A Differential Fault Attack on Plantlet

Subhamoy Maitra, Akhilesh Siddhanti

***Abstract*—Lightweight stream ciphers have received serious attention in the last few years. The present design paradigm considers very small state (less than twice the key size) and use of the secret key bits during pseudo-random stream generation. One such effort, Sprout, had been proposed two years back and it was broken almost immediately. After carefully studying these attacks, a modified version named Plantlet has been designed very recently. While the designers of Plantlet do not provide any analysis on fault attack, we note that Plantlet is even weaker than Sprout in terms of Differential Fault Attack (DFA). Our investigation, following the similar ideas as in the analysis against Sprout, shows that we require only around 4 faults to break Plantlet by DFA in a few hours time. While fault attack is indeed difficult to implement and our result does not provide any weakness of the cipher in normal mode, we believe that these initial results will be useful for further understanding of Plantlet.**

***Index Terms*—Cryptanalysis, Fault Attack, Plantlet, Stream Cipher.**

I. Introduction

A new light-weight stream cipher has been introduced by Mikhalev, Armknecht and Müller [22], which claims to be built upon by taking care of various cryptographic weaknesses of Sprout [4]. Called Plantlet, it involves the secret key bits both in key-scheduling phase and in the pseudo-random bit generation phase. This unique feature has been carried over from Sprout, and sets it apart from most of the common stream ciphers. The designers of such proposals claim that this feature allows lower state sizes against Time-Memory-Data Trade-Off (TMDTO) attack and hence saves resources.

Sprout, the predecessor of Plantlet, was heavily attacked in many papers and serious concerns were voiced regarding its design. A key-recovery based on divide-and-conquer technique [14] and a TMDTO attack [13] are two important cryptanalytic results on the cipher. Additionally, one may refer to [15], [2], [8], [27] for further cryptanalytic results. Although Sprout was attacked immediately, it was an initial attempt in realization of lightweight stream ciphers with the secret key stored in non-volatile memory. Plantlet uses an 80-bit key, a 90-bit initialization vector, two shift registers, namely LFSR and NFSR, a counter and a Boolean function. Like other stream ciphers, Plantlet has an initial Key Loading Algorithm, followed by a Key Scheduling Algorithm (involving the secret key bits) and then finally the Pseudo-Random Generation Algorithm (again involving secret key bits) for providing the key stream.

S. Maitra is with Indian Statistical Institute, 203 B T Road, Kolkata 700 108, India. E-mail: subho@isical.ac.in

A. Siddhanti is with BITS Pilani KK Birla Goa Campus, Zuarinagar 403726, Goa, India. E-mail: akhileshsiddhanti@gmail.com

The designers of Sprout [4] claimed the security of the cipher against fault attacks, though it was disproved immediately [2]. However, for Plantlet [22] no specific comment has been made in terms of fault attack. In this paper, we present a Differential Fault Attack (DFA) against Plantlet in the same line as against Sprout in [2]. The attack succeeds with only 4 random faults in contrast with around 120 faults in random locations (20 faults, if the locations are known) for Sprout [2]. For Plantlet we first obtain the fault locations from respective signatures. Then we exploit the differential key streams using a SAT solver to obtain the entire states of LFSR and NFSR. Then one can immediately discover the secret key.

Before proceeding any further, let us describe Plantlet in detail.

A. Description of Plantlet

Plantlet is an improvisation over Sprout, which adapts from the Grain family of stream ciphers, more specifically Grain128a [1]. Table 1 compares the various stream ciphers.

Cipher	Key size	IV size	State size	Initialization rounds
Plantlet	80	90	101 (61 LFSR + 40 NFSR)	320
Sprout	80	70	80 (40 LFSR + 40 NFSR)	320
Grain 128a	128	96	256 (128 LFSR + 128 NFSR)	256
Grain v1	80	64	160 (80 LFSR + 80 NFSR)	160

TABLE I
COMPARISON OF PLANTLET WITH ITS PREDECESSORS IN TERMS OF LFSR AND NFSR SIZES.

Like the members of the Grain family [1], [3], [16], [17] and Sprout [4], Plantlet has two registers – one LFSR and one NFSR, which we denote by L_t and N_t respectively (for some round t) with LFSR being 61 bits long and NFSR being 40 bits in length. For a given round t, we denote LFSR bits as $l_t, l_{t+1}, \ldots, l_{t+60}$ and NFSR bits as $n_t, n_{t+1}, \ldots, n_{t+39}$ respectively. We also denote the secret key by k and its corresponding bits by $\mathsf{k}_0, \mathsf{k}_1, \ldots, \mathsf{k}_{79}$.

Plantlet incorporates the secret key in the KSA and PRGA by XORing a specially selected key bit, $\tilde{k}$, with the last bit of NFSR, n_{t+39}. The selection happens through a Round-Key function, which is cyclically selected from the secret key:

$$\tilde{k} = k_{(t \bmod 80)} \ \forall \ t \geq 0 \tag{1}$$

This cipher uses a 9-bit counter divided in two halves. The lower 7 bits are used for a modulo 80 counter, denoted by $(c_t^6, c_t^5, c_t^4, c_t^3, c_t^2, c_t^1, c_t^0)$ for a given round t. The last two bits, namely c_t^7, c_t^8 constitute a modulo 4 counter incremented once after completion of one round of the modulo 80 counter. This is merely to keep track of the completion of 320 rounds of KSA, post which the counter resets to zero. However, only the 5th LSB (c_t^4) of this counter is actually used in the evolution of the states. Hence, it might not be necessary to include the higher counter bits in the implementation.

As discussed before, the clocking of the cipher can be divided into three routines: KLA (Key Loading Algorithm), KSA (Key Scheduling Algorithm) and PRGA (Pseudo Random Generating Algorithm).

*1) **The KLA Routine**:* The NFSR is initialized with the first 40 bits of IV. The rest 50 bits of the IV are used to initialize the first 50 bits of LFSR. The remaining 11 bits of LFSR are initialized with a specific padding in the particular sequence: $(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1)$ from l_{50} to l_{60}.

*2) **The KSA Routine**:* The NFSR and LFSR registers are clocked 320 times, with one bit shifted in every clock pulse. The highest bit of NFSR is updated as

$$n_{t+39} = z_t \oplus g(N_t) \oplus \tilde{k}^t \oplus l_t \oplus c_t^4,$$

where $g(N_t)$ is a polynomial in GF(2):

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

and z_t is the key stream bit computed as:

$$z_t = h(x) + l_{t+30} + \sum_{j \in B} n_{t+j}. \quad (3)$$

Here $h(x)$ is a non-linear function:

$$\begin{aligned} h(x) = {} & n_{t+4}l_{t+6} \oplus l_{t+8}l_{t+10} \oplus l_{t+17}l_{t+32} \\ & \oplus l_{t+19}l_{t+23} \oplus n_{t+4}l_{t+32}n_{t+38} \end{aligned} \quad (4)$$

and $B = \{1, 6, 15, 17, 23, 28, 34\}$.

We would also like to present a clarification regarding the feedback of output z to NFSR during KSA. The specification of Plantlet [22, Section 4.3] mentions it while the equations do not agree with the same. However, here we consider the more complex version – where the output is fed back to both LFSR and NFSR during KSA. Note that faults are injected during the rounds of PRGA only. Hence it makes no difference in implementation of our fault attack.

The highest bit of LFSR, l_{t+60} is always initialized to 1 during KSA. The second-highest bit l_{t+59} is updated as:

$$l_{t+59} = l_{t+54} \oplus l_{t+43} \oplus l_{t+34} \oplus l_{t+20} \oplus l_{t+14} \oplus l_t \oplus z_t \quad (5)$$

Note that in any clock cycle, the output is generated first and only then the registers are updated. It is XORed to l_{t+59} and n_{t+39} (used as feedback) instead of producing any output during KSA.

*3) **The PRGA Routine**:* The update function for PRGA remains the same for NFSR except that the output is no longer XORed with the key stream bit: $n_{t+39} = g(N_t) \oplus \tilde{k}^t \oplus l_0^t \oplus c_t^4$. Further, the slightly modified update function for LFSR during PRGA phase is:

$$l_{t+60} = l_{t+54} \oplus l_{t+43} \oplus l_{t+34} \oplus l_{t+20} \oplus l_{t+14} \oplus l_t \quad (6)$$

The key stream bit z_t is produced using the output function in (3).

B. Description of Differential Fault Attack

Fault attacks were first introduced in the works of [9], [11], and since then such techniques have gained attention in cryptanalytic literature. In the domain of stream ciphers, one of the first fault attacks was introduced by Hoch and Shamir in [18]. Generally, faults are injected into the state of a cipher to introduce a "change" in the key stream bits, which can help deduce information about the internal state. The method of injecting faults can be laser shots, clock glitches, ionizing radiation, unsupported voltage, etc. Faults can be both transient or permanent. Though the approach of the attack seems quite complicated to be implemented in practical scenario, many implementations of well known ciphers like RSA, AES, DES etc. have been cryptanaylsed by this technique. In fact, all the ciphers in eStream [12] hardware portfolio, namely Grain v1, Mickey 2.0 and Trivium, have been cryptanalyzed against DFA [5], [6], [7], [19], [20], [21], [23]. As mentioned before, even Sprout has been cryptanalyzed [2]. In all these cases, it was enough to recover the cipher state by DFA as the KSA and PRGA of such ciphers are reversible, providing the secret key once the state is known. The same is true for Plantlet.

Let us now briefly explain the connection between Differential Cryptanalysis and Differential Fault Attacks (DFA). In differential attack, we generally put a "difference" in IV while KLA. Thus, if one goes for a sufficient number of rounds during KSA (namely 320 for Plantlet), the difference might not be exploited during the PRGA. In DFA, the adversary is allowed to inject faults during the rounds of PRGA as well. Here the difference is reflected immediately in the key stream and thus, the cipher becomes much weaker in this model. Thus, here we show that Plantlet is even weaker than Sprout in this very specific line of attack.

One difference between the ciphers from the eStream portfolio (Grain v1, Mickey 2.0 and Trivium) is that the secret key bits are not only involved in the KSA but also during PRGA (similar to Sprout). However, that does not resist DFA kind of cryptanalysis. We exploit the probability of matching between each corresponding pair of fault-free and faulty keystream bits to compute the signatures. Further we consider correlations between the differential stream and the signatures to obtain the good matches and thereby identifying the fault location (Section II). After finding the location of the faults, we can use corresponding differential key streams to obtain a system of nonlinear equations and those can be solved using an efficient SAT Solver tool (Section III). Once all the state variables are known for some round t, we can immediately deduce the secret key.

As with every fault attack, we need to lay down the assumptions while mounting the DFA. Naturally, this is important as too many assumptions can make the attack impractical. Also, the number of faults injected should be low, as there is a chance of damaging the device. We consider the following assumptions which are generally accepted throughout in cryptanalytic literature on fault attacks. The adversary

1) can restart the cipher and re-key it with the original Key/IV multiple times,

2) can inject the fault at the exact timings during the execution,
3) has the equipment/required technology for implementing the fault,
4) does not need to know the exact location during fault injection.

The paper is organized as follows. In Section II, we discuss how to obtain the fault signatures. Then, in Section III, we discuss the recovery of the entire state of LFSR, NFSR and consequently the recovery of the secret key bits by solving nonlinear equations. Section IV concludes the paper.

II. Fault Signatures, and How to Identify the Fault Locations

Suppose we introduce a "change" in the key stream bits by injecting a fault at some random location f. From now on, by injecting a fault at location f, we mean (for $0 \leq f \leq 60$) injecting a fault at LFSR bit l_f, and by injecting a fault at any location f, for $(61 \leq f \leq 100)$ we mean injecting a fault in NFSR bit $n_{(f-61)}$.

We obtain the respective fault-free key stream z_i and faulty key stream $z_i^{(f)}$ for λ key stream bits. To form a unique pattern of the key stream sequence, we compute a signature vector $Q^{(f)}$ which we define as:

$$Q^{(f)} = (q_0^{(f)}, q_1^{(f)}, \ldots, q_{\lambda-1}^{(f)}) \tag{7}$$

where

$$q_i^{(f)} = \frac{1}{2} - Pr(z_i \neq z_i^{(f)}), \forall\ i \in [0, \lambda - 1]. \tag{8}$$

We obtain this probability by sufficient number of experiments beforehand. The sharpness of a signature is defined as follows:

$$\sigma(Q^{(f)}) = \frac{1}{\lambda} \sum_{i=0}^{\lambda-1} |q_i^{(f)}|. \tag{9}$$

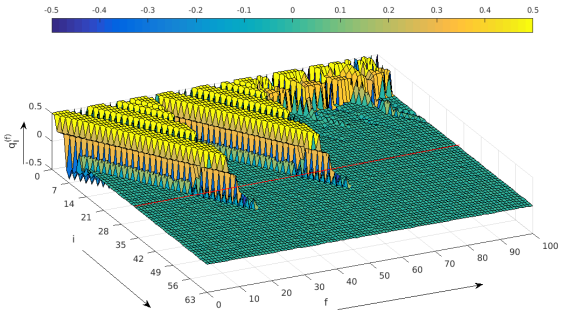

Fig. 1. Plot of $Q^{(f)}$ for all f in $[0, 100]$ with $\lambda = 64$.

As we can see in Figure 1, q_i^f are close to zero for $i > 32$, hence $\lambda = 32$ could have been sufficient. However, we choose $\lambda = 64$ instead as evaluation of correlations and ranks (which we will discuss soon) becomes more accurate with higher number of key stream bits. Now, we can use the sharpness of a location (we just defined) to compare the faults at different locations. For example, one may note the cases for $f = 30$ and $f = 31$ in Figure 2. It is very clear that identifying the location if the fault is indeed injected at 30 (blue) has much better chance than that of 31 (red). With $\lambda = 64$, we execute

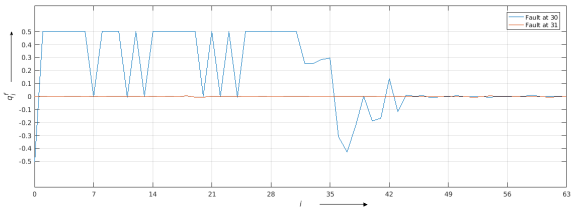

Fig. 2. Plot of $Q^{(30)}$ (blue) and $Q^{(31)}$ (red). This is a snapshot of Figure 1 for $f = 30, 31$.

2^{15} runs with random key-IV pairs to prepare the signatures $Q^{(0)}, Q^{(1)}, \ldots, Q^{(100)}$. As mentioned earlier, we pre-compute the signatures during the offline phase of the attack, and store it for comparisons with differential key stream.

Suppose we inject a fault in a random unknown location g $(0 \leq g \leq 100)$ and obtain the fault-free and faulty key streams z_i and $z_i^{(g)}$ respectively. Then we define the following:

$$\nu_i^{(g)} = \frac{1}{2} - \eta_i^{(g)} \tag{10}$$

where $\eta_i^{(g)} = z_i \oplus z_i^{(g)}$. Let us now continue with a few definitions and notations.

Definition 1: The vector

$$\Gamma^{(g)} = (\nu_0^{(g)}, \nu_1^{(g)}, \ldots, \nu_{\lambda-1}^{(g)})$$

is called trail of the fault at the unknown location g.

Note that there is no probability term in this scenario, since we are actually injecting a fault and checking against our signatures. We compare $\Gamma^{(g)}$ for each of the $Q^{(f)}$'s, for $f = 0, \ldots, 100$ to identify the exact fault location.

Definition 2: We call a relation between the signature $Q^{(f)} = (q_0^{(f)}, q_1^{(f)}, \ldots, q_{\lambda-1}^{(f)})$ and a trail $\Gamma^{(g)} = (\nu_0^{(g)}, \nu_1^{(g)}, \ldots, \nu_{\lambda-1}^{(g)})$ a mismatch, if there exists at least one $i, (0 \leq i \leq \lambda - 1)$ such that $(q_i^{(f)} = \frac{1}{2}, \nu_i^{(g)} = -\frac{1}{2})$ or $(q_i^{(f)} = -\frac{1}{2}, \nu_i^{(g)} = \frac{1}{2})$ hold true.

However, it is quite natural that this may not always happen, and thus we need to extend this definition. For this purpose, we incorporate the correlation coefficient between two sets of data.

Definition 3: We use correlation coefficient $\mu(Q^{(f)}, \Gamma^{(g)})$ between the signature $Q^{(f)} = (q_0^{(f)}, q_1^{(f)}, \ldots, q_{\lambda-1}^{(f)})$ and a trail $\Gamma^{(g)} = (\nu_0^{(g)}, \nu_1^{(g)}, \ldots, \nu_{\lambda-1}^{(g)})$ for checking a match. Naturally, $-1 \leq \mu(Q^{(f)}, \Gamma^{(g)}) \leq 1$. In case of a mismatch, (as per the Definition 2), then $\mu(Q^{(f)}, \Gamma^{(g)}) = -1$.

Then we experiment how one can locate the faults. For each known fault g, we calculate the trail $\Gamma^{(g)} = (\nu_0^{(g)}, \nu_1^{(g)}, \ldots, \nu_{\lambda-1}^{(g)})$. and hence the corresponding $\mu(Q^{(f)}, \Gamma^{(g)})$ for each of the faults $f, (0 \leq f < 100)$. We note:

1) $\max_{f=0}^{100} \mu(Q^{(f)}, \Gamma^{(g)})$,
2) $\mu(Q^{(g)}, \Gamma^{(g)})$, and

3) $\alpha(Q^{(g)}) = n(\mu(Q^{(f)}, \Gamma^{(g)}) > \mu(Q^{(g)}, \Gamma^{(g)}))$.

As we can see in Figure 3, when $\mu(Q^{(g)}, \Gamma^{(g)})$ (drawn in blue) is close to $\max_{f=0}^{100} \mu(Q^{(f)}, \Gamma^{(g)})$ (drawn in red), $\alpha(Q^{(g)})$ is small, it is easier to locate these faults. However, if $\mu(Q^{(g)}, \Gamma^{(g)})$ is much smaller than $\max_{f=0}^{100} \mu(Q^{(f)}, \Gamma^{(g)})$ (red), i.e., $\alpha(Q^{(g)})$ is large, that means it is harder to locate the fault.

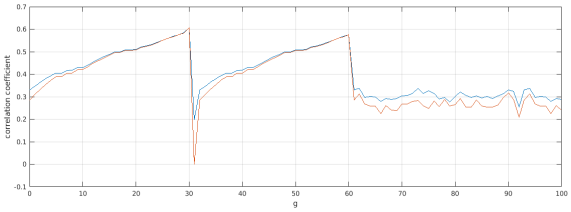

Fig. 3. Plot of $\max_{f=0}^{100} \mu(Q^{(f)}, \Gamma^{(g)})$ (red) and $\mu(Q^{(g)}, \Gamma^{(g)})$ (blue).

Given $\alpha(Q^{(g)})$, for each g, we can estimate how many attempts we should require to obtain the actual fault location. Our experimental results show that obtaining random fault locations is easier in Plantlet as compared to Sprout. Further, the fault requirement is much lower here, making a practical fault attack significantly easier.

As we will describe in Section III, obtaining the exact state is always possible (during the experiments) from the differential key stream corresponding to 4 correct fault locations (details are in Table II). Hence, we need to locate 4 correct fault locations from a large number of random faults. In fact, we also provide examples for the complete attack with only 4 faults. However, we sometimes fail to get LSB of NFSR n_0 in some cases during the experiments.

The exact algorithm for mounting a fault is as follows. Consider that every fault is injected at the same round t of PRGA routine:

- Inject a fault at some random fault location.
- Obtain the differential trail (for some unknown g) $\Gamma^{(g)} = (\nu_0^{(g)}, \nu_1^{(g)}, \ldots, \nu_{\lambda-1}^{(g)})$.
- For each f in $[0, 100]$, calculate $\mu(Q^{(f)}, \Gamma^{(g)})$.
- For the fault, prepare a ranked table T_g arranging the possible fault locations f with more priority according to $\mu(Q^{(f)}, \Gamma^{(g)})$.
- After creating tables T_g for the required faults (say 4), compute using SAT solvers as mentioned in Section III for each of the combinations.

When the correct fault set will be selected in the above algorithm, we will be able to obtain the correct state, which will in turn provide the secret key bits. Then we can check and match with the existing fault free and faulty key streams at hand. To obtain the streams, we need to re-key the cipher a few times and inject those many faults (4 is enough as per our experiments). Naturally, the DFA will be more efficient when the faults are in the locations where it is easier to identify them. That is a location g such that $\alpha(Q^{(g)})$ is small will provide better result. To clarify, lower the $\alpha(Q^{(g)})$, lesser is the number of possible combinations of faults, and lesser the number of times we have to run the SAT solver. However, for Plantlet, we note that the signature of the faults are quite sharp and thus we need not try for specific fault locations during the attack.

Now let us consider the set W such that it contains the 4 faults with maximum possible $\alpha(Q^{(g)})$ values from the experiments (average over 2^{15} runs. Then we calculate that $\prod_{g \in W}(1+\alpha(Q^{(g)})) \approx 2^{11.16}$ combinations (for 4 faults). This is the worst case. That is, after injecting random faults, if these faults appear, then we may need to run the SAT solver these many times. However, the average case scenario is much better than this. For example, we may consider the set of four fault locations $S = \{5, 26, 2, 12\}$. As we have mentioned earlier, the values less than 61 belong to LFSR, and the other values belong to NFSR. The expected number of combinations to check in this case is $\prod_{g \in S}(1 + \alpha(Q^{(g)})) \approx 2^{5.02}$.

III. Obtaining the State from the Differential Key-streams

Let us assume that a fault is injected in the LFSR location ϕ at PRGA round t. The same method will work if the fault is injected in the NFSR. Since we re-key the cipher with the same (key, IV) pair before injecting a fault, after fault injection we get the state $l_t, l_{t+1}, \ldots, l_{t+\phi-1}, 1 \oplus l_{t+\phi}, l_{t+\phi+1} \ldots, l_{t+60}$ and $n_t, n_{t+1}, \ldots, n_{t+39}$ at the t-th round of PRGA. Then corresponding to each faulty key-stream bit z_t^{ϕ}, we introduce two new variables $\mathcal{L}_t^{(\phi)}, \mathcal{N}_t^{(\phi)}$ and obtain three more equations (LFSR, NFSR and output function). Thus we have additional 2ℓ variables and 3ℓ equations for each faulty key-streams, when we involve ℓ many bits of key stream. Here we have considered $\ell = 60$ for solving such equations. Naturally, this implies that we will not involve all the secret key bits (as we need at least 80 key stream bits for involvement of each of the secret key bits). However, at this stage we are not looking for solving the secret key bits. Instead we only try to obtain the LFSR and NFSR states. By trial and error we found $\ell = 60$ to be sufficient for that. Once we know the state (N_t, L_t) for some round t, then we may use SAT solver to obtain the secret key bits as explained in [22, Table 4]. In fact, knowledge of 95 (out of 101) state bits are enough to get the state bits very quickly.

Experimental Results: We solve the equations using a SAT solver, Cryptominisat-2.9.6, under SAGE 7.5.rc2 [26] on a laptop running with Ubuntu-16.10. The hardware configuration is based on Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz and 8 GB RAM. For each guessed set of faults, a system of equations are formulated, and the equations are then fed into the SAT solver.

These experiments are made for 5 runs (for each row) considering that the correct faults have been identified. Thus the time given in the table should be multiplied by $\prod_{g \in S}(1 + \alpha(Q^{(g)}))$ while estimating the effort for the complete fault attack. Note that we have experimented with more number of faults too. The results show that having more faults does not mean that the equations can be solved in less time. This is the reason, we get faster processing time in case of 9 faults than 10 faults (see Table II). However, as we continue towards

# Faults	Solution time (seconds)		
	Maximum	Minimum	Average
4	20914.7	1879.04	7871.17
5	6568.77	1278.76	3306.89
6	4902.52	1392.12	3210.38
9	5209.26	748.72	2178.49
10	2981.80	1708.66	3231.10
12	2266.23	656.93	1261.23

TABLE II
RESULTS OBSERVED WHILE OBTAINING STATE FROM FAULT ATTACK.

very few faults (such as 4 to 6), the processing time generally increases with lesser number of faults. The following are two examples with 4 faults each.

Example 1: Consider that we inject 4 faults at random and the the locations turn out to be $S = \{17, 19, 60, 42\}$. In this case, the expected number of combinations so that we arrive at the right set of fault locations is $2^{3.12}$. We have to check for every such combination in SAT solver. For the correct fault locations, the SAT solver takes 5841.80 seconds to compute the entire states of LFSR and NFSR.

Example 2: Here is another example with 4 random faults. Consider the set of locations $S = \{13, 36, 28, 45\}$. Here, the expected number of possible combinations to check for is $2^{4.08}$. Note that in this case, the SAT solver is unable to compute the NFSR bit n_0. However, this should not be a problem to solve the complete system and the key finally. For the correct fault locations, the SAT solver takes 2215.01 seconds to compute the states of LFSR and NFSR (without n_0).

IV. CONCLUSION

In this paper, we have applied Differential Fault Attack (DFA) on Plantlet. Plantlet has evolved from Sprout after discovery of many of its cryptographic weaknesses. It is thus believed that Plantlet will be cryptographically stronger than Sprout. However, we note that this is not true in terms of DFA. By using as little as 4 faults, we show that the LFSR and NFSR states can be obtained in reasonable time and this in turn provides the secret key immediately. Our technique first finds the locations of the random faults using fault signatures and then it formulates equations to be fed into a SAT solver. Experiments for the same have been performed on a software implementation of the cipher and the exact algorithm was described. It might happen that the larger state size and simple involvement of the secret key bits during the PRGA phase for Plantlet compared to Sprout make the DFA more efficient on Plantlet than its previous instantiation. This indeed requires further attention. While fault attacks can be implemented in very restricted scenarios, we believe our observations can help in understanding Plantlet better.

Acknowledgments: The authors like to thank Dr. Santanu Sarkar of IIT Madras for detailed technical discussion and implementation issues in SAGE.

REFERENCES

[1] M. Ågren, M. Hell, T. Johansson and W. Meier. A New Version of Grain-128 with Authentication. Symmetric Key Encryption Workshop 2011, DTU, Denmark.
[2] S. Maitra and S. Sarkar and A. Baksi and P. Dey. Key Recovery from State Information of Sprout: Application to Cryptanalysis and Fault Attack. http://eprint.iacr.org/2015/236
[3] M. Ågren, M. Hell, T. Johansson and W. Meier. Grain-128a: a new version of Grain-128 with optional authentication. IJWMC, 5(1): 48–59, 2011. This is the journal version of [1].
[4] F. Armknecht and V. Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. FSE 2015. http://eprint.iacr.org/2015/131
[5] S. Banik, S. Maitra and S. Sarkar. A Differential Fault Attack on the Grain Family of Stream Ciphers. In CHES 2012, LNCS, Vol. 7428, pp. 122–139.
[6] S. Banik and S. Maitra. A Differential Fault Attack on MICKEY 2.0. CHES 2013, LNCS, Vol. 8086, pp. 215–232, 2013.
[7] S. Banik, S. Maitra and S. Sarkar. Improved differential fault attack on MICKEY 2.0. Journal of Cryptographic Engineering, 5(1):13–29, 2015. http://link.springer.com/article/10.1007\%2Fs13389-014-0083-9, 2014
[8] S. Banik. Some results on Sprout. INDOCRYPT 2015, LNCS, Vol. 9462, pp. 124–139, 2015. http://eprint.iacr.org/2015/327
[9] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In CRYPTO 1997, LNCS, Vol. 1294, pp. 513–525.
[10] E. Biham and O. Dunkelman. Differential Cryptanalysis in Stream Ciphers. Cryptology ePrint Archive, Report 2007/218, https://eprint.iacr.org/2007/218
[11] D. Boneh, R. A. DeMillo and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In EUROCRYPT 1997, LNCS, Vol. 1233, pp. 37–51.
[12] The ECRYPT Stream Cipher Project. eSTREAM Portfolio of Stream Ciphers. SAC 2015, Canada. http://www.ecrypt.eu.org/stream/
[13] M. F. Esgin and O. Kara. Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks. http://eprint.iacr.org/2015/289
[14] V. Lallemand and M. Naya-Plasencia. Cryptanalysis of Full Sprout. In Crypto 2015, LNCS. In CRYPTO 2015, LNCS, Vol. 9215, pp. 663–682. http://eprint.iacr.org/2015/232
[15] Y. Hao. A Related-Key Chosen-IV Distinguishing Attack on Full Sprout Stream Cipher. http://eprint.iacr.org/2015/231
[16] 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.
[17] M. Hell, T. Johansson, A.Maximov and W. Meier. A Stream Cipher Proposal: Grain-128. In IEEE International Symposium on Information Theory (ISIT 2006).
[18] J. J. Hoch and A. Shamir. Fault Analysis of Stream Ciphers. In CHES 2004, LNCS, Vol. 3156, pp. 1–20.
[19] M. Hojsík and B. Rudolf. Differential Fault Analysis of Trivium. In FSE 2008, LNCS, Vol. 5086, pp. 158–172.
[20] M. Hojsík and B. Rudolf. Floating Fault Analysis of Trivium. In INDOCRYPT 2008, LNCS, Vol. 5365, pp. 239–250.
[21] Y. Hu, J. Gao, Q. Liu and Y. Zhang. Fault analysis of Trivium. Designs, Codes and Cryptography, 62(3): 289–311, 2012.
[22] V. Mikhalev, F. Armknecht and C. Müller. On ciphers that continuously access the non-volatile key. FSE 2017. TOSC, Volume 2016, Issue 2, pp. 52–79, 2016. http://tosc.iacr.org/index.php/ToSC/article/view/565/507
[23] S. Sarkar, S. Banik and S. Maitra. Differential Fault Attack against Grain family with very few faults and minimal assumptions. IEEE Transactions on Computers, 64(6):1647–1657, 2015.
[24] S. P. Skorobogatov and R. J. Anderson. Optical Fault Induction Attacks. In CHES 2002, LNCS, Vol. 2523, pp. 2–12.
[25] S. P. Skorobogatov. Optically Enhanced Position-Locked Power Analysis. In CHES 2006, LNCS, Vol. 4249, pp. 61–75.
[26] 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).
[27] B. Zhang and X. Gong. Another Tradeoff Attack on Sprout-Like Stream Ciphers. In Asiacrypt 2015, LNCS, Vol. 9453, pp. 561–585.