_{t+29} \oplus n_{t+1} \oplus n_{t+12} \oplus n_{t+21} = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ l_{t+4} \oplus l_{t+21} \oplus l_{t+37} \oplus n_{t+9} \oplus n_{t+20} \oplus n_{t+29} = 0 \\ l_{t+5} \oplus l_{t+22} \oplus l_{t+38} \oplus n_{t+10} \oplus n_{t+21} \oplus n_{t+30} = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ l_{t+33} \oplus l_{t+50} \oplus l_{t+66} \oplus n_{t+38} \oplus n_{t+49} \oplus n_{t+58} = 0 \\ l_{t+34} \oplus l_{t+51} \oplus l_{t+67} \oplus n_{t+39} \oplus n_{t+50} \oplus n_{t+59} = 0 \end{cases}$$

$$\mathcal{NS} := \begin{cases} c_4^t \oplus l_t \oplus n_{t+40} \oplus g(N_t) = 0 \\ c_4^{t+1} \oplus l_{t+1} \oplus n_{t+41} \oplus g(N_{t+1}) = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ c_4^{t+18} \oplus l_{t+18} \oplus n_{t+58} \oplus g(N_{t+18}) = 0 \\ c_4^{t+19} \oplus l_{t+19} \oplus n_{t+59} \oplus g(N_{t+19}) = 0 \end{cases}$$

First of all, we can easily write all l_i's in $\mathcal{LS}$ as linear combinations of l_j's for $t \le j \le t+39$. Let $\mathcal{LS}'$ be the new system of equations of $\mathcal{LS}$ where all l_i's for $t-5 \le i \le t-1$ and $t+40 \le i \le t+67$ are replaced with l_j's for $t \le j \le t+39$ in accordance with the LFSR feedback function. Now denoting $\mathcal{L} := (l_t, l_{t+1}, \ldots, l_{t+39})^T$ and

$$\mathcal{B} := (n_t \oplus n_{t+11} \oplus n_{t+20}, n_{t+1} \oplus n_{t+12} \oplus n_{t+21}, \ldots, n_{t+39} \oplus n_{t+50} \oplus n_{t+59})^T,$$

we can write $\mathcal{LS}$ as $\mathcal{M} \cdot \mathcal{L} = \mathcal{B}$ where $\mathcal{M}$ is the 40×40 coefficient matrix of l_j's and ^{T} is the transpose operation.

We see that $\mathcal{M}$ is invertible. Hence, $\mathcal{L} = \mathcal{M}^{-1}\mathcal{B}$ implying we can equate each l_j, $t \leq j \leq t+39$, to linear combinations of some n_i's for $t \leq i \leq t+59$. Plugging in the values of l_j's for $t \leq k \leq t+19$ in $\mathcal{NS}$, we end up with a system, denoted by $\mathcal{NS}'$, of 20 non-linear equations in 60 variables (ignoring the counter values for the moment). As a result, the main goal is to find all the solutions of $\mathcal{NS}'$ and store them in a table with their outputs.

It's expected that there exist 2^{40} solutions for the system. One approach for solving it may be to use a SAT solver. However, this approach is quite inefficient compared to solving a system of linear equations. Having a more detailed look at the equations in $\mathcal{NS}'$, we see that by carefully guessing 40 n_i values, the system becomes almost linear. To do that, one possible choice for selected n_i values, $SEC\{n_i\}$ is as follows:

$$
\begin{array}{cccccccccc}
n_{t+5} & n_{t+6} & n_{t+8} & n_{t+9} & n_{t+11} & n_{t+12} & n_{t+14} & n_{t+15} & n_{t+17} & n_{t+18} \\
n_{t+19} & n_{t+21} & n_{t+22} & n_{t+23} & n_{t+25} & n_{t+26} & n_{t+27} & n_{t+29} & n_{t+30} & n_{t+31} \\
n_{t+32} & n_{t+33} & n_{t+34} & n_{t+35} & n_{t+36} & n_{t+38} & n_{t+39} & n_{t+40} & n_{t+41} & n_{t+42} \\
n_{t+43} & n_{t+45} & n_{t+46} & n_{t+47} & n_{t+48} & n_{t+49} & n_{t+50} & n_{t+52} & n_{t+54} & n_{t+55}
\end{array}
$$

Fixing these values, there still exists (at most 3) non-linear terms with probability $\frac{1}{2}$. We see some nonlinear terms when $(n_{t+48}, n_{t+52}, n_{t+54}, n_{t+55}) \in S$ where

$$
\begin{aligned}
S := \{&(1,1,1,1),(0,1,1,1),(1,0,1,1),(1,1,0,1) \\
&,(1,1,1,0),(0,0,1,1),(0,1,0,1),(1,1,0,0)\}
\end{aligned}
$$

It is expected that each 40-bit guess in both cases,

$$(n_{t+48}, n_{t+52}, n_{t+54}, n_{t+55}) \in S \text{ and } (n_{t+48}, n_{t+52}, n_{t+54}, n_{t+55}) \notin S$$

yields 1 solution on average, which we have verified experimentally. Hence, by solving all the cases where the system is linear, we can obtain around 2^{39} solutions. As a result, we can obtain half of the solutions with an effort of solving $\frac{1}{2} \cdot 2^{40} = 2^{39}$ systems of linear equations. If one wishes to obtain all the solutions, then in order to make the system linear, 2 more n_i values (e.g adding n_{t+48} and n_{t+55} to $SEC\{n_i\}$ and forming $EXSEC\{n_i\}$) need to be guessed. Hence, the effort for finding all the solutions is equivalent to solving $\frac{1}{2} \cdot 2^{42} + 2^{39} \approx 2^{41.32}$ systems of linear equations. The non-linear cases can be solved using a SAT solver as well. However, this is still more inefficient than guessing 2 more bits and solving a linear system.

We can repeat the computations for each candidate of the counter values. For $d = 40$, there are 39 different values of $(c_4^t, c_4^{t+1}, \ldots, c_4^{t+d-21})$, which we will refer as *counter array*. Observe that the counter values are added to the system $\mathcal{NS}'$ linearly. Therefore, we can do row reduction operation once and find solutions for different counter arrays. However, we still need to store separate tables for

each counter array to produce 43 bits of output for each internal state. Since there are 74 possible counter arrays to produce 43 output bits, we need to store 74 separate tables. Algorithm 1 and Algorithm 2 summarizes the full attack.

Algorithm 1 Creating Tables

for each choice of 40 n_i values in $SEC\{n_i\}$ with $(n_{t+48}, n_{t+52}, n_{t+54}, n_{t+55}) \in S$ **do**
  $N_t \leftarrow Solve(NS')$
  for each C_i where C_i's denote possible counter arrays **do**
    Find $N_t^{C_i}$ from N_t by plugging in the values in C_i
    $L_t^{C_i} \leftarrow \mathcal{M}^{-1} \cdot \mathcal{B}$
    $Z_{t-10}^{t+32} \leftarrow (z_{t-10}, \ldots, z_{t+32})$ // generate keystream for 43 clocks.
    Store $(Z_{t-10}^{t+32}, N_t^{C_i}, L_t^{C_i})$ in a table $T_Z^{C_i}$ sorted by Z_{t-10}^{t+32}
  end for
end for
for each choice of 42 n_i values in $EXSEC\{n_i\}$ with $(n_{t+48}, n_{t+52}, n_{t+54}, n_{t+55}) \notin S$ **do**
  $N_t \leftarrow Solve(NS')$
  for each C_i where C_i's denote possible counter arrays **do**
    Find $N_t^{C_i}$ from N_t by plugging in the values in C_i
    $L_t^{C_i} \leftarrow \mathcal{M}^{-1} \cdot \mathcal{B}$
    $Z_{t-10}^{t+32} \leftarrow (z_{t-10}, \ldots, z_{t+32})$ // generate keystream for 43 clocks.
    Store $(Z_{t-10}^{t+32}, N_t^{C_i}, L_t^{C_i})$ in a table $T_Z^{C_i}$ sorted by Z_{t-10}^{t+32}
  end for
end for

We verified that Algorithm 1 produces internal states ((N_t, L_t) pairs) for which $(\delta_{t-9}, \delta_{t-8}, \ldots, \delta_{t+30}) = (0, 0, \ldots, 0)$ and that Algorithm 2 finds the correct key very easily if the guess for the internal state is correct.

The data complexity $D = 2^{40}$ bits of output, not necessarily produced by just one IV, and we have 74 tables of each having 2^{40} rows. Each row contains 80-bit internal state and 3 output bits, indexed by 40 bits of the output. Hence, the memory requirement is roughly 770 Terabytes (this can be reduced by storing some of the tables and increasing the data complexity). The precomputation for creating the tables is $2^{41.32}$ number of row reducing operations of 20 by 20 matrices along with producing 43 bit outputs for each solution. So, we have $74 \cdot 2^{40} \cdot 43$ clocks of Sprout keystream generation function which is equivalent to roughly $2^{43.3}$ encryptions. The workload of adding an output with its internal state to the appropriate table in a sorted way is negligible. The time complexity during the online phase is 2^{40} table look-ups along with the $2^{-3} \cdot 2^{40} \cdot 2^{3} \cdot 2^{-8.34} = 2^{32.66}$ number of encryptions.

4.2 Reducing the data complexity

Let $\Delta_t^d := (\delta_t, \delta_{t+1}, \ldots, \delta_{t+d-1})$. We focused on the case when $\Delta_t^{40} = (0, 0, \ldots, 0)$ for some t. However, suppose Δ_t^{40} contains a single 1 at i-th index and k_i is

Algorithm 2 Online Phase of the Attack

```
Take 2^40 keystream bits (not necessarily generated using the same IV)
for each 43-bit keystream block do
    // C_j := corresponding counter array for the current clock-cycle
    if Keystream block exists in T_Z^{C_j} then
        Fill NLFSR and LFSR according to values in T_Z^{C_j}
        while Internal state does not produce a contradiction do
            Clock cipher backwards
            if keystream is in {0, 1} then
                Check state!
            else
                Determine key bit value involved
            end if
        end while
    end if
end for
```

incorporated into the NLFSR feedback at clock-cycle $t + i$. In that case, we can create the tables for $k_i = 0$ and $k_i = 1$ separately (which would double the table size) and apply the same attack. If we do this for each possible index i, M increases by a factor of $2 \cdot 40 = 80$ and D decreases by a factor of 40. Generalizing this idea, we can create tables for each possible Δ_t^d. If there are n 1s in Δ_t^d, M increases by a factor of $2^n \cdot \binom{d}{n}$ while D decreases by a factor of $\binom{d}{n}$.

5 Conclusion

We have illustrated a Time-Memory-Data tradeoff attack mounted to Sprout stream cipher. The attack combines both guess-and-determine and divide-and-conquer techniques. We have guessed some taps of the internal state satisfying certain particular property and then determine the remaining taps by solving a specific system of linear equations. After storing all such internal states in a table with their outputs (we can produce some output bits without knowing the key bits, thanks to the special property that the internal states satisfy), we mount a divide-and-conquer attack to recover the key from the given output keystream bits. The attack complexities are highly feasible.

One countermeasure against the attack may be decreasing the throughput of Sprout and giving only one bit output in for instance 16 Sprout clocks. That output may be the sum of all the 16 bit outputs of original Sprout.

References

1. Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: A new version of grain-128 with optional authentication. *Int. J. Wire. Mob. Comput.*, 5(1):48–59, December 2011.

2. Frederik Armknecht and Vasily Mikhalev. On lightweight stream ciphers with shorter internal states. Cryptology ePrint Archive, Report 2015/131, 2015. http://eprint.iacr.org/.
3. Frederik Armknecht and Vasily Mikhalev. On lightweight stream ciphers with shorter internal states. In *Fast Software Encryption FSE 2015*, Lecture Notes in Computer Science. Springer, 2015. (To appear).
4. Steve Babbage. Improved exhaustive search attacks on stream ciphers. Security and Detection 1995, European Convention IET, 1995.
5. Steve Babbage and Matthew Dodd. The mickey stream ciphers. In Matthew Robshaw and Olivier Billet, editors, *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 191–209. Springer Berlin Heidelberg, 2008.
6. Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 1–21. Springer Berlin Heidelberg, 2006.
7. Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.
8. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer Berlin Heidelberg, 2007.
9. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. Prince – a low-latency block cipher for pervasive computing applications. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer Berlin Heidelberg, 2012.
10. Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer Berlin Heidelberg, 2006.
11. Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. Katan and ktantan – a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer Berlin Heidelberg, 2009.
12. Jovan Dj. Golić. Cryptanalysis of alleged A5 stream cipher. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 1997.
13. Yonglin Hao. A related-key chosen-iv distinguishing attack on full sprout stream cipher. Cryptology ePrint Archive, Report 2015/231, 2015. http://eprint.iacr.org/.

14. Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A stream cipher proposal: Grain-128. In *Information Theory, 2006 IEEE International Symposium on*, pages 1614–1618, July 2006.
15. Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The grain family of stream ciphers. In Matthew Robshaw and Olivier Billet, editors, *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 179–190. Springer Berlin Heidelberg, 2008.
16. Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *Int. J. Wire. Mob. Comput.*, 2(1):86–93, May 2007.
17. Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
18. Ferhat Karakoç, Hüseyin Demirci, and A. Emre Harmancı. Itubee: A software oriented lightweight block cipher. In Gildas Avoine and Orhun Kara, editors, *Lightweight Cryptography for Security and Privacy*, volume 8162 of *Lecture Notes in Computer Science*, pages 16–27. Springer Berlin Heidelberg, 2013.
19. Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of full sprout. Cryptology ePrint Archive, Report 2015/232, 2015. http://eprint.iacr.org/.
20. Subhamoy Maitra, Santanu Sarkar, Anubhab Baksi, and Pramit Dey. Key recovery from state information of sprout: Application to cryptanalysis and fault attack. Cryptology ePrint Archive, Report 2015/236, 2015. http://eprint.iacr.org/.
21. Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE: A lightweight block cipher for multiple platforms. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 339–354. Springer Berlin Heidelberg, 2013.
22. Wenling Wu and Lei Zhang. Lblock: A lightweight block cipher. In Javier Lopez and Gene Tsudik, editors, *Applied Cryptography and Network Security*, volume 6715 of *Lecture Notes in Computer Science*, pages 327–344. Springer Berlin Heidelberg, 2011.