# Some results on Sprout

Subhadeep Banik

DTU Compute, Technical University of Denmark, Lyngby.
Email: subb@dtu.dk

**Abstract.** Sprout is a lightweight stream cipher proposed by Armknecht and Mikhalev at FSE 2015. It has a Grain-like structure with two State Registers of size 40 bits each, which is exactly half the state size of Grain v1. In spite of this, the cipher does not appear to lose in security against generic Time-Memory-Data Tradeoff attacks due to the novelty of its design. In this paper, we first present improved results on Key Recovery with partial knowledge of the internal state. We show that if 50 of the 80 bits of the internal state are guessed then the remaining bits along with the Secret Key can be found in a reasonable time using a SAT solver. Thereafter we show that it is possible to perform a distinguishing attack on the full Sprout stream cipher in the multiple IV setting using around $2^{40}$ randomly chosen IVs on an average. The attack requires around $2^{48}$ bits of memory. Thereafter we will show that for every Secret Key, there exist around $2^{30}$ IVs for which the LFSR used in Sprout enters the all zero state during the Keystream generating phase. Using this observation, we will first show that it is possible to enumerate Key-IV pairs that produce keystream bits with period as small as 80. We will then outline a simple Key recovery attack that takes time equivalent to $2^{66.7}$ encryptions with negligible memory requirement. This although is not the best attack reported against this cipher in terms of the Time complexity, it is the best in terms of the memory required to perform the attack.

**Keywords:** Grain v1, Sprout, Stream Cipher.

## 1 Introduction

Lightweight stream ciphers have become immensely popular in the cryptological research community, since the advent of the eStream project [1]. The three hardware finalists included in the final portfolio of eStream i.e. Grain v1 [9], Trivium [5] and MICKEY 2.0 [4], all use bitwise shift registers to generate keystream bits. After the design of Grain v1 was proposed, two other members Grain-128 [10] and Grain-128a were added to the Grain family mainly with an objective to provide a larger security margin and include the functionality of message authentication respectively. All the aforementioned ciphers have one trait in common: the sizes of their internal states are at least twice the length of the Key used in their designs. In FSE 2015, Armknecht and Mikhalev proposed the Grain-like stream cipher Sprout [2] with a startlingly opposite trend, the size of the internal state of Sprout was equal to the size of its Key. In [2], the authors argue that due to novel design strategy of the cipher, it can resist generic Time-Memory-Data Tradeoff attacks. The security against some other known classes of attacks was also discussed.

### 1.1 Previous Attacks on Sprout

To the best of our knowledge, four attacks have been reported against Sprout . We explain the feasibility of these attacks a follows:

- In [7], a very obvious Related Key-Chosen IV distinguisher is reported against Sprout . Let $K, V$ denote a Key-IV pair and let $K'$ denote $K$ with the first bit flipped and similarly let $V'$ denote $V$ with the first bit flipped. Then it is easy to see that the probability that the first $80n$ keystream bits produced by $K, V$ and $K', V'$ are equal is given by $\frac{1}{8 \cdot 2^n}$.

- In [11], a fault attack against Sprout is presented. Another attack based on solving a system of non-linear equations by a SAT solver is also presented. The authors guess the values of 54 out of the 80 bits of the internal state. The remaining 106 unknowns, i.e. the remaining 26 internal state bits and the 80 Key bits are found as follows. The authors use the first 400 keystream bits produced by the cipher to populate a bank of non-linear equations in the unknown variables. The resulting system is solved via a SAT solver in around 77 seconds on average on a system running on a 1.83 GHz processor and 4 GB RAM. The SAT solver on an average returns 6.6 candidate Keys. Thus the authors argue that their findings amount to an attack on Sprout in $2^{54}$ *attempts*, since 54 bits are initially guessed in this process. However, the authors do not discuss the computational complexity associated with one *attempt* at solution by a SAT solver. If one can perform around $2^e$ Sprout encryptions in 77 seconds, then in terms of number of encryptions performed, the attack takes time equivalent to $6.6 \times 2^{54} \times 2^e$ encryptions which is more than $2^{80}$ if $e > 23$ (which may be achievable with a good implementation of the cipher), and so it is not certain that the work in [11] translates to a feasible attack on Sprout .
- In [8], a list merging technique is employed to determine the internal state and Secret Key of Sprout that is faster than exhaustive search by $2^{10}$. The attack has a memory complexity of $2^{46}$ bits.
- In [6], a TMD tradeoff attack is outlined using an online time complexity of $2^{33}$ Encryptions and 770 TB of memory. The paper first observes that it is easy to deduce the Secret Key from the knowledge of the internal state and the keystream. The paper then makes an elegant observation on special states of Sprout that produce keystream without the involvement of the Secret Key. A method to generate and store such states in tables is first outlined. The online stage consists of inspecting keystream bits, retrieving the corresponding state from the table, assuming of course that the state in question is a special state, and then computing the Secret Key. The process, if repeated a certain number of times, guarantees that a special state is encountered, from where the correct Secret Key is found.

### 1.2 Organization of the Paper

In Section 2, we present the structural and algebraic description of the Sprout stream cipher. In Section 3, we show that by guessing 50 out of the 80 bits of the internal state, one can determine the remaining bits of the state and the Secret Key by using a SAT solver. This improves the results presented in [11], but due to reasons mentioned earlier, this does not necessarily amount to Cryptanalysis of the cipher. In 4, we show that it is possible to generate two IVs for every Secret Key that generate 80-bit shifted Keystream sequences. Making use of this result we mount a distinguishing attack on Sprout using keystream bits from around $2^{40}$ randomly chosen IVs, and a memory complexity of around $2^{48}$ bits. We also show that the Time complexity of this attack can be reduced at the cost of more memory. Finally in Section 5, we observe that for every Secret Key there exist around $2^{30}$ IVs that result in the LFSR landing in the all zero state during the Keystream generating phase. Based on this observation, we first show how it is possible to generate Key-IV pairs that generate Keystream sequences with period as small as 80. Thereafter, we mount a simple Key recovery attack that requires time equivalent to $2^{66.7}$ encryptions and negligible memory. Section 6 concludes the paper.

## 2 Description of Sprout

The exact structure of Sprout is explained in Figure 1. It consists of a 40-bit LFSR and a 40-bit NFSR. Certain bits of both the shift registers are taken as inputs to a combining Boolean function, whence the keystream is produced. The keystream is produced after performing the following steps:

**Initialization Phase:** The cipher uses an 80 bit Key and a 70 bit IV. The first 40 most significant bits of the IV is loaded on to the NFSR and the remaining IV bits are loaded on to the first 30 most

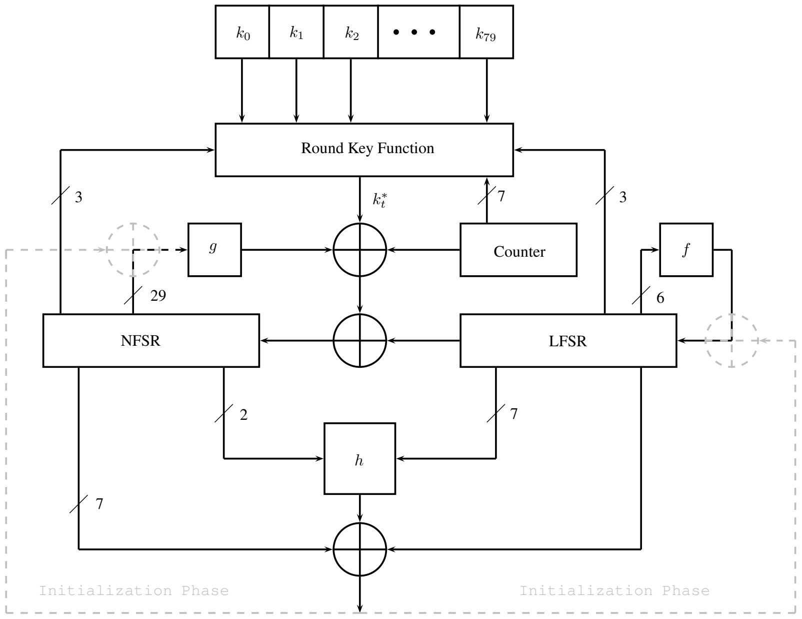

Fig. 1: Block Diagram of Sprout

significant bits of the LFSR. The last 10 bits of the LFSR are initialized with the constant `0x3fe`, i.e. the string of nine $1'$s followed by a 0. Let $L_t = [l_t, l_{t+1}, \ldots, l_{t+39}]$ and $N_t = [n_t, n_{t+1}, \ldots, n_{t+39}]$ denote the 40-bit vectors that denote respectively LFSR and NFSR states at the $t^{th}$ clock interval. During the initialization phase, the registers are updated as follows.

**(a)** In the first 320 rounds (i.e. $0 \leq t \leq 319$) of the initialization phase the cipher produces the keystream bit $z_t$ which is not produced as output. This is computed as

$$z_t = l_{t+30} + \sum_{i \in \mathcal{A}} n_{t+i} + h(N_t, L_t).$$

where $\mathcal{A} = \{1, 6, 15, 17, 23, 28, 34\}$ and $h(N_t, L_t) = n_{t+4}l_{t+6} + l_{t+8}l_{t+10} + l_{t+32}l_{t+17} + l_{t+19}l_{t+23} + n_{t+4}l_{t+32}n_{t+38}$.

**(b)** The LFSR updates as $l_{t+40} = z_t + f(L_t)$, where

$$f(L_t) = l_t + l_{t+5} + l_{t+15} + l_{t+20} + l_{t+25} + l_{t+34}.$$

**(c)** The NFSR updates as $n_{t+40} = z_t + g(N_t) + c_t^4 + k_t^* + l_0^t$, where $c_t^4$ denotes the $4^{th}$ LSB of the modulo 80 up-counter which starts at $t = 0$, $k_t^*$ is the output of the Round Key function defined as:

$$k_t^* = \begin{cases} K_{t \bmod 80}, & \text{if } t < 80, \\ K_{t \bmod 80} \cdot (l_{t+4} + l_{t+21} + l_{t+37} + n_{t+9} + n_{t+20} + n_{t+29}), & \text{otherwise.} \end{cases}$$

Here $K_i$ simply denotes the $i^{th}$ bit of the Secret Key. The non-linear function $g(N_t)$ is given as:

$$\begin{aligned} g(N_t) = {} & n_{t+0} + n_{t+13} + n_{t+19} + n_{t+35} + n_{t+39} + n_{t+2}n_{t+25} + n_{t+3}n_{t+5} + n_{t+7}n_{t+8} + \\ & n_{t+14}n_{t+21} + n_{t+16}n_{t+18} + n_{t+22}n_{t+24} + n_{t+26}n_{t+32} + n_{t+33}n_{t+36}n_{t+37}n_{t+38} + \\ & n_{t+10}n_{t+11}n_{t+12} + n_{t+27}n_{t+30}n_{t+31}. \end{aligned}$$

**Keystream Phase:** After the Initialization phase is completed, the cipher discontinues the feedback of the keystream bit $z_t$ to the update functions of the NFSR and LFSR and makes it available as the output bit. During this phase, the LFSR and NFSR update themselves as $l_{t+40} = f(L_t)$ and $n_{t+40} = g(N_t) + c_t^4 + k_t^* + l_0^t$ respectively.

## 3 Key Recovery from partial knowledge of State

In [11], results were presented pertaining to the recovery of the Secret Key with partial knowledge of the State. The authors claimed that if all the NFSR bits are known and 14 bits of the LFSR are also known then by using the algebraic equations resulting from the first 450 keystream bits, the Keyspace can be reduced to a set of 6.6 candidates on average, by solving the equations through a SAT solver. It was also mentioned that the solver took around 77 seconds on average to solve the system. Although this does not necessarily lead to an attack, we show in this section that it is possible to propose a better algorithm. Before proceeding we present a brief outline of the algorithm used in [11]:

1. Assume that the entire NFSR state and around $m$ bits of the LFSR are known just after the completion of the Key Initialization phase. Let us label the time index as $t = 0$ at this instant. The remaining $40 - m$ bits of the LFSR and the 80 bits of the Secret Key are unknown at this point. The vectors $L_t = [l_t, l_{t+1}, \ldots, l_{t+39}]$ and $N_t = [n_t, n_{t+1}, \ldots, n_{t+39}]$. So initially it is assumed that $N_0$ is completely known and $L_0$ is known partially.

2. For $t = 0$ to $N_r - 1$ do

   - Introduce two new unknowns $l_{40+t}, n_{40+t}$ defined as $l_{40+t} = f(L_t)$ and $n_{40+t} = g(N_t)+c_t^4+k_t^*+l_t$.
   - Form the keystream equation

   $$z_t = l_{t+30} + \sum_{i \in \mathcal{A}} n_{t+i} + n_{t+4}l_{t+6} + l_{t+8}l_{t+10} + l_{t+32}l_{t+17} + l_{t+19}l_{t+23} + n_{t+4}l_{t+32}n_{t+38}.$$

3. After forming the above bank of $3N_r$ equations, pass them to a SAT solver.

The authors of [11] claimed that for $m = 14,\ N_r = 450$, the SAT solver was able to narrow down the set of candidate Secret Keys to 6.6 on average in around 77 seconds.

### 3.1 A few observations

The ease with which a SAT solver is able to solve a given bank of equations depends on the algebraic degree of the equations so formed [12]. It is clear that the algebraic degree of $z_t$ with respect to the unknowns in $L_0$ and the Secret Key increases for increasing $t$. It is also known that, if the Key is known, then the state update during both the Keystream phase and the Initialization phase are one-to-one and invertible. Indeed, rewriting the functions $f,\ g$ as $f(L_t) = l_t + f'(L'_t)$ and $g(N_t) = n_t + g'(N'_t)$ (here $L'_t = [l_{t+1}, l_{t+2}, \ldots, l_{t+39}]$ and $N'_t = [n_{t+1}, n_{t+2}, \ldots, n_{t+39}]$, then if $L_t,\ N_t$ denote the state at time $t$, then $L_{t-1}$ is given as $[l_{t-1}, l_t, \ldots, l_{t+38}]$ where $l_{t-1} = l_{t+39} + f'(L'_{t-1})$, and since $L'_{t-1}$ is a subset of $L_t$, we can see that $L_{t-1}$ is completely defined by $L_t$. Similarly $N_{t-1} = [n_{t-1}, n_t, \ldots, n_{t+38}]$ where

$$n_{t-1} = n_{t+39} + l_{t-1} + k_{t-1}^* + c_{t-1}^4 + g'(N'_{t-1}).$$

Here too since $N'_{t-1} \subset N_t$, the previous state $N_{t-1}$ is completely defined by $L_t, N_t$. Keeping this in mind, we formulate the following strategy for Key recovery from the partial knowledge of state.

1. We assume that at $t = 320$, all the bits of $N_{320}$ and $m$ bits of $L_{320}$ are known. Thereafter we do the following:

2. For $t = 0$ to 319 do
   - Introduce two new unknowns $l_{360+t}, n_{360+t}$ defined as $l_{360+t} = f(L_{t+320})$ and $n_{360+t} = g(N_{t+320}) + c^4_{t+320} + k^*_{t+320} + l_{t+320}$.
   - Form the keystream equation $z_{t+320} = l_{t+350} + \sum_{i\in\mathcal{A}} n_{t+320+i} + h(N_{t+320}, L_{t+320})$.
3. We now take help of the keystream generated before $t = 320$
4. For $t = 320$ to 1 do
   - Introduce two new unknowns $l_{t-1}, n_{t-1}$ defined as $l_{t-1} = l_{t+39} + f'(L'_{t-1})$ and $n_{t-1} = n_{t+39} + l_{t-1} + k^*_{t-1} + c^4_{t-1} + g'(N'_{t-1})$.
   - Form the keystream equation $z_{t-1} = l_{t+29} + \sum_{i\in\mathcal{A}} n_{t-1+i} + h(N_{t-1}, L_{t-1})$.
5. After preparing this bank of $320 * 3 * 2 = 1920$ equations, we forward it to a SAT Solver.

Since the algebraic degrees of $z_{320+t}$ and $z_{320-t}$ are expected to be the same with respect to the unknowns in $L_{320}$ and the Secret Key, we achieve the dual purpose of populating our bank of equations with more entries and at the same time control the algebraic degree of the equations to some extent. We performed the experiments with Cryptominisat 2.9.5 [13] solver installed with the SAGE 5.7 [14] computer algebra system on a computer with a 2.1 GHz CPU and 16 GB memory. For $m = 10$, (after guessing 50 bits of the internal state), we were able to find the remaining bits of the state and the correct Secret Key in around 31 seconds on average.

## 4 A Distinguishing Attack

Before we get into details of the distinguisher, let us revisit a few facts about Sprout . If the Secret Key is known, then the state updates in both the Keystream and Initialization phases are one-to-one and efficiently invertible. Before proceeding, we give an algorithmic description of the state update inversion routines in the Keystream and Initialization phases. We denote the algorithms by $\mathsf{KS}^{-1}$ and $\mathsf{Init}^{-1}$ respectively.

```
Input: $L_t, N_t$: The LFSR, NFSR state at time $t$;
Output: $L_{t-1}, N_{t-1}$: The LFSR, NFSR state at
        time $t-1$;

$l_{t-1} \leftarrow l_{t+39} + f'(L'_{t-1})$;
$n_{t-1} \leftarrow n_{t+39} + l_{t-1} + k^*_{t-1} + c^4_{t-1} + g'(N'_{t-1})$;
$L_{t-1} \leftarrow [l_{t-1}, l_t, l_{t+1}, \ldots, l_{t+38}]$;
$N_{t-1} \leftarrow [n_{t-1}, n_t, n_{t+1}, \ldots, n_{t+38}]$;
Return $L_{t-1}, N_{t-1}$
```

**Algorithm 1**: Algorithm $\mathsf{KS}^{-1}$

```
Input: $L_t, N_t$: The LFSR, NFSR state at time $t$;
Output: $L_{t-1}, N_{t-1}$: The LFSR, NFSR state at
        time $t-1$;

$z_{t-1} \leftarrow l_{t+29} + \sum_{i\in\mathcal{A}} n_{t-1+i} + h(N_{t-1}, L_{t-1})$;
$l_{t-1} \leftarrow l_{t+39} + f'(L'_{t-1}) + z_{t-1}$;
$n_{t-1} \leftarrow n_{t+39} + l_{t-1} + k^*_{t-1} + c^4_{t-1} + g'(N'_{t-1}) + z_{t-1}$;
$L_{t-1} \leftarrow [l_{t-1}, l_t, l_{t+1}, \ldots, l_{t+38}]$;
$N_{t-1} \leftarrow [n_{t-1}, n_t, n_{t+1}, \ldots, n_{t+38}]$;
Return $L_{t-1}, N_{t-1}$
```

**Algorithm 2**: Algorithm $\mathsf{Init}^{-1}$

We will use the above subroutines to generate Key-IV pairs that generate 80-bit shifted keystream sequences. To do that we follow the following steps:

1. Fix the Secret Key $K$ to some constant in $\{0,1\}^{80}$
2. Fix $\mathsf{Success} \leftarrow 0$
3. Do the following till $\mathsf{Success} = 1$

   - Select $\mathbf{S} = [s_0, s_1, \ldots, s_{79}] \xleftarrow{\mathsf{R}} \{0,1\}^{80}$ randomly.
   - Assign $N_0 \longleftarrow [s_0, s_1, \ldots, s_{39}]$, $L_0 \longleftarrow [s_{40}, s_{41}, \ldots, s_{79}]$
   - Run $\mathsf{Init}^{-1}$ over $N_0, L_0$ for 320 rounds and store the result as $\mathbf{U} = [u_0, u_1, \ldots, u_{79}]$.

   - Assign $N_{80} \longleftarrow [s_0, s_1, \ldots, s_{39}]$, $L_{80} \longleftarrow [s_{40}, s_{41}, \ldots, s_{79}]$
   - Run $\mathsf{KS}^{-1}$ over $N_{80}, L_{80}$ for 80 rounds, followed by $\mathsf{Init}^{-1}$ for 320 rounds.
   - Store the result as $\mathbf{V} = [v_0, v_1, \ldots, v_{79}]$.

   - If $u_{70} = u_{71} = \cdots = u_{78} = v_{70} = v_{71} = \cdots = v_{78} = 1$ and $u_{79} = v_{79} = 0$ then $\mathsf{Success} = 1$

The above algorithm fixes the Secret Key $K$, and randomly chooses a state $\mathbf{S}$ and assumes that for two different IVs $V_1, V_2$, the state in the $0^{th}$ round of the Keystream phase for $(K, V_1)$ and the $80^{th}$ round of the Keystream phase for $(K, V_2)$ are both equal to $\mathbf{S}$. The algorithm then performs the State inversion routines in each case and tries to find $V_1$ and $V_2$. A Success occurs when the last 10 bits of both $\mathbf{U}$, $\mathbf{V}$ are equal to the padding `0x3fe` used in Sprout . In that case $V_1 = [u_0, u_1, \ldots, u_{69}]$ and $V_2 = [v_0, v_1, \ldots, v_{69}]$ produces exactly 80-bit shifted Keystream sequences for the Key $K$. Of course, a Success requires 20 bit conditions to be fulfilled and assuming that $\mathbf{U}, \mathbf{V}$ are i.i.d, each iteration of the the above algorithm has a success probability of $2^{-20}$ for any randomly selected $\mathbf{S}$. So running the iteration $2^{20}$ times guarantees one Success on average. By running the above algorithm we were able to obtain several Key-IV pairs that generates 80 bit shifted keystream sequences, which we tabulate in Table 1.

| # | $K$ | $V_1$ | $V_2$ |
|---|---|---|---|
| 1 | `8b0b c4c3 781e fe4b 925c` | `1 03c2cb34d8b8870e5` | `1 f208a4661d50a1f72` |
| 2 | `be8d d8e2 a818 80c5 eda7` | `2 d7d0162c62f256ad7` | `2 5f7c58576e05e3c52` |

Table 1: Key-IV pairs that produce 80 bit shifted Keystream bits. (Note that the first hex character in $V_1, V_2$ encodes the first 2 IV bits, the remaining 17 hex characters encode bits 3 to 70)

Note that it is possible to generate such a Key-IV pair in $2^{10}$ attempts instead of $2^{20}$, if instead of choosing $\mathbf{S}$, we first choose $K, V_1$ randomly, run the forward Initialization algorithm to generate $\mathbf{S}$, and then assume that $\mathbf{S}$ is the $80^{th}$ Keystream phase state for some $K, V_2$ and thereafter run 80 rounds of $\mathsf{KS}^{-1}$ and 320 rounds of $\mathsf{Init}^{-1}$ to generate $\mathbf{V}$. In such a case, Success would be dependent on only the last 10 bits of $\mathbf{V}$ and hence expected once in $2^{10}$ attempts. However we stick with the first algorithm in order to explain the distinguishing attack.

### 4.1 The Distinguisher

In the above algorithm for determining Key-IV pairs that generate shifted Keystream sequences, once the Key is fixed, a Success is expected every $2^{20}$ attempts and since there are $2^{80}$ ways of choosing $\mathbf{S}$, this implies that for every Key $K$, there exist $2^{80-20} = 2^{60}$ IV pairs $V_1, V_2$ such that the Key-IV pairs $(K, V_1)$ and $(K, V_2)$ produce exactly 80-bit shifted Keystream sequences. So our Distinguisher is as follows

1. Generate around 240 keystream bits for the unknown Key $K$ and some randomly generated Initial Vector $V$.

2. Store the Keystream bits in some appropriate data structure like a Binary Search Tree.

3. Continue the above steps with more randomly generated IVs $V$ till we obtain two Initial Vectors for $K$ that generate 80-bit shifted Keystream.

The only question now remains how many random Initial Vectors do we need to try before we get a match. The answer will become clearer if (for a fixed $K$) we imagine the the space of Initial Vectors as an undirected Graph $G = (W, E)$, where $W = \{0,1\}^{70}$ is the Vertex set which contains all the possible 70 bit Initial vector values as nodes. An edge $(V_1, V_2) \in E$ if and only if $(K, V_1)$ and $K(V_2)$ produce 80-bit shifted Keystream sequence. From the above discussion, it is clear that the cardinality of $E$ is expected to be $2^{60}$. When we run the Distinguisher algorithm for $N$ different Initial Vectors, we effectively add $\binom{N}{2}$ edges to the coverage and a match occurs when one of these edges is actually a member of the Edge-set $E$. Since there are potentially $\binom{2^{70}}{2}$ edges in the IV space, by the Birthday bound, a match will occur when the product of $\binom{N}{2}$ and the cardinality of $E$ which is around $2^{60}$ is equal to $\binom{2^{70}}{2}$. From this equation solving for $N$, we get $N \approx 2^{40}$. This gives a bound for the Time and Memory complexity of the Distinguisher. The Time complexity is around $2^{40}$ encryptions, and the memory required is of the order of $2^{40} * 240 \approx 2^{48}$ bits.

In general for Sprout like structures that have an $n$ bit LFSR and NFSR with a $2n$-bit Secret Key and $2n - \Delta$ bit IV (for some $\Delta > 0$), the above equation boils down to $\binom{N}{2} * 2^{2n-2\Delta} = \binom{2^{2n-\Delta}}{2}$, which gives $N \approx 2^n$. In order to verify our theoretical results, we performed experiments on smaller versions of Sprout with $n = 8, 9, 10, 11$ to find the expected value of $N$ in each case. The results have been tabulated in Table 2.

| # | $n$ | $N$ (Experimental) | $N$ (Theoretical) |
|---|---|---|---|
| 1 | 8 | 222.4 | 256 |
| 2 | 9 | 446.9 | 512 |
| 3 | 10 | 911.7 | 1024 |
| 4 | 11 | 1865.7 | 2048 |

Table 2: Experimental values of $N$ for smaller versions of Sprout

## 4.2 Decreasing the Time Complexity

So far we have been restricting ourselves to 80-bit shifts of Keystream sequences. We could easily consider shifts of the form $80 * P$ where $P$ can be any positive integer. The algorithm to find two Initial Vectors $V_1, V_2$ for any Key $K$ that generates $80 * P$-bit shifted Keystream sequence is not very different from the one which finds IVs that generate 80-bit shifted Keystream. We present the explicit form of the algorithm for convenience.

1. Fix the Secret Key $K$ to some constant in $\{0,1\}^{80}$

2. Fix Success $\leftarrow 0$

3. Do the following till $\mathsf{Success} = 1$

   - Select $\mathbf{S} = [s_0, s_1, \ldots, s_{79}] \overset{\mathsf{R}}{\longleftarrow} \{0,1\}^{80}$ randomly.
   - Assign $N_0 \longleftarrow [s_0, s_1, \ldots, s_{39}]$, $L_0 \longleftarrow [s_{40}, s_{41}, \ldots, s_{79}]$
   - Run $\mathsf{Init}^{-1}$ over $N_0, L_0$ for 320 rounds and store the result as $\mathbf{U} = [u_0, u_1, \ldots, u_{79}]$.

   - Assign $N_{80*P} \longleftarrow [s_0, s_1, \ldots, s_{39}]$, $L_{80*P} \longleftarrow [s_{40}, s_{41}, \ldots, s_{79}]$
   - Run $\mathsf{KS}^{-1}$ over $N_{80*P}, L_{80*P}$ for $80 * P$ rounds, followed by $\mathsf{Init}^{-1}$ for 320 rounds.
   - Store the result as $\mathbf{V} = [v_0, v_1, \ldots, v_{79}]$.

   - If $u_{70} = u_{71} = \cdots = u_{78} = v_{70} = v_{71} = \cdots = v_{78} = 1$ and $u_{79} = v_{79} = 0$ then $\mathsf{Success} = 1$

The only change is that we assume that $\mathbf{S}$ is the round 0 state for some $K, V_1$ and the round $80 * P$ state for some $K, V_2$. We perform the inversion operations accordingly and look for a $\mathsf{Success}$. Arguing just as before, we can say that, for any fixed $K$ and $P$, there exist $2^{60}$ IV pairs that generate $80 * P$-bit shifted Keystream Sequences. So we redefine our Distinguishing attack as follows:

1. Generate around $80 * P$ keystream bits for the unknown Key $K$ and some randomly generated Initial Vector $V$.

2. Store the Keystream bits in some appropriate data structure like a Binary Search Tree.

3. Continue the above steps with more randomly generated IVs $V$ till we obtain two Initial Vectors for $K$ that generate $80 * i$-bit shifted Keystream for some $1 \leq i \leq P$.

We can calculate the expected number of attempts $N$ before we get a match as follows. Redefine the undirected graph $G = (W, E)$, where $W = \{0,1\}^{70}$ is the Vertex set which contains all the possible 70 bit Initial vector values as nodes. An edge $(V_1, V_2) \in E$ if and only if $(K, V_1)$ and $K(V_2)$ produce $80 * i$-bit shifted Keystream sequence for some $0 \leq i \leq P$. The expected cardinality of $E$ is approximately $P * 2^{60}$. Again choosing $N$ Initial Vectors adds $\binom{N}{2}$ edges to the coverage and so the required value of $N$ is given by $\binom{N}{2} * P * 2^{60} = \binom{2^{70}}{2} \Rightarrow N \approx \frac{2^{40}}{\sqrt{P}}$. This implies that the Time complexity can be reduced to $\frac{2^{40}}{\sqrt{P}}$ encryptions with the Memory complexity at $80 * P * 2^{40}$ bits. For $P = 2^{10}$ say, this results in a Time Complexity of $2^{35}$ Encryptions and Memory of $2^{57}$ bits.

## 5 A Key Recovery Attack

We make another observation to begin this Section. During the Keystream phase, the LFSR pretty much runs autonomously. Which means that if after the Initialization phase, the LFSR lands on the all zero state then it remains in this state for the remainder of the Keystream phase, i.e. if $L_0 = \mathbf{0}$, then $L_t = \mathbf{0}$ for all $t > 0$. Assuming uniform distribution of $L_0$, we can argue that for every Key $K$, this event occurs for $2^{-40}$ fraction of IVs on average. So for each $K$, there exists on an average $2^{70-40} = 2^{30}$ IVs which lead to an all zero LFSR after the Initialization phase. We shall see two implications of this event.

### 5.1 Keystream with period 80

Now once the LFSR enters the all zero state the NFSR runs autonomously. Since the NFSR is a finite state machine of 40 bits only, we can always expect Keystream of period less than $80 * 2^{40}$, once the LFSR becomes all zero. Hence for every Key, we expect to find $2^{30}$ Initial vectors that produce keystream sequences of less than $80*2^{40}$. With some effort, we can even find Key-IV pairs that produce Keystream with period 80. We will take help of SAT solvers for his. The procedure may be outlined as follows:

1. Select a Key $K \xleftarrow{R} \{0,1\}^{80}$ randomly.
2. Assume $L_0 = [0, 0, 0, \ldots, 0]$.
3. Assign $N_0 \leftarrow [n_0, n_1, n_2, \ldots, n_{39}]$, where all the $n_i$ are unknowns.
4. For $i = 0$ to 79 do
    - Introduce the unknown $n_{40+i}$, and add the equation $n_{40+i} = g(N_i) + c_i^4 + k_i^*$ to the equation bank.
5. Add the 40 Equations $n_i = n_{80+i}$, $\forall\ i \in [0, 39]$ to the equation bank.
6. Pass the equations to the Solver. This effectively asks the solver to solve the vector equation $N_0 = N_{80}$ for the given Key $K$.
7. If the solver returns the solution $N_0 = [s_0, s_1, \ldots, s_{39}]$ then run the $\mathsf{Init}^{-1}$ routine 320 times on $N_0 = [s_0, s_1, \ldots, s_{39}], L_0 = [0, 0, \ldots, 0]$.
8. Store the result in $\mathbf{B} = [b_0, b_1, \ldots, b_{79}]$.
9. If $b_{70} = b_{71} = \cdots = b_{78} = 1$ and $b_{79} = 0$ then Exit else repeat the above steps with another random Secret Key.

The steps in the above the above algorithm can be summarized as follows. First select a random Secret Key $K$. Then assume that the LFSR is all zero after the Initialization phase, and fill the corresponding NFSR state with unknowns. We then populate the equation bank accordingly for the first 80 rounds and ask the solver to solve the vector equation $N_0 = N_{80}$, in the unknowns $n_0, n_1, \ldots, n_{119}$. If the solver returns the solution $N_0 = [s_0, s_1, \ldots, s_{39}]$ then $N_0 = [s_0, s_1, \ldots, s_{39}], L_0 = [0, 0, \ldots, 0]$ is a valid initial state for the Sprout Keystream phase if we can find an IV for the given Key $K$ that results in this state. So we run the $\mathsf{Init}^{-1}$ routine 320 times and obtain the resultant vector $\mathbf{B}$. Now if the last ten bits of $\mathbf{B}$ are equal to the `0x3fe` pattern used in Sprout , then we can be sure that for the Key $K$ and the Initial Vector $V = [b_0, b_1, \ldots, b_{69}]$, the Keystream sequence produced is of period exactly 80 since the same state $N_0 = [s_0, s_1, \ldots, s_{39}], L_0 = [0, 0, \ldots, 0]$ will repeat in the Keystream phase every 80 iterations. The above process is expected to produce one such Key-IV pair in $2^{10}$ attempts. For example, for $K =$ `2819 5612 323c 2357 3518` and $V =$ `2 fbfc75bfcb4396485`, we do obtain a Keystream sequence of period 80 (Note that the first hex character of $V$ encodes the first 2 bits, the remaining 17 hex characters encode bits 3 to 70).

## 5.2 Application to Key recovery

It is clear, that for every Key, on average one out of every $2^{40}$ Initial Vectors lands the LFSR in the all zero state after Initialization. In such a situation the algebraic structure of the cipher becomes simpler to analyze. The NFSR update equation becomes

$$n_{t+40} = g(N_t) + c_t^4 + k_t^*,$$

where $k_t^* = K_{t \bmod 80} \cdot (n_{t+9} + n_{t+20} + n_{t+29})$ and the output Keystream bit is generated as

$$z_t = n_{t+1} + n_{t+6} + n_{t+15} + n_{t+17} + n_{t+23} + n_{t+28} + n_{t+34}.$$

Given such a situation, this greatly simplifies the guess and determine approach of [8] both in terms of time and memory. To explain the attack better let us define $x_i = n_{i+1}$, for all $i \geq 0$ and so we have $N_1 = [x_0, x_1, x_2, \ldots, x_{39}]$. So for $i = 0$ to 6 we have

$$z_i = x_i + x_{i+5} + x_{i+14} + x_{i+16} + x_{i+22} + x_{i+27} + x_{i+33}.$$

This means that if the attacker knows that $L_0 = \mathbf{0}$, then the first 7 Keystream bits $z_0, z_1, z_2, \ldots, z_6$ is dependent on only $N_1$ and the Secret Key is not involved directly in the computation. This implies that if the attacker intends to guess $N_1$ then by observing the first seven keystream bits he can narrow down $N_1$ to a set of $2^{33}$ possible candidates in the following way:

1. Guess $x_0, x_1, x_2, \ldots, x_{32}$ first. There are $2^{33}$ possible candidates.
2. Calculate $x_{i+33} = z_i + x_i + x_{i+5} + x_{i+14} + x_{i+16} + x_{i+22} + x_{i+27}$ for $i = 0$ to 6.

For each of these $2^{33}$ candidates, the attacker proceeds as follows: he calculates $x_{40}$ from the equation for $z_7$ as $x_{40} = z_7 + x_7 + x_{12} + x_{21} + x_{23} + x_{24} + x_{31}$ and from $x_{40}$ he calculates $k_0^*$ as $k_0^* = x_{40} + c_0^4 + g(N_1)$. Now we know that $k_0^* = K_0 \cdot (x_8 + x_{19} + x_{28})$. So if $k_0^* = 0$ and $x_8 + x_{19} + x_{28} = 0$ then nothing can be deduced. If $k_0^* = 0$ and $x_8 + x_{19} + x_{28} = 1$ then it can be deduced that $K_0 = 0$. If $k_0^* = 1$ and $x_8 + x_{19} + x_{28} = 1$ then it can be deduced that $K_0 = 1$. If $k_0^* = 1$ and $x_8 + x_{19} + x_{28} = 0$, then a contradiction is reached and it is concluded that the guess for $N_1$ was incorrect. Thereafter the same procedure with $x_{41}, x_{42} \ldots$ is followed sequentially. We outline the above procedure formally as follows:

1. For Each of the $2^{33}$ choices of $N_1$ do the following till a contradiction is arrived at

   **A.** Assign $i \leftarrow 0$

   **B.** Do the following:

   - Calculate $x_{i+40} = z_{i+7} + x_{i+7} + x_{i+12} + x_{i+21} + x_{i+23} + x_{i+24} + x_{i+31}$
   - Calculate $k_i^* = x_{i+40} + c_i^4 + g(N_{i+1})$
   - If $k_i^* = 0$ and $x_{i+8} + x_{i+19} + x_{i+28} = 0 \Rightarrow$ No Deduction
   - If $k_i^* = 0$ and $x_{i+8} + x_{i+19} + x_{i+28} = 1 \Rightarrow K_{i \bmod 80} = 0$
   - If $k_i^* = 1$ and $x_{i+8} + x_{i+19} + x_{i+28} = 1 \Rightarrow K_{i \bmod 80} = 1$
   - If $k_i^* = 1$ and $x_{i+8} + x_{i+19} + x_{i+28} = 0 \Rightarrow$ we have a contradiction. In this case we restart the process with a new guess of $N_1$.
   - If there is no contradiction then assign $i \leftarrow i + 1$ and repeat the process if the entire Secret Key has not already been found.

**Analysis of Time Complexity:** In the above algorithm, the probability that any guess for $N_1$ is eliminated in 1 round itself is $\frac{1}{4}$, i.e. when $k_i^* = 1$ and $x_{i+8} + x_{i+19} + x_{i+28} = 0$. The probability therefore that it takes 2 rounds to eliminate is $\left(1 - \frac{1}{4}\right) * \frac{1}{4}$. In general, the probability that it takes $i$ steps is roughly $\left(1 - \frac{1}{4}\right)^{i-1} * \frac{1}{4}$. Therefore the average number of rounds $\theta$ that a guess takes to eliminate is given by

$$\theta = \sum_{i=1}^{\infty} \frac{i}{4} * \left(1 - \frac{1}{4}\right)^{i-1} = 4.$$

The attacker obtains the Keystream for some random IV and then tries all the possible $2^{33}$ guesses. This takes $\theta \cdot 2^{33} = 2^{35}$ steps for any IV that does not lead to $L_0 = \mathbf{0}$. It has already been pointed out in [6], clocking each Sprout step is equivalent to $2^{-8.34}$ encryptions. And so for every any IV that does not yield $L_0 = \mathbf{0}$ the total work done is equivalent to $2^{35-8.34} = 2^{26.66}$ encryptions. Now the attacker has to try out around $2^{40}$ IVs to succeed in getting $L_0 = \mathbf{0}$, and so the total Time complexity in this process equals $2^{40+26.66} = 2^{66.66}$ encryptions.

**Analysis of Memory Complexity:** The memory complexity of the algorithm is surprisingly negligible. Testing each guess of $N_1$ can be done on the fly and hence the memory complexity is limited to that required to run the loop and store the computed values of the Key and the values of the $x_i$

bits. This is in stark contrast to the $2^{46}$ bits (8 TB) required in [8] or the 770 TB required in [6]. Thus although, the algorithm that we provide is not the best in terms of Time complexity, it is certainly best in terms of Memory.

## 6 Discussion and Conclusion

In this paper we outline a Distinguishing attack and a Key Recovery attack on the Sprout stream cipher. We also present some results on Key Recovery from partial knowledge of the state, shifted Keystream sequence producing Key-IV pairs and Key-IV pairs producing Keystream sequences with period 80. The Key recovery attack that we propose is not the best in terms of Time complexity but certainly best in terms of the total memory required. It can be pointed out that the attack in [6] was possible due to the non-linear mixing of the Secret Key during the Keystream phase, i.e. $k_t^* = K_{t \bmod 80} \cdot (l_{t+4} + l_{t+21} + l_{t+37} + n_{t+9} + n_{t+20} + n_{t+29})$. This enabled the attacker to identify and generate special internal states that for 40 rounds or so do not involve the Secret Key bit in the computation of the Keystream bit, i.e. those for which $l_{t+4}+l_{t+21}+l_{t+37}+n_{t+9}+n_{t+20}+n_{t+29} = 0$, for 40 consecutive rounds. The attack in [6] would not be directly applicable if the Key mixing was linear, for example if $k_t^* = K_{t \bmod 80}$. However even if the Key mixing were done linearly, all the attacks presented in this paper would still hold. This reiterates the point that when it comes to designing stream ciphers with shorter internal states, the Sprout architecture is possibly the wrong choice. However this does open up a fascinating new research discipline of designing stream ciphers with reduced internal state sizes, one in which the scope to experiment could be boundless.

## References

1. The ECRYPT Stream Cipher Project. eSTREAM Portfolio of Stream Ciphers. Revised on September 8, 2008.
2. F. Armknecht and V. Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. To appear in FSE 2015. Preprint available at `http://eprint.iacr.org/2015/131.pdf`.
3. M. Ågren, M. Hell, T. Johansson and W. Meier. A New Version of Grain-128 with Authentication. Symmetric Key Encryption Workshop 2011, DTU, Denmark, February 2011.
4. S. Babbage and M. Dodd. The stream cipher MICKEY 2.0. ECRYPT Stream Cipher Project Report. Available at `http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf`.
5. C. De Cannière and B. Preneel. TRIVIUM -Specifications. ECRYPT Stream Cipher Project Report. Available at `http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf`.
6. M. F. Esgin and O. Kara. Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks. Available at `http://eprint.iacr.org/2015/289.pdf`.
7. Y. Hao. A Related-Key Chosen-IV Distinguishing Attack on Full Sprout Stream Cipher. Available at `http://eprint.iacr.org/2015/231.pdf`.
8. V. Lallemand and M. Naya-Plasencia. Cryptanalysis of Full Sprout. Available at `http://eprint.iacr.org/2015/232.pdf`.
9. M. Hell, T. Johansson and W. Meier. Grain - A Stream Cipher for Constrained Environments. ECRYPT Stream Cipher Project Report 2005/001, 2005. Available at `http://www.ecrypt.eu.org/stream`.
10. M. Hell, T. Johansson and W. Meier. A Stream Cipher Proposal: Grain-128. In IEEE International Symposium on Information Theory (ISIT 2006), 2006.
11. S. Maitra, S. Sarkar, A. Baksi and P. Dey. Key Recovery from State Information of Sprout: Application to Cryptanalysis and Fault Attack. Available at `http://eprint.iacr.org/2015/236.pdf`.
12. S. Sarkar, S.Banik and S.Maitra. Differential Fault Attack against Grain family with very few faults and minimal assumptions. Available at `http://eprint.iacr.org/2013/494.pdf`.
13. M. Soos. CryptoMiniSat-2.9.5. `http://www.msoos.org/cryptominisat2/`.
14. W. Stein. Sage Mathematics Software. Free Software Foundation, Inc., 2009. Available at `http://www.sagemath.org`. (Open source project initiated by W. Stein and contributed by many).