Differential Fault Attack against Grain family with very few faults and minimal assumptions

Santanu Sarkar[1], Subhadeep Banik[2] and Subhamoy Maitra[2]

[1] National Institute of Standards and Technology, 100 Bureau Drive, Stop 8930, Gaithersburg, MD 20899-8930, USA
sarkar.santanu.bir@gmail.com
[2] Applied Statistics Unit, Indian Statistical Institute, 203 B T Road, Kolkata 700 108, India
s.banik_r@isical.ac.in, subho@isical.ac.in

Abstract. The series of published works, related to Differential Fault Attack (DFA) against the Grain family, require (i) quite a large number (hundreds) of faults (around $n \ln n$, where $n = 80$ for Grain v1 and $n = 128$ for Grain-128, Grain-128a) and also (ii) several assumptions on location and timing of the fault injected. In this paper we present a significantly improved scenario from the adversarial point of view for DFA against the Grain family of stream ciphers. Our model is the most realistic one so far as it considers that the cipher to be re-keyed a very few times and fault can be injected at any random location and at any random point of time, i.e., no precise control is needed over the location and timing of fault injections. We construct equations based on the algebraic description of the cipher by introducing new variables so that the degrees of the equations do not increase. In line of algebraic cryptanalysis, we accumulate such equations based on the fault-free and faulty key-stream bits and solve them using the SAT Solver Cryptominisat-2.9.5 installed with SAGE 5.7. In a few minutes we can recover the state of Grain v1, Grain-128 and Grain-128a with as little as 10, 4 and 10 faults respectively (and may be improved further with more computational efforts).

Keywords: Differential Fault Attack, Grain v1, Grain-128, Grain-128a, LFSR, NFSR, SAT Solver, Stream Cipher.

1 Introduction

Fault attacks study the robustness of a cryptosystem in a setting that is weaker than its original or expected mode of operation. Though optimistic, this model of attack can successfully be employed against a number of proposals. In a practical setting, it is indeed possible to mount such an attack when the number of faults is very low and we do not require precise controls over fault injection, both in terms of exact register locations as well as timing. In this paper we achieve these goals and present the most efficient DFA against the Grain family. Our main contribution in this paper is to generate a large number of equations from each of the faults and further, introduce new variables at each stage to keep the degree of each equation as low as possible. We use the SAT solver Cryptominisat-2.9.5 [30] installed in SAGE [32] to solve the equations towards obtaining the complete secret key.

Fault attacks have received serious attention in literature for quite some time [10, 11]. Such attacks on stream ciphers have been studied in [21] where a typical attack scenario consists of an adversary who can inject a random fault (using laser shots/clock glitches [28, 29]) in a cryptographic device as a result of which one or more bits of its internal state get altered. The faulty output from this altered device is then used to deduce information about its internal state/secret key. Here the adversary requires certain privileges like the ability to re-key the device, control the timing of the fault etc. The more privileges the adversary is granted, the more the attack becomes impractical and unrealistic.

The Grain family of stream ciphers [1,2,19,20] has received a lot of attention and in particular, Grain v1 is in the hardware profile of eStream [17]. In most of the fault attacks reported so far [4, 8, 24] on this cipher, the adversary is granted far too many privileges to make the attacks practical. In particular, these works consider reproducing the faults in the same location more than once. This particular assumption has been relaxed in the work [5], where faulting the same location more than once could be avoided. Further this work [5] considered some restricted cases to accommodate situations when a fault affects more than one register location. For detailed cryptanalytic results related to this family, the reader may refer to [3,7,9,13–16,18,25,26,31,33,34].

A DFA on Trivium using SAT solvers has been presented in [27] (see also [22, 23] for other DFAs on Trivium). For more than a decade, there has been seminal research in the area of algebraic cryptanalysis. The main idea here is to solve multivariate polynomial systems that describe a cipher. For a very brief introduction in this, one may refer [27, Section 5]. The DFA on Trivium [27] requires only 2 faults and this is far fewer than the fault requirements against the Grain family. This motivates us to see how this kind of algebraic cryptanalysis can be exploited towards DFA against Grain family.

Our Contribution. The idea of [5] considers introducing faults in all the LFSR locations. Given an n-bit LFSR, this requires an expected $n \ln n$ fault injections which is around $2^{8.5}$ for $n = 80$ as in Grain v1. Our preliminary intuition was that the number of faults used in the existing works [4, 5, 8, 24] is considerably high. This is because, for each fault, one can generate a significant number of equations using the faulty key-stream bits that might be enough to solve the system. This convinced us that one actually requires a very few faults. We generated the equations carefully, by introducing new variables at each stage, so that although the number of variables increased, the degrees of the equations were kept in check. Since the degrees of the equations were low, they were fed into a good SAT solver to obtain the solutions in the direction of [27]. We found that this strategy indeed succeeds and it is possible to recover the secret key from a very few faults (less than ten). Furthermore, all the published literature on DFAs against the Grain family require the adversary to be able to inject fault at a precise stage of operation i.e. at the beginning of the PRGA. In this work we will show that in Grain v1 and Grain-128, the adversary does not need to impose such precise control over fault injections. If the adversary is able to to inject fault at some PRGA round $\tau \in [0, \tau_{max} - 1]$ (the value of τ_{max} varies for Grain v1 and Grain-128 and is related to the algebraic structure of each cipher), then with high probability she is able to deduce the value of τ and also the register location that the fault has affected. The same can not be reproduced for Grain-128a because the cipher does not make all the key-stream bits directly available to the attacker. Thus, we have the following advantages.

- We require very few faults and thus the actual hardware will be minimally stressed and the DFA can be implemented in practice.
- So far all the DFAs on Grain family considered injecting fault either in the LFSR or in the NFSR. Here we can tackle the cases where fault is injected randomly considering both the LFSR and the NFSR at the same time.
- We also improve upon the technique of fault location identification proposed in [5] by introducing two new sets of signature vectors $\mathcal{Q}^3_\phi, \mathcal{Q}^4_\phi \in \{0,1\}^{2n}$ (in addition to the two sets $\mathcal{Q}^1_\phi, \mathcal{Q}^2_\phi \in \{0,1\}^{2n}$ described in [5]) so that the probability of success in exact fault location identification improves.
- For Grain v1 and Grain-128 we outline techniques that allows the adversary to relax requirements related to the timing of fault injections.

Table 1 summarizes the contribution of our work with previous work in this topic.

Table 1. Comparison of this work with previous DFA′s on Grain

		CHES 2012 [4]	Indocrypt 2012 [5]	This work
# Faults	Grain v1	$2^{9.05}$	$2^{8.45}$	**10**
	Grain-128	-	$2^{9.27}$	**4**
	Grain-128a	-	$2^{9.27}$	**10**
Faults in		LFSR	LFSR	**LFSR/NFSR/Both**
Control over Fault Timing	Grain v1	Required	Required	**Not Required**
	Grain-128	Required	Required	**Not Required**
	Grain-128a	Required	Required	Required
Distinguish multiple bit-fault		No	Yes	Yes

2 Algebraic description of the Grain family

Consider $a_i, b_i, c_i \in \{0,1\}$ for $i \in [0, \ldots, n-1]$. Any cipher in the Grain family consists of an n-bit LFSR and an n-bit NFSR (see Figure 1). The update function of the LFSR is given by the equation

$$y_{t+n} = f(Y_t) = \bigoplus_{i=0}^{n-1} c_i y_{t+i},$$

where $Y_t = [y_t, y_{t+1}, \ldots, y_{t+n-1}]$ is an n-bit vector that denotes the LFSR state at the t^{th} clock interval and f is a linear function on the LFSR state bits obtained from a primitive polynomial in $GF(2)$ of degree n.

The NFSR state is updated as

$$x_{t+n} = y_t \oplus g(X_t) = y_t \oplus g(x_t, \ldots, x_{t+n-1}).$$

Here, $X_t = [x_t, x_{t+1}, \ldots, x_{t+n-1}]$ is an n-bit vector that denotes the NFSR state at the t^{th} clock interval and g is a non-linear function of the NFSR state bits. It may very well happen that g is degenerate on some of its variables, i.e., all the NFSR bits may not contribute in the function g.

The output key-stream is produced by combining the LFSR and NFSR bits as

$$z_t = \bigoplus_{i=0}^{n-1} b_i y_{t+i} \oplus \bigoplus_{i=0}^{n-1} a_i x_{t+i} \oplus h(y_t, \ldots, y_{t+n-1}, x_t, \ldots, x_{t+n-1}),$$

where h is a non-linear Boolean function, and may be degenerate on some of the variables.

Key Scheduling Algorithm (KSA). The Grain family uses an n-bit key K, and an m-bit initialization vector IV, with $m < n$. The key is loaded in the NFSR and the IV is loaded in the 0^{th} to the $(m-1)^{th}$ bits of the LFSR. The remaining m^{th} to $(n-1)^{th}$ bits of the LFSR are loaded with some fixed pad $P \in \{0,1\}^{n-m}$. Then, for the first $2n$ clocks, the key-stream bit z_t is XOR-ed to both the LFSR and NFSR update functions.

Pseudo-Random key-stream Generation Algorithm (PRGA). After the KSA, z_t is no longer XOR-ed to the LFSR and the NFSR but it is used as the Pseudo-Random key-stream bit. Thus, during this phase, the LFSR and NFSR are updated as $y_{t+n} = f(Y_t), x_{t+n} = y_t \oplus g(X_t)$.

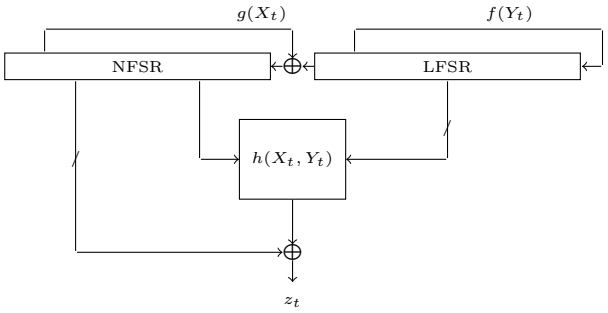

Fig. 1. Structure of Stream Cipher in Grain Family

MAC Generation Algorithm (MGA) in Grain-128a. Grain-128a [1, 2] also considers generation of MAC. Here we follow the description given in [2]. Let $z_0, z_1, z_2, \ldots$ denote the key-stream bits produced by the cipher. Assume that we have a message of length L defined by the bits $m_0, \ldots, m_{L-1}$. Set $m_L = 1$ as padding. To provide authentication, two registers, called accumulator and shift register of size 32 bits each, are used. The content of accumulator and shift register at time t is denoted by $a_t^0, \ldots, a_t^{31}$ and $r_t, \ldots, r_{t+31}$ respectively. The accumulator is initialized through $a_0^j = z_j, 0 \leq j \leq 31$ and the shift register is initialized through $r_j = z_{32+j}, 0 \leq j \leq 31$. The shift register is updated as $r_{t+32} = z_{64+2t+1}$. The accumulator is updated as $a_{t+1}^j = a_t^j \oplus m_t r_{t+j}$ for $0 \leq j \leq 31$ and $0 \leq t \leq L$. The final content of accumulator, $a_{L+1}^0, \ldots, a_{L+1}^{31}$ is used for authentication. For fault attack, we need to consider the key-stream bits and thus it is important to note that we cannot use the first 64 bits of key-stream and also then each alternative key-stream bit as those are used in MAC for Grain-128a.

3 Description of the attack using SAT solver

Let us first consider that the exact location (only a single register) of the fault is known. We will later (in Section 4) discuss the issues related to **(1)** how we can obtain the exact location of the fault after injecting the fault at a random location, **(2)** how we can reject the cases where fault has disturbed more than one locations and **(3)** how he can guess the time of injection of a randomly timed fault. To explain our idea for exploiting the SAT solver more precisely, let us now consider that the fault will be injected after the KSA, i.e., just before the PRGA starts.

3.1 Populating the bank of equations for Grain v1 and Grain-128

We will now explain the method of obtaining a large number of equations that will be used for algebraic cryptanalysis. For the time being we will consider the case that will work for Grain v1 or Grain-128. The case of Grain-128a will be little different that we will discuss next.

Equations from fault-free key-stream Let us first consider the equations from the ℓ-bit fault-free key-stream $z_0, \ldots, z_{\ell-1}$. As discussed, the LFSR state just after the KSA (at the beginning of the 0-th clock) is $Y_0 = [y_0, y_1, \ldots, y_{n-1}]$ and the NFSR state is $X_0 = [x_0, x_1, \ldots, x_{n-1}]$. In general, the value of ℓ required to complete the attack is more than 160 for all the three versions of Grain. But it is not feasible to compute the Algebraic Normal Form (ANF) of z_{159} on any standard PC, for any version of Grain. For example using a workstation with 1.83 GHz processor, 3 GHz RAM

Table 2. Exact description of the three ciphers following [5].

	Grain v1	Grain-128	Grain-128a
n	80	128	128
m	64	96	96
Pad	FFFF	FFFFFFFF	FFFFFFFE
$f(\cdot)$	$y_{t+62} \oplus y_{t+51} \oplus y_{t+38}$ $\oplus y_{t+23} \oplus y_{t+13} \oplus y_t$	$y_{t+96} \oplus y_{t+81} \oplus y_{t+70}$ $\oplus y_{t+38} \oplus y_{t+7} \oplus y_t$	$y_{t+96} \oplus y_{t+81} \oplus y_{t+70}$ $\oplus y_{t+38} \oplus y_{t+7} \oplus y_t$
$g(\cdot)$	$x_{t+62} \oplus x_{t+60} \oplus x_{t+52}$ $\oplus x_{t+45} \oplus x_{t+37} \oplus x_{t+33}$ $x_{t+28} \oplus x_{t+21} \oplus x_{t+14}$ $x_{t+9} \oplus x_t \oplus x_{t+63}x_{t+60} \oplus$ $x_{t+37}x_{t+33} \oplus x_{t+15}x_{t+9}$ $x_{t+60}x_{t+52}x_{t+45} \oplus x_{t+33}$ $x_{t+28}x_{t+21} \oplus x_{t+63}x_{t+60}$ $x_{t+21}x_{t+15} \oplus x_{t+63}x_{t+60}$ $x_{t+52}x_{t+45}x_{t+37} \oplus x_{t+33}$ $x_{t+28}x_{t+21}x_{t+15}x_{t+9} \oplus$ $x_{t+52}x_{t+45}x_{t+37}x_{t+33}$ $x_{t+28}x_{t+21}$	$y_t \oplus x_t \oplus x_{t+26} \oplus$ $x_{t+56} \oplus x_{t+91} \oplus x_{t+96} \oplus$ $x_{t+3}x_{t+67} \oplus x_{t+11}x_{t+13}$ $\oplus x_{t+17}x_{t+18} \oplus x_{t+27}x_{t+59}$ $\oplus x_{t+40}x_{t+48} \oplus x_{t+61}$ $x_{t+65} \oplus x_{t+68}x_{t+84}$	$y_t \oplus x_t \oplus x_{t+26} \oplus$ $x_{t+56} \oplus x_{t+91} \oplus x_{t+96} \oplus$ $x_{t+3}x_{t+67} \oplus x_{t+11}x_{t+13}$ $\oplus x_{t+17}x_{t+18} \oplus x_{t+27}x_{t+59}$ $\oplus x_{t+40}x_{t+48} \oplus x_{t+61}$ $x_{t+65} \oplus x_{t+68}x_{t+84}$ $\oplus x_{t+88}x_{t+92}x_{t+93}x_{t+95}$ $\oplus x_{t+22}x_{t+24}x_{t+25} \oplus$ $x_{t+70}x_{t+78}x_{t+82}$
$h(\cdot)$	$y_{t+3}y_{t+25}y_{t+46} \oplus y_{t+3}$ $y_{t+46}y_{t+64} \oplus y_{t+3}y_{t+46}$ $x_{t+63} \oplus y_{t+25}y_{t+46}x_{t+63} \oplus$ $y_{t+46}y_{t+64}x_{t+63} \oplus y_{t+3}$ $y_{t+64} \oplus y_{t+46}y_{t+64} \oplus y_{t+64}$ $x_{t+63} \oplus y_{t+25} \oplus x_{t+63}$	$x_{t+12}x_{t+95}y_{t+95} \oplus x_{t+12}$ $y_{t+8} \oplus y_{t+13}y_{t+20} \oplus x_{t+95}$ $y_{t+42} \oplus y_{t+60}y_{t+79}$	$x_{t+12}x_{t+95}y_{t+94} \oplus x_{t+12}$ $y_{t+8} \oplus y_{t+13}y_{t+20} \oplus x_{t+95}$ $y_{t+42} \oplus y_{t+60}y_{t+79}$
z_t	$x_{t+1} \oplus x_{t+2} \oplus x_{t+4} \oplus$ $x_{t+10} \oplus x_{t+31} \oplus x_{t+43}$ $x_{t+56} \oplus h$	$x_{t+2} \oplus x_{t+15} \oplus x_{t+36} \oplus$ $x_{t+45} \oplus x_{t+64} \oplus x_{t+73}$ $\oplus x_{t+89} \oplus y_{t+93} \oplus h$	$x_{t+2} \oplus x_{t+15} \oplus x_{t+36} \oplus$ $x_{t+45} \oplus x_{t+64} \oplus x_{t+73}$ $\oplus x_{t+89} \oplus y_{t+93} \oplus h$

and 2 MB system cache, computing the ANF of any z_ℓ in Grain v1, for $\ell > 44$ is infeasible. The ANF of z_{44} itself has algebraic degree 17 and consists of 80643 monomials.

Also we have to keep in mind that we expect to solve these solutions using SAT solver. SAT solvers solve a polynomial equation system by converting the ANF′s to their equivalent Conjunctive Normal Forms (CNF′s). As we will see in Section 3.3, the representation of each degree d monomial requires $d + 1$ CNF clauses. Thus to enable the SAT solver to solve the system efficiently, the degrees of the expressions in the equation system must also be controlled.

In order to overcome both limitations, we use a technique popularly used in ANF-CNF conversions [6]. At each PRGA round $t > 0$, we introduce two new variables y_{t+n}, x_{t+n} to update the LFSR and NFSR state respectively. To illustrate the technique, let us denote the states at the beginning of the t-th ($t \geq 0$) PRGA round as

$$Y_t = [y_t, y_{t+1}, \ldots, y_{t+n-1}], \quad X_t = [x_t, x_{t+1}, \ldots, x_{t+n-1}].$$

Given these, we formulate the following equations.

1. LFSR equation: $y_{t+n} = f(Y_t)$.
2. NFSR equation: $x_{t+n} = y_t \oplus g(X_t)$.
3. Key-stream equation: $z_t = \bigoplus_{i=0}^{n-1} b_i y_{t+i} \oplus \bigoplus_{i=0}^{n-1} a_i x_{t+i} \oplus h(y_t, \ldots, y_{t+n-1}, x_t, \ldots, x_{t+n-1})$.

In the Grain family, while the first equation is linear, the degrees of the other two equations are also not very high. We initially start with $2n$ variables, $y_0, y_1, \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$. Then corresponding to each key-stream bit z_t, we introduce two new variables y_{t+n}, x_{t+n} and obtain three more equations. Thus we have in total $2n + 2\ell$ variables and 3ℓ equations. The advantage of using such a technique is as follows.

- First of all it allows us to formulate the expression for z_ℓ (via a series of equations) for values of $\ell \geq 159$. Instead, if at each round $t > 0$, the variables y_{t+n}, x_{t+n} were replaced by their equivalent algebraic expressions in $y_0, y_1, \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$, this would never have been possible on an ordinary PC.
- Since the expressions in the LFSR and NFSR cells always stay linear, this allows us to control the algebraic degree and the number of monomials in each of the 3ℓ equations so obtained.

Equations from faulty key-streams We use a similar technique to extract equations from faulty key-streams. Let us assume that a fault is injected in the LFSR location ϕ at PRGA round 0. The same method will work if the fault is injected in the NFSR. Since we re-key the cipher with the same Key-IV before injecting a fault, after fault injection we obtain the state $y_0, y_1, \ldots, y_{\phi-1}, 1 \oplus y_\phi, y_{\phi+1} \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$ at the start of PRGA. Then corresponding to each key-stream bit z_t, we introduce two new variables $y^{(\phi)}_{t+n}, x^{(\phi)}_{t+n}$ and obtain three more equations. Thus we have additional 2ℓ variables and 3ℓ equations.

Total number of variables and equations Given that we introduce ν faults after these many re-keyings, the total number of variables is $2n + 2(\nu + 1)\ell$ and the total number of equations is $3(\nu + 1)\ell$.

3.2 Populating the bank of equations for Grain-128a

We will now explain the formation of equations for Grain-128a. Here the first 64 key-stream bits $z_0, \ldots, z_{63}$ and every other (alternating) key-stream bits thereafter are used to construct MAC. Hence these bits are unavailable to the attacker.

Equations from fault-free key-stream Let us first consider the equations from the ℓ-bit fault-free key-stream $z_{64}, z_{66}, \ldots, z_{64+2\ell-2}$ as only alternative key-stream bits are used for encryption. Hence, similar to the above, we have the following equations.

1. 2 LFSR equations: $y_{t+n} = f(Y_t)$ and $y_{t+n+1} = f(Y_{t+1})$.
2. 2 NFSR equations: $x_{t+n} = y_t \oplus g(X_t)$ and $x_{t+n+1} = y_{t+1} \oplus g(X_{t+1})$.
3. 1 Key-stream equation: $z_t = \bigoplus_{i=0}^{n-1} b_i y_{t+i} \oplus \bigoplus_{i=0}^{n-1} a_i x_{t+i} \oplus h(y_t, \ldots, y_{t+n-1}, x_t, \ldots, x_{t+n-1})$.

We initially start with $2n$ variables, $y_0, y_1, \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$. Then corresponding to each key-stream bit z_t, we introduce four new variables $y_{t+n}, y_{t+n+1}, x_{t+n}, x_{t+n+1}$ and obtain five more equations. Thus we have in total $2n + 4\ell$ variables and 5ℓ equations.

Equations from faulty key-streams Let us consider that a fault is injected in the LFSR location ϕ at the beginning of the PRGA. Again, the method works similarly if the fault is injected in the NFSR. Since we will re-key the cipher with the same Key-IV, in such a case we will obtain the state $y_0, y_1, \ldots, y_{\phi-1}, 1 \oplus y_\phi, y_{\phi+1} \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$. Then corresponding to each key-stream bit z_t, we introduce four new variables $y^{(\phi)}_{t+n}, y^{(\phi)}_{t+n+1}, x^{(\phi)}_{t+n}, x^{(\phi)}_{t+n+1}$ and obtain five more equations. Thus we have additional 4ℓ variables and 5ℓ equations.

Total number of variables and equations Given that we introduce ν faults after these many re-keyings, the total number of variables is $2n+4(\nu+1)\ell$ and the total number of equations is $5(\nu+1)\ell$. All these equations are used in the SAT solver to obtain $y_0, y_1, \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$. This completes the attack. As the ciphers in the Grain family are invertible both in KSA and PRGA, one can also get the secret key efficiently.

3.3 Using the SAT Solver

To solve polynomial systems of multivariate equations by SAT solvers, the attacker initially converts the system from Algebraic Normal Form (ANF) to Conjunctive Normal Form (CNF). We will show some standard techniques of how this can be done:

Conversion of monomials Any monomial of the form $x_1x_2\cdots x_d$ is first equated to another variable β (say). The tautological equivalent of $\beta = x_1x_2\cdots x_d$ is $\beta \Leftrightarrow x_1x_2\cdots x_d$, which is same as the boolean expression β XNOR $x_1x_2\cdots x_d$. This therefore can be expressed as

$$(\overline{\beta} \vee x_1) \wedge (\overline{\beta} \vee x_2) \wedge \cdots \wedge (\overline{\beta} \vee x_d) \wedge (\beta \vee \overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_d})$$

As can be seen this adds $d+1$ clauses to the system each of which need to be TRUE for the correct solution.

Conversion of linear expressions A linear system of the form $x_1 + x_2 + \ldots + x_k = 1$ can be expressed equivalently as the boolean expression x_1 XOR x_2 XOR $\cdots x_k$. For example $x_1 + x_2 + x_3 + x_4 = 1$ can be expressed as

$$(x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge$$
$$(\overline{x_1} \vee \overline{x_2} \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$

A linear system of the form $x_1+x_2+\ldots+x_k = 0$ can be expressed equivalently as the boolean expression $(x_1$ XOR x_2 XOR $\cdots x_k)'$. For example $x_1 + x_2 + x_3 = 1$ can be expressed as

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

Conversion of large XOR chains In the above system, the number of clauses obtained depends on the value k. it can easily be shown that this number is 2^{k-1}. To prevent accumulation of such large chain of clauses, we introduce dummy variables γ_i at each stage. For example, $x_1 + x_2 + \ldots + x_k = 0$ is equivalent to:

$$x_1 + x_2 + x_3 + \gamma_1 = 0, \quad \gamma_1 + x_4 + x_5 + \gamma_2 = 0, \quad \cdots \gamma_M + x_{k-2} + x_{k-1} + x_k = 0.$$

This gives rise to $M = \frac{k}{2} - 1$ equations each having $2^{4-1} = 8$ clauses.

Example 1. To solve the equation system $x_1x_2+x_2x_3+x_4 = 0$, $x_2x_3+x_3+x_1 = 0$, we translate the system into the following form **[1]** $\beta_1 = x_1x_2$, **[2]** $\beta_2 = x_2x_3$, **[3]** $\beta_1 + \beta_2 + x_4 = 0$, **[4]** $\beta_2 + x_3 + x_1 = 0$.

[1] gives us $(\overline{\beta_1} \vee x_1) \wedge (\overline{\beta_1} \vee x_2) \wedge (\beta_1 \vee \overline{x_1} \vee \overline{x_2}) =$ TRUE.
[2] gives us $(\overline{\beta_2} \vee x_2) \wedge (\overline{\beta_2} \vee x_3) \wedge (\beta_2 \vee \overline{x_2} \vee \overline{x_3}) =$ TRUE.
[3] gives us $(\beta_1 \vee \beta_2 \vee \overline{x_4}) \wedge (\beta_1 \vee \overline{\beta_2} \vee x_4) \wedge (\overline{\beta_1} \vee \beta_2 \vee x_4) \wedge (\overline{\beta_1} \vee \overline{\beta_2} \vee \overline{x_4}) =$ TRUE.
[4] gives us $(x_1 \vee \beta_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{\beta_2} \vee x_3) \wedge (\overline{x_1} \vee \beta_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{\beta_2} \vee \overline{x_3}) =$ TRUE.

After the system of algebraic equations have been converted to their equivalent CNF, they are passed on to the SAT solver for extracting a solution.

4 How to identify the location of a random fault

Initially, we will assume that the adversary is able to inject faults at the beginning of the PRGA. In the next section, we will relax this requirement and show how to deduce the injection time of a randomly timed fault (for Grain v1 and Grain-128). Now so far, in all the published literature on fault analysis of Grain [4, 5, 8, 24], either the LFSR or the NFSR has been chosen for fault injection. Techniques have been proposed in all of the above works, to identify the location of a randomly applied bit fault in the internal state, provided the adversary knows apriori whether it is the LFSR or the NFSR she is injecting faults in. In the attack we propose, the adversary does not need apriori knowledge of this information, i.e., she injects a fault in a random bit location of the internal state without knowing whether the fault has affected a location in the LFSR or the NFSR. We will propose a technique (along the lines of [5]) that will enable the adversary to not only identify the location of an injected fault but also help him determine whether the fault was injected in the LFSR or NFSR.

The idea of determining the location of a randomly applied fault in the LFSR of the Grain family by comparing the difference of the faultless and faulty key-stream sequence with certain pre-computed signature vectors was first introduced in [4]. This technique however required the adversary to be able to fault the same LFSR location more than once to conclusively determine the fault location. This idea was further developed in [5] where the differential key-stream was compared with two sets of vectors called the First and Second signature vectors. Using this technique it was no longer necessary to fault the same location more than once and it enabled the adversary to exercise less control over fault injections.

4.1 First and Second signature Vectors

We will summarize the basic ideas in [4,5]. Consider 2 initial PRGA states S and S^ϕ which differ only in the LFSR location ϕ i.e. S^ϕ is produced when a random fault toggles the LFSR location ϕ of S. Let $Z = [z_0, z_1, z_2, \ldots]$ and $Z^\phi = [z_0^\phi, z_1^\phi, z_2^\phi, \ldots]$ be the faultless and faulty key-stream sequence produced by S and S^ϕ respectively.

a. At certain PRGA rounds i, z_i and z_i^ϕ are guaranteed to be equal, irrespective of the exact value of S. This is because at each PRGA round i, only a few bits of the internal state are used to produce z_i. Therefore at all rounds i when the faulty and faultless internal states differ in bit locations which have no contribution towards z_i, the faulty and faultless key-stream bits are guaranteed to be equal.

b. Furthermore at certain other PRGA rounds j, z_j and z_j^ϕ are guaranteed to be unequal for all values of S. Again certain bits of the internal state are linearly xor-ed to the output function h to produce the output key-stream bit. It is when an induced fault causes a deterministic difference between S and S^ϕ in an odd number of these bits that the output bits are guaranteed to be unequal.

c. For every fault location ϕ $(0 \leq \phi < n)$ in the LFSR, one can define [5] two $2n$ length vectors $\mathcal{Q}_\phi^1, \mathcal{Q}_\phi^2$ (called the **First and Second signature Vectors** for the location ϕ) as follows:

$$\mathcal{Q}_\phi^1(i) = \begin{cases} 1, \text{if } z_i = z_i^\phi, \ \forall\ S, \\ 0, \ \text{otherwise.} \end{cases} \qquad \mathcal{Q}_\phi^2(i) = \begin{cases} 1, \text{if } z_i \neq z_i^\phi, \ \forall\ S, \\ 0, \ \text{otherwise.} \end{cases}$$

The signature vectors can be efficiently computed by performing analysis of the differential trail of the Grain PRGA following the methods described in [5].

d. The location identification algorithm consists of comparing $E^\phi = Z \oplus Z^\phi$ with $\mathcal{Q}^1_\phi$, $\mathcal{Q}^2_\phi$ for all $\phi \in [0, n-1]$ and finding a match. For any element $V \in \{0,1\}^{2n}$, define the support of V as

$$\Pi_V = \{i : 0 \leq i < 2n,\ V(i) = 1\}.$$

Now define a relation $\preceq$ in $\{0,1\}^{2n}$ such that for 2 elements $V_1, V_2 \in \{0,1\}^{2n}$, $V_1 \preceq V_2$ if $\Pi_{V_1} \subseteq \Pi_{V_2}$. For the correct value of ϕ, $\Pi_{\mathcal{Q}^1_\phi} \subseteq \Pi_{E^\phi}, \Pi_{\mathcal{Q}^2_\phi} \subseteq \Pi_{1 \oplus E^\phi}$ and hence $\mathcal{Q}^1_\phi \preceq E^\phi,\ \mathcal{Q}^2_\phi \preceq 1 \oplus E^\phi$. So the strategy is to formulate the first candidate set $\Psi_{0,\phi} = \{\psi : 0 \leq \psi \leq n-1,\ \mathcal{Q}^1_\psi \preceq E^\phi\}$. If $|\Psi_{0,\phi}|$ is 1, then the single element in $\Psi_{0,\phi}$ will give us the fault location ϕ. If not, we then formulate the second candidate set $\Psi_{1,\phi} = \{\psi : \psi \in \Psi_{0,\phi},\ \mathcal{Q}^2_\psi \preceq 1 \oplus E^\phi\}$. If $|\Psi_{1,\phi}|$ is 1, then the single element in $\Psi_{1,\phi}$ will give us the fault location ϕ. If $\Psi_{1,\phi}$ has more than one element, then the strategy fails.

4.2 Our results

We use the basic technique outlined in [5], but with a few tweaks. First of all, the work in [5] considers faults in LFSR locations only. We have extended the technique to determine the location of a fault introduced in either the LFSR or the NFSR. For this we increase the number of pre-computed first and second signature vectors to $2n$, i.e. one for each register location in the NFSR and LFSR. We now compare the differential vector E^ϕ with the first and second signature vectors of all the $2n$ register locations using the strategy outlined in **[d]**. After the comparison with the signature vectors the algorithm will either output

1. The LFSR or NFSR location ϕ of the induced fault, OR
2. If $|\Psi_{1,\phi}| > 1$, then it outputs a failure message.

Further, we also introduce two additional signature vectors over the two described in [5].

We performed computer experiments by simulating random single bit faults for 2^{20} randomly chosen Key-IVs. The probability that the new algorithm identifies the correct fault location in the LFSR or the NFSR i.e. $P(|\Psi_{1,\phi}| = 1)$ is around 1.00 for Grain v1, 1.00 for Grain-128 and 0.81 for Grain-128a (improving the success probabilities of [5] that were around 0.99 for Grain v1, 1.00 for Grain-128 and 0.79 for Grain-128a).

4.3 Improving the success probabilities: Third and Fourth signature Vectors

While the probabilities of success of fault location identification are very high (close to 1) for both Grain v1 and Grain-128, it is around 0.79 for Grain-128a. One of the reasons why the success probability relatively low for Grain-128a is because the cipher does not make each and every key-stream bit directly available to the adversary. As has been explained, the key-stream bits of the first 64 rounds and every alternate round thereafter contribute to the computation of the MAC and is not directly available to the adversary. This limits the information available to the location identification algorithm and hence the slightly low success probability.

This leaves plenty of room for improving the success probabilities towards 1. We already know that given a single bit fault in the internal state of the cipher, the faulty and the faultless key-streams at certain PRGA rounds are guaranteed to be equal and they are also guaranteed to be different at certain other rounds. However there may be situations when the difference of the faulty and faultless key-stream bits at a certain PRGA round i i.e. $z_i \oplus z_i^\phi$ is deterministically equal or unequal to the difference of the faulty and faultless key-stream bits at some other PRGA round

j even though the difference of these bits at either rounds i or j themselves is not guaranteed to be either 0 or 1.

That is to say $z_i \oplus z_i^\phi = z_j \oplus z_j^\phi$ holds for all values of S, even if $z_i = z_i^\phi,\ z_j = z_j^\phi$ or $z_i = z_i^\phi \oplus 1,\ z_j = z_j^\phi \oplus 1$ does not hold. Alternatively, for some other values of i, j, ϕ, we may get $z_i \oplus z_i^\phi = 1 \oplus z_j \oplus z_j^\phi$ for all values of S, even if $z_i = z_i^\phi \oplus 1,\ z_j = z_j^\phi$ or $z_i = z_i^\phi,\ z_j = z_j^\phi \oplus 1$ does not hold. We will explain the occurrence of these events with the help of the following theorem.

Theorem 1. *Let $S,\ S^\phi$ be two internal states in Grain that differ only in the LFSR location ϕ at the beginning of the PRGA. Then for certain values of ϕ, there exists integers t_0, t_1 such that if $\delta_{t_0} = z_{t_0} \oplus z_{t_0}^\phi,\ \&\ \delta_{t_1} = z_{t_1} \oplus z_{t_1}^\phi$, then*

A. *Even if $\delta_{t_0} = 0,\ \delta_{t_1} = 0$ or $\delta_{t_0} = 1,\ \delta_{t_1} = 1$ does not hold for all values of S, $\delta_{t_0} = \delta_{t_1}$ always holds.*

B. *Even if $\delta_{t_0} = 0,\ \delta_{t_1} = 1$ or $\delta_{t_0} = 1,\ \delta_{t_1} = 0$ does not hold for all values of S, $\delta_{t_0} = 1 \oplus \delta_{t_1}$ always holds.*

Proof. To prove this, we will require the tool D-Grain(ϕ, r) proposed in [5, Section 2.1] that can be used to analyze all the 3 versions of Grain. Briefly recalling, D-Grain(ϕ, r) is an algorithm that performs simple truncated differential analysis of the Grain cipher. It takes two inputs: **(a)** the difference location $\phi \in [0, n-1]$ of the LFSR, and **(b)** the number of PRGA rounds r for which the analysis is to be performed. The algorithm initializes a differential engine Δ_ϕ-GRAIN, which consists of an n-integer LFSR and NFSR with the same taps as a given version of Grain, but with different update functions. Table 3 presents a comparison.

Table 3. The engine Δ_ϕ-GRAIN

	Grain cipher	Δ_ϕ-GRAIN
LFSR Update	$y_{t+n} = y_t \oplus y_{t+f_1} \oplus y_{t+f_2} \oplus \cdots \oplus y_{t+f_a}$	$u_{t+n} = u_t + u_{t+f_1} + u_{t+f_2} + \ldots + u_{t+f_a} \bmod 2$
NFSR Update	$x_{t+n} = y_t \oplus g(x_t, x_{t+g_1}, x_{t+g_2}, \ldots, x_{t+g_b})$	$v_{t+n} = u_t + 2 \cdot OR(v_t, v_{t+g_1}, \ldots, v_{t+g_b})$

Here $L_t = [u_t, u_{t+1}, \ldots, u_{t+n-1}]$ and $N_t = [v_t, v_{t+1}, \ldots, v_{t+n-1}]$ denote respectively the LFSR and NFSR states of Δ_ϕ-GRAIN at the PRGA round t and OR is a map from $\mathbb{Z}^{b+1} \to \{0, 1\}$ which roughly represents the logical 'or' operation and is defined as

$$OR(k_0, k_1, \ldots, k_b) = \begin{cases} 0, \text{ if } k_0 = k_1 = k_2 = \cdots = k_b = 0, \\ 1, \text{ otherwise.} \end{cases}$$

Let us define the equation for output key-stream bit in Grain as (this accommodates all the 3 versions of Grain)

$$z_t = \bigoplus_{k=1}^{c} x_{t+l_k} \oplus \bigoplus_{k=1}^{d} y_{t+i_k} \oplus h(y_{t+h_1}, y_{t+h_2}, \ldots, y_{t+h_e}, x_{t+j_1}, x_{t+j_2}, \ldots, x_{t+j_w})$$

Now we define the vectors (for $0 \le t < r$)

$$\chi_t = [v_{t+l_1}, \ldots, v_{t+l_c}, u_{t+i_1}, \ldots, u_{t+i_d}],\ \eta_t = [x_{t+l_1}, \ldots, x_{t+l_c}, y_{t+i_1}, \ldots, y_{t+i_d}]$$

$$\Upsilon_t = [u_{t+h_1}, \ldots, u_{t+h_e}, v_{t+j_1}, \ldots, v_{t+j_w}],\ \theta_t = [y_{t+h_1}, \ldots, y_{t+h_e}, x_{t+j_1}, \ldots, x_{t+j_w}].$$

The key-stream element Δz_t output from the engine Δ_ϕ-GRAIN is given as

$$\Delta z_t = \begin{cases} 0, \text{if } \Upsilon_t = \mathbf{0} \text{ AND } \chi_t \sqsubseteq 1 \text{ AND } |\chi_t| \text{ is even} \\ 1, \text{if } \Upsilon_t = \mathbf{0} \text{ AND } \chi_t \sqsubseteq 1 \text{ AND } |\chi_t| \text{ is odd} \\ 2, \text{ otherwise.} \end{cases}$$

$\mathbf{V} \sqsubseteq \alpha$, implies that all elements of $\mathbf{V}$ are less than or equal to α. The algorithm D-GRAIN(ϕ, r) initializes the LFSR and NFSR of Δ_ϕ-GRAIN to all 0′s except the ϕ^{th} LFSR element which is initialized to 1. It then runs the engine for r PRGA rounds. For each t, $(0 \leq t < r)$ it returns the 3-tuple $[\chi_t, \Upsilon_t, \Delta z_t]$.

- χ_t and Υ_t that contain elements from $\{0, 1, 2, 3\}$.
- Δz_t which is an integer from the set $\{0, 1, 2\}$.

Let us denote the symbols $S_t = X_t || Y_t$ and $S_t^\phi = X_t^\phi || Y_t^\phi$ the corresponding internal states at round t, which differed in the LFSR location ϕ at the beginning of the PRGA. Also $\eta_t^\phi, \theta_t^\phi$ are the t^{th} round vectors of S_t^ϕ that contribute to the output key-stream bit as a linear mask and input to the function h respectively. Then it has been proven in [5], that if the i^{th} element of χ_t (Υ_t) is

(1) 0, then the i^{th} bits of η_t and η_t^ϕ (θ_t and θ_t^ϕ) is equal for all values of S,
(2) 1, then the i^{th} bits of η_t and η_t^ϕ (θ_t and θ_t^ϕ) is unequal for all values of S,
(3) 2 or 3, then the difference between the i^{th} bits of η_t and η_t^ϕ (θ_t and θ_t^ϕ) is probabilistic.

Similarly, if Δz_t is 0 or 1, it implies that z_t and z_t^ϕ are respectively equal or unequal for all S. However if this output is 2 then the difference is probabilistic.

Consider the situation when for some particular value of ϕ the output in the t_0^{th} PRGA round of D-GRAIN(ϕ, r) i.e. $[\chi_{t_0}, \Upsilon_{t_0}, \Delta z_{t_0}]$ be such that **(i)** $\Upsilon_{t_0} = \mathbf{0}$ and **(ii)** χ_{t_0} has all but one element equal to 0, and this non-zero element is strictly greater than 0, i.e. $v_{t_0+l_w} > 1$ for some $w \leq c$ and all other $v_{t_0+l_k}, u_{t_0+i_k}$ equals 0. Then following **(1) - (3)**, for all values of S, we must have $\theta_{t_0} = \theta_{t_0}^\phi$ and η_{t_0} and $\eta_{t_0}^\phi$ have all but their w^{th} element deterministically equal. Let us call the difference of the w^{th} elements of these vectors equal to δ. If $\mathcal{P}(\cdot)$ denotes the GF(2) sum of the elements of a vector, then we have

$$\delta_{t_0} = z_{t_0} \oplus z_{t_0}^\phi = \mathcal{P}(\eta_{t_0}) \oplus h(\theta_{t_0}) \oplus \mathcal{P}(\eta_{t_0}^\phi) \oplus h(\theta_{t_0}^\phi) = \delta$$

Consider the output of D-GRAIN(ϕ, r) at the PRGA round $t_1 = t_0 - l_e + l_w$ for some $l_e < l_w$. Note that due to the evolution of the LFSR of Δ_ϕ-GRAIN the difference of the e^{th} element of χ_{t_1} must be equal to the w^{th} element of χ_{t_0}. Now if **(iii)** all the remaining elements of χ_{t_1} and the entire of Υ_{t_1} are all 0′s then following the previous argument we have

$$\delta_{t_1} = z_{t_1} \oplus z_{t_1}^\phi = \mathcal{P}(\eta_{t_1}) \oplus h(\theta_{t_1}) \oplus \mathcal{P}(\eta_{t_1}^\phi) \oplus h(\theta_{t_1}^\phi) = \delta$$

Thus at PRGA rounds t_0, t_1 we have $\delta_{t_0} = \delta_{t_1} = \delta$, even though δ itself is not deterministic. Now all that remains to be shown is that for every version of Grain, there exist values of ϕ, t_0, t_1 for which the conditions **(i),(ii),(iii)** hold. This can be shown by construction i.e. for each version of Grain, we need to construct the engine Δ_ϕ-GRAIN for all values of ϕ and check the outputs of this engine for sufficiently large values of r. Experimental results have shown that for all the three versions of Grain, taking $r = 2n$, there exist such pairs t_0, t_1 for almost all values of ϕ.

Part **B** of the proof can be arrived at by following similar arguments. Consider some other PRGA round $t_1 = t_0 - l_{e'} + l_w$ for some $l_{e'} < l_w$. Then the difference of the e'-th element of

χ_{t_1} must be equal to the w^{th} element of χ_{t_0}. Now if **(iv)** $\Upsilon_{t_1} = \mathbf{0}$ and **(v)** there exists some w' such that $\chi_{t_1}[w'] = 1$ and all the remaining elements of of χ_{t_1} is 0, this implies that **(vi)** all elements of θ_{t_1} and $\theta^{\phi}_{t_1}$ are deterministically equal and **(vii)** the w'-th elements of η_{t_1} and $\eta^{\phi}_{t_1}$ are deterministically unequal, **(viii)** the difference δ between the e'-th element of η_{t_1} and $\eta^{\phi}_{t_1}$ is probabilistic and **(ix)** all other elements of η_{t_1} and $\eta^{\phi}_{t_1}$ are deterministically equal. Then,

$$\delta_{t_1} = z_{t_1} \oplus z^{\phi}_{t_1} = \mathcal{P}(\eta_{t_1}) \oplus h(\theta_{t_1}) \oplus \mathcal{P}(\eta^{\phi}_{t_1}) \oplus h(\theta^{\phi}_{t_1}) = 1 \oplus \delta$$

Thus at PRGA rounds t_0, t_1 we have $\delta_{t_0} = 1 \oplus \delta_{t_1} = \delta$ even though δ is non-deterministic. As above, the existence of ϕ for which there exist PRGA rounds t_0, t_1 that satisfy **(i),(ii),(iv),(v)** for all the three versions of Grain, can be shown by construction. Also, the above analysis may be similarly extended for the case when the difference at the beginning of the PRGA exists in some NFSR location. □

Example 2. Let $S,\ S^0$ be two internal states in Grain v1, that differ only in the LFSR location 0 at the beginning of the PRGA. Then although $z_{41} = z^0_{41},\ z_{66} = z^0_{66}$ or $z_{41} = z^0_{41} \oplus 1,\ z_{66} = z^0_{66} \oplus 1$ does not hold for all values of S, $z_{41} \oplus z^0_{41} = z_{66} \oplus z^0_{66}$ always holds.

Example 3. Let $S,\ S^{38}$ be two internal states in Grain v1, that differ only in the LFSR location 38 at the beginning of the PRGA. Then although $z_{79} = z^{38}_{79},\ z_{104} = z^{38}_{104} \oplus 1$ or $z_{79} = z^{38}_{79} \oplus 1,\ z_{104} = z^{38}_{104} \oplus 1$ does not hold for all values of S, $z_{79} \oplus z^{38}_{79} = z_{104} \oplus z^{38}_{104} \oplus 1$ always holds.

Now by performing a differential trail analysis using D-Grain(ϕ, r) for all the register locations ϕ in the LFSR and the NFSR we will obtain a set of PRGA rounds for most of the register locations at which the difference between the faulty and faultless key-stream bits are thus related. This fact can be further utilized to improve the success probability of the identification algorithm. For example, suppose the given identification algorithm using the first two signature vectors, narrows down the set $\Psi_{1,\phi}$ to $\{0, 25\}$. Now suppose, we observe that $z_{41} \oplus z^{\phi}_{41} = 1 \oplus z_{66} \oplus z^{\phi}_{66}$. We can immediately conclude that $\phi \neq 0$, and hence $\phi = 25$ must be the actual fault location.

We will now formalize the above ideas by first defining the **Third and Fourth signature vectors** $\mathcal{Q}^3_{\phi}, \mathcal{Q}^4_{\phi} \in \{0,1\}^{2n}$ that had not been considered in [5]. Index the n NFSR locations as $0, 1, \ldots, n-1$ and the n LFSR locations as $n, n+1, \ldots, 2n-1$. Then for every register location $\phi \in [0, 2n-1]$ define the set of tuples

$$\mathcal{C}^{\phi}_3 = \{(i,j) : i \neq j \text{ and } z_i \oplus z^{\phi}_i = z_j \oplus z^{\phi}_j = \delta,\ \forall\ S,\ \text{but } 0 < P(\delta = 0) < 1\}$$

$$\text{and } \mathcal{C}^{\phi}_4 = \{(i,j) : i \neq j \text{ and } z_i \oplus z^{\phi}_i = z_j \oplus z^{\phi}_j \oplus 1 = \delta,\ \forall\ S,\ \text{but } 0 < P(\delta = 0) < 1\}$$

Now we define $\mathcal{Q}^3_{\phi}, \mathcal{Q}^4_{\phi}$ as

$$\mathcal{Q}^3_{\phi}(i) = \mathcal{Q}^3_{\phi}(j) = \begin{cases} \max(i,j), & \text{if } (i,j) \in \mathcal{C}^{\phi}_3 \\ 0, & \text{otherwise.} \end{cases} \quad \mathcal{Q}^4_{\phi}(i) = \mathcal{Q}^4_{\phi}(j) = \begin{cases} \max(i,j), & \text{if } (i,j) \in \mathcal{C}^{\phi}_4 \\ 0, & \text{otherwise.} \end{cases}$$

The max() function has been chosen to ensure that for two distinct pairs $(i_0, j_0) \neq (i_1, j_1) \in C^t_{\phi}$, we have $\mathcal{Q}^t_{\phi}(i_0) \neq \mathcal{Q}^t_{\phi}(i_1)$. Now for ϕ to be the correct fault location for some differential vector E^{ϕ} we must have $E^{\phi}(i) = E^{\phi}(j)$ whenever $\mathcal{Q}^3_{\phi}(i) = \mathcal{Q}^3_{\phi}(j) \neq 0$ and $E^{\phi}(i) = 1 \oplus E^{\phi}(j)$ whenever $\mathcal{Q}^4_{\phi}(i) = \mathcal{Q}^4_{\phi}(j) \neq 0$. Let us denote these conditions by the notations $E^{\phi} \lhd \mathcal{Q}^3_{\phi}$ and $E^{\phi} \lhd \mathcal{Q}^4_{\phi}$.

So let us formally define our fault location identification algorithm.

e. Let $\Psi_{1,\phi}$ be the set of candidate fault locations obtained after performing step [**d**] in the previous subsection.

f. Formulate the set $\Psi_{2,\phi} = \{\phi : \phi \in \Psi_{1,\phi} \text{ and } E^{\phi} \lhd \mathcal{Q}_{\phi}^{3}\}$. If $|\Psi_{2,\phi}| = 1$, the output the only element in $\Psi_{2,\phi}$.

g. Else, formulate the set $\Psi_{3,\phi} = \{\phi : \phi \in \Psi_{2,\phi} \text{ and } E^{\phi} \lhd \mathcal{Q}_{\phi}^{4}\}$. If $|\Psi_{3,\phi}| = 1$, the output the only element in $\Psi_{3,\phi}$. If $|\Psi_{3,\phi}| > 1$, then our strategy fails.

We again performed computer experiments by simulating random single bit faults for 2^{20} randomly chosen Key-IVs. The probability that the new algorithm identifies the correct fault location in the LFSR or the NFSR i.e. $Prob(|\Psi_{3,\phi}| = 1)$ is around 1.00 for Grain v1, 1.00 for Grain-128 and 0.81 for Grain-128a.

Implication of the success probabilities In Table 4, the number of faults required to determine the internal states of Grain v1, Grain-128 and Grain-128a are given. As can be seen, the attack may be carried out in very little time by employing around $10, 4, 10$ faults for Grain v1, Grain-128 and Grain-128a family. While the probabilities of success of fault location identification are very high (close to 1) for both Grain v1 and Grain-128, it is around 0.81 for Grain-128a. Since the success probabilities in Grain v1 and Grain-128 are very high, it is expected that for any set of 10 (for Grain v1) and 4 (for Grain-128) randomly applied faults in the internal state, the algorithm will succeed in finding the fault location of all the faults with very high probability and hence help complete the attack. But this is not the case for Grain-128a. For Grain-128a, the location identification algorithm is expected to succeed with 0.81 and so if the adversary wants to complete the attack, she has to apply around $10 \cdot \frac{1}{0.81} \approx 12.3$ faults to succeed.

4.4 Identifying Multiple bit faults

In [5], a preliminary study was made of the situation when a single fault injection affects the value of upto three consecutive locations in the LFSR. It gave rise to $4n-5$ possible cases of faults out of which n were due to single bit faults and the other $3n-5$ due to double or triple bit faults. The same fault identification routine is used to determine the fault location of faulty streams arising due to double or triple bit faults. In [5], it was shown that if the faults are restricted to the LFSR then the location identification will be able with a very high probability (close to 1 for Grain v1, Grain-128 and Grain-128a) identify that a faulty stream produced due to a double or triple bit fault could not have been produced due to a single bit fault (this happens when $\Psi_{3,\phi} = \emptyset$) and in all such cases the algorithm outputs a null message. In our experiments we have explored this situation with respect to faults in both the LFSR and the NFSR. After experimenting with randomly chosen single, double and triple bit faults for around 2^{20} Key-IV pairs, it was found that the probability that the algorithm successfully rejects a faulty stream produced due to a multiple bit fault i.e. $Pr(\Psi_{1,\phi} = \emptyset)$ is 0.94 for Grain v1, 0.99 Grain-128 and 0.86 for Grain-128a.

5 Identifying Fault Locations for Injections at random time

Thus far we have assumed that the adversary is able to inject all faults at the beginning of a fixed PRGA round. This is usually practical as fault injections are usually synchronized with the power consumption curves of the device implementing the cryptosystem [12]. In this section we show that it is possible to attack Grain even this requirement is relaxed. We will show that if the adversary injects a fault at a PRGA round τ where $\tau \in [0, \tau_{max} - 1]$. In such an event, it is

possible for the adversary, with high probability, determine the values of the fault location ϕ and the injection time τ. Before we get into further details, let us recap a few things and look at a definition that we will be using extensively.

The location identification algorithm that presented so far (call it $\mathsf{FLI}(E^\phi)$ takes the difference vector $E^\phi = Z \oplus Z^\phi$, performs the seven steps **(a) -(g)** and returns the following

- The fault location ϕ if the set $\Psi_{3,\phi}$ has cardinality 1.
- The $\emptyset$ message if the set $\Psi_{3,\phi}$ has cardinality 0, which is indicative of the fact that Z^ϕ was generated due to multiple bit fault.
- A failure message if the set $\Psi_{3,\phi}$ has cardinality strictly greater than 1. This case may may arise for both single and multiple bit faults.

Definition 1. *Two distinct fault location and time injection pairs (ϕ, τ) and (ϕ', τ') are said to be* equivalent *if they produce the same faulty key-stream.*

For example in Grain v1, faulting the NFSR location 70 at PRGA round 0 would produce the same faulty key-stream as faulting NFSR location 69 at PRGA round 1. This is because the difference that is induced in location 70 at PRGA round 0 shifts to location 69 in PRGA round 1 anyway. Thus $(70, 0)$ and $(69, 1)$ are equivalent pairs. However $(62, 0)$ and $(61, 1)$ are not equivalent since 62 is a tap for the update function of the NFSR for Grain v1. A difference induced in PRGA round 0 in location 62 travels to both locations 61 and 79 in the next round. Whereas a fault at location 61 in round 1 would affect only this location and not location 79.

Let us denote the elements of $E^\phi = [e_0, e_1, e_2, \ldots]$. Also define the vector $E_i^\phi = [e_i, e_{i+1}, \ldots]$. Let us assume that the vector E^ϕ has been produced due to fault injection at some LFSR or NFSR location ϕ at time τ where $0 \leq \tau \leq \tau_{max} - 1$. To identify (ϕ, τ) the adversary runs the routine $\mathsf{FLI}(E_i^\phi)$ for all $i \in [0, \tau_{max} - 1]$. As a result, he adversary could obtain

1. The output $S + i$ for all values of i. Note that since the pairs $(S + i, i)$ are equivalent, he can assume that $(S, 0)$ are the true values of (ϕ, τ).
2. The output $S + i$ for some values of i and a failure messages for some other values of i. The adversary then takes the minimum value of $i = i_{min}$ for which $\mathsf{FLI}(E_i^\phi)$ succeeds and assumes $(S + i_{min}, i_{min})$ to be the true values of (ϕ, τ).
3. The failure message for all values of i. In this event he rejects the key-stream. However, the probability of this outcome is quite low.
4. If he obtains $\emptyset$ for some value of i he deduces multiple-bit injection and rejects the key-stream.
5. If he obtains the outputs S_1 for $i = i_1$ and S_2 for $i = i_2$ such that (S_1, i_1) and (S_2, i_2) are not equivalent then he deduces that the algorithm has failed.

Experiments performed for around 2^{20} random Key-IVs the probability of Case 5 occurring is only about 0.089 for Grain v1 if we take $\tau_{max} = 10$. For for Grain-128, taking $\tau_{max} = 15$, the failure probability comes to 0.079. For higher values of τ_{max} the failure probability becomes non-negligible.

This approach, however, fails for Grain-128a. Recall, that every alternate key-stream bit in Grain-128a is used for the computation of MAC and is therefore not directly available to the attacker. It is easy to see that the given approach will fail in all cases when the injection time is an odd number.

6 Experimental Results

In this section we present the experimental results in detail. After the fault location and injection time of a particalar faulty key-stream vector have been identified using the signature vectors, a system of equations are formulated using the steps outlined in Section 3, and the equations are then fed into a SAT solver. There are several issues to be considered.

- The number of faults is the most significant figure that we minimize using the SAT solvers. This implies that we also reduce the number of rekeyings of the cipher.
- We deduce the fault location and the injection time using the idea of the four signature vectors. Note that, in [5], faults have been introduced only in LFSR, but here we can handle the situation when faults may be introduced either the LFSR, NFSR or both of them.
- The number of faulty key-stream bits required to solve the system is also important. In our experiments, We have used $2n$ key-stream bits corresponding to each fault, i.e., $2 \cdot 80 = 160$ for Grain v1 and $2 \cdot 128 = 256$ for Grain-128 and Grain-128a. In fact, for Grain-128a, we use even fewer key-stream bits as we obtain more equations per key-stream bit.

We have solved the equations using SAT solver Cryptominisat-2.9.5 [30] installed with SAGE 5.7 on Linux Ubuntu 2.6. The hardware platform is an HP Z800 workstation with 3GHz Intel(R) Xeon(R) CPU. We have considered three different cases: (i) the faults are introduced in LFSR only, (ii) the faults are introduced in NFSR only, and (iii) the faults are introduced in both LFSR and NFSR (here we consider that expected half of the faults are injected in LFSR and the other half in NFSR). The results have been presented in Table 4. We have presented the time required for the SAT solver part only as the time for identifying the location of the fault using signature vectors is negligible. For each row, we consider a set of ten (10) experiments. As it is not easy to count the exact number of computational steps required in the SAT solver, we have reported the amount of time required in seconds.

One may note that our method requires far fewer faults that what earlier known for Grain family (of the order of hundreds) in literature so far [4, 5, 8, 24]. Several issues may be optimized in the experiments. We note that with very little amount of key-stream, the attack takes longer time. It is also clear that the number of faults may be reduced further with more computational effort.

7 Conclusion

The Differential Fault Analysis (DFA) against the Grain family of stream ciphers has been a fairly well researched topic [4, 5, 8, 24] and has been studied under various fault models some more restrictive and some more relaxed. In this work, we propose a DFA of the Grain family that requires the adversary to have the least control over fault injections, i.e., same as that of [5] but requires far fewer faults than that required in [5]. Furthermore the adversary need not restrict the fault injections to either the LFSR or the NFSR, a stipulation that has been imposed in all the previous fault attacks on the Grain family. For Grain v1 and Grain-128, the adversary need not even exercise precise control over timing of fault injection. The algorithm we propose first finds the location and injection time of a randomly applied bit fault (it rejects the faulty stream if it infers that it was produced due to multiple bit fault) and then populates a bank of equations in the internal state variables of the cipher at the start of the PRGA. The algorithm then tries to solve the equations using the Cryptominisat-2.9.5 SAT solver [30]. For all the three ciphers

Table 4. Experimental Results

Faults in LFSR only					
Cipher	Number of faults	Amount of key-stream	Time (in sec.)		
			Minimum	Maximum	Average
Grain v1	10	160	16.48	49.23	27.40
	9	160	22.10	32.71	40.50
	8	160	18.62	92.34	48.40
Grain-128	5	256	5.21	9.43	7.10
	4	256	9.03	96.68	34.40
	3	256	24.52	361.53	163.70
Grain-128a	11	175	14.47	37.85	23.60
	10	175	26.82	253.15	52.74
Faults in NFSR only					
Cipher	Number of faults	Amount of key-stream	Time (in sec.)		
			Minimum	Maximum	Average
Grain v1	11	160	27.93	105.44	55.35
	10	160	21.14	89.50	43.64
	9	160	29.64	123.98	56.35
Grain-128	6	256	16.64	196.32	93.45
	5	256	22.87	380.01	147.70
Grain-128a	11	175	179.62	8453.14	1542.27
	10	175	175.07	8387.21	1495.54
Faults in both LFSR and NFSR					
Cipher	Number of faults	Amount of key-stream	Time (in sec.)		
			Minimum	Maximum	Average
Grain v1	11	160	54.96	1420.71	220.90
	10	160	19.17	452.30	352.20
Grain-128	6	256	6.48	14.32	10.41
	5	256	12.18	37.56	22.15
	4	256	27.63	4876.53	581.80
Grain-128a	11	175	46.45	259.34	101.10
	10	175	69.63	5144.56	1472.35

the solver is able to recover the entire internal state using equations generated by less than or equal to 10 random faults in a few minutes. This is, to the best of our knowledge, the best fault analysis that has been reported against the Grain family.

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. 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].
3. J. P. Aumasson, I. Dinur, L. Henzen, W. Meier, and A. Shamir. Efficient FPGA Implementations of High-Dimensional Cube Testers on the Stream Cipher Grain-128. In SHARCS - Special-purpose Hardware for Attacking Cryptographic Systems. 2009.
4. 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.
5. S. Banik, S. Maitra and S. Sarkar. A Differential Fault Attack on the Grain Family under Reasonable Assumptions. In INDOCRYPT 2012, LNCS Vol. 7668, pp. 191–208.
6. G. V. Bard. Algebraic Cryptanalysis. Springer-Verlag, London, New York, 2009.
7. C. Berbain, H. Gilbert and A. Maximov. Cryptanalysis of Grain. In FSE 2006, LNCS, Vol. 4047, pp. 15–29, 2006.
8. A. Berzati, C. Canovas, G. Castagnos, B. Debraize, L. Goubin, A. Gouget, P. Paillier, S. Salgado. Fault Analysis of Grain-128. In: IEEE International Workshop on Hardware-Oriented Security and Trust 2009, pp. 7–14.
9. T. E. Bjørstad. Cryptanalysis of Grain using Time/Memory/Data tradeoffs (v1.0 / 2008-02-25). Available at `http://www.ecrypt.eu.org/stream`.
10. E. Biham, A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In CRYPTO 1997, LNCS Vol. 1294, pp. 513–525.
11. D. Boneh, R. A. DeMillo, R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In EUROCRYPT 1997, LNCS Vol. 1233, pp. 37–51.

12. B. Debraize, I. M. Corbella. Fault Analysis of the Stream Cipher Snow 3G. In FDTC 2009, pp. 103–110.
13. C. De Cannière, O. Küçük and B. Preneel. Analysis of Grain's Initialization Algorithm. In AFRICACRYPT 2008, LNCS, Vol. 5023, pp. 276–289, 2008.
14. I. Dinur, T. Güneysu, C. Paar, A. Shamir, R. Zimmermann. An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware. In ASIACRYPT 2011, LNCS Vol. 7073, pp. 327–343, 2011.
15. I. Dinur, A. Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In FSE 2011, LNCS, Vol. 6733, pp. 167–187, 2011.
16. H. Englund, T. Johansson, and M. Sönmez Turan. A Framework for Chosen IV Statistical Analysis of Stream Ciphers. In INDOCRYPT 2007, LNCS, Vol. 4859, pp. 268–281, 2007.
17. The ECRYPT Stream Cipher Project. eSTREAM Portfolio of Stream Ciphers. Revised on September 8, 2008.
18. S. Fischer, S. Khazaei, and W. Meier. Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers. In AFRICACRYPT 2008, LNCS, Vol. 5023, pp. 236–245, 2008.
19. 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`.
20. M. Hell, T. Johansson, A.Maximov and W. Meier. A Stream Cipher Proposal: Grain-128. In IEEE International Symposium on Information Theory (ISIT 2006).
21. J. J. Hoch, A. Shamir. Fault Analysis of Stream Ciphers. In CHES 2004, LNCS, Vol. 3156, pp. 1–20.
22. M. Hojsík, B. Rudolf. Differential Fault Analysis of Trivium, In FSE 2008, LNCS, Vol. 5086, pp. 158–172.
23. M. Hojsík, B. Rudolf. Floating Fault Analysis of Trivium, In INDOCRYPT 2008, LNCS, Vol. 5365, pp. 239–250.
24. S. Karmakar and D. Roy Chowdhury. Fault analysis of Grain-128 by targeting NFSR. In AFRICACRYPT 2011, LNCS, Vol. 6737, pp. 298–315.
25. S. Khazaei, M. Hassanzadeh and M. Kiaei. Distinguishing Attack on Grain. ECRYPT Stream Cipher Project Report 2005/071, 2005. Available at `http://www.ecrypt.eu.org/stream`
26. S. Knellwolf, W. Meier and M. Naya-Plasencia. Conditional Differential Cryptanalysis of NLFSR-based Cryptosystems. In ASIACRYPT 2010, LNCS, Vol. 6477, pp. 130–145, 2010.
27. M. S. E. Mohamed , S. Bulygin and J. Buchmann. Improved Differential Fault Analysis of Trivium. In COSADE 2011, Darmstadt, Germany, February 24–25, 2011.
28. S. P. Skorobogatov. Optically Enhanced Position-Locked Power Analysis. In CHES 2006, LNCS, Vol. 4249, pp. 61–75.
29. S. P. Skorobogatov and R. J. Anderson. Optical Fault Induction Attacks. In CHES 2002, LNCS, Vol. 2523, pp. 2–12.
30. M. Soos. CryptoMiniSat-2.9.5. `http://www.msoos.org/cryptominisat2/`.
31. P. Stankovski. Greedy Distinguishers and Nonrandomness Detectors. In INDOCRYPT 2010, LNCS, Vol. 6498, pp. 210–226, 2010.
32. 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).
33. B. Zhang and Z. Li. Near Collision Attack on the Grain v1 Stream Cipher. To appear in FSE 2013.
34. H. Zhang and X. Wang. Cryptanalysis of Stream Cipher Grain Family. IACR Cryptology ePrint Archive 2009: 109. Available at `http://eprint.iacr.org/2009/109`.