A Differential Fault Attack on the Grain Family under Reasonable Assumptions

Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar

Applied Statistics Unit, Indian Statistical Institute,
203 B T Road, Kolkata 700 108, India
{s.banik_r,subho}@isical.ac.in, sarkar.santanu.bir@gmail.com

Abstract. In this paper we study a differential fault attack against ciphers having the same physical structure as in the Grain family. In particular we demonstrate our attack against Grain v1, Grain-128 and Grain-128a. The existing attacks by Berzati et al. (HOST 2009), Karmakar et al. (Africacrypt 2011) and Banik et al. (CHES 2012) assume a fault model that allows them to reproduce a fault at a particular register location more than once. However, we assume a realistic fault model in which the above assumption is no longer necessary, i.e., re-injecting the fault in the same location more than once is not required. In addition, towards a more practical framework, we also consider the situation in which more than one consecutive locations of the LFSR are flipped as result of a single fault injection.

Keywords: Differential fault attacks, Grain v1, Grain-128, Grain-128a, LFSR, NFSR, Stream Cipher.

1 Introduction

Fault attacks have received serious attention in cryptographic literature for more than a decade [1,2]. Such attacks on stream ciphers have gained momentum ever since the work of Hoch and Shamir [10] and this model of cryptanalysis, though optimistic, could successfully be employed against a number of proposals. Fault attacks study the mathematical robustness of a cryptosystem in a setting that is weaker than its original or expected mode of operation. A typical attack scenario [10] consists of an adversary who injects a random fault (using laser shots/clock glitches [14, 15]) in a cryptographic device as a result of which one or more bits of its internal state are altered. The faulty output from this altered device is then used to deduce information about its internal state/secret key. In order to perform the attack, 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 [4, 8, 9] has received a lot of attention and it is in the hardware profile of eStream [3]. In all the fault attacks reported so far [5, 6, 12] on this cipher, the adversary is granted far too many privileges

S. Galbraith and M. Nandi (Eds.): INDOCRYPT 2012, LNCS 7668, pp. 191–208, 2012.

to make the attacks practical. In fact the designers of Grain themselves have underlined a set of reasonable privileges that may be granted to the adversary while performing the fault attack. Unfortunately, all the existing fault attacks on the Grain family exploited additional assumptions. In this regard, let us refer to the following quote from the designers of Grain [8, Section 6.5] (see also similar comments in [9, Section IV D] and [4, Section 4.4]):

"If an attacker is able to reset the device and to induce a single bit fault many times and at different positions that he can correctly guess from the output difference, we cannot preclude that he will get information about a subset of the state bits in the LFSR. Such an attack seems more difficult under the (more realistic) assumption that the fault induced affects several state bits at (partially) unknown positions, since in this case it is more difficult to determine the induced difference from output differences."

	[6]	[12]	[5]	This work
Multiple fault at same location	Required	Required	Required	Not Required
Multiple IV Initialization	No	Yes	No	No
Single bit Fault	Yes	Yes	Yes	Upto 3-bit toggle allowed
Control over Fault Timing	Required	Required	Required	Required
Multiple Re-Keying	Allowed	Allowed	Allowed	Allowed

Fig. 1. Differential Fault Attack on Grain: Survey of Fault Models

In the published fault attacks [5, 6, 12] on Grain, it has been assumed that attacker has the ability to inject a single bit fault in the same register location over and over again. This is clearly rather optimistic and does not follow the fault model prescribed by the designers. In our work, we have assumed that the adversary has only those privileges that have been allowed by the designers, i.e., we follow the exact fault model provided by the designers and demonstrate that in such a scenario too, the adversary can not only recover "a subset of the LFSR state bits" but also recover the secret key. Furthermore, we consider a situation in which the adversary is unable to induce a single bit toggle every time he injects a fault. The best he can do is to 'influence' upto k bits in random but consecutive LFSR locations without knowing the exact number of bits the injected fault has altered or their locations. We show that for certain small values of k, even under this added constraint the secret key can be recovered. The idea used here is that, with very high confidence, the adversary should be able to identify the situations when the injected fault alters the binary value in only a single register location. He can then use the algorithms described for a single location identification and proceed with the attack.

In this work we assume that the adversary has the following privileges: **(a)** he can reset the cipher with the original Key-IV and restart cipher operations as many times he wishes; this is not a problem if the device ouputs different faulty ciphertexts of the same or known messages and such a model has been used in [5, 6, 11, 12] (actually the attack model requires the original and several faulty key-streams), **(b)** he has full control over the timing of fault injection, and **(c)** he can inject a fault that may affect upto k consecutive LFSR locations but he is unaware of the exact number of bits altered or their locations. In this work we have concentrated on the cases till $k = 3$. As pointed out earlier, these assumptions about the fault model are far more realistic and practical than the ones assumed in [5, 6, 12].

Organization of this paper. In this section, we continue with a detailed description of the Grain family. In Section 2, we introduce certain tools and definitions that will help us launch the attack. To demonstrate the attack, initially we assume that the attacker is able to induce a single bit toggle at any random LFSR location. The fault location identification routine is explained in Section 3. The general strategy to attack a cipher with the physical structure of Grain is outlined in Section 4. Section 5 demonstrates the attacks on Grain v1, Grain-128 and Grain-128a. In Section 6, we explore a stricter fault model in which the attacker is able to flip the binary values of upto 3 consecutive LFSR locations. Section 7 concludes the paper.

1.1 Brief Description of Grain Family

Any cipher in the Grain family consists of an n-bit LFSR and an n-bit NFSR (see Figure 2). The update function of the LFSR is given by the equation $y_{t+n} = f(Y_t) = y_t \oplus y_{t+f_1} \oplus y_{t+f_2} \oplus \cdots \oplus y_{t+f_a}$, 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 $x_{t+n} = y_t \oplus g(X_t) = y_t \oplus g(x_t, x_{t+\tau_1}, x_{t+\tau_2}, \ldots, x_{t+\tau_b})$. Here, $X_t = [x_t, x_{t+1}, \ldots, x_{t+n-1}]$ is an n-bit

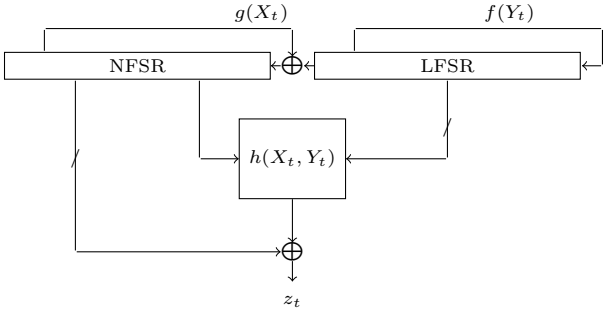

Fig. 2. Structure of Stream Cipher in Grain Family

Table 1. Grain at a glance

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

vector that denotes the NFSR state at the t^{th} clock interval and g is a non-linear function of the NFSR state bits. The output key-stream is produced by combining the LFSR and NFSR bits as $z_t = x_{t+l_1} \oplus x_{t+l_2} \oplus \cdots \oplus x_{t+l_c} \oplus y_{t+i_1} \oplus y_{t+i_2} \oplus \cdots \oplus y_{t+i_d} \oplus h(y_{t+h_1}, y_{t+h_2}, \ldots, y_{t+h_e}, x_{t+j_1}, x_{t+j_2}, \ldots, x_{t+j_w})$. Here h is a non-linear Boolean function.

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 $y_{t+n} = f(Y_t), x_{t+n} = y_t \oplus g(X_t)$.

2 Tools and Definitions

2.1 Differential Grain

Let $S_0 = [X_0||Y_0] \in \{0,1\}^{2n}$ be the initial state of the Grain family PRGA and S_{0,Δ_ϕ} be the initial state which differs from S_0 in an LFSR location $\phi \in [0, n-1]$. The task is to ascertain how the corresponding internal states in the t^{th} round

S_t and S_{t,Δ_ϕ} will differ from each other, for some integer $t > 0$. One such tool appeared in [6], but our approach is improved and more involved. We present the following algorithm which we will call D-Grain that takes as input the difference location $\phi \in [0, n-1]$ and the round r, and returns (i) a set of r integer arrays χ_t, for $0 \leq t < r$, each of length $c+d$, (ii) a set of r integer arrays Υ_t, for $0 \leq t < r$, each of length $e+w$ and (iii) an integer array ΔZ of length r. Note that as defined in Section 1.1, c, d are the number of NFSR, LFSR bits (respectively) which are linearly added to the output function h. Further, w, e are the number of NFSR, LFSR bits (respectively) that are input to the function h.

Now consider the corresponding generalized differential engine Δ_ϕ-Grain with an n-cell LFSR ΔL and an n-cell NFSR ΔN. All the elements of ΔL and ΔN are integers. We denote the t^{th} round state of ΔL as $\Delta L_t = [u_t, u_{t+1}, \ldots, u_{t+n-1}]$ and that of ΔN as $\Delta N_t = [v_t, v_{t+1}, \ldots, v_{t+n-1}]$. Initially all the elements of $\Delta N, \Delta L$ are set to 0, with the only exception that the cell numbered $\phi \in [0, n-1]$ of ΔL is set to 1. The initial states $\Delta N_0, \Delta L_0$ are indicative of the difference between S_0 and S_{0,Δ_ϕ} and we will show that the t^{th} states $\Delta N_t, \Delta L_t$ are indicative of the difference between S_t and S_{t,Δ_ϕ}. ΔL updates itself as $u_{t+n} = u_t + u_{t+f_1} + u_{t+f_2} + \cdots + u_{t+f_a} \bmod 2$ and ΔN updates itself as $v_{t+n} = u_t + 2 \cdot OR(v_t, v_{t+\tau_1}, v_{t+\tau_2}, \ldots, v_{t+\tau_b})$. The rationale behind the update functions will be explained later. Here 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 $\chi_t = [v_{t+l_1}, v_{t+l_2}, \ldots, v_{t+l_c}, u_{t+i_1}, u_{t+i_2}, \ldots, u_{t+i_d}]$ and $\Upsilon_t = [u_{t+h_1}, u_{t+h_2}, \ldots, u_{t+h_e}, v_{t+j_1}, v_{t+j_2}, \ldots, v_{t+j_w}]$. Note that χ_t (respectively Υ_t) is the set of cells in Δ_ϕ-Grain which corresponds to the bits that are linearly added to the output function h (respectively, input to h) in the t^{th} PRGA stage of the actual cipher.

If $\mathbf{V}$ is a vector having non-negative integral elements, then $\mathbf{V} \sqsubseteq \beta$, (for some positive integer β), implies that all elements of $\mathbf{V}$ are less than or equal to β. The t^{th} key-stream element Δz_t produced by this engine is given as

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

Here $\mathbf{0}$ denotes the all zero vector, and $|\cdot|$ denotes the number of non-zero elements in a vector. Initially $\Delta N_0, \Delta L_0$ represent the difference of S_0 and S_{0,Δ_ϕ}. As the PRGA evolves, the only non-zero element (having value 1) of ΔL propagates and so does the difference between S_t and S_{t,Δ_ϕ}. Since the LFSR in Grain is updated by a linear function, whenever the difference between S_t and S_{t,Δ_ϕ} is fed back via the update function, a 1 is fed back in ΔL. Now when the difference between S_t and S_{t,Δ_ϕ} propagates to some NFSR tap location τ_i (for some value of t), then this difference may or may not be fed back, depending on the nature of the Boolean function g and the current state S_t. Hence in such a case the

propagation of the differential is probabilistic. Note that in all such situations, either the integer 2 or 3 is fed back in ΔN as is apparent from the update equation for v_{t+n}. Therefore if

1. some cell in ΔL_t or ΔN_t is 0, then the corresponding bits are equal in S_t and S_{t,Δ_ϕ} with probability 1;
2. some cell in ΔL_t or ΔN_t is 1, then the corresponding bits are different in S_t and S_{t,Δ_ϕ} with probability 1;
3. some cell in ΔL_t or ΔN_t is 2 or 3, then that the corresponding bits are different in S_t and S_{t,Δ_ϕ} with some probability $0 < p_d < 1$.

Also, note that if Υ_t is $\mathbf{0}$, then all the bits of S_t and S_{t,Δ_ϕ} that provide inputs to the non-linear function h are the same (for all choices of S_0). If all elements of χ_t are less than or equal to 1, then each one of the elements of S_t and S_{t,Δ_ϕ} which linearly adds on to the output function h to produce the output key-stream bit is either equal or different with probability 1. When both these events occur, the key-stream bits produced by S_t and S_{t,Δ_ϕ} are definitely the same if $|\chi_t|$ is an even number, as an even number of differences cancel out in GF(2). When this happens, Δ_ϕ-Grain outputs $\Delta z_t = 0$. If $|\chi_t|$ is an odd number, then the key-stream bits produced by S_t and S_{t,Δ_ϕ} are different with probability 1. In this case $\Delta z_t = 1$. In all other cases, the difference of the key-stream bits produced by S_t and S_{t,Δ_ϕ} is equal to 0 or 1 with some probability, and then $\Delta z_t = 2$. We describe the routine D-Grain(ϕ, r) in Algorithm 1 which returns the arrays χ_t, Υ_t for $0 \leq t < r$ and $\Delta Z = [\Delta z_0, \ldots, \Delta z_{r-1}]$.

Input: ϕ: An LFSR location $\in [0, n-1]$, an integer $r(> 0)$;
Output: An integer array ΔZ of r elements;
Output: Two integer arrays χ_t, Υ_t for $0 \leq t < r$;

$[u_0, u_1, \ldots, u_{n-1}] \leftarrow \mathbf{0}, [v_0, v_1, \ldots, v_{n-1}] \leftarrow \mathbf{0}, u_\phi \leftarrow 1, t \leftarrow 0$;
while $t < r$ **do**
  $\Upsilon_t \leftarrow [u_{h_1}, u_{h_2}, \ldots, u_{h_e}, v_{j_1}, v_{j_2}, \ldots, v_{j_w}]$;
  $\chi_t \leftarrow [v_{l_1}, v_{l_2}, \ldots, v_{l_c}, u_{i_1}, u_{i_2}, \ldots, u_{i_d}]$;
  if $\Upsilon_t = \mathbf{0}$ *AND* $\chi_t \sqsubseteq 1$ **then**
    if $|\chi_t|$ *is EVEN* **then**
      $\Delta z_t \leftarrow 0$;
    end
    if $|\chi_t|$ *is ODD* **then**
      $\Delta z_t \leftarrow 1$;
    end
  end
  else
    $\Delta z_t \leftarrow 2$;
  end
  $t_1 \leftarrow u_0 + u_{f_1} + u_{f_2} + \ldots + u_{f_a} \bmod 2$;
  $t_2 \leftarrow u_0 + 2 \cdot OR(v_0, v_{\tau_1}, v_{\tau_2}, \ldots, v_{\tau_b})$;
  $[u_0, u_1, \ldots, u_{n-2}, u_{n-1}] \leftarrow [u_1, u_2, \ldots, u_{n-1}, t_1]$;
  $[v_0, v_1, \ldots, v_{n-2}, v_{n-1}] \leftarrow [v_1, v_2, \ldots, v_{n-1}, t_2]$;
  $t = t + 1$;
end
$\Delta Z = [\Delta z_0, \Delta z_1, \ldots, \Delta z_{r-1}]$;
Return $[\chi_0, \chi_1, \ldots, \chi_{r-1}]$, $[\Upsilon_0, \Upsilon_1, \ldots, \Upsilon_{r-1}]$, ΔZ

Algorithm 1. D-Grain(ϕ, r)

2.2 Derivative of a Boolean Function

Certain properties of Boolean functions have been exploited for the fault attack described in [5]. We use some other properties of the Boolean functions to mount our attack and these are described here. A q-variable Boolean function is a mapping from the set $\{0,1\}^q$ to the set $\{0,1\}$. One important way to represent a Boolean function is by its Algebraic Normal Form (ANF). A q-variable Boolean function $h(x_1,\ldots,x_q)$ can be considered to be a multivariate polynomial over $GF(2)$. This polynomial can be expressed as a sum of products representation of all distinct k-th order products ($0 \leq k \leq q$) of the variables. The number of variables in the highest order product term with nonzero coefficient is called the *algebraic degree*, or simply the degree of h and denoted by $deg(h)$. Functions of degree at most one are called affine functions.

Definition 1. *Consider a q-variable Boolean function F and any vector $\alpha \in \{0,1\}^q$. We refer to the function $F(\mathbf{x}+\alpha)$ as a translation of F. The set of all possible translations of a given function F is denoted by the term 'Translation Set' and by the symbol $\mathcal{A}_F$. Since a q-variable function can have at most 2^q translations, the cardinality of $\mathcal{A}_F$ is atmost 2^q.*

Definition 2. *Consider a q-variable Boolean function F and its translation set $\mathcal{A}_F$. Any GF(2) linear combination $\hat{F}$ of the functions in $\mathcal{A}_F$, i.e., $\hat{F}(\mathbf{x}) = c_1F(\mathbf{x}\oplus\alpha_1)\oplus c_2F(\mathbf{x}\oplus\alpha_2)\oplus\cdots\oplus c_iF(\mathbf{x}\oplus\alpha_i)$, where $c_1,c_2,\ldots,c_i \in \{0,1\}$ is said to be a derivative of F. If $\hat{F}$ happens to be an affine Boolean function and $c_1 = c_2 = \cdots = c_i = 1$ then the set of vectors $\pi = [\alpha_1,\alpha_2,\ldots,\alpha_i]$ is said to be an affine differential tuple of F. If none of the vectors in π is $\mathbf{0}$ then π is said to be a weight i affine differential tuple of F otherwise π is said to be a weight $(i-1)$ affine differential tuple.*

3 Differential Fault Analysis of the Grain Family

In this section, we assume that the attacker has the ability to induce exactly a single bit toggle at a random LFSR location by applying a fault. Later, in Section 6, we will consider a more practical fault model in which an injected fault toggles more than one consecutive bits in LFSR locations.

3.1 Obtaining the Location of the Fault

Let $S_0 \in \{0,1\}^{2n}$ be the initial state of the Grain family PRGA described in Section 1.1 and S_{0,Δ_ϕ} be the initial state resulting after injecting a single bit fault in LFSR location $\phi \in [0,n-1]$. Let $Z = [z_0,z_1,\ldots,z_{2n-1}]$ and $Z^\phi = [z_0^\phi,z_1^\phi,\ldots,z_{2n-1}^\phi]$ be the first $2n$ key-stream bits produced by S_0 and S_{0,Δ_ϕ} respectively. The task for the fault location identification routine is to determine the fault location ϕ by analyzing the difference between Z and Z^ϕ. Of course, in Grain-128a the entire Z and Z_ϕ are not available to the attacker. Thus, we will deal with Grain-128a separately.

Our approach to determine the fault location is an improvement over the one presented in [5]. The basic idea used in [5] was the fact that if a fault is injected in single LFSR location at the beginning of the PRGA, then at certain specific PRGA rounds the key-stream bits are guaranteed to be equal. However, this technique requires multiple fault injections at the same LFSR bit to conclusively identify the fault location. In our work we utilize the added fact that due to a single bit fault at the beginning of the PRGA, the key-stream bits at certain other PRGA rounds are guaranteed to be different as well. This removes the requirement for multiple single-bit fault injection at the same LFSR location.

Grain v1 and Grain-128. As in [5], we define a $2n$ bit vector E_ϕ over GF(2) defined as follows. Let E_ϕ be the bitwise logical XNOR (complement of XOR) of Z and Z_ϕ, i.e., $E_\phi = 1 \oplus Z \oplus Z^\phi$. Similarly we define $\overline{E}_\phi = 1 \oplus E_\phi$. Since S_0 can have 2^{n+m} values (each arising from a different combination of the n bit key and m bit IV, the remaining $n-m$ padding bits are fixed), each of these choices of S_0 may lead to different patterns of E_ϕ. The bitwise logical AND of all such vectors E_ϕ is denoted as the First Signature vector Sgn^1_ϕ for the fault location ϕ. Similarly the bitwise logical AND of all such vectors $\overline{E}_\phi$ is denoted as the Second Signature vector Sgn^2_ϕ for the fault location ϕ. Note that if $Sgn^1_\phi(i)$ ($Sgn^2_\phi(i)$) is 1 for any $i \in [0, 2n-1]$ then the i^{th} key-stream bit produced by S_0 and S_{0,Δ_ϕ} is equal (different) for all choices of S_0.

This implies that if $\Delta_\phi Z = [\Delta_\phi z_0, \Delta_\phi z_1, \ldots, \Delta_\phi z_{2n-1}]$ is the third output of the routine D-Grain$(\phi, 2n)$, then

$$Sgn^1_\phi(i) = \begin{cases} 1, \text{ if } \Delta_\phi z_i = 0, \\ 0, \text{ otherwise.} \end{cases} \qquad Sgn^2_\phi(i) = \begin{cases} 1, \text{ if } \Delta_\phi z_i = 1, \\ 0, \text{ otherwise.} \end{cases}$$

Grain-128a. Grain-128a has a different encryption strategy in which the first 64 key-stream bits and every alternate key-stream bit thereof is used to construct the message authentication code and therefore unavailable to the attacker. To circumvent this problem, in Grain-128a every re-keying is followed by a fault injection at the beginning of round 64 of the PRGA instead of round 0. Hence the vectors Z, Z^ϕ are defined as $Z = [z_{64}, z_{66}, \ldots, z_{318}]$ and $Z^\phi = [z^\phi_{64}, z^\phi_{66}, \ldots, z^\phi_{318}]$. As before, we define $E(\phi) = 1 \oplus Z \oplus Z^\phi$ and $\overline{E}(\phi) = 1 \oplus E(\phi)$ and Sgn^1_ϕ, Sgn^2_ϕ are defined as the bitwise AND of all possible $E(\phi), \overline{E}(\phi)$ respectively. Note that if a fault is applied at a random LFSR location ϕ at the 64^{th} PRGA round, then the t^{th} state of Δ_ϕ-Grain will align itself with the $(64+t)^{th}$ state of Grain-128a. This implies that if $\Delta_\phi Z = [\Delta_\phi z_0, \Delta_\phi z_1, \ldots, \Delta_\phi z_{255}]$ is the third output of the routine D-Grain$(\phi, 256)$, then

$$Sgn^1_\phi(i) = \begin{cases} 1, \text{ if } \Delta_\phi z_{2i} = 0, \\ 0, \text{ otherwise.} \end{cases} \qquad Sgn^2_\phi(i) = \begin{cases} 1, \text{ if } \Delta_\phi z_{2i} = 1, \\ 0, \text{ otherwise.} \end{cases}$$

3.2 Steps for Location Identification

The task for the fault identification routine is to determine the value of ϕ given the vector E_ϕ. For any element $V \in \{0,1\}^l$, define the set $B_V = \{i : 0 \leq i <$

$l,\ V(i) = 1\}$ i.e. B_V is the support of of V. Now define a relation $\preceq$ in $\{0,1\}^l$ such that for any two elements $V_1, V_2 \in \{0,1\}^l$, we will have $V_1 \preceq V_2$ if $B_{V_1} \subseteq B_{V_2}$. Now we check the elements in B_{E_ϕ} and $B_{\overline{E}_\phi}$. By definition, these are the PRGA rounds i during which $z_i = z_i^\phi$ and $z_i \neq z_i^\phi$ respectively. By the definition of the first and second Signature vector proposed above, we know that for the correct value of ϕ, $B_{Sgn_\phi^1} \subseteq B_{E_\phi}, B_{Sgn_\phi^2} \subseteq B_{\overline{E}_\phi}$ and hence $Sgn_\phi^1 \preceq E_\phi, Sgn_\phi^2 \preceq \overline{E}_\phi$. So our strategy would be to search all the n many first Signature vectors and formulate the first candidate set $\Psi_{0,\phi} = \{\psi : 0 \leq \psi \leq n-1,\ Sgn_\psi^1 \preceq E_\phi\}$. 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 $\Psi_{1,\phi} = \{\psi : \psi \in \Psi_{0,\phi},\ Sgn_\psi^2 \preceq \overline{E}_\phi\}$. 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, we will be unable to decide conclusively at this stage.

However, the task is made simpler if we can access the faulty key-streams Z^ϕ and hence get E_ϕ for all $\phi \in [0, n-1]$. This is possible since our fault model allows multiple re-keying with the same but unknown Key-IV. We need to reset the cipher each time and then inject a fault at any random unknown LFSR location at the beginning of the PRGA. By performing this process around $n \cdot \sum_{i=1}^{n} \frac{1}{i} \approx n \ln n$, we can expect to hit all the LFSR locations in $[0, n-1]$ and obtained n different faulty key-streams Z^ϕ.

The remaining task is to label each Z^ϕ with a unique $\phi \in [0, n-1]$. Using the techniques outlined above for all the faulty key-streams, we will be able to uniquely label them if (i) for all $\phi \in [0, n-1]$, $|\Psi_{1,\phi}| = 1$, i.e., all the faulty key-streams were assigned unique labels, or (ii) for all $\phi_i \in \mathcal{W} = \{\phi_1, \phi_2, \ldots, \phi_j\}$, $|\Psi_{1,\phi_i}| > 1$ AND $|\Psi_{1,\phi_i} - \overline{\mathcal{W}}| = 1$, where $\overline{\mathcal{W}} = [0, n-1] - \mathcal{W}$, i.e., $\overline{\mathcal{W}}$ is the set of labels that have already been assigned. The second condition states that even if some faulty key-stream Z^{ϕ_i} has not been labelled uniquely, its second candidate set Ψ_{1,ϕ_i} (along with the element ϕ_i) must contain only those labels that have already been uniquely assigned. Given a random key $K \in_R \{0,1\}^n$ and a random Initial Vector $IV \in_R \{0,1\}^m$ the probability that all n faulty key-streams can be labelled uniquely has been experimentally found to be 1 for all the 3 ciphers Grain v1, Grain-128 and Grain-128a. The experiments were performed over a set of 2^{20} randomly chosen Key-IV pairs. We sum up the fault location identification routine in the following steps.

A. Reset the cipher with the unknown key K and Initial Vector IV and record the first $2n$ fault-free key-stream bits Z.

B. Reset the cipher again with K, IV, and inject a single bit fault in a random LFSR location $\phi,\ 0 \leq \phi \leq n-1$ at the beginning of the PRGA. Record the faulty key-stream bits Z^ϕ, calculate E_ϕ and $\Psi_{1,\phi}$

C. Repeat Step **[B.]** around $n \ln n$ times so that n different faulty key-stream vectors corresponding to all LFSR locations $\phi \in [\ 0, n-1]$ are obtained and calculate the corresponding $\Psi_{1,\phi}$ vector.

D. Once all the faulty key-stream vectors have been labelled we proceed to the next stage of the attack.

4 Beginning the Attack

Let us first describe some notations that we will henceforth use.

1. $S_t = [x_0^t, x_1^t, \ldots, x_{n-1}^t \quad y_0^t, y_1^t, \ldots, y_{n-1}^t]$ is used to denote the internal state of the cipher at the beginning of round t of the PRGA. Thus x_i^t (y_i^t) denotes the i^{th} NFSR (LFSR) bit at the start of round t of the PRGA. When $t = 0$, we use $S_0 = [x_0, x_1, \ldots, x_{n-1} \quad y_0, y_1, \ldots, y_{n-1}]$ to denote the internal state for convenience.
2. S_{t,Δ_ϕ} is used to denote the internal state of the cipher at the beginning of round t of the PRGA, when a fault has been injected in LFSR location ϕ at the beginning of the PRGA round.
3. z_i^ϕ denotes the key-stream bit produced in the i^{th} PRGA round, after faults have been injected in LFSR location ϕ at the beginning of the PRGA round. z_i is the fault-free i^{th} key-stream bit.
4. $\eta_t = [x_{l_1}^t, x_{l_2}^t \ldots, x_{l_c}^t, y_{i_1}^t, y_{i_2}^t \ldots, y_{i_d}^t]$ is the set of elements in S_t which contribute to the output key-stream bit function linearly and $\theta_t = [y_{h_1}^t, y_{h_2}^t, \ldots, y_{h_e}^t, x_{j_1}^t, x_{j_2}^t, \ldots, x_{j_w}^t]$ be the subset of S_t which forms the input to the combining function h.
5. If $\mathbf{v}$ is an integer vector all elements of which are either 0 or 1, then we express $\mathbf{v}$ as a vector over GF(2) and denote it by the symbol $\widetilde{\mathbf{v}}$.
6. If $\mathbf{w}$ is a vector over GF(2) then $\mathcal{P}(\mathbf{w})$ denotes the GF(2) sum of the elements of $\mathbf{w}$.

Determining the LFSR. During PRGA, the LFSR evolves linearly and independent of the NFSR. Hence, y_i^t for any $i \in [0, n-1]$ and $t \geq 0$ is a linear function of $y_0, y_1, \ldots, y_{n-1}$. Let S_0 and S_{0,Δ_ϕ} be two initial states of the Grain PRGA (as described in Section 1.1) that differ in only the LFSR location $\phi \in [0, n-1]$. Let $[\chi_{0,\phi}, \chi_{1,\phi}, \ldots, \chi_{2n-1,\phi}], [\Upsilon_{0,\phi}, \Upsilon_{1,\phi}, \ldots, \Upsilon_{2n-1,\phi}]$, $\Delta_\phi Z$ be the outputs of D-Grain$(\phi, 2n)$.

Let $[\mathbf{0}, \alpha_1]$ be a weight 1 affine differential tuple of h, such that $h(\mathbf{x}) \oplus h(\mathbf{x} \oplus \alpha_1) = h_{01}(\mathbf{x})$ is a function of variables that takes input from LFSR locations only. If, for some round t of the PRGA, we have $\chi_{t,\phi} \sqsubseteq 1$, $\Upsilon_{t,\phi} \sqsubseteq 1$ and $\widetilde{\Upsilon}_{t,\phi} = \alpha_1$, then we can conclude that the t^{th} round fault-free and faulty internal states S_t and S_{t,Δ_ϕ} differ deterministically in the bit locations that contribute to producing the output key-stream bit at round t. In such a scenario, the GF(2) sum of the fault-free and faulty key-stream bit at round t is given by $z_t \oplus z_t^\phi = \mathcal{P}(\eta_t) \oplus h(\theta_t) \oplus \mathcal{P}(\eta_t \oplus \widetilde{\chi}_{t,\phi}) \oplus h(\theta_t \oplus \widetilde{\Upsilon}_{t,\phi}) = \mathcal{P}(\widetilde{\chi}_{t,\phi}) \oplus h(\theta_t) \oplus h(\theta_t \oplus \alpha_1) = \mathcal{P}(\widetilde{\chi}_{t,\phi}) \oplus h_{01}(\theta_t)$.

Note that in the above equation $\mathcal{P}(\widetilde{\chi}_{t,\phi}) \oplus h_{01}(\theta_t)$ is an affine Boolean function in the LFSR state bits of $S_t = [y_0^t, y_1^t, \ldots, y_{n-1}^t]$ and hence $[y_0, y_1, \ldots, y_{n-1}]$. Since $z_t \oplus z_t^\phi$ is already known to us, this gives us one linear equation in $[y_0, y_1, \ldots, y_{n-1}]$. The trick is to get n such linear equations which are linearly independent by searching over all possible values of ϕ and affine differential tuples of h. Of course h may not have an affine differential tuple $[\mathbf{0}, \alpha_1]$ of weight 1 or even if it does $\widetilde{\Upsilon}_{t,\phi} = \alpha_1$ and $\chi_{t,\phi} \sqsubseteq 1$ may not hold for any t or ϕ. In such situations, one can look at other higher weight affine differential tuples.

Exploring Higher Weight Affine Differential Tuples. Consider λ many fault locations $\phi_i \in [0, n-1]$. Let $[\chi_{0,\phi_i}, \ldots, \chi_{2n-1,\phi_i}]$, $[\Upsilon_{0,\phi_i}, \ldots, \Upsilon_{2n-1,\phi_i}]$, $\Delta_{\phi_i} Z$ be the λ many outputs of D-GRAIN$(\phi_i, 2n)$ for $i \in [1, \lambda]$. Let $[\mathbf{0}, \alpha_1, \alpha_2, \ldots, \ldots, \alpha_\lambda]$ be a weight λ (where λ is an odd number) affine differential tuple of h, such that $h(\mathbf{x}) \oplus \bigoplus_{i=1}^{\lambda} h(\mathbf{x} \oplus \alpha_i) = H_1(\mathbf{x})$ is a function of variables that takes input from LFSR locations only. If for some round t of the PRGA, $\chi_{t,\phi_i} \sqsubseteq 1$, $\Upsilon_{t,\phi_i} \sqsubseteq 1$ and $\widetilde{\Upsilon}_{t,\phi_i} = \alpha_i$ for all $i \in [1, \lambda]$, then by the arguments outlined in the previous subsection, we conclude $z_t \oplus \bigoplus_{i=1}^{\lambda} z_t^{\phi_i} = \mathcal{P}(\eta_t) \oplus h(\theta_t) \oplus \bigoplus_{i=1}^{\lambda} \left(\mathcal{P}(\eta_t \oplus \widetilde{\chi}_{t,\phi_i}) \oplus h(\theta_t \oplus \widetilde{\Upsilon}_{t,\phi_i}) \right) = \bigoplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_1(\theta_t)$.

Note that if λ is odd then we can not exploit differential tuples of the form $[\alpha_1, \alpha_2, \ldots, \ldots, \alpha_\lambda]$ where all $\alpha_i \neq \mathbf{0}$ as an odd number of terms do not cancel out in GF(2). Instead, if $[\alpha_1, \alpha_2, \ldots, \ldots, \alpha_\lambda]$ is a weight λ (λ is an even number) affine differential tuple of h, such that $\bigoplus_{i=1}^{\lambda} h(\mathbf{x} \oplus \alpha_i) = H_2(\mathbf{x})$ is a function of variables, that takes inputs from LFSR locations only, then by the previous arguments we have $\bigoplus_{i=1}^{\lambda} z_t^{\phi_i} = \bigoplus_{i=1}^{\lambda} \left(\mathcal{P}(\eta_t \oplus \widetilde{\chi}_{t,\phi_i}) \oplus h(\theta_t \oplus \widetilde{\Upsilon}_{t,\phi_i}) \right) = \bigoplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_2(\theta_t)$.

Note that each of the above cases gives us one linear equation in $[y_0, y_1, \ldots, y_{n-1}]$. We formally state the routine $\mathrm{FLE}_L(\lambda)$ in Algorithm 2 that attempts to find such linear equations by investigating weight λ affine differential tuples.

Solving the System. Ideally we should get n linearly independent equations in $[y_0, y_1, \ldots, y_{n-1}]$ to solve the LFSR. If a call to $\mathrm{FLE}_L(1)$ does not give us the requisite number of equations then we must call $\mathrm{FLE}_L(2)$ and if required $\mathrm{FLE}_L(3)$ to obtain the required number of equations. Note that the number of iterations in the outer most **'for'** loop is of $\mathrm{FLE}_L(\lambda)$ is $\binom{n}{\lambda} \approx O(n^\lambda)$, so beyond a certain value of λ, it may not be practically feasible to call $\mathrm{FLE}_L(\lambda)$. Assuming that we have n outputs from the successive $\mathrm{FLE}_L(\lambda)$ routines of the form

$$t_i, \quad [\phi_{1,i}, \phi_{2,i}, \ldots, \phi_{\lambda_i,i}], \ \gamma_i \oplus \bigoplus_{j=0}^{n-1} c_{i,j} y_j, \ [\alpha_{1,i}, \alpha_{2,i}, \ldots, \alpha_{\lambda_i,i}],$$

$\forall i \in [0, n-1]$, if λ_i is even. Else the last output will be of the form $[\mathbf{0}, \alpha_{1,i}, \alpha_{2,i}, \ldots, \alpha_{\lambda_i,i}]$. Then we can write the equations so obtained in matrix form $LY = W$, where L is the $n \times n$ coefficient matrix $\{c_{i,j}\}$ over GF(2), Y is the column vector $[y_0, y_1, \ldots, y_{n-1}]^t$ and W is a column vector. The i^{th} element $W(i)$ is given by $\gamma_i \oplus z_{t_i} \oplus \bigoplus_{j=1}^{\lambda_i} z_{t_i}^{\phi_{j,i}}$, if λ_i is odd, and $\gamma_i \oplus \bigoplus_{j=1}^{\lambda_i} z_{t_i}^{\phi_{j,i}}$ if λ_i is even, $\gamma_i \in \{0, 1\}$.

If the equations are linearly independent then L is invertible. Thus, the solution Y of the above system are obtained by computing $L^{-1}W$. Both L and its inverse may be precomputed and hence the solution can be obtained immediately after recording the faulty bits.

Determining the NFSR. Once the LFSR state has been determined, we proceed to finding the NFSR state. Since the NFSR updates itself non-linearly, the method used to determine the NFSR initial state will be slightly different from the LFSR. If λ is odd, let $[\mathbf{0}, \alpha_1, \alpha_2, \ldots, \alpha_\lambda]$ be a weight λ (where λ is an odd number) tuple of h (not necessarily affine differential), such that

Input: λ: An integer > 0;
Output: Set of Rounds t, locations $[\phi_1, \phi_2, \ldots, \phi_\lambda]$, Affine expression in $[y_0, y_1, \ldots, y_{n-1}]$; Tuples $[\alpha_1, \ldots, \alpha_\lambda]$

for $\phi_1 = 0$ *to* $n-1$, $\phi_2 = 0$ *to* $n-1$, $\ldots$, $\phi_\lambda = 0$ *to* $n-1$ **do**
  if *All* ϕ_j*'s are pairwise unequal* **then**
    for $i = 1$ *to* λ **do**
      $([\chi_{0,\phi_i}, \ldots, \chi_{2n-1,\phi_i}],\ [\Upsilon_{0,\phi_i}, \ldots, \Upsilon_{2n-1,\phi_i}],\ \Delta_{\phi_i} Z) = \text{D-Grain}(\phi_i, 2n)$
    end
    for $t = 0$ *to* $2n-1$ **do**
      if $\chi_{t,\phi_i} \sqsubseteq 1$ *AND* $\Upsilon_{t,\phi_i} \sqsubseteq 1$, $\forall i \in [1, \lambda]$ **then**
        if λ *is odd* **then**
          $H_1(\mathbf{x}) = h(\mathbf{x}) \oplus_{i=0}^{\lambda} h(\mathbf{x} \oplus \widetilde{\Upsilon}_{t,\phi_i})$;
          if H_1 *is a function only on LFSR bits* **then** **1.1**
            Output Round t, Locations $[\phi_1, \phi_2, \ldots, \phi_\lambda]$, Expression $\oplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_1(\theta_t)$;
            Output Tuple $[\mathbf{0}, \widetilde{\Upsilon}_{t,\phi_1}, \ldots, \widetilde{\Upsilon}_{t,\phi_\lambda}]$
          end
        end
        else
          $H_2(\mathbf{x}) = \oplus_{i=0}^{\lambda} h(\mathbf{x} \oplus \widetilde{\Upsilon}_{t,\phi_i})$;
          if H_2 *is a function only on LFSR bits* **then** **1.2**
            Output Round t, Locations $[\phi_1, \phi_2, \ldots, \phi_\lambda]$, Expression $\oplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_2(\theta_t)$;
            Output Tuple $[\widetilde{\Upsilon}_{t,\phi_1}, \ldots, \widetilde{\Upsilon}_{t,\phi_\lambda}]$
          end
        end
      end
    end
  end
end

Algorithm 2. $\text{FLE}_L(\lambda)$

$h(\mathbf{x}) + \bigoplus_{i=1}^{\lambda} h(\mathbf{x} \oplus \alpha_i) = H_1(\mathbf{x}) = x' \oplus H_{11}(\mathbf{x})$ where x' is a variable that takes input from an NFSR location and $H_{11}(\mathbf{x})$ is a function only on the LFSR variables. If for some round t of the PRGA $\chi_{t,\phi_i} \sqsubseteq 1$ and $\Upsilon_{t,\phi_i} \sqsubseteq 1$ and $\widetilde{\Upsilon}_{t,\phi_i} = \alpha_i$ for all $i \in [1, \lambda]$, then by the arguments outlined in the previous subsection we conclude $z_t \oplus \bigoplus_{i=1}^{\lambda} z_t^{\phi_i} = \mathcal{P}(\eta_t) \oplus h(\theta_t) \oplus \bigoplus_{i=1}^{\lambda} \left(\mathcal{P}(\eta_t \oplus \widetilde{\chi}_{t,\phi_i}) \oplus h(\theta_t \oplus \widetilde{\Upsilon}_{t,\phi_i}) \right) = \bigoplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_1(\theta_t) = \bigoplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_{11}(\theta_t) \oplus x_{j_r}^t$, for some $r \in [1, w]$. Since, the LFSR is already known, $H_{11}(\theta_t)$ can be calculated and that leaves $x_{j_r}^t$ as the only unknown in the equation, whose value is also calculated immediately after recording the faulty bits and solving the LFSR.

The λ even case can be dealt with similarly. We can describe another routine $\text{FLE}_N(\lambda)$ which will help in determining the NFSR state. This routine is similar to the $\text{FLE}_L(\lambda)$ routine described in Algorithm 2. The only differences are that line **1.1** will change to

if $H_1(\mathbf{x}) = x' \oplus H_{11}(\mathbf{x})$ where x' is an NFSR term and $H_{11}(\mathbf{x})$ depends on LFSR variables only.

Line **1.2** of Algorithm 2 will also change accordingly. With the help of $\text{FLE}_N(\lambda)$ routine, we can obtain specific NFSR state bits at various rounds of operation of

the PRGA. Due to the shifting property of shift registers, the following equation holds $x_i^t = x_{i-1}^{t+1}$. For example, calculating x_{46}^{30} and x_{50}^{32} is the same as determining the two NFSR state bits of the internal state S_{30}: x_{46}^{30} and x_{52}^{30}.

Hence by using the $\text{FLE}_N(\lambda)$ for successive values of λ, one can obtain all the n NFSR state bits of S_t for some $t \geq 0$. Since the LFSR initial state of S_0 is already known and due to the fact that the LFSR operates independent of the NFSR in the PRGA, the attacker can compute the LFSR state bits of S_t by simply running the Grain PRGA forward for t rounds and thus compute the entire of S_t.

4.1 Finding the Secret Key and Complexity of the Attack

It is known that the KSA, PRGA routines in the Grain family are invertible (see [6, 12]). Once we have all the bits of S_t, by running the PRGA^{-1} (inverse PRGA) routine for t rounds one can recover S_0. Thereafter the KSA^{-1} (inverse KSA) routine can be used to find the secret key.

The attack complexity directly depends on the number of re-keyings to be performed such that all of locations in $[0, n-1]$ of the LFSR are covered. Since each re-keying is followed by exactly one fault injection, the expected number of fault injection is $n \cdot \sum_{i=1}^{n} \frac{1}{i} \approx n \cdot \ln\ n$. Thereafter, the attack requires one matrix multiplication between an $n \times n$ matrix and an $n \times 1$ vector to recover the LFSR, and solving a few equations to get the NFSR state. After this, t invocations of the PRGA^{-1} and a single invocation of the KSA^{-1} gives us the secret key.

Note that, construction of the matrix L and running the $\text{FLE}_L(\lambda)$ and $\text{FLE}_N(\lambda)$ can be done beforehand and thus do not add to the attack complexity. However, these routines are a part of the pre-processing phase, the exact runtime of which will depend on the nature of the functions g, h and also the choice of taps used in the cipher design.

5 Attacking the Actual Ciphers

Now we will provide the details of the actual attack on Grain v1, Grain-128 and Grain-128a.

Grain v1. In Grain v1 the non linear combining function is of the form $h(s_0, s_1, s_2, s_3, s_4) = s_1 \oplus s_4 \oplus s_0s_3 \oplus s_2s_3 \oplus s_3s_4 \oplus s_0s_1s_2 \oplus s_0s_2s_3 \oplus s_0s_2s_4 \oplus s_1s_2s_4 \oplus s_2s_3s_4$. Here only s_4 corresponds to an NFSR variable. This function has 4 affine differential tuples of weight 1, only one of which ($[\mathbf{0}, \alpha = 11001]$) leads to a derivative which is a function of only LFSR variables. However, $\widetilde{\Upsilon}_{t,\phi} = \alpha$ and $\chi_{t,\phi} \sqsubseteq 1$ does not hold for any t or ϕ. Hence one needs to look at higher weight tuples.

A call to $\text{FLE}_L(3)$ returns 78 linearly independent equations. The result is given in Table 2. A call to $\text{FLE}_L(2)$ gives us the 2 other equations required to solve the system. The result is shown in Table 3. One can verify that the linear equations so obtained are linearly independent and thus LFSR can be solved

Table 2. Output of $\text{FLE}_L(3)$ for Grain v1 (ADT implies Affine Differential Tuple)

t	ϕ_1	ϕ_2	ϕ_3	Range	Expr.	ADT
$45+i$	$62+i$	$24+i$	$70+i$	$i \in [0,9]$	y_{46}^t	00000, 00100, 00110, 01000
$55+i$	$72+i$	$16+i$	$51+i$	$i \in [0,7]$		
$63+i$	$13+i$	$24+i$	$59+i$	$i \in [0,9]$		
$73+i$	$33+i$	$26+i$	$51+i$	$i \in [0,10]$		
$84+i$	$44+i$	$37+i$	$38+i$	$i \in [0,6]$		
$91+i$	$53+i$	$44+i$	$41+i$	$i \in [0,8]$		
$100+i$	$70+i$	$53+i$	$60+i$	$i \in [0,8]$		
109	79	71	69			
$77+i$	$45+i$	$51+i$	$38+i$	$i \in [0,5]$	$y_3^t \oplus y_{25}^t \oplus y_{64}^t$	00000, 01100, 10000, 10110
$83+i$	$72+i$	$57+i$	$44+i$	$i \in [0,4]$		
94	62	79	55			
95	78	63	56		$y_3^t \oplus y_{25}^t \oplus y_{46}^t \oplus y_{64}^t$	00000, 01001, 01100, 10110

Table 3. Output of $\text{FLE}_L(2)$ for Grain v1

t	ϕ_1	ϕ_2	Range	Expr.	ADT
$110+i$	$64+i$	$77+i$	$i \in [0,1]$	y_{46}^t	00001, 11000

Table 4. Output of $\text{FLE}_N(1)$ for Grain v1

t	ϕ_1	Range	Expr.	ADT
$55+i$	$23+i$	$i \in [0,14]$	$1 \oplus y_3^t \oplus y_{46}^t \oplus x_{63}^t$	00000, 01010
$70+i$	$77+i$	$i \in [0,2]$		
$91+i$	$62+i$	$i \in [0,5]$		

Table 5. Output of $\text{FLE}_N(3)$ for Grain v1

t	ϕ_1	ϕ_2	ϕ_3	Range	Expr.	ADT
$17+i$	i	$1+i$	$20+i$	$i \in [0,27]$	$1 \oplus y_3^t \oplus y_{46}^t \oplus x_{63}^t$	00000, 00001, 00010, 10000
$45+i$	$28+i$	$13+i$	$48+i$	$i \in [0,9]$		
$73+i$	$53+i$	$33+i$	$26+i$	$i \in [0,17]$	$1 \oplus y_3^t \oplus x_{63}^t$	00000, 00010, 00100, 00110

readily. A call each to $\text{FLE}_N(1)$ and $\text{FLE}_N(3)$ gives us all the NFSR bits of S_{80}. The output of these routines are given as in Tables 4 and 5. A look at these tables shows that the attacker can calculate the values of x_{63}^t for all $t \in [17, 96]$. This is equivalent to calculating x_i^{80} for all $i \in [0, 79]$. Thereafter, S_0 and the secret key may be obtained as per the techniques outlined in Section 4.1.

Grain-128. In Grain-128 the non linear combining function is of the form $h(s_0, s_1, \ldots, s_8) = s_0s_1 \oplus s_2s_3 \oplus s_4s_5 \oplus s_6s_7 \oplus s_0s_4s_8$. Only s_0, s_4 correspond to the NFSR variables. This function has 4 affine differential tuples of weight 1 which produce derivatives on LFSR variables. A call to $\text{FLE}_L(1)$ produces all the 128 equations needed to solve the LFSR. The output of this routine is given in Table 6.

A call to $\text{FLE}_N(1)$ gives us all the NFSR bits of S_{12}. The output of this routine is in Table 7. Thus, $\text{FLE}_N(1)$ gives us x_{12}^t for all $t \in [0, 115]$, and x_{95}^t for all $t \in [0, 11]$. This is equivalent to all the NFSR state bits of S_{12}. Thereafter, S_0 and the secret key may be obtained as per the techniques outlined in Section 4.1.

Table 6. Output of $\text{FLE}_L(1)$ for Grain-128

t	ϕ_1	Range	Expr.	ADT
i	$20+i$	$i \in [0, 107]$	y^t_{13}	000 000 000, 000 100 000
$61+i$	$50+i$	$i \in [0, 19]$	y^t_{60}	000 000 000, 000 000 010

Table 7. Output of $\text{FLE}_N(1)$ for Grain-128

t	ϕ_1	Range	Expr.	ADT
i	$8+i$	$i \in [0, 115]$	x^t_{12}	000 000 000, 010 000 000
$33+i$	$75+i$	$i \in [0, 11]$	x^t_{95}	000 000 000, 000 001 000

Table 8. Output of $\text{FLE}_L(1)$ for Grain-128a

t	ϕ_1	Range	Expr.	ADT
$6+2i$	$26+2i$	$i \in [0, 50]$	y^t_{13}	000 000 000, 000 100 000
$108+2i$	$70+2i$	$i \in [0, 12]$		
$2i$	$13+2i$	$i \in [0, 33]$	y^t_{20}	000 000 000, 001 000 000
$28+2i$	$107+2i$	$i \in [0, 10]$	y^t_{60}	000 000 000, 000 000 010
$50+2i$	$1+2i$	$i \in [0, 18]$		

Table 9. Output of $\text{FLE}_N(1)$ for Grain-128a

t	ϕ_1	Range	Expr.	ADT
$50+2i$	$58+2i$	$i \in [0, 34]$	x^t_{12}	000 000 000, 010 000 000
$120+2i$	$96+2i$	$i \in [0, 15]$		
$152+2i$	$102+2i$	$i \in [0, 12]$		
$2i$	$42+2i$	$i \in [0, 42]$	x^t_{95}	000 000 000, 000 001 000
$86+2i$	$38+2i$	$i \in [0, 4]$		

Grain-128a. In Grain-128a, the first 64 key-stream bits and every alternate key-stream bit thereof are used to construct the message authentication code and therefore unavailable to the attacker. To resolve this problem, in Grain-128a every re-keying is followed by a fault injection at the beginning round 64 of the PRGA instead of round 0 and the goal of the attacker is to reconstruct the internal state at the 64^{th} instead of the 0^{th} PRGA round. Note that if a fault is applied at a random LFSR location ϕ at the 64^{th} PRGA round, then the t^{th} state of Δ_ϕ-Grain will align itself with the $(64+t)^{th}$ state of the actual cipher. Hence, in a slight departure from the notation introduced in the previous section we will call the 64^{th} PRGA state S_0 and all other notations are shifted with respect to t accordingly (e.g., S_t refers to the $(64+t)^{th}$ PRGA state etc).

The key-stream bit at every odd numbered round (after round 64 of the PRGA) is used for making the MAC and is unavailable to the attacker. Hence after calling $\text{FLE}_L(1)$ the attacker must reject all outputs with an odd value of t. Even then the attacker obtains all the equations required to solve the LFSR. The output is presented in Table 8. Similarly a call to $\text{FLE}_N(1)$ after rejecting outputs with odd values of t, gives us 112 NFSR bits of S_{62}. The output is given in Table 9.

At this point, the attacker could simply guess the remaining 16 bits of S_{62} or give a call to $\text{FLE}_N(2)$ and thus increase the complexity of the preprocessing stage. As it turns out, the attacker can do even better without going for these two options. The 16 NFSR bits not determined at this point are x^{62}_{2i+1}, for $0 \le i \le 15$. Let us now look at the equations for the key-stream bits z_{62+2j} for $j \in [0, 8]$,

$$z_{62+2j} = \bigoplus\nolimits_{i \in \mathcal{B}} x^{62}_{i+2j} \oplus x^{62}_{15+2j} \oplus y^{62}_{93+2j} \oplus h(\theta_{62+2j}),$$

where $\mathcal{B} = \{2, 36, 45, 64, 73, 89\}$. Now, $x^{62}_{15+2j},\ j \in [0, 8]$ is the only unknown in each of these equations and so its value can be calculated immediately. This leaves us with the 7 unknown bits $x^{62}_1, x^{62}_3, \ldots, x^{62}_{13}$. In addition to the entries in Table 9, $\text{FLE}_N(1)$ also gives the output

$$t = 96 + 2i, \phi_1 = 48 + 2i, x_{95}^t, [\mathbf{0}, 000\ 001\ 000],\ \forall i \in [0, 6].$$

This gives us the bits x_{95}^{96+2i} or equivalently x_{127}^{64+2i} for $i \in [0, 6]$. Let us write the NFSR update function g in the form $g(X) = x' \oplus g'(X)$, where x' corresponds to the variable that taps the 0^{th} NFSR location. Then looking at the NFSR update rule for Grain-128a, we have

$$x_{127}^{64+2i} = y_0^{63+2i} \oplus x_0^{63+2i} \oplus g'(X_{63+2i}) = y_{1+2i}^{62} \oplus x_{1+2i}^{62} \oplus g'(X_{63+2i}),$$

$\forall i \in [0, 6]$. Again, $x_{1+2i}^{62},\ i \in [0, 6]$ is the only unknown in these equations and so its value can be calculated immediately. This gives us all the NFSR bits of S_{62}. Using the techniques in Section 4.1, S_0 can be calculated. Since this state corresponds to the 64^{th} PRGA state, the PRGA^{-1} routine needs to be run 64 more times before invoking the KSA^{-1} routine which would then reveal the secret key.

6 When a Fault Injection Affects More Than One Locations: Some Preliminary Observations

So far we have discussed an attack scenario where an injected fault flips exactly one bit value at a random LFSR location. We now relax the requirements of the attack, and assume a fault model that allows the user to inject a fault that affects more than one locations. Our strategy is that, if the fault injection affects more than one location, we will be able to identify that scenario, and will not use those cases for further processing.

We consider the case when at most three consecutive locations can be disturbed by a single fault injection. Thus, four cases are possible: (a) exactly one LFSR bit is flipped (n cases), (b) 2 consecutive locations $i, i+1$ of the LFSR are flipped ($n-1$ cases), (c) 3 consecutive locations $i, i+1, i+2$ of the LFSR are flipped ($n-2$ cases) and (d) locations $i, i+2$ are flipped but not $i+1$ ($n-2$ cases). Studying such a model makes sense if we attack an implementation of Grain where the LFSR register cells are physically positioned linearly one after the other.

It is clear that such a fault model allows a total of $n+n-1+2(n-2) = 4n-5$ types of faults out of which only n are single bit-flips. We assume that each of these $4n-5$ cases are equally probable. The success of our attack that we have described in Section 4 will depend on the ability of the attacker to deduce whether a given faulty key-stream vector has been produced as a result of a single bit toggling of any LFSR location or a multiple-bit toggle. Thus, we need to design a fault location identification algorithm that analyzes a faulty key-stream and (i) if the faulty key-stream has been produced due to a single bit toggling of any LFSR location, the algorithm should output that particular position, and (ii) if the faulty key-stream has been produced due to multiple-bit toggling of LFSR locations, the algorithm should infer that the faulty key-stream could not have been produced due to a single bit toggle.

To solve the problem, will use the same fault location identification technique used in Section 3.2. For the method to be a success, the routine would return the fault location numbers for all possible cases when a single LFSR location is toggled (n out of $4n-5$ cases), and the empty set $\emptyset$ for all the other $3n-5$ cases. Let p_s be the probability that the fault location identification technique has succeeded (theoretically, the probability is defined over all possible Key-IV pairs). By performing computer simulations over 2^{20} randomly chosen Key-IV pairs, the value of p_s was found to be 0.99994 for Grain v1, 1.00 for Grain-128 and 0.993 for Grain-128a. Note that assuming this fault model increases the number of re-keyings and hence fault injections to $(4n-5) \cdot \ln(4n-5)$.

As the experiments show, the probability of the location identification technique failing is very small. In case the method fails for some particular Key-IV pair, we reset the cipher with the same Key-IV and repeat the fault identification routine and this time inject the fault at PRGA round 1 instead of 0 (round 66 for Grain-128a) and then try to reconstruct this first (66^{th} for Grain-128a) PRGA state using the methods outlined in Section 4. The probability that the location identification routine will fail for both PRGA round 0 and 1 is $(1-p_s)^2$ (assuming independence) and is thus even smaller. In case the method fails for both round 0 and 1, we repeat the routine on PRGA round 2 and so on.

7 Conclusion

In this paper we outline a general strategy to perform differential fault attack on ciphers with the physical structure of Grain. In particular, the attack is demonstrated on Grain v1, Grain-128 and Grain-128a. The attack also uses a much more practical and realistic fault model compared to the fault attacks on the Grain family reported in literature [5, 6, 12].

References

1. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
2. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997)
3. The ECRYPT Stream Cipher Project. eSTREAM Portfolio of Stream Ciphers (revised on September 8, 2008)
4. Ågren, M., Hell, M., Johansson, T., Meier, W.: A New Version of Grain-128 with Authentication. In: Symmetric Key Encryption Workshop. DTU, Denmark (2011)
5. Banik, S., Maitra, S., Sarkar, S.: A Differential Fault Attack on the Grain Family of Stream Ciphers. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 122–139. Springer, Heidelberg (2012)
6. Berzati, A., Canovas, C., Castagnos, G., Debraize, B., Goubin, L., Gouget, A., Paillier, P., Salgado, S.: Fault Analysis of Grain-128. In: IEEE International Workshop on Hardware-Oriented Security and Trust 2009, pp. 7–14 (2009)

7. Dinur, I., Güneysu, T., Paar, C., Shamir, A., Zimmermann, R.: An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware. In: Lee, D.H. (ed.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 327–343. Springer, Heidelberg (2011)
8. Hell, M., Johansson, T., Meier, W.: Grain - A Stream Cipher for Constrained Environments. ECRYPT Stream Cipher Project Report 2005/001 (2005), http://www.ecrypt.eu.org/stream
9. Hell, M., Johansson, T., Maximov, A., Meier, W.: A Stream Cipher Proposal: Grain-128. In: IEEE International Symposium on Information Theory, ISIT 2006 (2006)
10. Hoch, J.J., Shamir, A.: Fault Analysis of Stream Ciphers. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 240–253. Springer, Heidelberg (2004)
11. Hojsík, M., Rudolf, B.: Differential Fault Analysis of Trivium. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 158–172. Springer, Heidelberg (2008)
12. Karmakar, S., Roy Chowdhury, D.: Fault analysis of Grain-128 by targeting NFSR. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 298–315. Springer, Heidelberg (2011)
13. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional Differential Cryptanalysis of NLFSR-based Cryptosystems. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 130–145. Springer, Heidelberg (2010)
14. Skorobogatov, S.P.: Optically Enhanced Position-Locked Power Analysis. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 61–75. Springer, Heidelberg (2006)
15. Skorobogatov, S.P., Anderson, R.J.: Optical Fault Induction Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003)