# Differential Fault Attack on Grain v1, ACORN v3 and Lizard

Akhilesh Anilkumar Siddhanti, Santanu Sarkar, Subhamoy Maitra and Anupam Chattopadhyay
BITS Pilani, Goa, India; IIT Madras, India; Indian Statistical Institute, Kolkata; NTU Singapore
Communicating email: subho@isical.ac.in

*Abstract*—**Differential Fault Attack (DFA) is presently a very well known technique to evaluate security of a stream cipher. This considers that the stream cipher can be weakened by injection of the fault. In this paper we study DFA on three ciphers, namely Grain v1, Lizard and ACORN v3. We show that Grain v1 (an eStream cipher) can be attacked with injection of only 5 faults instead of 10 that has been reported in 2012. For the first time, we have mounted the fault attack on Lizard, a very recent design and show that one requires only 5 faults to obtain the state. ACORN v3 is a third round candidate of CAESAR and there is only one hard fault attack on an earlier version of this cipher. However, the 'hard fault' model requires a lot more assumption than the generic DFA. In this paper, we mount a DFA on ACORN v3 that requires 9 faults to obtain the state. In case of Grain v1 and ACORN v3, we can obtain the secret key once the state is known. However, that is not immediate in case of Lizard. While we have used the basic framework of DFA that appears in literature quite frequently, specific tweaks have to be explored to mount the actual attacks that were not used earlier. To the best of our knowledge, these are the best known DFA on these three ciphers.**

**Keywords:** Differential Fault Attack, Stream Cipher, Grain v1, ACORN v3, Lizard.

## I. Introduction

In search of stream ciphers suitable for widespread adoption, the eStream portfolio [20] was started in 2004 by EU ECRYPT network. By this date, three ciphers form the hardware profile of the portfolio, namely Grain v1 [10], Trivium [7] and MICKEY 2.0 [1]. Stream ciphers find a special application in designing securities in case of resource-constrained or low power scenarios like RFID tags or hearing aids, due to their very low gate requirements. A natural attention was drawn towards Grain v1 for being the 'lightest' in terms of the state size and the Boolean functions used among the three, and many lightweight stream ciphers have been hence proposed based on Grain v1. The DFA proposed in [2], [18] shows that by injecting 10 or more faults during the Pseudo Random bit Generation Algorithm (PRGA), the secret key can be deduced hence compromising the security of the cipher. However, the attack can be further optimized. In this work, we claim that the fault requirement can be further brought down to just 5 using optimized techniques.

Following the estream portfolio, a new competition for authenticated ciphers called CAESAR [21] has been hosted. Enlisting fifteen different ciphers as final candidates, a cipher with unique design has emerged called ACORN v3 [22]. A lightweight stream cipher composed of 6 Linear Feedback Shift Registers (LFSRs) making a state size of 293 bits, ACORN v3 promises a 128-bit security keyed using a 128-bit secret key and IV. Since very limited study has been done on such type of cipher constructs, we explore how the design performs against mounting of a DFA. As per our experiments, cryptanalysis is possible in this case with a requirement of 9 faults. We are aware of a fault attack on an earlier version of ACORN as in [8], but that is only in a restricted model of hard fault, which considers a fault to be permanent. Our model of DFA here is much more well accepted in literature.

Inheriting the ideas from Grain v1, another interesting lightweight stream cipher Lizard has been designed. A unique feature of Lizard [9] is that the secret key cannot be found even if the secret state is known. This ensures additional security, specifically in places where the secret state can get compromised. Till now, there has been no reported cryptanalysis on Lizard apart from a related key/IV attack shown in [5]. We show that a successful DFA can be performed against Lizard using a minimum of 5 faults.

For all the above mentioned ciphers viz. Grain v1, Lizard and ACORN v3, we follow a similar approach as in [14] that has been used many times in earlier papers too (see references in [14]). The correct location of the fault is obtained by finding the correlation between faulty and fault-free key streams. Using the given set of faulty and fault-free key streams, equations are generated and fed into a SAT solver. The outline of such DFA will be discussed in Section II. In Section III, we describe the process of finding the exact location of fault. In Section IV, we explain the procedure of finding the state variables and the recovery of secret key once the exact location of fault is known. For optimizing the SAT solver to find solutions faster, we consider key stream bits from previous rounds as well. This is the main tweak in our approach over the existing works and briefly mentioned in Sections IV-A, IV-B, IV-C. In Section V, we conclude the paper summarizing our work. The description of the ciphers are available in the Appendix.

### A. Our Contribution

While a specific mode of DFA, that we discuss in this paper, is well standardized, most of the stream cipher designers do not consider evaluating such attack on the new designs. This leaves an open space towards implementing such attacks on specific ciphers. Further, blind implementation of some standard techniques do not immediately help in mounting a successful DFA. For this the exact implementation related to a specific cipher requires certain optimization. In this paper, we have two specific modes of optimization.

- What we feed to the SAT solver for obtaining the states are some equations based on the differential key streams. For the first time we show that corresponding to a state we should consider the equations forward and backward both. Earlier we have only considered the equations while moving forward. This drastically reduces the number of faults as experienced in Grain v1 and the method succeeded for Lizard too.
- Due to the large state of ACORN v3 and the clever state update, it is not easy to obtain the solutions through the SAT solver directly. Thus we need to consider fixing some bits before exploiting the SAT solver. This indeed increases the overall complexity, but at the same time makes the DFA possible. The exhaustive search over the assumed bits can be trivially parallelized keeping the complete attack practical.

## II. Proposed Outline of DFA

Fault Attacks have always been studied in cryptanalytic literature with great interest. By inducing a fault, we mean flipping one bit ($1 \rightarrow 0$ or $0 \rightarrow 1$) for some particular state of the cipher. Such faults can be induced at the beginning of the PRGA round, hence causing a change in the key stream bits. The difference between the key stream bits can be used to deduce the internal state of the cipher. Fault attack techniques range from simple glitches (caused by perturbations in the clock or power supply), focused laser beam injection, Body Bias injection to Electromagnetic injection. The range of attacks is much wider if one considers the non-volatile memories, for which, one may use hot air gun or even software-based Rowhammer attack. Depending on the level of intrusion that is enabled by the attack setup, attacks can be classified to be non-invasive, semi-invasive and invasive.

Fault injection attacks of various forms [6] is becoming an important tool in the arsenal of modern cryptanalysts. Rapidly evolving techniques for attacks and their countermeasures [17] indicate that a proper feasibility analysis of the implementation is imperative. Although inducing a fault might seem quite complicated, there have been many works in this area. Implementations of well-known ciphers like RSA, AES and DES have already been cryptanalyzed. In fact, all the final candidates of eStream [20] hardware portfolio (namely Grain v1, MICKEY 2.0 and Trivium) have been cryptanalyzed using DFA [2], [18], [3], [4], [11], [12], [13], [18]. This work aims to highlight that ACORN v3 and Lizard can also be cryptanalyzed using DFA and the existing knowledge against Grain v1 can be improved.

Let us now clearly explain the assumptions while mounting the DFA. Generally too many assumptions can make an attack impractical. Further, the number of faults injected should be low, as there is a chance of damaging the device completely. Based on the documents in cryptanalytic literature on fault attacks, we consider that the attacker:

1) can restart the cipher and re-key it as well with the original Key/IV more than once,
2) can inject the fault with certain precision of timing,
3) has the equipment/required technology for injecting the fault,
4) does not need to know the exact location during fault injection.

Next we will discuss several steps of DFA. Note that the basic methodology is the same which is basically the Differential Attack, but the Key Scheduling Algorithm (KSA) is ignored. That is, we consider that one can inject the fault during the PRGA. We will follow the basic methodology as in [14] and the references in this work which are in the same line. Our specific tweaks will be described in the process.

## III. Identifying Fault Locations

The first step of the DFA requires identification of fault signatures. We consider the most common signature methods that had been used in [14] too. Consider that the certain changes in the key stream bits are achieved by injecting a fault at some random location $f$. By random location, we mean some LFSR or NFSR bit, which is the part of secret state of the cipher. Thus, by injecting a fault at location $f$ means it might be a location in the LFSR or NFSR according to the specific description of the cipher. For example, in case of Grain v1, $f \in [0, 79]$ means injecting a fault at LFSR bit $l_f$, whereas $f \in [80, 159]$ underlines injecting a fault in NFSR bit $n_{(f-80)}$.

In the attack model, we consider that for some fault location $f$, it is possible to obtain the respective fault-free key stream $z_i$ and faulty key stream $z_i^{(f)}$ for $\lambda$ key stream bits. To form a unique pattern of the key stream sequence, we compute a signature vector $Q^{(f)}$ which we define as:

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

where

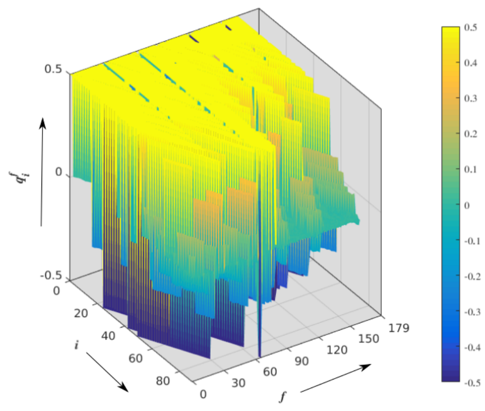
$$q_i^{(f)} = \frac{1}{2} - Pr(z_i \neq z_i^{(f)}), \forall\ i \in [0, \lambda - 1]. \tag{2}$$

This probability is estimated by sufficient number of experiments beforehand. The sharpness of a signature is defined as follows:

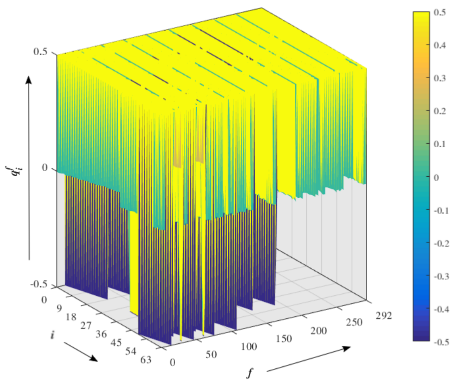
$$\sigma(Q^{(f)}) = \frac{1}{\lambda} \sum_{i=0}^{\lambda-1} |q_i^{(f)}|. \tag{3}$$

Following similar convention for ACORN v3, the fault in location $f$ simply corresponds to fault in bit $S_f$. The corresponding plot is presented in Figure 1. For Lizard, the convention is fault location in $S_f$ for first 31 bits and fault location in $B_{(f-90)}$ for next 90 bits. With $\lambda = 90, 64, 64$ respectively, we execute $2^{15}$ runs with random key-IV pairs to obtain the signatures $Q^{(0)}, Q^{(1)}, \dots$ for each of Grain v1, ACORN v3 and Lizard.

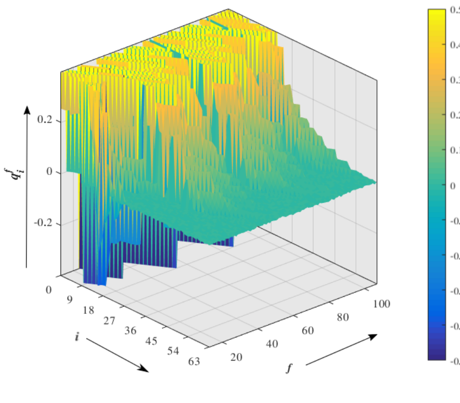
As we can see in Figure 1, the Z-axis has been plotted from $-0.5$ to $0.5$. The signatures are said to be strong if the curve is closer to $-0.5$ or $0.5$ for some fault location $f$. In all the three cases of Grain v1, ACORN v3 and Lizard, the signatures are quite strong, in fact stronger than Plantlet [14] and Sprout [15]. Hence, the identification of the fault will



(a) Grain v1: Signature



(b) ACORN v3: Signature



(c) Lizard: Signature

Fig. 1: Signatures for Grain v1 (plot of $Q^{(f)}\ \forall f\ \in [0,159]$), ACORN v3 (plot of $Q^{(f)}\ \forall f\ \in [0,292]$) and Lizard (plot of $Q^{(f)}\ \forall f\ \in [0,120]$) with $\lambda = 64$ for ACORN v3 and Lizard, and $\lambda = 90$ for Grain v1.

be easier for these ciphers. The signatures are pre-computed during the offline phase of the attack, and they are stored for comparisons with differential key stream later. To clarify this, we require to explain a few more definitions.

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

$$\nu_i^{(g)} = \frac{1}{2} - \eta_i^{(g)} \tag{4}$$

where $\eta_i^{(g)} = z_i \oplus z_i^{(g)}$.

*Definition 1:* The vector

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

is called trail of the fault at the unknown location $g$.
Note that there is no probability involved in this scenario, as one actually injects a fault and checks against the signatures. That is, one can compare $\Gamma^{(g)}$ for each of the $Q^{(f)}$'s, to estimate the exact fault location.

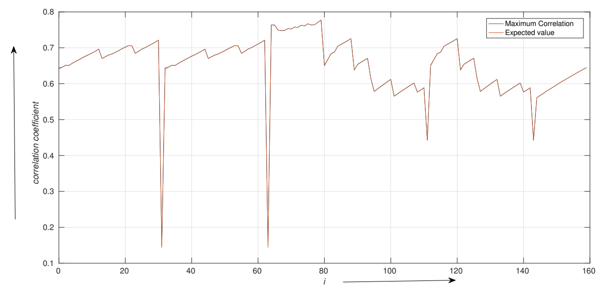
*Definition 2:* We call a relation between the signature $Q^{(f)} = (q_0^{(f)}, q_1^{(f)}, \dots, q_{\lambda-1}^{(f)})$ and a trail $\Gamma^{(g)} = (\nu_0^{(g)}, \nu_1^{(g)}, \dots, \nu_{\lambda-1}^{(g)})$ a mismatch, if there exists at least one $i, (0 \leq i \leq \lambda - 1)$ such that $(q_i^{(f)} = \frac{1}{2},\ \nu_i^{(g)} = -\frac{1}{2})$ or $(q_i^{(f)} = -\frac{1}{2},\ \nu_i^{(g)} = \frac{1}{2})$ hold true.
However, this is for excluding some locations for possible faults, but to identify the location, this definition needs to be extended. For this purpose, we incorporate the correlation coefficient between two sets of data.

*Definition 3:* It is natural to use correlation coefficient $\mu(Q^{(f)}, \Gamma^{(g)})$ between the signature $Q^{(f)} = (q_0^{(f)}, q_1^{(f)}, \dots, q_{\lambda-1}^{(f)})$ and a trail $\Gamma^{(g)} = (\nu_0^{(g)}, \nu_1^{(g)}, \dots, \nu_{\lambda-1}^{(g)})$ 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 $\mu(Q^{(f)}, \Gamma^{(g)}) = -1$.
Let us now explain how one can locate the faults. For each known fault $g$, it is possible to calculate the trail $\Gamma^{(g)} = (\nu_0^{(g)}, \nu_1^{(g)}, \dots, \nu_{\lambda-1}^{(g)})$, and hence the corresponding $\mu(Q^{(f)}, \Gamma^{(g)})$ for each of the faults $f$. The following quantities are noted:

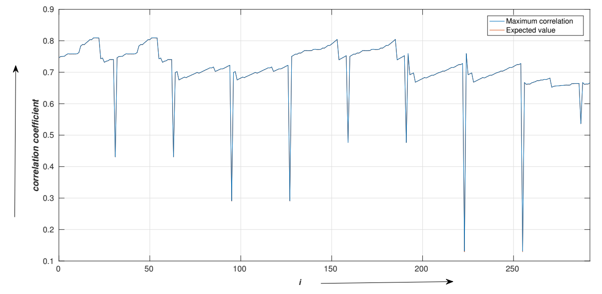
1) $\max_f \mu(Q^{(f)}, \Gamma^{(g)})$,
2) $\mu(Q^{(g)}, \Gamma^{(g)})$, and
3) $\alpha(Q^{(g)}) = \#_f|\{\mu(Q^{(f)}, \Gamma^{(g)}) > \mu(Q^{(g)}, \Gamma^{(g)})\}|$.

In the following Figure 2, when $\mu(Q^{(g)}, \Gamma^{(g)})$ (drawn in red) is close to $\max_{f=0}^{100} \mu(Q^{(f)}, \Gamma^{(g)})$ (drawn in blue), $\alpha(Q^{(g)})$ is small, it is easier to locate these faults. However, if $\mu(Q^{(g)}, \Gamma^{(g)})$ is much smaller than $\max_f \mu(Q^{(f)}, \Gamma^{(g)})$ (blue), i.e., $\alpha(Q^{(g)})$ is large, that means it is harder to locate the fault for that particular fault location $f$ from differential key stream. In fact, the difference between the red and blue lines for ACORN v3 is so small, it is barely visible. Hence, we should expect ACORN v3 to have better expected ranks than Grain v1 and Lizard.

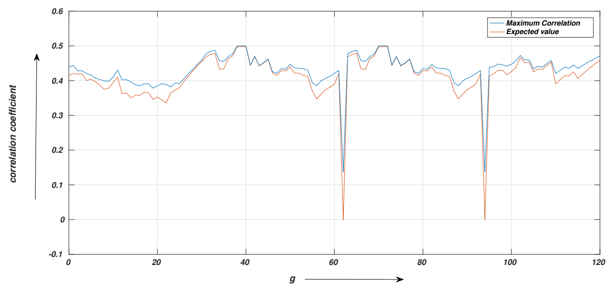
Given $\alpha(Q^{(g)})$, for each $g$, we can estimate how many attempts we should require to obtain the actual fault location. As one can see in Figure 3, the rank of the correct set of fault locations is very low for all three ciphers, with ranks for ACORN v3 being the strongest. The ranks for ACORN v3 and Grain v1 lie between 1 and 2, hence we can get the correct set



(a) Grain v1

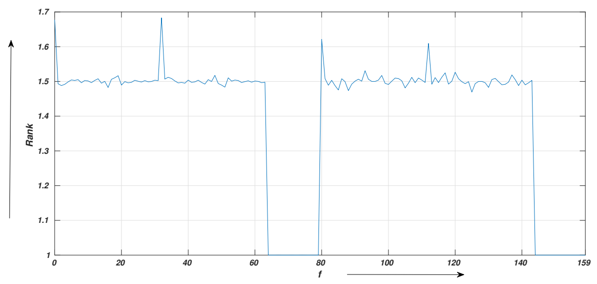


(b) ACORN

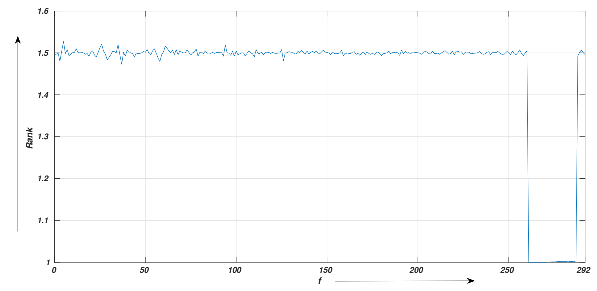


(c) Lizard

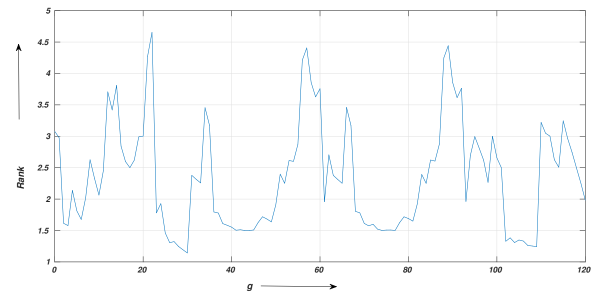
Fig. 2: Plot of $\max_{f=0}^{100} \mu(Q^{(f)}, \Gamma^{(g)})$ (blue) and $\mu(Q^{(g)}, \Gamma^{(g)})$ (red) for all three ciphers.



(a) Grain v1



(b) ACORN



(c) Lizard

Fig. 3: Ranks of actual fault locations in list of predicted fault locations for all the three ciphers. (lower the better).

of fault locations very quickly using this technique. The ranks of correct set of fault locations for Lizard also comes very close to the other two ciphers. However ACORN v3 has the highest fault requirement (9 faults) due to its large state size, and also due to an additional complexity of $2^{20}$ incorporated (explained in Section IV) for faster solving, ACORN v3 has higher complexity ($2^{25.40}$) than Grain v1 ($2^{3.49}$) and Lizard ($2^{10.69}$).

Thus, to summarize, 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 $[0, 159]$ (for e.g. Grain v1), 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 number of faults, compute using SAT solvers as mentioned in Section IV for each of the combinations.

In case, the correct fault set can be selected in the above algorithm, one can obtain the correct state, which will in turn discover the secret key bits. This can be confirmed as one can check and match with the existing fault free and faulty key streams at hand. To obtain the streams, the attacker needs to re-key the cipher a few times and inject the required number of faults.

## A. *Estimated complexity to find the correct set of faults*

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. That is, lower the $\alpha(Q^{(g)})$, lesser the the number of possible combinations of faults, and lesser the number of times one needs to run the SAT solver. It has been noted in [14] that for Plantlet, the signature of the faults are quite sharp.

| Cipher | Faults Reqd. | Maximum | Average |
|---|---|---|---|
| Grain v1 | 5 | 3.49 | 2.44 |
| | 6 | 4.10 | 2.93 |
| | 7 | 4.71 | 3.42 |
| ACORN v3 | 7 | 4.21 | 3.78 |
| | 8 | 4.80 | 4.32 |
| | 9 | 5.40 | 4.86 |
| Lizard | 5 | 10.69 | 6.16 |
| | 6 | 12.76 | 7.39 |
| | 7 | 14.71 | 8.62 |

TABLE I: Maximum and average number of combinations to check for all three ciphers for different number of faults. The values (except for faults) have been given in logarithm to the base 2.

Interestingly, signatures of Grain v1, Lizard and ACORN v3 are sharper than Plantlet. As we will see later, for the actual attack, we require at least 5, 9 and 5 faults for Grain v1, ACORN v3 and Lizard respectively. In the following table we provide the experimental estimation of the number of attempts to get the exact fault locations for these three ciphers. Note that the data provided in Table I is logarithm to the base 2.

## IV. Deducing the state variables and Secret Key

Once we obtain the differential key streams for some set of fault locations, we need to find at least one state of the cipher for some round $t$ which in-turn can help us find the secret key. We start off by noting that for every key stream bit produced, we can formulate following three equations:

1) The output function,
2) The NFSR feedback function,
3) The LFSR feedback function.

Hence, at the beginning of the first round of PRGA, we have 160 unknown variables (80 for each LFSR and NFSR) in case of Grain v1, 293 variables in case of ACORN v3 and 121 variables in case of Lizard. With every new round of the ciphers, the complexity of the above equations increase sharply. To combat this, new variables are introduced at every step, and hence new equations are formed. Two new variables are added and three new equations are formed for every round of the ciphers. Note than in case of ACORN v3, there are 6 LFSRs hence 7 new variables are added with each cycle (6 from LFSRs and 1 from feedback) and 8 new equations are formed. We collect all these equations and feed them into a SAT solver. However, the number of equations becomes very high and hence the SAT solver cannot find a solution, hence steps specific to each cipher need to be taken.

### A. *Optimizing SAT solver for Grain v1*

Grain v1 has been constructed in such a way that the higher 16 bits of LFSR and NFSR are not used at all. Hence, we can safely discard the equations formed during the last 16 rounds of our set of equations. Next, we observe that if the fault has taken place in LFSR, the NFSR equations do not change till the fault reaches location $l_0$. Hence, we remove all such NFSR equations. Now, if the fault has taken place in NFSR, we need not consider any equation of LFSR because LFSR remains unaffected throughout the clocking. Since LFSR equations are linear and easier to calculate for the SAT solver, we consider injecting faults in LFSR only.

Note that Grain v1 is reversible, i.e. given one state, we can easily determine the previous state of the cipher by solving feedback equations. Considering that the fault has been injected at PRGA round $t$, we can form more equations by considering key stream bits of round $t-1, t-2$ and so on. Although the number of equations increase, it is added only once (not for every fault) and helps in finding a solution faster.

After the performing the above optimizations, the fault requirement for Grain v1 is 5 faults with a time complexity of $2^{3.49}$.

*Example 1:* Consider the following set of 5 fault locations for Grain v1: $S = \{6, 16, 50, 51, 69\}$. (This set of numbers is randomly generated and not specifically chosen.) The estimated number of fault locations to check for is $2^{2.29}$. The equations are formed and fed into the SAT solver. The number of key stream bits considered is 250, with 40 reverse key stream bits considered, and the total time required by the SAT solver for the correct set of fault locations is 1756.45 seconds.

### B. *Optimizing SAT solver for ACORN v3*

The state size of ACORN v3 is much larger than Grain v1, Lizard, Plantlet and Sprout. Also, the number of equations added at each clock cycle is much higher than compared to the latter. Hence, we propose a different approach - we consider that some $n$ bits for example $l_0, l_1, \ldots, l_{n-1}$ are known. Now we try to find a solution assuming these $n$ bits are correct. Now the SAT solver is able to find a solution much faster. Note that this raises our attack complexity by $2^n$, but we can try getting as small value of $n$ as possible while still being able to find solutions faster. Our experiments show that for $n = 20$ we can deduce the state using 9 faults, whereas with $n = 40$ or $n = 60$ we can deduce the state even with 8 or 7 faults. From Table I, we know the maximum number of combinations to be $2^{4.21}$, $2^{4.80}$, $2^{5.40}$ in case of 7, 8 and 9 faults. Considering the above optimizations, the complexity will be $2^{64.21}$ in case of 7 faults, $2^{44.80}$ and $2^{25.40}$ in case of 8 and 9 faults. However, there are some cases in which we cannot solve for the entire state with 7 or 8 faults, and hence we consider 9 faults to be minimum for a successful attack.

Since the solving time depends upon which $n$ bits (say 20) are known, a good choice would be choosing the 15 tap locations of ACORN v3 and then further considering higher bits like $S_{292}, S_{291}, \ldots, S_{287}$ and so on. Like Grain v1 and Lizard, we can further reduce the number of faults by using key stream bits from rounds prior to injecting of the fault. We have not performed this optimization in our work for ACORN v3, but we believe this could better our results further.

*Example 2:* Suppose we have the following set of 9 locations for ACORN v3, $S = \{279, 238, 10, 129, 9, 121, 271, 225, 166\}$. The number of variables considered to be known are $s_0, \ldots, s_{19}$, i.e. $n = 20$ bits. The number of combinations to check, for this set of fault locations will be $2^{4.92}$. Thus the number of times SAT solver is run will be $2^{20} \times 2^{4.92} = 2^{24.92}$. The number of key stream bits considered is 1200. For solving the correct set of fault locations, the SAT solver takes 342.43 seconds.

### C. Optimizing SAT solver for Lizard

In case of Lizard, the fault requirement is comparatively very high (more than ten) when we adopt the strategy used in case of ACORN v3 and Grain v1. However, we use some optimizations to improve our results. Firstly, we have used 90 key stream bits $z_t, z_{t+1}, z_{t+2}, \ldots, z_{t+89}$ to formulate equations, where $t$ refers to the round in which the fault has been injected. Since Lizard is reversible without using key bits during the PRGA, we reverse the state $(S_t, B_t)$ upto $(S_{(t-90)}, B_{(t-90)})$ and formulate equations for $z_{t-1}, z_{t-2}, \ldots, z_{t-89}$. Next, we consider that if we are able to inject faults in NFSR2 (register $B$) only, we can reduce the number of variables drastically, and hence obtain results faster. This is because the $S$ register is independent of $B$ register, and we need not include more variables for NFSR1 update equations (NFSR1 remains same post fault injection in NSFR2). Also, we note that the highest bit used in NFSR2 update function is $B_{84}$, hence we need not include any variables from round 85 for all faults.

As mentioned before, we can only solve for the secret state and not for the secret key in case of Lizard. However, we can obtain the secret key once the secret state is known in case of Grain v1 and ACORN v3. Solving for the state of Lizard takes a fault requirement of 5 faults with a time complexity of $2^{10.69}$.

*Example 3:* Considering 5 fault locations $S = \{33, 59, 10, 5, 43\}$ and combinations to check for being $2^{5.52}$, the SAT solver takes 2092.41 seconds to compute the states of LFSR and NFSR. The number of key stream bits considered is 90 and 40 key stream bits are taken from the previous rounds.

### D. Summary of Comparison

Here we present the summary of DFA on the three ciphers based on our theory and experiments. According to our study, all the ciphers could be attacked using DFA with very few faults. The above experiments were performed on ciphers implemented in Sage-7.6 [19] along with Cryptominisat-2.9.6 as SAT solver on a laptop running Ubuntu-17.04. The hardware configuration is based on Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz and 8 GB RAM.

| Cipher | #Faults | Time Complexity | Time taken by SAT solver | | |
|---|---|---|---|---|---|
| | | | Max. | Avg. | Min |
| Grain v1 | 5 | $2^{3.49}$ | 26798.64 | 7165.48 | 204.48 |
| ACORN v3 | 9 | $2^{25.40}$ | 369.56 | 293.75 | 194.80 |
| Lizard | 5 | $2^{10.69}$ | 720.42 | 201.82 | 20.46 |

TABLE II: Results observed while obtaining state from fault attack.

## V. Conclusion

Most of the popular and commercial Feedback Shift Register (FSR) based stream ciphers have come out to be vulnerable against Differential Fault Attack. In this paper, we presented successful DFA against a finalist of eStream portfolio Grain v1 (improvisation over previous DFA), a phase-3 candidate of CAESAR called ACORN v3 and a lightweight stream cipher Lizard. We explored the identification of fault locations using correlation of signatures and trail of a faulty key stream for all the three ciphers and expected number of checks required to obtain a correct state was presented. Equations were formed from faulty and fault-free key streams and fed into a SAT solver. Further cipher-specific optimizations were performed towards minimizing the number of faults as well as to speed up solving time. This is the novel contribution of this work. The analysis performed in this work can be further extended to other stream ciphers as well, and future work in this area could be promising. We are working towards optimizing our attacks on these three ciphers to succeed with even fewer faults. Further, our technique on Grain v1 can also be implemented on Grain 128 and Grain 128a. These we will include in the final version of the paper. Based on our work and the development in this domain, it is evident that FSR based ciphers in nonlinear combiner/filter generator model will generally be vulnerable against DFA. Implementors need to come up with new ways to protect against such fault attack scenarios.

## References

[1] S. Babbage and M. Dodd. The stream cipher MICKEY 2.0. ECRYPT Stream Cipher Project Report. Online available at http://ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf

[2] 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.

[3] S. Banik and S. Maitra. A Differential Fault Attack on MICKEY 2.0. CHES 2013, LNCS, Vol. 8086, pp. 215–232, 2013.

[4] 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

[5] S. Banik and T. Isobe. Some cryptanalytic results on Lizard. Online available at http://eprint.iacr.org/2017/346.pdf

[6] A. Barenghi, L. Breveglieri, I. Koren and D. Naccache. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. In Proceedings of the IEEE vol. 100, no. 11, pp. 3056–3076, Nov. 2012, doi: 10.1109/JPROC.2012.2188769

[7] C. De Cannire and B. Preneel. TRIVIUM Specifications. eSTREAM, ECRYPT Stream Cipher Project, Report.

[8] P. Dey, R. S. Rohit and A. Adhikari. Full key recovery of ACORN with a single fault. Journal of Information Security and Applications, Volume 29, Issue C, August 2016, Pages 57-64, doi: 10.1016/j.jisa.2016.03.003 Elsevier Science Inc. New York, NY, USA

[9] M. Hamann, M. Krause, W. Meier. LIZARD - A Lightweight Stream Cipher for Power-constrained Devices. IACR Transactions on Symmetric Cryptology, Volume 2017, Issue 1, http://tosc.iacr.org/index.php/ToSC/article/view/584

[10] 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.

[11] M. Hojsík and B. Rudolf. Differential Fault Analysis of Trivium. In FSE 2008, LNCS, Vol. 5086, pp. 158–172.

[12] M. Hojsík and B. Rudolf. Floating Fault Analysis of Trivium. In INDOCRYPT 2008, LNCS, Vol. 5365, pp. 239–250.

[13] Y. Hu, J. Gao, Q. Liu and Y. Zhang. Fault analysis of Trivium. Designs, Codes and Cryptography, 62(3): 289–311, 2012.

[14] S. Maitra, A. Siddhanti and S. Sarkar. A Differential Fault Attack on Plantlet. To appear in IEEE Transactions on Computers. DOI: 10.1109/TC.2017.2700469, Date of Publication: 02 May 2017. An earlier version is available at Cryptology ePrint Archive: Report 2017/088, 4 February, 2017. http://eprint.iacr.org/2017/088

[15] S. Maitra, S. Sarkar, A. Baksi and P. Dey. Key Recovery from State Information of Sprout: Application to Cryptanalysis and Fault Attack, 2015. http://eprint.iacr.org/2015/236

[16] 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. Available at http://tosc.iacr.org/index.php/ToSC/article/view/565/507

[17] T. Sugawara, D. Suzuki, R. Fujii, S. Tawa, R. Hori, M. Shiozaki, T. Fujino. Reversing Stealthy Dopant-Level Circuits. Cryptographic Hardware and Embedded Systems CHES 2014, vol. 8731 of the series Lecture Notes in Computer Science, pp. 112–126.

[18] 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.

[19] 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).

[20] The ECRYPT Stream Cipher Project. eSTREAM Portfolio of Stream Ciphers. Online at http://www.ecrypt.eu.org/stream/

[21] The Project CAESAR on Authenticated Ciphers. Online at http://competitions.cr.yp.to/caesar.html

[22] H. Wu. ACORN: A Lightweight Authenticated Cipher (v3). https://competitions.cr.yp.to/round3/acornv3.pdf, 2016

## Appendix: Description of the ciphers

### *A1: Grain v1*

Grain v1 has two registers, LFSR and NFSR of 80 bits each and we use the notation $s_i, s_1 + i, \ldots, s_{79+i}$ and $b_0 + i, b_1 + i, \ldots, b_{79+i}$ for state bits of LFSR and NFSR respectively. The output function calculates the key stream bit and then the LFSR and NFSR states are updated. The output function is given by:

$$z_t = b_{i+1} \oplus b_{i+2} \oplus b_{i+4} \oplus b_{i+10} \oplus b_{i+31} \oplus b_{i+43} \oplus b_{i+56} \oplus h(x) \tag{5}$$

where $h(x)$ is given by:

$$h(x) = x_1 \oplus x_4 \oplus x_0x_3 \oplus x_2x_3 \oplus x_3x_4 \oplus x_0x_1x_2 \oplus x_0x_2x_3 \oplus x_0x_2x_4 \oplus x_1x_2x_4 \oplus x_2x_3x_4 \tag{6}$$

and $x_1, x_2, x_3, x_4, x_5$ correspond to tap positions $s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, b_{i+63}$.

The LFSR feedback bit $s_{i+80}$ is calculated as:

$$s_{i+80} = s_{i+62} \oplus s_{i+51} \oplus s_{i+38} \oplus s_{i+23} \oplus s_{i+13} s_i \tag{7}$$

and the NFSR feedback bit is calculated as:

$$\begin{aligned} b_{i+80} = {} & s_i \oplus b_{i+62} \oplus b_{i+60} \oplus b_{i+52} \oplus b_{i+45} \oplus b_{i+37} \\ & \oplus b_{i+33} \oplus b_{i+28} \oplus b_{i+9} \oplus b_i \oplus b_{i+63}b_{i+60} \oplus b_{i+37}b_{i+33} \\ & \oplus b_{i+15}b_{i+9} \oplus b_{i+60}b_{i+52}b_{i+45} \oplus b_{i+33}b_{i+28}b_{i+21} \\ & \oplus b_{i+63}b_{i+45}b_{i+28}b_{i+9} \oplus b_{i+60}b_{i+52}b_{i+37}b_{i+33} \\ & \oplus b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} \oplus b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9} \\ & \oplus b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21} \end{aligned} \tag{8}$$

The cipher is initialized using the key and IV bits as per the following:

$$b_i = k_i \quad for \quad 0 \le i \le 79, \tag{9}$$

$$s_i = IV_i \quad for \quad 0 \le i \le 79. \tag{10}$$

After initialization, the cipher is clocked 160 times without producing any key stream bit. In fact, the key stream bit is XOR'd with the feedback bit during the KSA. After 160 rounds, we get our first key stream bit.

### *A2: ACORN v3*

We briefly state here the description of ACORN v3 relevant to our work, i.e. we assume the plaintext message to be a stream of 0's and are concerned only about the key stream generation process (PRGA), hence initialization of the cipher has been omitted. As stated before, ACORN v3 has 6 LFSRs concatenated to form a 293 bit state. We denote the state of the cipher by $S^t$ and its respective bits as: $S^t_0 \ldots S^t_{292}$. The cipher has the following three functions:

1) **Output Function.** The output bit $z_t$ for any state $t$ is generated as:

$$z_t = S^t_{12} \oplus S^t_{154} \oplus maj(S^t_{235}, S^t_{61}, S^t_{193}) \oplus ch(S^t_{230}, S^t_{111}, S^t_{66}) \tag{11}$$

2) **Feedback Function.** The feedback bit $f_t$ for any state $t$ is generated as:

$$f_t = S^t_0 \oplus (\sim S^t_{107}) \oplus maj(S^t_{244}, S^t_{23}, S^t_{160}) \oplus (ca_i \,\& S^t_{196}) \oplus (cb_i \& ks_i) \tag{12}$$

3) **State Update Function.** Before performing the shift, the bits $S^t_{289}, S^t_{230}, S^t_{193}, S^t_{154}, S^t_{107}, S^t_{61}$ are updated as follows:

$$S^t_{289} = S^t_{289} \oplus S^t_{235} \oplus S^t_{230} \tag{13}$$

$$S^t_{230} = S^t_{230} \oplus S^t_{196} \oplus S^t_{193} \tag{14}$$

$$S^t_{193} = S^t_{193} \oplus S^t_{160} \oplus S^t_{154} \tag{15}$$

$$S^t_{154} = S^t_{154} \oplus S^t_{111} \oplus S^t_{107} \tag{16}$$

$$S^t_{107} = S^t_{107} \oplus S^t_{66} \oplus S^t_{61} \tag{17}$$

$$S^t_{61} = S^t_{61} \oplus S^t_{23} \oplus S^t_0 \tag{18}$$

And then the bits are shifted in the following manner:

$$S^{t+1}_i = S^t_{i+1} \;\forall\; i \;\in\; [0, 291] \tag{19}$$

with the last bit initialized with the feedback bit:

$$S^{t+1}_{292} = f_t \tag{20}$$

### *A3: Lizard*

The 121-bit inner state of Lizard is divided into two NFSRs namely NFSR1 and NFSR2. At time $t$, the first NFSR, NFSR1 is denoted by $(S^t_0, \ldots, S^t_{30})$ and the second NFSR, NFSR2 by $(B^t_0, \ldots, B^t_{89})$. NFSR1 is of 31 bit and the update rule of this NFSR is

$$\begin{aligned} S^{t+1}_{30} = {} & S^t_0 \oplus S^t_2 \oplus S^t_5 \oplus S^t_6 \oplus S^t_{15} \oplus S^t_{17} \oplus S^t_{18} \oplus S^t_{20} \\ & \oplus S^t_{25} \oplus S^t_8S^t_{18} \oplus S^t_8S^t_{20} \oplus S^t_{12}S^t_{21} \oplus S^t_{14}S^t_{19} \\ & \oplus S^t_{17}S^t_{21} \oplus S^t_{20}S^t_{22} \oplus S^t_4S^t_{12}S^t_{22} \oplus S^t_4S^t_{19}S^t_{22} \\ & \oplus S^t_7S^t_{20}S^t_{21} \oplus S^t_8S^t_{18}S^t_{22} \oplus S^t_8S^t_{20}S^t_{22} \oplus S^t_{12}S^t_{19}S^t_{22} \\ & \oplus S^t_{20}S^t_{21}S^t_{22} \oplus S^t_4S^t_7S^t_{12}S^t_{21} \oplus S^t_4S^t_7S^t_{19}S^t_{21} \\ & \oplus S^t_4S^t_{12}S^t_{21}S^t_{22} \oplus S^t_4S^t_{19}S^t_{21}S^t_{22} \oplus S^t_7S^t_8S^t_{18}S_{21} \\ & \oplus S^t_7S^t_8S^t_{20}S^t_{21} \oplus S^t_7S^t_{12}S^t_{19}S^t_{21} \oplus S^t_8S^t_{18}S^t_{21}S^t_{22} \\ & \oplus S^t_8S^t_{20}S^t_{21}S^t_{22} \oplus S^t_{12}S^t_{19}S^t_{21}S^t_{22} \end{aligned}$$

The second register NFSR2 is of 90 bit and the update rule of this NFSR is

$$\begin{aligned}B_{89}^{t+1} = {} & S_0^t \oplus B_0^t \oplus B_{24}^t \oplus B_{49}^t \oplus B_{79}^t \oplus B_{84}^t \oplus B_3^t B_{59}^t \\ & \oplus B_{10}^t B_{12}^t \oplus B_{15}^t B_{16}^t \oplus B_{25}^t B_{53}^t \oplus B_{35}^t B_{42}^t \\ & \oplus B_{55}^t B_{58}^t \oplus B_{60}^t B_{74}^t \oplus B_{20}^t B_{22}^t B_{23}^t \\ & \oplus B_{62}^t B_{68}^t B_{72}^t \oplus B_{77}^t B_{80}^t B_{81}^t B_{83}\end{aligned}$$

Output bit $z_t$ is a function from $\{0,1\}^{53}$ to $\{0,1\}$. At time $t$, $z_t = \mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t \oplus \overline{\mathcal{T}}_t$, where

- $\mathcal{L}_t = B_7^t \oplus B_{11}^t \oplus B_{30}^t \oplus B_{40}^t \oplus B_{45}^t \oplus B_{54}^t \oplus B_{71}^t$
- $\mathcal{Q}_t = B_4^t B_{21}^t \oplus B_9^t B_{52}^t \oplus B_{18}^t B_{37}^t \oplus B_{44}^t B_{76}^t$
- $\mathcal{T}_t = B_5^t \oplus B_8^t B_{82}^t \oplus B_{34}^t B_{67}^t B_{73}^t \oplus B_2^t B_{28}^t B_{41}^t B_{65}^t \oplus B_{13}^t B_{29}^t B_{50}^t B_{64}^t B_{75}^t \oplus B_6^t B_{14}^t B_{26}^t B_{32}^t B_{47}^t B_{61}^t \oplus B_1^t B_{19}^t B_{27}^t B_{43}^t B_{57}^t B_{66}^t B_{78}^t$
- $\overline{\mathcal{T}}_t = S_{23}^t \oplus S_3^t S_{16}^t \oplus S_9^t S_{13}^t B_{48}^t \oplus S_1^t S_{24}^t B_{38}^t B_{63}^t$

The state initialization process is divided into 4 phases.

**Phase 1: Key and IV Loading:** Let $K = (K_0, \ldots, K_{119})$ be the 120-bit key and $IV = (IV_0, \ldots, IV_{63})$ the 64-bit public IV. The state is initialized as follows:

$$B_j^0 = \begin{cases} K_j \oplus IV_j, & \text{for } 0 \leq j \leq 63 \\ K_j, & \text{for } 64 \leq j \leq 89 \end{cases}$$

$$S_j^0 = \begin{cases} K_{j+90}, & \text{for } 0 \leq j \leq 28 \\ K_{119+1}, & \text{for } j = 29 \\ 1, & \text{for } j = 30 \end{cases}$$

**Phase 2: Grain-like Mixing:** In this phase the output bit $z_t$ is fed back into both NFSRs for $0 \leq t \leq 127$. This type of approach is used in Grain family.

**Phase 3: Second Key Addition:** In this phase, the 120-bit key is XORed to both NFSRs as follows:

$$B_j^{129} = B_j^{128} \oplus K_j, \quad \text{for } 0 \leq j \leq 89$$

$$S_j^{129} = \begin{cases} S_j^{128} \oplus K_{j+90}, & \text{for } 0 \leq j \leq 29 \\ 1, & \text{for } j = 30 \end{cases}$$

**Phase 4: Final Diffusion** This phase is exactly similar to phase 2 except $z_t$ is not fed back into the NFSRs. In this phase, one has to run both NFSRs 128 rounds. So after this phase, registers are $(S_0^{257}, \ldots, S_{30}^{257})$ and $(B_0^{257}, \ldots, B_{89}^{257})$. Now Lizard is ready to produce output key stream bits.