Cryptanalysis of Plantlet

Subhadeep Banik[1], Khashayar Barooti[1] and Takanori Isobe[2,3]

[1] Security and Cryptography Laboratory (LASEC), École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, {subhadeep.banik,khashayar.barooti}@epfl.ch
[2] National Institute of Information and Communications Technology, Tokyo, Japan
[3] University of Hyogo, Hyogo, Japan, takanori.isobe@ai.u-hyogo.ac.jp

Abstract. Plantlet is a lightweight stream cipher designed by Mikhalev, Armknecht and Müller in IACR ToSC 2017. It has a Grain-like structure with two state registers of size 40 and 61 bits. In spite of this, the cipher does not seem to lose in security against generic Time-Memory-Data Tradeoff attacks due to the novelty of its design. The cipher uses a 80-bit secret key and a 90-bit IV. In this paper, we first present a key recovery attack on Plantlet that requires around $2^{76.26}$ Plantlet encryptions. The attack leverages the fact that two internal states of Plantlet that differ in the 43rd LFSR location are guaranteed to produce keystream that are either equal or unequal in 45 locations with probability 1. Thus an attacker can with some probability guess that when 2 segments of keystream blocks possess the 45 bit difference just mentioned, they have been produced by two internal states that differ only in the 43rd LFSR location. Thereafter by solving a system of polynomial equations representing the keystream bits, the attacker can find the secret key if his guess was indeed correct, or reach some kind of contradiction if his guess was incorrect. In the latter event, he would repeat the procedure for other keystream blocks with the given difference. We show that the process when repeated a finite number of times, does indeed yield the value of the secret key.
In the second part of the paper, we observe that the previous attack was limited to internal state differences that occurred at time instances that were congruent to 0 mod 80. We further observe that by generalizing the attack to include internal state differences that are congruent to all equivalence classed modulo 80, we lower the total number of keystream bits required to perform the attack and in the process reduce the attack complexity to $2^{69.98}$ Plantlet encryptions.

Keywords: Grain v1, Plantlet, Stream Cipher.

1 Introduction

Lightweight stream ciphers have become immensely popular in the cryptological research community, since the advent of the eStream project [est08]. The three hardware finalists included in the final portfolio of eStream i.e. Grain v1 [HJM07], Trivium [CP08] and MICKEY 2.0 [BD08], all use bitwise shift registers to generate keystream bits. After the design of Grain v1 was proposed, two other members Grain-128 [HJMM06] 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. In FSE 2015, Armknecht and Mikhalev proposed the Grain-like stream cipher Sprout [AM15] with a startling trend: the size of the internal state of Sprout was equal to the size of its key. After the publication of [BS00], it was widely accepted that to be secure against generic Time-Memory-Data tradeoff attacks, the internal state of a stream cipher needed to be at least twice the size of the secret key. However the novelty of the Sprout design ensured that the cipher remained secure against generic TMD tradeoffs. The smaller internal state

IACR Transactions on Symmetric Cryptology ISSN 2519-173X, Vol. 2019, No. 3, pp. 103–120
DOI:10.13154/tosc.v2019.i3.103-120

makes the cipher particularly attractive for compact lightweight implementations. However, Sprout has been cryptanalyzed in more ways than one [Ban15, EK15, LNP15, ZG15] and so naturally there has been a lot of research going into the design of secure lightweight stream ciphers.

At the FSE 2017 conference of IACR ToSC, two lightweight stream ciphers Lizard [HKM17] and Plantlet [MAM16] were proposed. Plantlet was essentially a re-design of Sprout after patching some existing weaknesses. The main differences between Plantlet and Sprout are as follows:

- Plantlet uses a 61-bit LFSR, whereas Sprout used a 40-bit LFSR. This slight increase in state size is done to prevent guess and determine attacks [LNP15, Ban15].
- In [EK15], 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 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 which the correct secret key is found. The attack leveraged the fact that the addition of the key bits to the state was done via a non-linear function, due to which it became easy to find special states. Plantlet patched this weakness by making the key addition strictly linear, thus preventing this attack.
- In [Ban15], it was observed that it was easy to find Key-IV pairs that led the LFSR to enter the all zero state after Key-IV mixing. Using this fact the authors were able to compute Key-IV pairs that produced keystream of period 80 bits. The authors were further able to use this fact to mount a guess and determine attack, that required around $2^{66.7}$ Sprout encryptions. To counter this attack, the designers of Plantlet kept the 61st LFSR bit fixed to 1 during the entire Key-IV mixing phase. This ensured that after the Key-IV phase terminated, the LFSR would never enter the all zero state and hence both the above weaknesses were patched.

No cryptanalytic advances have yet been reported against Plantlet that recovers the secret key without the use of side channels. In [HKMZ18], a distinguishing attack against Plantlet was reported that uses data and memory complexity of 2^{61} bits, and time complexity of 2^{55} steps. In [MSS17], a differential fault attack was reported against Plantlet that recovered the secret key using 4 fault injections.

1.1 Contribution and Organization of the Paper

In this paper, we present a key recovery attack on Plantlet that requires a computational complexity of around $2^{76.26}$ encryptions. As mentioned in the abstract, it is first observed that two internal states of Plantlet that differ in the 43rd LFSR location are guaranteed to produce keystream that are either deterministically equal or unequal in 45 locations. Thus, it can, with some probability, be guessed that when 2 segments of keystream blocks that possess the above 45 bit difference is encountered, they have been generated by two internal states that differ in the 43rd LFSR location.

Thereafter the set of polynomial expressions representing each keystream bit is computed and tabulated in an equation bank. If the guess was correct, the secret key is computed by solving the above set of polynomial equations. If not, the attacker reaches some kind of contradiction in the computations, concludes that his guess was incorrect, and starts

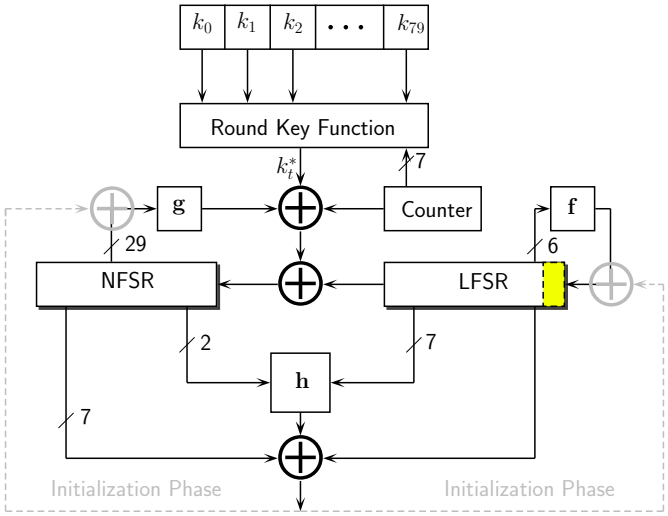

Figure 1: Block Diagram of Plantlet

afresh. We show that the process when repeated a finite number of times, does indeed yield the value of the secret key. After this, we observe that the previous attack was limited to internal state differences that occurred at time instances that were congruent to 0 mod 80. We further observe that by generalizing the attack to include internal state differences that are congruent to all equivalence classed modulo 80, we lower the total number of keystream bits required to perform the attack and in the process reduce the attack complexity. The rest of the paper is organized in the following manner.

1. In Section 2, we present the mathematical description of the Plantlet stream cipher.
2. In Section 3, we introduce some lemmas which serve to lay the mathematical foundations and form the building blocks of the attack.
3. In Section 4, we present a mathematical description of the attack. We clearly outline that stages of the attack serially and estimate the time and memory complexities of each phase. We also present experimental evidence in support of the claims made in the complexity analysis.
4. In Section 5, we show how the attack can be extended to all equivalence classes modulo 80, and hence reduce the attack complexity.
5. Section 6, concludes the paper.

2 Description of Plantlet

The exact structure of Plantlet is explained in Figure 1. It consists of a 61-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 90 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 50 most significant bits of the LFSR. The last 11 bits of the LFSR are initialized with the 11 bit constant `0x7fd`, i.e. the string of nine 1′s followed by 01. Let $L_t = [l_t, l_{t+1}, \ldots, l_{t+60}]$ and $N_t = [n_t, n_{t+1}, \ldots, n_{t+39}]$ be

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 $l_{t+59} = z_t + f(L_t)$ (the last bit is fixed to 1), where

$$f(L_t) = l_t + l_{t+14} + l_{t+20} + l_{t+34} + l_{t+43} + l_{t+54}.$$

(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:

$$k_t^* = K_{t \bmod 80}$$

Here K_i simply denotes the i^{th} bit of the secret key. The non-linear functions $g(N_t)$ and $g'(N_t)$ is given as:

$$\begin{aligned} g(N_t) = {} & n_{t+0} + n_{t+13} + n_{t+19} + n_{t+35} + n_{t+39} + n_{t+2}n_{t+25} + n_{t+3}n_{t+5} + \\ & n_{t+7}n_{t+8} + n_{t+14}n_{t+21} + n_{t+16}n_{t+18} + n_{t+22}n_{t+24} + n_{t+26}n_{t+32} + \\ & n_{t+33}n_{t+36}n_{t+37}n_{t+38} + n_{t+10}n_{t+11}n_{t+12} + n_{t+27}n_{t+30}n_{t+31}. \\ g'(N_t) = {} & g(N_t) + n_t \end{aligned}$$

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 $l_{t+60} = f(L_t)$ and $n_{t+40} = g(N_t) + c_t^4 + k_t^* + l_t$ respectively. Thus the LFSR now behaves as a 61 bit linear register, whereas during key-IV mixing, it was essentially functioning as a 60 bit register. It is recommended by the designers that one single key-IV pair not be used to generate more than 2^{30} keystream bits.

3 Observations on the differential structure of Plantlet

Before outlining the attack on Plantlet, we proceed to list some observations on the structure of Plantlet, that will help us construct the attack. The following observations and lemmas should be seen as building blocks of our attack.

Observation 1: The first of them is as follows: if the secret key is known, then the state update in the keystream phase is one-to-one and efficiently invertible. Before proceeding, we give a formal algorithmic description of the state update inversion routine in the keystream and initialization phase. We denote the algorithm by the notation KS^{-1} .

Lemma 1. *Given two time instances during the keystream phase t_1, t_2 (with $t_2 > t_1$ and both less than 2^{30}), and the 61-bit difference vector $\delta = L_{t_1} \oplus L_{t_2}$. Then it is possible to compute the LFSR states L_{t_1}, L_{t_2} efficiently.*

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 $t-1$;

$l_{t-1} \leftarrow l_{t+60} + f'(L'_{t-1})$;

$n_{t-1} \leftarrow n_{t+39} + l_{t-1} + k^*_{t-1} + c^4_{t-1} + g'(N'_{t-1})$;

$L_{t-1} \leftarrow [l_{t-1}, l_t, l_{t+1}, \ldots, l_{t+59}]$;

$N_{t-1} \leftarrow [n_{t-1}, n_t, n_{t+1}, \ldots, n_{t+38}]$;

Return L_{t-1}, N_{t-1}

Algorithm 1: Algorithm KS^{-1}

Proof. If M is the companion matrix over $GF(2)$ of the connection polynomial $p(x)$ of the LFSR, then we can write L_{t+1} as a matrix-vector product between M and L_t. Thus we have $L_{t+1} = M \cdot L_t$. We thus have $L_{t_2} = M^{t_2 - t_1} \cdot L_{t_1}$. And so we have,

$$\delta = \ L_{t_2} \oplus L_{t_1} = (M^{t_2 - t_1} \oplus I) \cdot L_{t_1}$$

The above is a system of linear equations with the 61 variables in the L_{t_1} vector as unknowns. Further it is known that the minimal polynomial of M is the connection polynomial $p(x)$ of the LFSR itself. Since $p(x)$ is primitive, its roots α^{2^i}, $\forall\ i \in [0, 60]$ are the eigenvalues of M (here α denotes any root of $p(x)$). Define $T = t_2 - t_1$. The eigenvalues of $M^T \oplus I$ are the set $\{1 + \alpha^{T \cdot 2^i},\ \forall\ i \in [0, 60]\}$. If for some i, $1 + \alpha^{T \cdot 2^i} = 0$, this implies $\alpha^{T \cdot 2^i} = 1$. Because α is primitive, this implies that $T \cdot 2^i \equiv 0 \bmod 2^{61} - 1$. Since 2^i is coprime with $2^{61} - 1$, we must have $T \equiv 0 \bmod 2^{61} - 1$, which is a contradiction, since $0 < T < 2^{30}$. Thus $M^T \oplus I$ has nonzero eigenvalues and is hence invertible. So the above system of equations can be solved efficiently by using Gaussian elimination to compute L_{t_1} and hence L_{t_2} . □ □

Lemma 2. *Consider two* Plantlet *internal states $S_{t_1} = (N_{t_1}, L_{t_1})$ and $S_{t_2} = (N_{t_2}, L_{t_2})$ during the keystream phase such that $N_{t_1} = N_{t_2}$ and $L_{t_1} \oplus L_{t_2} = e_{43}$, ($e_i$ is the 61-bit unit hamming weight vector, with 1 at location i). We further impose the condition that $t_1 \neq t_2$ and they are both multiples of 80. Then consider the vectors Z_{t_1} and Z_{t_2} of the first 80 keystream bits generated by S_{t_1} and S_{t_2} respectively. Also consider the vectors Y_{t_1} and Y_{t_2} of the first 80 keystream bits produced by S_{t_1} and S_{t_2} respectively, in the backward direction, i.e. by running the KS^{-1} routine. To be more specific*

$$Z_{t_i} = [z_{t_i+0}, z_{t_i+1}, z_{t_i+2}, \ldots, z_{t_i+79}]$$
$$Y_{t_i} = [z_{t_i-1}, z_{t_i-2}, z_{t_i-3}, \ldots, z_{t_i-80}],\ \textit{for } i = 1, 2.$$

*Then in the 160 bit difference vector $\Delta = Z_{t_1} || Y_{t_1} \oplus Z_{t_2} || Y_{t_2}$, there are **45** bits that take the value 1 or 0 with probability 1, i.e. when the probability is computed over all possible initial states S_{t_1}.*

Proof. The above result is not difficult to verify, if we analyze the differential trail of the difference when introduced in the 43rd LFSR location. First of all since both t_1 and t_2 are multiples of 80, the values of the key bit k^*_t and counter bit c^4_t used in their respective update functions are the same. In the forward direction, for $j \in [0, 10] \cup \{12\} \cup [14, 19] \cup [21, 23] \cup \{25\} \cup [27, 28] \cup \{30, 32, 34, 39, 41\}$, the differences (between the Plantlet LFSR states L_{t_1+j} and L_{t_2+j}) sit on LFSR locations that are not used in the computation of

the keystream bit. Hence for all such j, $z_{t_1+j} = z_{t_2+j}$. Whereas, for $j' \in \{13, 31, 40\}$, the difference appears at tap location l_{30}, that contributes to the keystream equation linearly. For all such j', we have $z_{t_1+j'} \oplus z_{t_2+j'} = 1$, with probability 1. Similarly in the backward direction, for $m \in \{-13\} \cup [-11, -1]$, the differences do not affect keystream equation, hence for all such m, we have $z_{t_1+m} = z_{t_2+m}$. At $m' = -12$, the difference is on the NFSR tap location n_1, which also contributes to the keystream equation linearly. Hence we have $z_{t_1+m'} \oplus z_{t_2+m'} = 1$. There are, in total, 45 time instances where these events take place, hence a total of 45 bits in the difference vector are guaranteed to be either 0 or 1, with probability 1.

Note that the 43rd LFSR bit was chosen as the initial difference location because it maximizes the number of bits in Δ that are deterministically equal to 0 or 1. □ □

Lemma 3. *Consider the same setting as in the previous lemma, i.e. we have two* Plantlet *internal states $S_{t_1} = (N_{t_1}, L_{t_1})$ and $S_{t_2} = (N_{t_2}, L_{t_2})$ during the keystream phase such that $N_{t_1} = N_{t_2}$ and $L_{t_1} \oplus L_{t_2} = e_{43}$, ($e_i$ is the 61-bit unit hamming weight vector, with 1 at location i), with $t_1 \neq t_2$ and they are both multiples of 80. Then consider the vectors Z_{t_1}, Y_{t_1} and Z_{t_2}, Y_{t_2} as defined previously. We have the following identities:*

$$
\begin{aligned}
z_{t_1+11} \oplus z_{t_2+11} &= n_{t_1+15} \cdot n_{t_1+49} \oplus l_{t_1+28} \\
z_{t_1+20} \oplus z_{t_2+20} &= l_{t_1+39} \\
z_{t_1+24} \oplus z_{t_2+24} &= l_{t_1+47} \\
z_{t_1+26} \oplus z_{t_2+26} &= l_{t_1+58} \\
z_{t_1+29} \oplus z_{t_2+29} &= n_{t_1+33} \cdot n_{t_1+67} \oplus l_{t_1+46} \\
z_{t_1+33} \oplus z_{t_2+33} &= l_{t_1+41} \\
z_{t_1+35} \oplus z_{t_2+35} &= l_{t_1+45} \\
z_{t_1+36} \oplus z_{t_2+36} &= n_{t_1+40} \cdot n_{t_1+74} \oplus l_{t_1+53} \\
z_{t_1+37} \oplus z_{t_2+37} &= n_{t_1+41} \\
z_{t_1+38} \oplus z_{t_2+38} &= n_{t_1+42} \cdot n_{t_1+76} \oplus l_{t_1+55} \oplus l_{t_1+57} \oplus 1 \\
z_{t_1+42} \oplus z_{t_2+42} &= l_{t_1+65} \\
z_{t_1+43} \oplus z_{t_2+43} &= n_{t_1+47} \cdot n_{t_1+81} \oplus l_{t_1+60} \\
z_{t_1+44} \oplus z_{t_2+44} &= l_{t_1+76} \\
z_{t_1+46} \oplus z_{t_2+46} &= n_{t_1+50} \cdot l_{t_1+78}
\end{aligned}
$$

Proof. The above lemma is similarly verified by a study of how the single bit difference at LFSR location 43 propagates through the internal state. For $j = 20$, for example, the difference between S_{t_1}, S_{t_2} is at LFSR location 23. Then the sum of the keystream bits z_{t_1+20}, z_{t_2+20} can be essentially expressed as:

$$z_{t_1+20} \oplus z_{t_2+20} = l_{t_1+39} \cdot l_{t_1+43} \oplus l_{t_1+39} \cdot (1 \oplus l_{t_1+43}) = l_{t_1+39}$$

The other expressions can be verified similarly. Note that 7 of the 14 expressions listed above, depend only on L_{t_1}, whereas $z_{t_1+46} \oplus z_{t_2+46}$ consists of a single product term involving an LFSR bit. □

Lemma 4. *In the event that we generate uniformly randomly, N internal states of* Plantlet *$S_{t_i} \in \{0,1\}^{101}$, for $i \in [1, N]$, then the probability that there exists $i_1, i_2 \in [1, N]$, such that, $S_{t_{i_1}} \oplus S_{t_{i_2}} = 0^{40}||e_{43}$, is approximately $p = \frac{N^2}{2^{102}}$. The above can be modeled as a Bernoulli trial with success probability p (where "success" is defined as the event in which we sample two internal states with the given difference). Then by repeating the above experiment (in which we sample internal states randomly) around $\frac{1}{p}$ times, we can expect to obtain one successful event.*

Proof. The first probability value in the lemma is easy to prove, by birthday bound considerations. Given N samples in a domain of size 2^{101}, the probability q that there are no collisions of the required type is given by

$$q = \left(1 - \frac{1}{2^{101}}\right) \cdot \left(1 - \frac{2}{2^{101}}\right) \cdot \left(1 - \frac{3}{2^{101}}\right) \cdots \left(1 - \frac{N-1}{2^{101}}\right)$$
$$\approx 1 - \frac{1 + 2 + \cdots + (N-1)}{2^{101}} = 1 - \frac{N(N-1)}{2^{102}} \approx 1 - \frac{N^2}{2^{102}}$$

Thus $p = 1 - q$ results in the required expression. We repeat the above experiment a number of times. Therefore what we do is as follows:

1. Randomly sample N states and look for the given difference.
2. If the above trial fails, then erase the above samples and repeat step 1.

It is easy to see that the above results in a series of Bernoulli trials with probability of success p. The probability distribution of the number of such trials needed to get one success, is a geometric distribution with mean $\frac{1}{p}$. Hence the second claim in the lemma follows. □

4 Key recovery attack on Plantlet

Having made some preliminary observations about differential structure of Plantlet, we are now ready to mathematically describe the cryptanalytic steps. Note that, in the preceding experiment if $N = 2^{51}$, then by birthday bound, one such trial would be sufficient. However we limit the value of N, because there is a limit to the maximum amount of keystream bits that can be generated using a single key-IV pair, and this limit is 2^{30} bits. The attacker lets the cipher run for 2^{30} cycles and collects the required keystream, with the idea that the cipher during its operation hits two internal states at times t_1, t_2 (both multiples of 80) that differ in only the 43rd LFSR location. If it does, then the attacker can identify the states and the corresponding values of t_1, t_2 by looking at the difference keystream vector $\Delta = Z_{t_1}||Y_{t_1} \oplus Z_{t_2}||Y_{t_2}$ (which was defined in the previous section). But there are obvious obstacles to this idea:

- Firstly given the limited amount of keystream bits one is allowed to generate with one key-IV pair, by Lemma 4, it is extremely unlikely that the attacker will actually encounter two states with the required difference. Hence the attacker must repeat the experiment with the same key and some other randomly selected IV multiple number of times. Lemma 4, also enumerates the number of times (i.e. $\frac{1}{p}$) the experiments need to be repeated to get a success.
- Second, although it is true that two internal states with difference only at 43rd LFSR location, produces keystream bits whose differential is guaranteed to be 0 or 1 at 45 fixed locations, the opposite is not true. In fact there exist, with probability around 2^{-45}, two completely random internal states of Plantlet that produce a keystream differential of 0/1 at the same 45 locations enumerated in Lemma 2. Thus the attacker, when for some (t_1, t_2), observes a differential keystream having required 0/1 pattern in the locations enumerated in Lemma 2, may still proceed to the next steps, assuming that they were generated by two Plantlet states with difference in the 43rd LFSR bit. But if his assumption about the state difference is wrong, then in the subsequent steps he would certainly reach a contradiction that would invalidate the assumption. The attacker would then require to repeat the experiment to obtain some other t_1, t_2 until he is successful in getting internal states with required difference.

- Thus any attack, must compensate for these computational overheads listed above.

Thus, at the very top level, the strategy of the attacker will be as follows:

A: Generate 2^{30} keystream bits with the given secret key and any random IV. This generates $N = \lfloor \frac{2^{30}}{80} \rfloor \approx 2^{23.7}$ keystream segments of length 80-bits each.

B: For all $t = 80 \cdot i$ where $i \in [1, N-1]$, store in a hash table t, Z_t, Y_t as defined.

C: From this table, try to find, if it exists, t_1, t_2 so that $\Delta = Z_{t_1}||Y_{t_1} \oplus Z_{t_2}||Y_{t_2}$ exhibits the 1/0 pattern in the locations listed in Lemma 2. We refer to such an event as a keystream-collision.

D: If there exists one or multiple such t_1, t_2, then assuming that the state differential in between the states at time t_1, t_2 is $0^{40}||e_{43}$, try to solve for the remaining system of equations to find the key.

E: If a contradiction is reached, try other values of t_1, t_2, if they exist. If none exist then repeat step **A** with another IV. If the attacker does not encounter a contradiction, and is able to solve the equations, he would have computed the secret key. By Lemma 4, $\frac{1}{p} = 2^{54.6}$ such trials should be sufficient to solve for the secret key.

We now try to explain the finer details of the attack, starting with a precomputation step that would ease the computational burden in the online stage of the attack.

4.1 Precomputation Stage

If the attacker encounters a keystream-collision, regardless or not whether it was generated by two states with a single bit difference at the 43rd LFSR location, he will need to try to solve the resulting system of polynomial expressions for each keystream bit in $Z_{t_1}, Z_{t_2}, Y_{t_1}$ and Y_{t_2}, assuming that the corresponding internal state difference is $0^{40}||e_{43}$. These are a system of boolean polynomials in 181 variables over GF(2) (40 for the NFSR state, 61 for the LFSR state, and 80 for the key). Such a system should generally be intractable to solve. However the attacker can use the results in Lemma 1, to get the value of the states L_{t_1} and L_{t_2}, since a keystream-collision, would automatically provide the values of t_1, t_2. He simply solves the equation $e_{43} = (M^{t_2-t_1} \oplus I) \cdot L_{t_1}$ to compute L_{t_1} and then after that computes $L_{t_2} = L_{t_1} \oplus e_{43}$. Once the entire LFSR states at times t_1, t_2 are known, the resulting equation system is now defined over 120 unknowns which is much easier to solve by using any publicly available equation solver.

However since $T = t_2 - t_1$, is the only varying parameter in the equation $e_{43} = (M^{t_2-t_1} \oplus I) \cdot L_{t_1}$, one can pre-solve the above set of equations for all possible values of T. Note that $1 \leq t_1 < t_2 \leq 2^{30}$ and since t_1, t_2 are multiples of 80, there are only around $N - 1 = \lfloor \frac{2^{30}}{80} \rfloor - 1 \approx 2^{23.7}$ different values of T. Each equation can be solved offline, and the solutions stored in a table sorted along with the value of T. Thus in this way, in the online stage, finding the value of L_{t_1}, L_{t_2} from the values of t_1, t_2 amounts to only a table lookup.

The total computational complexity in the offline stage amounts to solving $N-1 \approx 2^{23.7}$ equations over 61 variables. Assuming conservatively, that it takes $O(n^3)$ steps to do Gaussian elimination to solve the system, the total number of steps involved is bounded by $61^3 \cdot 2^{23.7} \approx 2^{41.5}$. It takes 61 bits to store the solution of the equation (in the table cell indexed by T) and so the memory complexity of this stage is $61 \cdot 2^{23.7} \approx 2^{29.6}$ bits.

4.2 Online Stage I: Collecting and storing keystream bits

In the online stage, the attacker needs to collect and store keystream bits and store it in a judicious manner. Note that since $N = \lfloor \frac{2^{30}}{80} \rfloor \approx 2^{23.7}$, the value of p calculated in Lemma

4 is around $p = 2^{-54.6}$ and so the number of IVs we need to try is around $V = \frac{1}{p} = 2^{54.6}$. For each such IV, the attacker proceeds to generate N keystream bits.

To facilitate detection of keystream-collision, one must choose a data structure to efficiently store keystream segments. For all $t = 80 \cdot i$ where $i \in [1, N-1]$, the attacker has to store in a hash table t, Z_t, Y_t as defined in Lemma 2. However it is unwise to insert the tuple into the table location indexed by t. Instead we insert the tuple in the table location $I = z_{t+g_0}||z_{t+g_2}||\cdots||z_{t+g_{44}}$, where the g_i's are the locations enumerated in Lemma 2, where the differential keystream is guaranteed to be 0/1. Thus $(g_0, g_1, \ldots, g_{40}) = (0, 1, 2, \ldots, 10, 12, 14, 15, \ldots, 19, 21, 22, 23, 25, 27, 28, 30, 32, 34, 39, 41, -13, -11, -10, \ldots, -1)$ are the locations where the difference is 0, and $(g_{41}, g_{42}, g_{43}, g_{44}) = (13, 31, 40, -12)$ are locations where the difference is 1. Note that each entry in the table should be able to store multiple entries. It is not difficult to see that a keystream-collision will occur if during an insertion into index I, the attacker checks the index $I^* = z_{t+g_0}||z_{t+g_1}||\cdots||z_{t+g_{40}}||1 \oplus z_{t+g_{41}}||1 \oplus z_{t+g_{42}}||\cdots||1 \oplus z_{t+g_{44}}$, and finds one or multiple tuples already stored at I^*. For each such keystream-collision pair in (I, I^*), the attacker proceeds to the next steps of the attack.

It takes 30 bits to store t and 160 bits to store Z_t, Y_t and so each IV trial takes around $190 \cdot N \approx 2^{31.25}$ bits of memory on average.

4.3 Online Stage II: Further filtering

For each keystream-collision pair obtained in the previous step, the attacker can perform further filtering. First, let t_1, Z_{t_1}, Y_{t_1} and t_2, Z_{t_2}, Y_{t_2} be a pair filtered from the previous stage. The attacker can then compute $t_2 - t_1$, and retrieve the value of L_{t_1} from the precomputed table. By Lemma 3, there are 8 other bits in $Z_{t_1} \oplus Z_{t_2}$ that are directly related to L_{t_1}. Since during the keystream stage the LFSR evolves independently, all l_{t_1+i} can be computed with the knowledge of L_{t_1} alone. This provides us with an opportunity to further filter the keystream-collision pairs obtained from the stage. For example

1. If, the attacker finds that $z_{t_1+20} \oplus z_{t_2+20} \neq l_{t_1+39}$ he can reject the pair.
2. Also if, $l_{t_1+78} = 0$ and the attacker finds that $z_{t_1+46} \oplus z_{t_2+46} = 1$, such a pair can also be rejected.

So, let us calculate the probability that a given IV produces a keystream-collision pair that survives both the filter levels described above. Note that since a single IV can produce N tuples, the total number of pairs of tuples are $D = \frac{N(N-1)}{2} \approx 2^{46.36}$. Denote $\alpha_i = z_{t_1+g_i} \oplus z_{t_2+g_i}$ and also define the following notations:

- $\beta_0 = z_{t_1+20} \oplus z_{t_2+20} \oplus l_{t_1+39}$, $\beta_1 = z_{t_1+24} \oplus z_{t_2+24} \oplus l_{t_1+47}$
- $\beta_2 = z_{t_1+26} \oplus z_{t_2+26} \oplus l_{t_1+58}$, $\beta_3 = z_{t_1+33} \oplus z_{t_2+33} \oplus l_{t_1+41}$
- $\beta_4 = z_{t_1+35} \oplus z_{t_2+35} \oplus l_{t_1+45}$, $\beta_5 = z_{t_1+42} \oplus z_{t_2+42} \oplus l_{t_1+65}$
- $\beta_6 = z_{t_1+44} \oplus z_{t_2+44} \oplus l_{t_1+76}$, $\beta_7 = z_{t_1+46} \oplus z_{t_2+46}$

The probability that a pair is not rejected is given as

$$\rho = \prod_{i=0}^{40} Pr(\alpha_i = 0) \cdot \prod_{i=41}^{44} Pr(\alpha_i = 1) \cdot \prod_{i=0}^{6} Pr(\beta_i = 0) \cdot \Big(1 - Pr(\beta_7 = 1 \ \& \ l_{t_1+78} = 0)\Big)$$
$$= 2^{-41} \cdot 2^{-4} \cdot 2^{-7} \cdot \frac{3}{4} = 2^{-52.41}$$

In the above calculation, we have assumed that the events $Pr(\beta_7 = 1)$ and $Pr(l_{t_1+78} = 0)$ are statistically independent, but under this situation this is a fair assumption to make.

Let X_{t_1,t_2} be the indicator variable that is 1 when the tuples at t_1, t_2 are not rejected by the filters, and zero otherwise. Then we have shown that $E(X_{t_1,t_2}) = \rho$. Let P_s be the expected number of pairs that survive during processing keystream generated a single IV. We have

$$\begin{aligned} P_s &= \sum_{i=1}^{N} \sum_{j=i+1}^{N} E[X_{i,j}] \\ &= \binom{N}{2} \cdot \rho \\ &= D \cdot \rho = 2^{-6.06} \end{aligned}$$

Thus the total number of pairs that survive trials with V different IVs is given as $P_u = V \cdot P_s = 2^{48.54}$. This is the number of pairs that proceed to the next stage of the attack.

4.4 Online Stage III: Solving Equation System

The final stage of the attack involves attempting to solve the equation system resulting from the keystream segment pairs. The attacker has to try to solve the P_u sets of equations assuming that they were generated by two Plantlet states that differ by $0^{40}||e_{43}$. Most of the times the assumption is wrong, so that a contradiction is arrived at. However the value of V has been chosen so that the attacker encounters, on average, at least one state pair with the required difference, which he can solve to find the secret key.

We construct the equation system over the polynomial ring $\mathbb{Z}_2[N, K]$, where $N = \{n_0, \ldots, n_{39}\}$, and $K = \{k_0, \ldots, k_{79}\}$, where the variables k_i correspond to the bits of the key, and the variables n_i correspond to the bits of the NFSR. As explained in Section 4.3, if Z_{t_1}, Y_{t_1} and Z_{t_2}, Y_{t_2} satisfy the filtering criteria, we assume that $N_{t_1} = N_{t_2}$, and $L_{t_1} = L_{t_2} \oplus e_{43}$. We can compute the value of L_{t_1}, and L_{t_2}, from the precomputed tables. So let's assume $L_{t_1} = (l_0, \ldots, l_{60})$, and $L_{t_2} = (l_0, \ldots, l_{60}) + e_{43}$. We can now generate the polynomial expressions for each keystream bit, by considering the content of the LFSR to be the 61 bit string $(l_0, \ldots, l_{60})$ over $GF(2)$, the content of the NFSR to be the boolean variables $(n_0, \ldots, n_{39})$, and the key denoted by the boolean variables $(k_0, \ldots, k_{79})$, and do all the computations in $R = \mathbb{Z}_2[N, K]$. Let the polynomial expressions generated this way be $(z^*_{t_1}, \ldots, z^*_{t_1+79})$, where all the entries are polynomials with unknowns in $K \cup N$. Now we consider the equations of form $z^*_{t_1+i} \oplus z_{t_1+i} = 0$ for $i \in [0, 79]$ and add them to the equation system. We can also do the same for the stream generated from time t_2, by loading the LFSR with the initial value $(l_0, \ldots, l_{60}) + e_{43}$, and the NFSR with the same variables $(n_0, \ldots, n_{39})$ and construct the equations of form $z^*_{t_2+i} \oplus z_{t_2+i} = 0$ for $i \in [0, 79]$.

We also generate the polynomial expressions for the keystream bits in the backward direction $(z^*_{t_1}, z^*_{t_1-1}, \ldots, z^*_{t_1-79})$ and $(z^*_{t_2}, z^*_{t_2-1}, \ldots, z^*_{t_2-79})$ with the same initial register values, and add the equations of form $z^*_{t_1-i} \oplus z_{t_1-i} = 0$ and $z^*_{t_2-i} \oplus z_{t_2-i} = 0$, to our system. This way we will have 4×80 equations over 120 unknowns.[1] As we are only looking for solutions in $\mathbb{Z}_2$, we can consider this system a SAT problem. We feed the system of equations to a SAT solver. For an incorrect assumption on the states generating a given differential keystream, a SAT based solver returns UNSAT, which is to say the system of equations fed to it are inconsistent. Thus this gives us an efficient method to arrive at a contradiction and reject an incorrect guess of initial state difference.

[1] In the real implementation we also add a lot of intermediate variables in order to control the degree of the polynomials.

4.5 Experimental Results

The computations in this section were done using the computer algebra software SAGE 8.7 [Dev17], and we used the Cryptominisat 5.0.1 [SNC09] package for solving the underlying equation system. All experiments were done on an AMD Opteron 8354 processor with CPU speed of 2200 MHz running on Ubuntu 14.04.6 LTS. We ran three experiments:

1. First we estimated the time it takes the SAT solver to return UNSAT, i.e. when the attacker incorrectly assumes that a given differential keystream (which satisfies all filter requirements in Section 4.3) is produced by two Plantlet states differing only by $0^{40}||e_{43}$. Note that most of the times (around $P_u - 1$ times), the attacker will have to face this situation, and hence it is important to measure the computational cost of this task.

2. Second we estimated the time it takes the SAT solver to return SAT, i.e. when the attacker correctly assumes that a given differential keystream (which satisfies all filter requirements in Section 4.3) is produced by two Plantlet states differing only by $0^{40}||e_{43}$. When this event occurs, the attacker would have successfully computed the value of the secret key.

3. Finally, we estimated the amount of time needed to perform one Plantlet encryption. This step is important because this way we can estimate the computational cost of solving an equation in terms of the computational cost of an encryption. Since there is no straightforward way to compute the number of steps taken by the solver to solve a given polynomial system, there is no good way of comparing the computational costs of solving an equation and performing one encryption. Due to this fact, many papers [ZLFL14, MAM16] in the past have measured the physical time to perform the above tasks to make a comparison. In [MAM16], in order to estimate the computational complexity of guess and determine attacks, the authors had measured the time of performing one encryption and concluded that it was possible to perform around 2^{10} encryptions per second on their system. Using this fact and after experimentally finding the average physical time required to solve a particular set of equations, they had concluded that guess and determine attacks on Plantlet did not perform better than a brute force attack. We adopt a similar method to estimate the bounds we present in this paper.

First we estimate the time it takes the SAT solver to return UNSAT. For this, we randomly generate a pair of keystream segments (Z_{t_1}, Y_{t_1} and Z_{t_2}, Y_{t_2}) of length 160 bits each and a 61 bit initial LFSR state L such that they satisfy all the filtering criterion in Section 4.3. Using the variables N, K and the bit-string L the polynomial expressions for Z_{t_1}, Y_{t_1} are computed. Similarly, using the variables N, K and the bit-string $\mathsf{L} \oplus e_{43}$ we generate the polynomial expressions for Z_{t_2}, Y_{t_2}. The polynomials and keystream bits generated earlier form the left and right sides of an equation bank we make. Since the keystream bits and polynomial equations were generated randomly and independent of each other, the system of equations when fed to a SAT solver, will with a high probability make the solver return UNSAT. Note that we have set up the above system of equations in a manner so as to simulate the event when the attacker observes a differential keystream that satisfies all filtering requirements and incorrectly assumes that the bits were generated by two states differing in the 43rd LFSR location.

While doing the the experiments, we found that the solver returns a SAT/UNSAT verdict faster when the last 4 bits of the NFSR are additionally guessed. So we essentially solve the system of equations over 116 unknowns. We ran the above set of experiments for 1000 randomly generated samples, and the results are presented in Figure 2. The figure is a probability distribution histogram of the time taken for the solver to return UNSAT. The x-axis represents the time taken in seconds and the y-axis the corresponding probability of

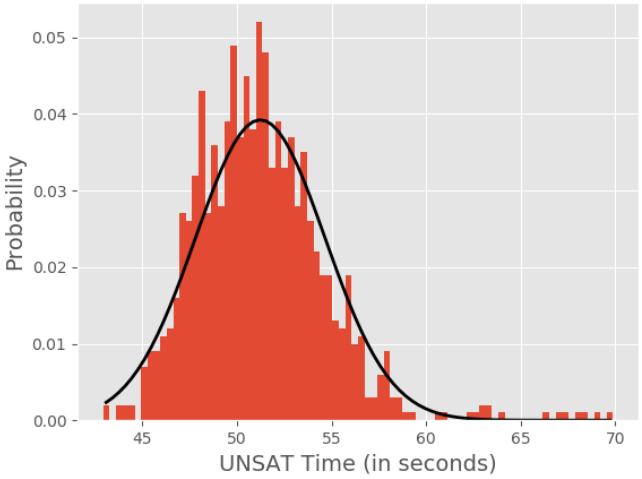

Figure 2: Histogram of the SAT solver abort time

occurrence. To plot the figure, we divided the x-axis range into 70 equal bins and counted the number of times, the physical time taken for the solver to return UNSAT occurred in the time range corresponding to each bin. The probability density function was close to a normal distribution (as indicated by the black curve), with mean $\mu_{UNS} = 51.22$ seconds, and standard deviation $\sigma_{UNS} = 3.42$ seconds.

Second we estimate the time it takes the solver to return SAT, i.e. when the attacker correctly predicts the state differential. For this, we randomly generate a 101-bit initial state N, L and a 80 bit key K. We generate Z_{t_1}, Y_{t_1} using N, L, K and Z_{t_2}, Y_{t_2} using N, $\mathsf{L} \oplus e_{43}$, K. Thereafter, we generate the polynomial expressions for Z_{t_1}, Y_{t_1} using the variables N, K and the bit string L. Similarly, we generate the polynomial expressions for Z_{t_2}, Y_{t_2} using the variables N, K and the bit string $\mathsf{L} \oplus e_{43}$. Again, an equation bank is created with the expressions and keystream bits in the left and right sides. Since the keystream bits and polynomial equations were generated consistent with each other, the system of equations when fed to a SAT solver, will return the correct solution $N = \mathsf{N}$ and $K = \mathsf{K}$. Note that we have set up the above system of equations in a manner so as to simulate the event when the attacker observes a differential keystream that satisfies all filtering requirements and correctly assumes that the bits were generated by two states differing in the 43rd LFSR location.

Again the last 4 bits of the NFSR are additionally guessed for getting faster solutions. We ran the above set of experiments for 1000 randomly generated samples, and the results are presented in Figure 3. The probability density function was again close to a normal distribution (as indicated by the black curve), with mean $\mu_{SAT} = 66.17$ seconds, and standard deviation $\sigma_{SAT} = 39.26$ seconds.

Third, we also computed the time it takes to perform a single encryption. As argued in [EK15, Ban15] one Plantlet encryption should be equal to the average number of Plantlet rounds required to be executed per trial with a guessed value of the key (in a brute force search). This comes to 320 initialization rounds and 4 rounds in the keystream generation phase. We have given a proof of this in Appendix A (at the end of this paper). Thus one Plantlet round is equivalent to around $\frac{1}{324} = 2^{-8.34}$ Plantlet encryptions. We measured the time to perform 320 initialization and 4 keystream generation rounds (for around 10,000 random Key-IV samples) and the results are presented in Figure 4. As expected the distribution is close to normal, with a mean of $\mu_{ENC} = 0.0057$ seconds, and

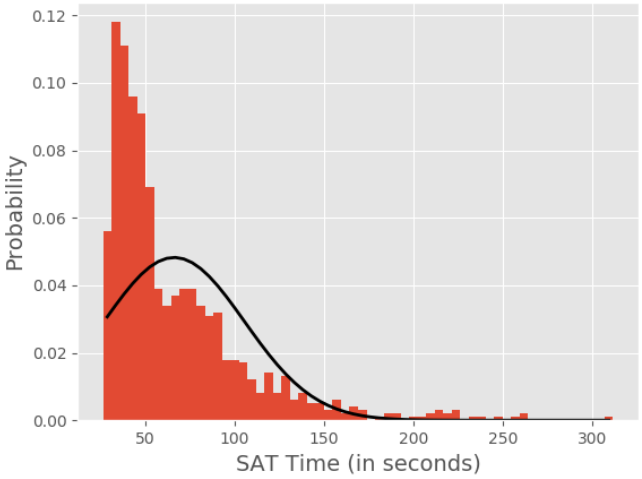

Figure 3: Histogram of the SAT solver runtime

the standard deviation is $\sigma_{ENC} = 0.000699$ seconds. We have provided SAGE codes used by us as auxiliary material attached to the paper. Note that in Grain-like designs, it is possible to optimize the encryption speed, by computing multiple rounds in a single clock cycle. For example, Grain v1 does not use any of the last 16 bits of both the linear and non-linear registers as inputs to the update or the keystream generating functions. As a result a 16 times speedup in software is possible by doing 16 round updates in one iteration. However that is not the case with Plantlet, as even the last NFSR bit is used in the non-linear update function g.

As pointed out earlier, the main aim of the previous experiment was to compare the cost of solving an equation and performing an encryption, so as to find the equivalent computational cost of solving P_u equations in terms of Plantlet encryptions. Since we guess 4 bits of NFSR, the computational cost of returning UNSAT can be estimated on average to be $C_u = \frac{2^4 \cdot \mu_{UNS}}{\mu_{ENC}} \approx 2^{17.13}$ Plantlet encryptions. Similarly the cost of returning SAT is around $C_s = \frac{2^4 \cdot \mu_{SAT}}{\mu_{ENC}} \approx 2^{17.5}$ Plantlet encryptions.

4.6 Total Complexity of attack

The total time complexity in the precomputation stage has been already calculated in Section 4.1 as bounded by $2^{41.5}$ bit operations. The online complexity is dominated by three computational tasks.

1. First, is the task of generating the requisite amount of keystream to perform the attack. We need to generate 2^{30} keystream bits from $V = 2^{54.6}$ IVs. This implies performing a total of $(320 + 2^{30}) \cdot 2^{54.6}$ Plantlet iterations. It has already been argued that each Plantlet iteration is computationally equivalent to $2^{-8.34}$ Plantlet encryptions. Hence the total computational complexity required to generate keystream is $(320 + 2^{30}) \cdot 2^{54.6-8.34} = 2^{76.26}$ Plantlet encryptions.

2. Second is the time required to solve equations. A total of $P_u = 2^{48.54}$ equations need to be solved. Only one of these equations are expected to yield the correct solution for the secret key. In the previous section we have argued that the time to solve an equation unsuccessfully (i.e. yielding UNSAT from the solver) is computationally equivalent to $C_u = 2^{17.13}$ encryptions, whereas to solve successfully is equivalent

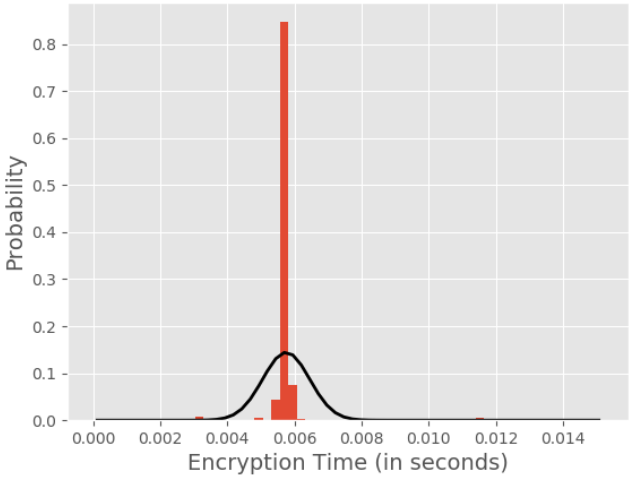

Figure 4: Histogram of the encryption runtime

to $C_s = 2^{17.5}$ encryptions. Thus the computational burden for this task is around $(P_u - 1) \cdot C_u + C_s \approx 2^{65.7}$ encryptions.

3. The total memory access is dominated by the number of table insertions done in the online stage of the attack. We need a total of $N \cdot V = 2^{54.6+23.7} = 2^{78.3}$ table insertions. Note that any point of time of the attack we do not need more than $2^{31.25} + 2^{29.6} \approx 304$ MB of memory. Thus the tables can be stored in the primary memory and accessed reasonably quickly. We do not have a method to reliably compare this to the number of encryptions, but $2^{78.3}$ memory accesses is not likely to take more time than $2^{76.26}$ Plantlet encryptions, by any fair estimation.

Thus the computational complexity is dominated by the task of generating keystream and equal to $2^{76.26}$ Plantlet encryptions. The memory required for the storage of the precomputed tables has been already calculated to be around $2^{29.6}$ bits. Other than that, for each IV trial in the online stage, we already shown that around $2^{31.25}$ bits are required.

5 Improving Attack Complexity

Based on the observations in the above section, we can propose a more efficient key recovery attack on Plantlet. First of all we note that one of the conditions of Lemma 2 and 3 is that both t_1 and t_2 are multiples of 80. It can be easily seen that both the lemmas also hold if we relax the conditions on t_1, t_2 to $t_1 \equiv t_2 \bmod 80$. This is because if $t_1,\ t_2$ belong to the same equivalence class modulo 80, then the sequence of key and counter bits used to update $S_{t_1} = (N_{t_1}, L_{t_1})$ and $S_{t_2} = (N_{t_2}, L_{t_2})$ are the same. Thus the differential evolution of S_{t_1}, S_{t_2} is independent of key and counter bits and hence the lemmas naturally hold.

Thus, we can see the attack in the previous section as limited to the equivalence class 0 mod 80. Thus one can naturally try to improve the attack complexity by extending the attack to all equivalence classes 0 mod 80. We will see how subtly tweaking the above steps can lead to a more efficient attack.

5.1 Precomputation Step

For any equivalence class, the number of values of $t_1 - t_2$ remain the same. In fact the exact values of $t_1 - t_2$ also remain the same since they are multiples of 80. So any additional precomputation steps are not required to extend the attack to other equivalence classes.

5.2 Online stage

Thus, at the very top level, the modified strategy of the attacker will be as follows:

A: Generate 2^{30} keystream bits with the given secret key and any random IV. This generates $N = \lfloor \frac{2^{30}}{80} \rfloor \approx 2^{23.7}$ keystream segments of length 80-bits each.

B: Take all equivalence classes j mod 80, for $0 \leq j \leq 79$. For all values of j:

- For all values $t = j + 80 \cdot i$ where $i \in [1, N-1]$, store in a hash table t, Z_t, Y_t.
- From this table, try to find, if it exists, t_1, t_2 so that $\Delta = Z_{t_1}||Y_{t_1} \oplus Z_{t_2}||Y_{t_2}$ exhibits the 1/0 pattern in the locations listed in Lemma 2, i.e. is a keystream-collision.
- If there exists one or multiple such t_1, t_2, then assuming that the state differential in between the states at time t_1, t_2 is $0^{40}||e_{43}$, try to solve for the remaining system of equations to find the key.
- If a contradiction is reached, try other values of t_1, t_2, j, if they exist. If none exist then repeat step **A** with another IV. If the attacker does not encounter a contradiction, and is able to solve the equations, he would have computed the secret key.

Thus the high level strategy for the modified attack, is the same as in the previous section, with the only exception that we do the attack for t_1, t_2 belonging to all the equivalence classes mod80. The only thing remaining to do is to re-evaluate the attack complexities and memory requirements. We do it in the following steps:

1. In Lemma 4, we know that to hit upon two internal states with the difference $0^{40}||e_{43}$, within one trial consisting of N random sampling of internal states, we need $\frac{1}{p} = \frac{2^{102}}{N^2}$ trials. Since $N \approx \frac{2^{30}}{80} = 2^{23.7}$, the number of trials is around $2^{54.6}$.

2. However each IV can produce keystream required to conduct 80 such trials, one for each equivalence class modulo 80. Hence the number of different IVs required is $\frac{2^{54.6}}{80} \approx 2^{48.32}$.

3. The online memory complexity need not change: since the attacker can construct the tables described in Section 4.2 sequentially for each equivalence class modulo 80. However the attacker choose to construct tables for all the equivalence class once he receives keystream corresponding a single IV. This only increases the online memory complexity by a factor of 80.

4. What does not change however is the total number of trials that need to be performed for encountering 2 states with the required difference which still stands at $2^{54.6}$. The only difference is that we need to produce keystream with much lesser number of different IVs. Because of this the remaining online complexities related to solving equations does not change.

5. **Online Complexity** : We recalculate the online complexity under the 3 heads:

a) Generating Keystream: We need 2^{30} to generate keystream bits from $2^{48.32}$ IVs. This implies performing a total of $(320 + 2^{30}) \cdot 2^{48.32}$ Plantlet iterations. It has already been argued that each Plantlet iteration is computationally equivalent to $2^{-8.34}$ Plantlet encryptions. Hence the total computational complexity required to generate keystream is $(320 + 2^{30}) \cdot 2^{48.32-8.34} = 2^{69.98}$ Plantlet encryptions.

b) Solving Equations: Since the total number of trials does not change, all the attack procedure described in Sections 4.3,4.4 do not change. A total of $P_u = 2^{48.54}$ equations satisfy both filtering levels and need to be solved. Thus the computational burden for this task is around $(P_u - 1) \cdot C_u + C_s \approx 2^{65.7}$ encryptions.

c) Memory access: The total memory access is proportional to the number of trials. Hence this complexity too does not change. We need a total of $2^{54.6+23.7} = 2^{78.3}$ table insertions.

Thus the dominant time complexity is the one required to generate keystream and is around $2^{69.98}$ Plantlet encryptions.

6 Conclusion

In this paper, we propose a key recovery attack on the Plantlet stream cipher. The first attack requires 2^{30} keystream bits to be generated with the secret key and $2^{54.6}$ randomly chosen IVs. This is computationally equivalent to performing $2^{76.26}$ Plantlet encryptions. The attack takes advantage of the sparse locations of bits tapped from the LFSR which are used as inputs to the filter function producing the keystream bit. As a result, two Plantlet states that differ in the 43rd LFSR location are guaranteed to produce keystream that are either equal or unequal in 45 locations with probability 1. This enables us to get a reasonably reliable probabilistic mapping from a differential keystream with a given pattern to a given difference in the internal state. Using precomputed tables, we can probabilistically extract the LFSR part of the two states, once we encounter a differential keystream with the required 0/1 pattern. We then try to solve for the remainder of the state and the secret key. The process, if repeated with $2^{54.6}$ randomly chosen IVs, each generating 2^{30} keystream bits, is expected to give us the correct value of secret key at least once.

In the second part of the paper, we observe that the previous attack was limited to values of t_1, t_2 limited to the equivalence class $0 \mod 80$. We extend the scope of the attack to all equivalence classes modulo 80. This requires the attacker to generate keystream from much lesser number IVs, and reduces the online complexity to $2^{69.98}$ Plantlet encryptions.

Acknowledgments

Subhadeep Banik is supported by the Ambizione Grant no. PZ00P2_179921, awarded by the Swiss National Science Foundation (SNSF). Takanori Isobe is supported by Grant-in-Aid for Scientific Research (B) (KAKENHI 19H02141) for Japan Society for the Promotion of Science.

References

[AM15] Frederik Armknecht and Vasily Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. In Gregor Leander, editor, *FSE*, volume 9054 of *Lecture Notes in Computer Science*, pages 451–470. Springer, 2015.

[Ban15] Subhadeep Banik. Some Results on Sprout. In Alex Biryukov and Vipul Goyal, editors, *INDOCRYPT*, volume 9462 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2015.

[BD08] Steve Babbage and Matthew Dodd. The MICKEY Stream Ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 191–209. Springer, 2008.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[Dev17] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.0)*, 2017. `http://www.sagemath.org`.

[EK15] Muhammed F. Esgin and Orhun Kara. Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks. In Orr Dunkelman and Liam Keliher, editors, *SAC*, volume 9566 of *Lecture Notes in Computer Science*, pages 67–85. Springer, 2015.

[est08] The ECRYPT Stream Cipher Project. *eSTREAM Portfolio of Stream Ciphers.*, September 2008.

[HJM07] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *IJWMC*, 2(1):86–93, 2007.

[HJMM06] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A Stream Cipher Proposal: Grain-128. In *2006 IEEE International Symposium on Information Theory*, pages 1614–1618, July 2006.

[HKM17] Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD - A Lightweight Stream Cipher for Power-constrained Devices. *IACR Trans. Symmetric Cryptol.*, 2017(1):45–79, 2017.

[HKMZ18] Matthias Hamann, Matthias Krause, Willi Meier, and Bin Zhang. Design and analysis of small-state grain-like stream ciphers. *Cryptography and Communications*, 10(5):803–834, Sep 2018.

[LNP15] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of Full Sprout. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO (1)*, volume 9215 of *Lecture Notes in Computer Science*, pages 663–682. Springer, 2015.

[MAM16] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On Ciphers that Continuously Access the Non-Volatile Key. *IACR Trans. Symmetric Cryptol.*, 2016(2):52–79, 2016.

[MSS17] Subhamoy Maitra, Akhilesh Siddhanti, and Santanu Sarkar. A differential fault attack on plantlet. *IEEE Trans. Computers*, 66(10):1804–1808, 2017.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, pages 244–257, 2009.

[ZG15] Bin Zhang and Xinxin Gong. Another Tradeoff Attack on Sprout-Like Stream Ciphers. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 561–585. Springer, 2015.

[ZLFL14] Bin Zhang, Zhenqi Li, Dengguo Feng, and Dongdai Lin. Near collision attack on the grain v1 stream cipher. In Shiho Moriai, editor, *Fast Software Encryption*, pages 518–538, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

A Cost of executing one round of Plantlet [EK15]

To do an exhaustive search, first an initialization phase has to be run for 320 rounds, and then generate 80-bits of keystream to do 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. This means that in the process of brute force search, the probability that for any random key, $(i+1)$ Plantlet keystream phase rounds need to be run, is $\frac{1}{2^i}$. Hence, the expected number of Plantlet rounds per trial is

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

Add to this the 320 rounds in the initialization phase, the average number of Plantlet rounds per trial is 324. As a result, we will assume that clocking the registers once will cost roughly $\frac{1}{320+4} = 2^{-8.34}$ encryptions.