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,
İstanbul Şehir University, İstanbul, Turkey

Abstract. The internal state size of a stream cipher is supposed to be at least twice the key length to provide resistance against the conventional Time-Memory-Data (TMD) tradeoff attacks. This well adopted security criterion seems to be one of the main obstacles in designing, particularly, ultra lightweight stream ciphers. At FSE 2015, Armknecht and Mikhalev proposed an elegant design philosophy for stream ciphers as fixing the key and dividing the internal states into equivalence classes where any two different keys always produce non-equivalent internal states.

The main concern in the design philosophy is to decrease the internal state size without compromising the security against TMD tradeoff attacks. If the number of equivalence classes is more than the cardinality of the key space, then the cipher is expected to be resistant against TMD tradeoff attacks even though the internal state (except the fixed key) is of fairly small length. Moreover, Armknecht and Mikhalev presented a new design, which they call Sprout, to embody their philosophy.

In this work, ironically, we mount a TMD tradeoff attack on Sprout within practical limits using 2^d output bits in 2^{71-d} encryptions of Sprout along with 2^d table lookups. The memory complexity is 2^{86-d} where $d \leq 40$. In one instance, it is possible to recover the key in 2^{31} encryptions and 2^{40} table lookups 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 $2^{41.3}$ systems of linear equations with 20 unknowns and an effort of about 2^{35} encryptions. Furthermore, we mount a guess-and-determine attack having a complexity about 2^{68} encryptions with negligible data and memory. We have verified our attacks by conducting several experiments. Our results show that Sprout can be practically broken.

Keywords: Sprout · Stream cipher · Keystream generator · Time Memory Data tradeoff attacks · LFSR · NLFSR · Guess-and-determine · Divide-and-conquer

This author's work is based upon work from COST Action CRYPTACUS, supported by COST (European Cooperation in Science and Technology).

O. Dunkelman and L. Keliher (Eds.): SAC 2015, LNCS 9566, pp. 67–85, 2016.
DOI: 10.1007/978-3-319-31301-6_4

1 Introduction

One of the main design principles for stream ciphers is that internal state size should be at least twice the key size. Otherwise, the cipher will be vulnerable to generic Time-Memory-Data (TMD) tradeoff attacks [3,5–7,11,16]. This principle makes lightweight stream cipher design more challenging in comparison to designing block ciphers even though the same generic attacks are valid for block ciphers, particularly, when they are used in stream cipher modes such as OFB or CTR mode.

There have been several lightweight block cipher designs appeared in the literature such as Present [8], ITUBee [17], Lblock [21], Prince [9], Ktantan [10], Twine [20] and many more in the last decade. An example that does not follow the common tendency of designing lightweight block ciphers rather than lightweight stream ciphers is the new stream cipher design Sprout, presented at FSE 2015 [2] 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 strictly less than twice the key size.

The property of having a smaller internal state size results from the uncommon design principle adopted by the designers of Sprout [2]. According to this principle, the key is also incorporated into the next state function of the cipher as a fixed vector and the internal states including the keys are divided into equivalence classes. Each key is in a different equivalence class. That is, it is impossible to obtain two equivalent internal states with different keys. Therefore, the conventional TMD tradeoff attacks are ineffective since the number of equivalence classes is not less than the cardinality of the key space and one needs to travel almost all the equivalence classes to recover one specific key, rendering the generic tradeoff attacks slower than the exhaustive search.

The design of Sprout is inspired by Grain 128a in [1]. The sizes of both the NLFSR and the LFSR of Grain family are reduced to half of their values and the functions are slightly changed. The design philosophy of output generation and feedback functions is almost conserved except adding a round key function to incorporate key bits for each clock. As a result, a stream cipher of area cost roughly 800 GE is developed whereas Grain needs 1162 GE for the same level of 80-bit security [2].

Sprout has immediately attracted the interest of the cryptology community intensively, indicating that the design philosophy itself introduces several open questions about the security of such ciphers. Even though it is a very recent cipher, there have been a couple of its analyses [4,12,18,19].

The first attack paper is by Lallemand and Naya-Plasencia, and they have shown that it is possible to recover the key with a time complexity equivalent to roughly 2^{70} Sprout encryptions by merging the sets of possible LFSRs and NLFSRs through a careful sieving [18]. Indeed, the actual workload is $2^{74.51}$ steps but since the cost of each step is considered to be $2^{5.64}$ times faster than one step of exhaustive search in [18], 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 [19]. The system of nonlinear equations generated from the output is quite easy to solve. The authors show that it is possible to solve the system with roughly 900 keystream bits in less than half a second on an ordinary PC. Moreover, the authors show that it is still possible to solve the system even though two thirds of LFSR bits are 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. Besides their algebraic attack, they give an example of a fault attack as well [19].

The guess-and-determine attack in [19] using a SAT solver is further improved in [4] by Banik. After guessing 50 bits of the state, the remaining bits including key bits can be solved in roughly half a minute on a standard PC by means of Cryptominisat 2.9.5 solver installed in SAGE 5.7. Moreover, Banik observed that the LFSR is expected to be initialized to the all zero vector in one of 2^{40} random synchronizations. When such an event occurs, the NLFSR is easily recovered with the key by guessing 33 bits of its state. The total time complexity is around 2^{67} encryptions with negligible memory and about 2^{42} bits of output. A distinguisher attack is also included by observing that it is possible to produce shifted versions of a keystream up to a factor of 80 with the same key by using different IVs. When the factor for shifting is limited as 2^{10}, it is possible to distinguish Sprout output with 2^{57} bits of memory and 2^{32} encryptions [4].

In [12], 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.

Table 1. The time complexities are given in terms of number of encryptions except when specified as number of table lookups (TLs). The memory complexities are given in terms of number of rows. Additionally, we assume an effort of 1 s = 2^{15} encryptions and that of 1 min = 2^{21} encryptions on a standard PC.

	Time	Data	Memory
[18]	2^{70}	Negligible	2^{46}
[19]	2^{75}	Negligible	Negligible
Sect. 3 in [4]	2^{70}	Negligible	Negligible
Sect. 5 in [4]	$2^{66.7}$	2^{42} bits	Negligible
Sect. 3 in this work	2^{68}	Negligible	Negligible
Sect. 4.1 in this work	2^{40} TLs + 2^{31}	2^{40} bits	2^{46}
TMD Tradeoff in this work	2^{d} TLs + 2^{71-d}	2^{d} bits	2^{86-d}

In Table 1, we compare the complexities of known attacks on Sprout. Our straightforward C/C++ implementation of Sprout runs more than $2^{23.5}$ clocks in a second on a standard PC. Thus, we make the assumption that an effort of a second is equivalent to 2^{15} Sprout encryptions and that of a minute to 2^{21} Sprout encryptions. One may further improve these numbers by a more efficient and performance-oriented implementation. As we show in Appendix A, one clock of Sprout costs about $2^{-8.33}$ Sprout encryptions.

Our Contribution: None of the attacks in the literature so far has practical workloads. We introduce a new TMD tradeoff cryptanalysis of full Sprout within the practical bounds where all data, memory and time complexities can be upperbounded 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 a combination of both guess-and-determine and divide-and-conquer approaches. We show that the key recovery part itself can be transformed to a guess-and-determine attack of a complexity 2^{68} encryptions with negligible data and memory.

Our TMD tradeoff attack is based on a specific occasion where incorporation of key bits into the register can be discarded during the keystream production. So, Sprout behaves like Grain family since the round key function is bypassed. We can compute the internal states in advance for all the predefined occasions and store them with the produced keystream bits in a table. We can produce some keystream bits for these special internal states without knowing the key.

In general, one may expect that the special internal states to be stored are deduced by exhaustively searching them. However, we can deduce these states without a search-and-eliminate mechanism, reducing the time complexity of the precomputation phase dramatically. We show that computing the special internal states which cause to bypass the key bits is as easy as solving some systems of linear equations. Each of our guesses gives a valid solution. So, unlike generic TMD tradeoff attacks where the offline phase is generally as expensive as an exhaustive search, our offline phase of the attack is also in practical limits. To sum up, we explore and exploit a special kind of lack of sampling resistance in Sprout. The objective of this procedure is similar to the one of BSW sampling applied to A5/1 cipher [7].

In the online phase of the attack, we check if a special occasion occurs in the given keystream. The time complexity is 2^{79-d} Sprout clocks when we have 2^d bits of keystream (not necessarily from the same IV). The memory requirement is about 2^{86-d}. For example, when $d = 40$, we can recover the key in only 2^{31} encryptions and 2^{40} table lookups by using 2^{40} bits of keystream with 770 Terabytes of memory. The precomputation cost of preparing the tables is equivalent to solving about $2^{41.32}$ systems of linear equations with 20 unknowns and about 2^{35} encryptions. We have verified our results by conducting several experiments.

Organization of the Paper: Section 2 describes the high level structure of Sprout. In Sect. 3, we show how to efficiently recover the secret key when the (variable) internal state of Sprout is known and introduce a guess-and-determine attack. Section 4 introduces a TMD tradeoff approach to deal with filling the internal state, including how to construct systems of linear equations during the off-line phase of the attack. We conclude the paper with some remarks and propose a solution in Sect. 5. Some details of Sprout is given in Appendix A and we provide experimental results verifying our attack in Appendix B.

2 High Level Description of Sprout

Sprout [2] is a lightweight stream cipher inspired by Grain family [1,13–15]. 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 Fig. 1 in the appendix). 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

A 9-bit counter is used in the algorithm to count the number of rounds for the initialization phase (which has 320 rounds). After initialization, its first seven bits run cyclically from 0 to 79 and determine the index of the key bit selected at the current time. Moreover, the fourth bit of the counter c_4^t is involved in the NLFSR feedback.

The linear relation of the LFSR is $l_{39}^{t+1} = f(L_t) = l_0^t \oplus l_5^t \oplus l_{15}^t \oplus l_{20}^t \oplus l_{25}^t \oplus l_{34}^t$. 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 as $n_{39}^{t+1} = g(N_t) \oplus k_t^* \oplus l_0^t \oplus c_4^t$ where $k_t^* = k_{t \bmod 80} \cdot \delta_t$ with $\delta_t := l_4^t \oplus l_{21}^t \oplus l_{37}^t \oplus n_9^t \oplus n_{20}^t \oplus n_{29}^t$. Remark that one clock of Sprout is equivalent to $2^{-8.33}$ encryption of an exhaustive key search. We give the necessary details in the analysis sections. Also, one can refer to [2] or Appendix A for details.

3 A Key Recovery Attack

It is easy to see 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

$$l_0^t = l_{39}^{t+1} \oplus l_4^{t+1} \oplus l_{14}^{t+1} \oplus l_{19}^{t+1} \oplus l_{24}^{t+1} \oplus l_{33}^{t+1} \tag{1}$$

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

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

where

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

and $n_{i+1}^t = n_i^{t+1}$ for $0 \leq i \leq 38$ (see [2] or Appendix A). Now, the keystream z_t can be generated 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 [19]. We have a similar problem indeed: We make a guess for the internal state and then, we do not just 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 below 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 NLFSR and LFSR, 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, the LFSR output 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 register 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.

Table 2. ✓ denotes a known value, ? a value that is either known or unknown, $D(k_i)$ determining the value of k_i and C is a check if the keystream bit generated matches the actual one.

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

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

$$\sum_{i=0}^{s} \frac{2 \cdot 2^{s-i}}{2^s} = \sum_{i=0}^{s} \frac{1}{2^{i-1}} \approx 4 \quad \text{for } 21 \leq s \leq 40.$$

We see that we can check if a given state is correct in 4 clocks on average. Moreover, it is possible to mount a guess-and-determine attack by using Proposition 1. We can guess 77 bits of the internal state and determine the three remaining bits that appear as XOR in the output (such as $l^t_{31}, l^t_{30}, l^t_{29}$) since three bits of keystream can be produced without knowing the key. Observe that the bits l^t_i for $28 \geq i \geq 25$ are incorporated into the output as XOR at time $t+i-30$ for the first time after clock t during the backward clocking. So, the guess-and-determine attack can be improved by further determining l^t_i for $28 \geq i \geq 25$, if a key bit is not involved in the output along with l^t_i (that is, $\delta_{t+i-29} = 0$). If $\delta_{t+i-29} = 1$, then guess l^t_i as well and determine the related key bit. So, we guess at least 73 and at most 77 bits according to the values of δ_{t+i-29}. The overall complexity is around 2^{70} encryptions. It can be further improved by assuming that $\delta_{t+i-29} = 0$ for $28 \geq i \geq 25$. In this case, we guess 69 bits and determine 11 bits ($l^t_{36}, l^t_{35}, l^t_{34}, l^t_{33}$ from $\delta_{t+i-29} = 0$ and $l^t_{31}, \ldots, l^t_{25}$ from keystream bits). The cipher is clocked 9 times on average to come up with a contradiction (5 clocks during determining $l^t_{31}, \ldots, l^t_{25}$ and 4 clocks for the key recovery and checking). Repeat the attack 16 times by using shifted keystreams to fulfill the assumption. Hence, the average complexity is $2^4 \cdot 2^{69} \cdot 2^{3.17} = 2^{76.17}$ clocks and thus $2^{76.17} \cdot 2^{-8.33} = 2^{67.84}$ encryptions of Sprout. The data and memory complexities are negligible.

4 A Time-Memory-Data Tradeoff Attack

Recall that, treating the bits of the registers as the terms of a sequence, we denote $l_{t+i+j} := l^{t+j}_i$ and $n_{t+i+j} := n^{t+j}_i$. We mount an attack on Sprout with 2^d data and a time complexity of 2^{79-d} clocks where $d \leq 40$ by enhancing the idea of the guess-and-determine attack given in Sect. 3. We make use of memory also, having roughly 2^{86-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 $d \leq 20$, we have d linear equations with at most 80 unknowns since there will be no feedback for NLFSR. 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 $d > 20$, the new unknowns from the feedback of the NLFSR will appear with some nonlinear equations. The new equations coming from the feedback are

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

which are adding $d - 20$ more equations with $d - 20$ new unknowns $n_{t+40}, \ldots, n_{t+d+19}$. We have $2d - 20$ equations with $60 + d$ unknowns. We see that by carefully choosing $80 - d$ unknowns to be guessed, we mostly come up with $2d-20$ linear equations with $2d-20$ unknowns. Solving the linear system for each counter set, we can determine $2d - 20$ unknown bits. 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 Sect. 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 at most 80 tables having 2^{80-d} rows each to produce a table for each counter. During the online 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 occur 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 Sect. 3.

For each output of $d + 3$ bits, we have on average 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, clocking 4 times is enough for the checking. Hence, the time complexity is $4 \cdot 2^{77-d} = 2^{79-d}$ clocks which is equivalent to 2^{71-d} encryptions of Sprout along with 2^d table lookups.

4.1 Detailed Workload for $d = 40$

We focus on the extreme case $d = 40$ (i.e., $\delta_t = 0$ for 40 consecutive t values) and give the workloads in detail for this case. We need to solve the following systems of equations: a linear system $\mathcal{LS}$ and a nonlinear system $\mathcal{NS}$.

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

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

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 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.

The sequence $l_{t-5} \oplus l_{t+12} \oplus l_{t+28}$ can be also produced by the LFSR of Sprout. Since its characteristic polynomial is primitive, the coefficient matrix $\mathcal{M}$ is a power of the next state matrix and hence it is invertible. We also have verified on a computer 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 $t \leq j \leq t+19$ in $\mathcal{NS}$, we end up with a system, denoted by $\mathcal{NS}'$, of 20 nonlinear 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:

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

Selection of these n_i values mostly (but not always) follows the rule that n_{i+1} and n_{i+2} should be guessed to make an n_i value appear as a linear term. Fixing the values in $SEC\{n_i\}$, there still exists (at most 3) nonlinear terms $n_{t+51}n_{t+53}$, $n_{t+51}n_{t+56}$ and $n_{t+56}n_{t+57}$ 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 (see Appendix B). 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+51} and n_{t+57} to $SEC\{n_i\}$ and forming $EXSEC\{n_i\}$) need to be guessed. Hence, the effort of 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 nonlinear cases can be solved using a SAT solver as well. However, this is still less efficient 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 a *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

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 $(Z_{t-10}^{t+32}, N_{t-9}^{C_i}, L_{t-9}^{C_i})$ 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 $(Z_{t-10}^{t+32}, N_{t-9}^{C_i}, L_{t-9}^{C_i})$ in a table $T_Z^{C_i}$ sorted by $Z_{t-10}^{t+32}$
    end for
end 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. The solutions give internal states at the t-th clock. Store the internal states at time $(t - 9)$ just to avoid clocking backwards 9 times before mounting the key recovery attack in Sect. 3.

Algorithms 1 and 2 summarizes the full attack. Algorithm 1 is the offline phase of the attack, related to preparing tables for each counter value. Algorithm 2 gives the set of instructions of how to recover key or come to a contradiction in a formal way for each trial of 43-bit keystream segment. We verified by producing some test values that Algorithm 1 generates 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 when our assumption about δ_t is fulfilled during keystream generation (see Appendix B).

The data complexity is given as $D = 2^{40}$ 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 the remaining 40 bits of the output. Hence, the memory requirement M is roughly 770 Terabytes[1] (this can be reduced by storing some of the tables in cost of increased data complexity). The precomputation for creating the tables is $2^{41.32}$ row reduction operations of at most 20 by 20 matrices along with producing 43 bit outputs for each solution. Since 60 n_i values and all the l_i values are known, 33-bit

[1] One may choose to store only 60 n_i values to reduce the memory requirement. In this case about 580TB of memory is needed. But, in the online phase, the values of LFSR need to be computed.

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
```

output may be generated by substituting the appropriate values in the output function without needing to clock the whole cipher. In addition, we need to clock backwards 9 times to produce the remaining 10 output bits, which brings an additional effort of about $9 \cdot 2^{40} \cdot 2^{-8.33} \approx 2^{34.84}$ 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 lookups along with $2^{-3} \cdot 2^{40} \cdot 5 \cdot 2^{-8.33} \approx 2^{31}$ encryptions. Recall that only 1/8 of the keystreams are in one of the tables. And, if a keystream is found in a table, the corresponding internal state obtained is for time $t-9$, but we already make use of z_{t-10}. Hence, a candidate is checked in $1+4=5$ clocks on average.

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 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 and Discussion

We illustrated Time-Memory-Data tradeoff attacks and a guess-and-determine attack mounted on full Sprout. The TMD attacks combine both guess-and-determine and divide-and-conquer techniques. We have guessed some taps of the internal state satisfying a certain property and then determined the remaining taps by solving a specific system of linear equations. After storing all such internal states in tables 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 complexities indicate that the attack is highly feasible. We have verified our statements by conducting several experiments. The detailed experimental results including an implementation of the attack with a small-scale table are given in Appendix B.

Designing ultra lightweight stream ciphers allocating less than 1K GE in hardware with a moderate security level such as 80 bits, is a new challenge, initiated by Armknecht and Mikhalev [2]. Even though their first design attempt does not achieve the required security level, we believe that several other designs will likely appear in the literature soon and some of them will probably be secure enough to be used in the industry.

We claim that producing output during each clocking of the registers is not a convenient design philosophy for ciphers whose internal state sizes are not large enough. Otherwise, they may be prone to guess-and-determine or divide-and-conquer attacks. We think that adopting the design philosophy of Grain family, particularly in bitwise operations and clockwise output generation, has brought the security failure to Sprout. The research question is about the optimization of the rate of the output generation over the register clocking in terms of security versus throughput. This can be considered as a generic question relating to all ultra lightweight stream ciphers. One straightforward countermeasure against all the attacks on Sprout may be decreasing the throughput of Sprout and giving only one bit output in, for instance, 16 clockings of Sprout registers. That output may be the sum of all the 16 bit outputs of the original Sprout.

Acknowledgments. We would like to thank anonymous reviewers for their helpful comments, especially the second reviewer who makes a comprehensive analysis of the paper and a lot of useful suggestions. We would also like to thank Ferhat Karakoç and Güven Yücetürk for commenting on the paper.

References

1. Ågren, M., Hell, M., Johansson, T., Meier, W.: Grain-128a: a new version of Grain-128 with optional authentication. Int. J. Wire. Mob. Comput. **5**(1), 48–59 (2011)
2. Armknecht, F., Mikhalev, V.: On lightweight stream ciphers with shorter internal states. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 451–470. Springer, Heidelberg (2015)
3. Babbage, S.: Improved exhaustive search attacks on stream ciphers. Security and Detection, European Convention IET (1995)
4. Banik, S.: Some results on Sprout. Cryptology ePrint Archive, Report 2015/327 (2015). http://eprint.iacr.org/
5. Barkan, E., Biham, E., Shamir, A.: Rigorous bounds on cryptanalytic time/memory tradeoffs. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 1–21. Springer, Heidelberg (2006)

6. 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)
7. Biryukov, A., Shamir, A., Wagner, D.: Real time cryptanalysis of A5/1 on a PC. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 1–18. Springer, Heidelberg (2001)
8. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
9. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
10. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — a family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
11. Golić, J.D.: Cryptanalysis of alleged A5 stream cipher. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 239–255. Springer, Heidelberg (1997)
12. Hao, Y.: A related-key chosen-iv distinguishing attack on full Sprout stream cipher. Cryptology ePrint Archive, Report 2015/231 (2015). http://eprint.iacr.org/
13. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. In: 2006 IEEE International Symposium on Information Theory, pp. 1614–1618, July 2006
14. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 179–190. Springer, Heidelberg (2008)
15. Hell, M., Johansson, T., Meier, W.: Grain: a stream cipher for constrained environments. Int. J. Wire. Mob. Comput. **2**(1), 86–93 (2007)
16. Hellman, M.E.: A cryptanalytic time-memory trade-off. IEEE Trans. Inf. Theor. **26**(4), 401–406 (1980)
17. Karakoç, F., Demirci, H., Harmancı, A.E.: ITUbee: a software oriented lightweight block cipher. In: Avoine, G., Kara, O. (eds.) LightSec 2013. LNCS, vol. 8162, pp. 16–27. Springer, Heidelberg (2013)
18. Lallemand, V., Naya-Plasencia, M.: Cryptanalysis of full Sprout. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology - CRYPTO 2015. LNCS, vol. 9215, pp. 663–682. Springer, Heidelberg (2015)
19. Maitra, S., Sarkar, S., Baksi, A., Dey, P.: Key recovery from state information of Sprout: application to cryptanalysis and fault attack. Cryptology ePrint Archive, Report 2015/236 (2015). http://eprint.iacr.org/
20. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE: a lightweight block cipher for multiple platforms. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 339–354. Springer, Heidelberg (2013)
21. Wu, W., Zhang, L.: LBlock: a lightweight block cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)

A Details of Sprout

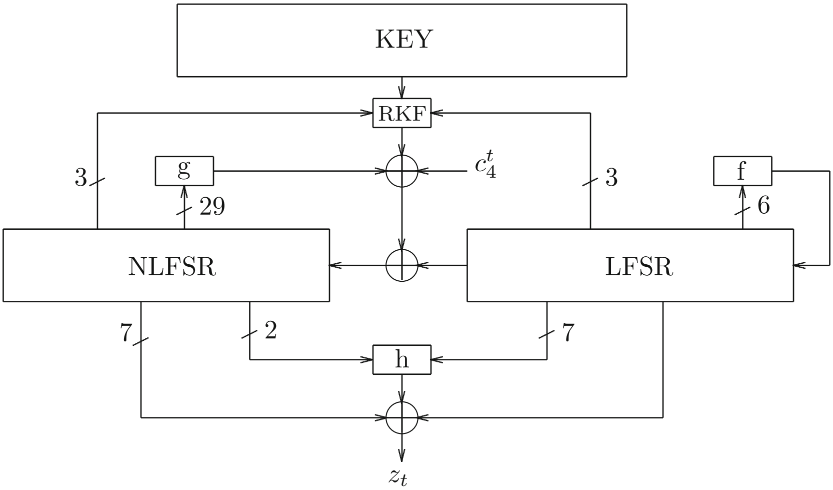

Fig. 1. The high level structure of Sprout

RKF in Fig. 1 represents the round key function. The LFSR is clocked as $l_{39}^{t+1} = f(L_t) = l_0^t \oplus l_5^t \oplus l_{15}^t \oplus l_{20}^t \oplus l_{25}^t \oplus l_{34}^t$ and $l_i^{t+1} = l_{i+1}^t$ for $0 \le i \le 38$. 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

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

with $\delta_t := l_4^t \oplus l_{21}^t \oplus l_{37}^t \oplus n_9^t \oplus n_{20}^t \oplus n_{29}^t$. Once the internal state is determined for clock-cycle t, the keystream bit z_t is generated as follows:

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

Initialization Phase: The feedback registers are initialized as follows:

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

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 corresponding actual keystream bit one with probability $\frac{1}{2}$, 2^{80} keys are tried for 1 clock and roughly half of them are eliminated, 2^{79} for one more clock and half of the remaining keys are eliminated, and so on. Hence, the average number of clocks per one trial after the initialization step among 2^{80} keys is

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

As a result, we will assume that clocking the registers once will cost roughly $\frac{1}{322} \approx 2^{-8.33}$ encryptions.

B Experiments

We did not implement the whole attack due to our memory shortage. Instead, we have conducted several experiments to verify our attack. We have solved millions of the systems of linear equations to collect some of the special internal states where $\delta_t = 0$ for consecutive 40 values of t and stored the solutions in tables. Then, we have accomplished to recover an internal state in the table with the key used in the given keystream sequence.

First of all, we performed a run test on δ_t values to check if the probability that 40 consecutive δ_t values vanish simultaneously is around 2^{-40} for the Sprout internals. We chose 30 random (IV, K) pairs, and run the cipher 2^{40} clocks for each selection. We depict the average number of runs having length i in Table 3. The values for $10 \leq i \leq 21$ are not given in the table since they are too big to display. However, we can simply say that the empirical result for any $10 \leq i \leq 21$ does not deviate from the corresponding expected value more than 0.0678 percent.

Moreover, we have implemented the attack with a small-scale table in Sage 6.5. We found more than 5 million solutions for the system $\mathcal{NS}'$ and generated tables for different counter values. The total size of all the tables is about 215

Table 3. Comparison between the empirical results and the expected number of runs.

	i		$i+1$		$i+2$	
	Empirical	Expected	Empirical	Expected	Empirical	Expected
$i = 22$	65596.66	65536	32802.60	32768	16398.17	16384
$i = 25$	8202.73	8192	4114.03	4096	2042.5	2048
$i = 28$	1025.9	1024	509.9	512	259.4	256
$i = 31$	127.8	128	64.2	64	31.33	32
$i = 34$	17.63	16	7.7	8	4.13	4
$i = 37$	1.83	2	1.2	1	0.400	0.500
$i = 40$	0.333	0.250	0.133	0.125	0	0.062
$i = 43$	0	0.031	0.067	0.016	0.033	0.008

Table 4. Number of state candidates eliminated at each clock i.

Clock-cycle number	# of states eliminated at i-th clock
$i = 1$	250071
$i = 2$	187325
$i = 3$	140862
$i = 4$	105620
$i = 5$	79061
$i = 6$	59067
$i = 7$	44714
$i = 8$	33287
$i = 9$	25044
$i = 10$	18590
Average # of clocks run	$3.999\cdots$

MB. For a randomly chosen state in the tables, we have found a valid (IV, K) pair generating the state. Then, we mounted the attack on the corresponding keystream sequence and successfully recovered the key. The (IV, K) pair was

$$\begin{aligned} IV &= 1101010110100000111010011001011101111010 \\ &\ 011011100110110011000001001010 \\ K &= 1000100100011001111111110000011000111110 \\ &\ 0011001110100100010110100010000100101001 \end{aligned}$$

where the left-most bit represents the value for index 0. At time $t = 25916$ (after initialization), $(\delta_{t-9}, \delta_{t-8}, \ldots, \delta_{t+30}) = (0, 0, \ldots, 0)$ is satisfied. The target

Table 5. Number of solutions for $\mathcal{NS}'$ for each cases when $(n_{t+48}, n_{t+52}, n_{t+54}, n_{t+55}) \in S$ or $(n_{t+48}, n_{t+52}, n_{t+54}, n_{t+55}) \notin S$.

# of solutions	# of eqns with i solns when $(n_{t+48}, n_{t+52}, n_{t+54}, n_{t+55}) \in S$	# of eqns with i solns when $(n_{t+48}, n_{t+52}, n_{t+54}, n_{t+55}) \notin S$
$i = 0$	6049	6478
$i = 1$	5198	4376
$i = 2$	3704	4637
$i = 3$	671	0
$i = 4$	344	501
$i = 5$	9	0
$i = 6$	20	0
$i = 7$	1	0
$i = 8$	4	8
Average	1.012	0.982

internal state at time t was

$$L_t = 01001111111100100000010111111011100101111$$
$$N_t = 11110110010111111110111111001000101100000$$

which is in one of the tables. Running the key recovery attack, we have found 62 key bits in 160 clocks, verifying our statements. The remaining bits can be recovered by an exhaustive search.

We also verified the average number of clocks iterated before arriving at a contradiction during the internal-state-check mechanism. We chose 1 million random internal states and ran the key recovery attack for each of them. Each state was clocked backwards once before starting the test since the output at this clock is definitely a check bit. Table 4 shows how many states were eliminated at each clock i. The average number of clocks run for a candidate was about 3.999, as expected (See Sect. 3). Another experiment was about verifying the assumption that there exists approximately 2^{40} solutions for the system $\mathcal{NS}'$ in total. We have solved the system 16000 times for each case when

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

Table 5 summarizes the results of the experiment, supporting our assumption about the number of solutions.