Fault Analysis of GRAIN-128

Alexandre Berzati*‡, Cécile Canovas*, Guilhem Castagnos‡, Blandine Debraize†,
Louis Goubin‡, Aline Gouget†, Pascal Paillier† and Stéphanie Salgado†
*CEA-LETI/MINATEC, 17 rue des Martyrs, 38054 Grenoble Cedex 9, France.
{alexandre.berzati,cecile.canovas}@cea.fr
†Gemalto, 6 rue de la Verrerie, 92190 Meudon, France.
{blandine.debraize,aline.gouget,pascal.paillier,stephanie.salgado}@gemalto.com
‡PRISM - Université de Versailles St-Quentin-en-Yvelines, 45 Avenue des Etats-Unis, 78035 Versailles Cedex, France.
{guilhem.castagnos,louis.goubin}@prism.uvsq.fr

***Abstract*—GRAIN-v1 is a stream cipher that has been selected in the final portfolio of the eSTREAM project. GRAIN-128 is a variant of GRAIN-v1. The best known mathematical attack against GRAIN-128 is the brute force key-search.**

This paper introduces a fault attack on GRAIN-128 based on a realistic fault model and explores possible improvements of the attack. We also discuss countermeasures to counteract our fault attack.

I. INTRODUCTION

The eSTREAM project [5] has federated a considerable research effort to identify new stream ciphers that might be interesting for widespread adoption. The stream cipher GRAIN-v1 [7] has been selected in the final portfolio of eSTREAM in Profile 2, *i.e.,* stream ciphers supporting a 80-bit key and a 64-bit initialization value (IV). However, due to the time complexity of recent time-memory-data trade-offs, the choice of a 80-bit key has been evaluated as inadequate and a 128-bit key is now considered as the minimum key size in secure applications. GRAIN-128 [6] is the 128-bit version of GRAIN-v1, a resized variant which preserves the main advantages of GRAIN-v1.

The resistance of GRAIN-128 against Differential Power Analysis (DPA) has been studied in [2]. Fault attacks, another type of *physical attacks*, constitute a powerful tool to retrieve the private key material of many different types of cryptosystems. Fault analysis was first used to break number-theoretic public-key cryptosystems [4], and was later extended to product block ciphers [3]. General techniques have been developed in [8], [1] to attack standard constructions of LFSR-based stream ciphers. However GRAIN-128 does not fall in this category of constructions since it relies on a non-linear feedback shift register.

In this paper, we suggest a fault attack on the stream cipher GRAIN-128. In Section II, we recall the description of GRAIN-128. The security model is discussed in Section III and an high-level description of our attack is given in Section IV. The main steps of the attack are described in Sections V, VI, VII, VIII. Finally, we discuss countermeasures in Section IX.

II. DESCRIPTION OF GRAIN-128

GRAIN-128 [6] supports a 128 bits key and a 96 bits IV. The design is based on two 128-bit shift registers, the first being linear (LFSR) and the second being nonlinear (NFSR). It also specifies an output function h. The internal state of GRAIN-128 has 256 bits. A schematic description of GRAIN-128 in keystream generation mode can be found in Figure 1.

The content of the LFSR (resp. NFSR) is denoted by $s_i, \ldots, s_{i+127}$ (resp. $b_i, \ldots, b_{i+127}$). The LFSR is updated by setting

$$s_{i+128} = s_i \oplus s_{i+7} \oplus_{i+38} \oplus_{i+70} \oplus_{i+81} \oplus_{i+96},$$

and the NFSR by

$$\begin{aligned} b_{i+128} \quad = \quad & s_i \oplus b_i \oplus b_{i+26} \oplus b_{i+56} \oplus b_{i+91} \oplus b_{i+96} \\ & \oplus b_{i+3}b_{i+67} \oplus b_{i+11}b_{i+13} \oplus b_{i+17}b_{i+18} \\ & \oplus b_{i+27}b_{i+59} \oplus b_{i+40}b_{i+48} \oplus b_{i+61}b_{i+65} \\ & \oplus b_{i+68}b_{i+84}. \end{aligned}$$

The filtering function h is a 9-variable Boolean function which outputs $b_{i+12}s_{i+8} \oplus s_{i+13}s_{i+20} \oplus b_{i+95}s_{i+42} \oplus s_{i+60}s_{i+79} \oplus b_{i+12}b_{i+95}s_{i+95}$. This output is then e-xored with $b_{i+2} \oplus b_{i+15} \oplus b_{i+36} \oplus b_{i+45} \oplus b_{i+64} \oplus b_{i+73} \oplus b_{i+89} \oplus s_{i+93}$ to define the output bit z_i.

Before the keystream is generated, GRAIN-128 is initialized with the 128-bit key K and the 96-bit IV. The key is loaded into the NFSR and the IV is loaded in the first 96 cells of the LFSR (the remaining 32 cells are filled with ones). Then the cipher is clocked 256 times without producing any keystream such that the output function is fed back and e-xored with the input both to the LFSR and to the NFSR.

III. FAULT ATTACK AND SECURITY MODEL

We consider that injected faults are transient and that the attacker is in possession of the physical device. The adversary is assumed to know the IV and the keystream generated by the stream cipher.

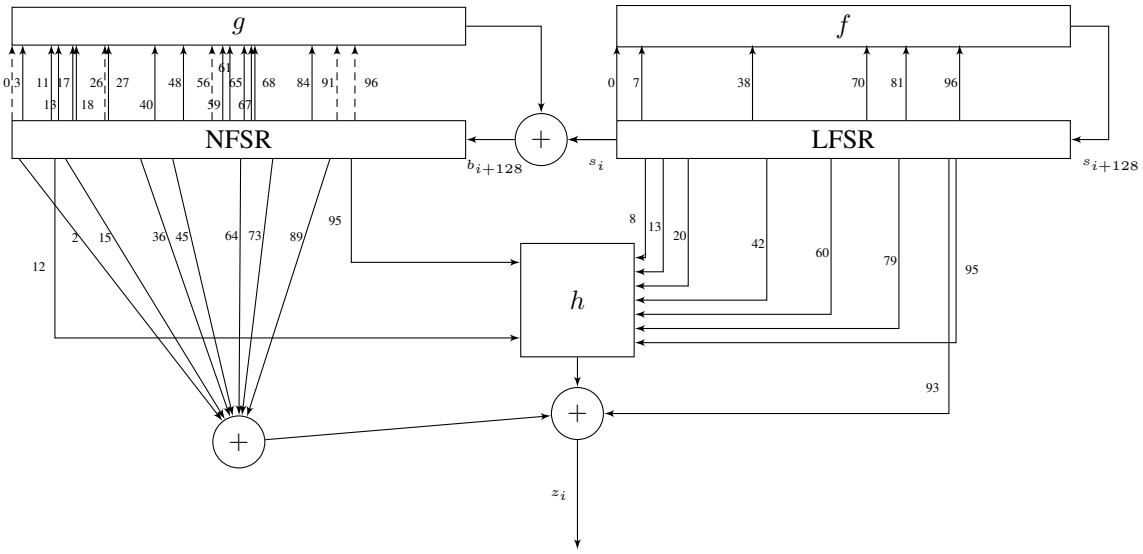

Fig. 1. Description of GRAIN-128

The ultimate goal of the attacker is to retrieve the secret key. In the following, the adversary will attempt to exploit only faults that are induced while generating the keystream, *i.e.*, after the initialization step. In this respect, the attacker first attempts to retrieve the secret initial state of the stream cipher, and then proceeds to recovering the secret key.

Depending on the fault model, the practicality of the attack may be debatable. In [8], the attacker is assumed to be able to apply bit-flipping faults with a partial control of their number, location and timing. The attacker can reset the cryptographic device to its original state. In [6], two variants of this fault model are envisioned in the particular case of GRAIN-128. We thus adopt a similar approach.

A. Definition of our Fault Model.

The adversary is assumed to be able to flip exactly one bit lying in one position in the LFSR without choosing its location but at a chosen point in time. Fault injection is performed *e.g.*, by lighting up the device with laser beams [9], [10]. The attacker has only partial control on the locations of the faults but he is assumed to be able to inject a fault over and over again at his will at the same position. In addition, the attacker is assumed to have full control over timing. The attacker is also assumed to be able to reset the cryptographic device to its original state and then apply another randomly chosen fault to the same device.

B. Practicality of our Fault Model

As we deal with an hardware implementation, choosing the time of the laser shot is possible by triggering it from the I/O signal. Shift registers are regularly clocked, and one keystream bit is computed per clock cycle. Hence, the attacker can identify steps in the execution, and it can safely be assumed that he has a full control over the exact timing of the fault injection.

The attacker can locate the position of the LFSR on the chip without increasing the number of faults by performing a preliminary fault setup stage. During this stage, he attacks a device architecturally close to the target device. He scans this device by performing laser shots on different areas and by analyzing the corresponding faulty outputs. Finally the attacker replaces the test device with the target device and attempts to inject faults with respect to the previous setup. The number of additional adjustments is as small as the test device is close to the target device.

Our model includes, as a particular case, a more restrictive fault model which is usually considered as being realistic where a fault is applied on the LFSR update function, f. The adversary is assumed to be able to randomly corrupt the result of f without knowing the computed faulty value. But in half of cases, this value is different from the correct one. Then, the perturbation of f can be seen as flipping the bit 127 of the LFSR at a chosen point in time.

This model can thus be considered as a particular application of our more general model.

IV. FAULT ATTACK ON FILTERED FSR

In [8], a general technique is introduced to attack LFSR-based stream ciphers filtered by a function f which takes j input bits. A high-level view is as follows:

1) Inject a fault and produce the keystream
2) Guess the nature of the fault
3) Check whether the guess is correct, otherwise make a new guess
4) Repeat steps 1-3 $\mathcal{O}(j)$ times
5) Solve a system of linear equations

This general framework exploits the linearity of the state update function. In this particular case, the correctness of a guess is easily checked by predicting the future differences on the input bits of the filtering function. Whenever this input difference is 0, we expect to see an output difference equal to 0. If our guess was incorrect, then we expect to see a nonzero output difference for half of these observations. So in average, we expect to reject incorrect guesses after 2^{j+1} output bits. We can easily construct a system of linear equations by collecting pairs of input/output differences corresponding to the same output bit location. Given about j pairs, we can narrow down exhaustive search on all possible input bits to one possibility. Once we have collected enough equations, we solve the system to determine the initial state of the LFSR.

Since the update function of the internal state in GRAIN-128 is not linear, the previous attack strategy cannot be applied directly. We now give a high-level description of our attack which can be summarized in five steps.

a) Characterization (Phase 1): During this phase, the attacker uses the test device. The goal is to find the position of the LFSR and also the location of one cell which can be perturbed, *i.e.*, the adversary will be able to flip the bit contained in the cell. Since the state update function of GRAIN-128 is nonlinear, we introduce in Section V a specific algorithm to check whether a guess is correct and the exact location of the LFSR cell, for any key the device may contain.

b) Check the correctness of a guess (Phase 2): The attacker uses the target device containing the wanted secret key. Based on the characterization phase, the attacker attempts to reproduce the same type of fault and possibly needs additional adjustments made with the specific algorithm. It is now assumed that the adversary is able to induce a fault at his will at the same position in the LFSR at different instants.

c) Recovering the LFSR state (Phase 3): This step is similar to the method described in [8] to construct and solve a system of linear equations in the LFSR state variables. This step is explained in more details in Section V.

d) Recovering the NFSR state (Phase 4): This step is not covered by any of the prior fault attacks on stream ciphers. We explain in Section V how to construct and solve a system of linear equations in the NFSR state variables.

e) Recovering the secret key (Phase 5): Given an internal state of GRAIN-128 at time t, the last part of the attack consists in recovering the key knowing the IV.

V. ATTACK DESCRIPTION: PHASES 1-2

The goal of this phase is to locate the position i of the flipped bit in the LFSR (at a known time t) by observing the differential bit sequence $\mathcal{S} = S \oplus S'$, where S and S' are the regular and faulted keystream, respectively. We suppose that the bit i is flipped *before* the keystream bit is computed.

For each of the 128 cells of the LFSR, it is possible to predict some *pattern* $\mathcal{P}$ in $\mathcal{S}$. If a given pattern $\mathcal{P}$ appears after a fault injection at position i in $\mathcal{S}$, and does never appear when the fault is injected at another position j, $\forall j \in [0, 127], j \neq i$, then we can deduce that the fault has been indeed injected at position i. If $\mathcal{P}$ *always* appears when the fault is injected at position i, then there an equivalence between the presence of the pattern $\mathcal{P}$ in $\mathcal{S}$ and the fact that the fault was injected at position i.

A. Description of ΔGrain Algorithm

For each and every position i, $0 \leq i \leq 127$, in the LFSR, we compute the related pattern $\mathcal{P}_i$ using a dedicated algorithm that we call ΔGrain. This algorithm makes use of two 128-bit registers denoted by ΔLFSR$= (\sigma_0, \ldots, \sigma_{127})$ and ΔNFSR$= (\beta_0, \ldots, \beta_{127})$. The 256-bit state is initialized to $\vec{0}$. When the fault is supposed to flip the bit at position i, we set $\sigma_i = 1$. We use the LFSR update function of GRAIN-128 to update the register ΔLFSR, and a variant of the NFSR update function to update the register ΔNFSR defined by:

$$g'(x_0, \cdots, x_{18}) = x_0 \vee x_1 \vee \cdots \vee x_{18} \ ,$$

where $\vee$ is an inclusive-or and $x_0, \cdots, x_{18}$ are plugged at the positions defined by the NFSR update function of GRAIN-128.

From the consecutive states of ΔLFSR and ΔNFSR, we compute the sequence of integers $\omega^0\omega^1 \cdots \omega^n \cdots$, where ω^i is the number of values equal to 1 among the 17 bit values implied in the computation of the keystream bit. Then, if the fault occurs at time t, the state of ΔLFSR after t_0 updates provides the complete knowledge of the XOR difference between the regular and the faulted LFSR state at time $t + t_0$. All the bits set to 0 in ΔNFSR after t_0 updates are, with probability 1, the bits having the same value in both the non faulted and the faulted NFSR (this does not depend on the value of the key and IV). The value of ω^{t+t_0} is the number of bits potentially flipped by the fault and also implied in the computation of the keystream bit z^{t+t_0}; all the other $17-\omega^{t+t_0}$ bits are the same in the faulted and non-faulted LFSR and NFSR.

B. Construction of Patterns

For every position, $0 \leq i \leq 127$, in the LFSR, we have to find a pattern $\{e_1, \cdots, e_n\}$ that will be used to locate faults. For example, if the first LFSR bit has been flipped at time 0, it is possible to predict that a 1-bit can appear in the differential keystream from tap s_{93} at time 35, another 1-bit at time 39 from tap b_{89}, etc. We call this pattern $\{35, 39, 55, 64\}$. The taps s_{93}, b_{64}, b_{73} and b_{89} have been chosen since they are *linear taps* meaning that they are linearly involved in the computation of the keystream. In the following, we use only linear taps.

Using ΔGrain we build the algorithm described in Fig. 2 to check if the pattern is suitable or not to locate a fault that flipped bit σ_i.

First, this algorithm allows to check that the input pattern always appears in the keystream if the fault has flipped bit p. Indeed, if ΔGrain returns $w = 1$ from a linear tap then

INPUT: Fault location p, pattern $\{e_1, e_2, \cdots, e_n\}$, range $[i_0 \cdots i_1]$
Initialize ΔLFSR and ΔNFSR to 0
Set σ_p to 1,
Clock ΔGrain e_n times,
If $\omega^{e_1} = \omega^{e_2} = \cdots = \omega^{e_n} = 1$:
  For each $i_0 \leq i < i_1$, $i \neq p$:
    Initialize ΔLFSR and ΔNFSR to 0
    Set σ_i to 1,
    Clock ΔGrain e_n times,
    If $\omega^{e_1}, \omega^{e_2}, \cdots, \omega^{e_n}$ are not all nonzero
      Return OK
    Else Return NOT OK
Else Return NOT OK

Fig. 2. Algorithm 1

- As $w < 2$, no other tap provides a bit that could be different between the faulted and not faulted execution for the computation of the keystream bit;
- As the tap is linear, this difference appears in the keystream with probability 1.

Secondly, the algorithm ensures that the pattern never appears if the fault has flipped another LFSR bit in range $i_0, \ldots, i_1$.

Algorithm 1 with $(i_0, i_1) = (0, 127)$ returns OK for patterns given in Table I. Patterns of Table II are OK for Algorithm 1 with smaller range $(i_0, i_1) = (42, 95)$ as well and pattern of Table III for $(i_0, i_1) = (68, 80)$.

Fault position	Patterns
$0 \leq i \leq 31$	$\{35+i, 39+i, 55+i, 64+i\}$
$32 \leq i \leq 37$	$\{35+(i-32), 42+(i-32), 46+(i-32), 62+(i-32), 71+(i-32)\}$
$38 \leq i \leq 41$	$\{35+(i-38), 66+(i-38), 73+(i-38), 77+(i-38)\}$
$96 \leq i \leq 127$	$\{3+(i-96), 35+(i-96), 50+(i-96), 61+(i-96)\}$

TABLE I

Fault position	Patterns
$42 \leq i \leq 67$	$\{35+(i-42), 66+(i-42), 73+(i-42), 77+(i-42)\}$
$81 \leq i \leq 95$	$\{35+(i-81), 46+(i-81), 67+(i-81), 82+(i-81)\}$

TABLE II

Fault position	Patterns
$70 \leq i \leq 80$	$\{35+(i-70), 82+(i-42), 93+(i-42), 98+(i-42)\}$

TABLE III

The presence in the keystream of one of the patterns given in Table I is a necessary condition for the fault to have been injected in position $0 \leq i \leq 41$ or $96 \leq i \leq 127$. When the attacker does not find a matching pattern in Table I (Step 1), then he deduces that the fault has been injected between position 42 and position 95. Hence if he finds a matching pattern (Step 2) in Table II, he has found a sufficient condition for the fault to have been injected at the corresponding position.

If not, he looks for the pattern given in Table III (Step 3). If he finds this pattern in the output difference, he learns the fault position. If not, the remaining positions are bits number 68 and 69 of the LFSR. Once the attacker has eliminated all the other possibilities, it is straightforward to locate the right position among the two by looking at the differential sequence.

VI. Attack Description: Phase 3

We have seen in Section V that if the fault flips one bit of the LFSR, it is always possible to learn its position by analyzing the keystream difference. Now that the fault is located, we explain how to make use of the information given by the output difference to obtain linear equations on a specific state of the LFSR. Let us recall that each keystream bit z_i is computed as the XOR of some LFSR and NFSR bits and the output of the filter h, $b_{i+12}s_{i+8} \oplus s_{i+13}s_{i+20} \oplus b_{i+95}s_{i+42} \oplus s_{i+60}s_{i+79} \oplus b_{i+12}b_{i+95}s_{i+95}$. It is easy to see that at clock i, if among all the state bits implied in the computation of z_i only one of the four bits s_{13}, s_{20}, s_{60} and s_{79} has been faulted, then the output difference is the value of an LFSR bit at clock t. For example, if $\sigma_{13} = 1$, the output difference is the value of s_{20}.

The number of LFSR bits which we can recover from the differential sequence depends on both the fault location and the number of times the cipher is clocked after the fault is injected. We describe the method to compute these numbers of bits in Figure 3.

INPUT: Fault location p, number of clock NC
OUTPUT:Number of known LFSR bits KB
Initialize ΔLFSR and ΔNFSR to 0
Set σ_p to 1 and KB to 0
For each $0 \leq i \leq NC$:
  If $\big[((\sigma_{13} = 1)$ OR $(\sigma_{20} = 1)$ OR $(\sigma_{60} = 1)$ OR $S(\sigma_{79} = 1))$ AND $(\Delta$Grain output $= 1)\big]$
  Then KB = KB +1
  If $\big[(\sigma_{13} \oplus \sigma_{20} = 1)$ AND $(\sigma_{60} \oplus \sigma_{79} = 1)$ AND $(\Delta$Grain output $= 2)\big]$
  Then KB = KB +1
  Clock ΔGrain
Return(KB)

Fig. 3. Algorithm 3

In a few words, the first If block (resp. the second block) counts the number of times it is possible to recover the value of an LFSR bit (resp. a linear equation on the LFSR bits).

Since the number of clock NC chosen by the attacker can be more than 128, and due to the linear update of the LFSR, we can consider that all these bits and equations recovered are equations in variables representing the 128 bits of an "initial state" of the LFSR. As linear dependencies can appear, the output of Algorithm 3 is not the rank of the system of linear equations. We build a second algorithm, derived from Algorithm 3, that we call "ComputeRank": Each time a linear equation is found by Algorithm 3, it is added as a column vector to a matrix with 128 rows $\mathcal{M}$. At the end, the number of column vectors in $\mathcal{M}$ is the number of linear equations found by the algorithm. $\mathcal{M}$'s rank is the number of linearly independent equations that the attacker can recover with this fault.

The "ComputeRank" algorithm can be extended by concatenating in $\mathcal{M}$ systems corresponding to several faults. This allowed us to compute the number of faults, injected at the same location but at different clocks, that are necessary to recover the initial state of the LFSR. Is is possible to improve this method but due to lack of space, we omit details here and just give the results we obtained (see Figure 4).

Fault positions	number of consecutive faults	number of faults each 4 clocks	number of faults each 30 clocks
0 to 6	90	62	44
7 to 12	81	32	19
13 to 19	41	23	16
20 to 37	34	19	15
38 to 59	20	17	11
60 to 69	16	17	11
70 to 78	14	15	10
79 to 80	11	14	9
81 to 127	6	8	6
Average nb	23.8	17.3	12.1

Fig. 4. Faults for recovering the LFSR state

Let us remark that these results are always true, for all key and IV values. We also made simulations validating this method.

VII. Description of the Attack: Phase 4

Once the LFSR state has been retrieved, the next step is to obtain the NFSR state. Note that if the LFSR state is known at time t, then it is also known at any time. In this section, we describe how to retrieve the NFSR state at a well chosen time following our fault model.

A. Obtaining equations on the NFSR state

Knowing the LFSR state, we can easily deduce linear equations on the NFSR bits from the regular keystream. The only non linear monomial in the expression of z_i is $b_{i+12}b_{i+95}s_{i+95}$. So if $s_{i+95} = 0$, then we get a linear equation in several bits of the current NFSR state.

Moreover, the differences between the faulted and non-faulted executions that have been used to recover the LFSR state in Section VI can be re-used to get extra linear equations. Suppose that a fault has been injected in position p, $0 \leq p \leq 127$, in the LFSR state $s_i, s_{i+1}, \ldots, s_{i+127}$ (*i.e.*, s_{i+p} has been flipped). At time $i + p + 32$, this fault will be in the bit 96 of the NFSR, and will not have affected the feedback function g. Then, the only differences between the state of the faulted and non-faulted execution will be at this bit of the NFSR and in known positions of the LFSR (that may have appeared if the fault has entered the feedback of the LFSR). Now, by analyzing the keystream difference after the time $t = i + p + 32$, we can get equations on the NFSR state at time t. Some linear equations can appear when the fault enters bits 12 and 95 due to the monomial $b_{i+12}b_{i+95}s_{i+95}$ in z_i. Others can appear when the fault enters locations that have a quadratic contribution in the feedback of the NFSR. For example, when a fault hits bit 84 at time t', the difference $b_{t'+84}$ will appear from the monomial $b_{t'+68}b_{t'+84}$ of $b_{t'+128}$. Eventually, we might get this bit from the keystream difference when it will be in position 89 of the NFSR, as this position contributes linearly in the keystream. All that equations can be recovered only if there are no others differences between the keystreams at the same time.

B. Number of equations from one fault

The number of linear equations that we can get from a keystream difference depends on the value of the bits of the LFSR state and on the fault position. We use Algorithm "CountEquations" of Fig. 5 with a computer algebra system to estimate this number.

1. Initialize the LFSR state with random values
2. Inject a fault in the bit at position $0 \leq p \leq 127$ of the LFSR state
3. Clock the non-faulted and the faulted LFSR states $32 + p$ times
4. Formally initialize a non-faulted NFSR state with variables $b_0, b_1, \ldots, b_{96}, \ldots, b_{127}$
   The corresponding formal faulted NFSR state is $b_0, b_1, \ldots, b_{96} + 1, \ldots, b_{127}$
5. Formally clock 117 times the non-faulted and the faulted GRAIN-128 states and count the linear equations in the variables $b_0, b_1, \ldots, b_{96}, \ldots, b_{127}$ obtained in the keystream difference

Fig. 5. Algorithm "CountEquations"

We use 117 iterations since for more iterations, the formal computation becomes heavy and very few new linear equations appear. The strategy of clocking $32+p$ times before doing the formal computation ensures that the location of differences are precisely known and that they will quickly produce exploitable equations. The number of equations we can obtain only depends on between which feedback taps of the LFSR the

Fault position in the LFSR state	Number of linear equations on the NFSR state
0 to 6	10.97
7 to 37	10.09
38 to 69	11.64
70 to 80	12.43
81 to 95	16.32
96 to 127	21.50

Fig. 6. Linear equations on the NFSR from a single fault

Fault positions in the LFSR state	number of faults needed to recover the whole NFSR state
0 to 6	8.29
7 to 37	7.48
38 to 69	6.75
70 to 80	6.09
81 to 95	4.53
96 to 127	4.40

Fig. 8. Consecutive faults for recovering the NFSR state

fault has occurred. It is notable that the majority of equations obtained involves only one variable, *i.e.,* directly gives the value of a bit of the NFSR without any linear algebra. The result obtained with this algorithm are depicted in Fig. 6. To sum up, the analysis from a single bit flipping in the LFSR state gives more than 10 bits of the NFSR state.

C. Number of equations from several faults

We now want to determine the number of keystream differences needed to recover the whole NFSR state. To figure this out, we have used Algorithm "RecoverNFSR" of Fig. 7 that recursively combined the information obtained from the keystream with the information obtained from each keystream difference.

1. For each keystream difference :
   a) collect in a system (S), linear and simple quadratic equations in the NFSR state at time t from the keystream difference as in Algorithm "CountEquations"
   b) append to (S) linear and simple quadratic equations in the NFSR state at time t obtained from the keystream
   c) compute a Groebner basis of (S) and solve the equations involving only one variable
   d) try to obtain new equations from the keystream and from the already used keystream differences thanks to the NFSR bits obtained in the previous step
2. Output (S)

Fig. 7. Algorithm "RecoverNFSR"

In our simulation, we use quadratic equations that involve at most 2 monomials. It experimentally ensures that the Groebner basis computation remains fast while providing more equations from a single fault analysis. The time t at which we recover the NFSR state depends on the type of fault used. For faults occurring in the same position p of the LFSR but at consecutive times, if starting with time i, then $t = i + p + 32$. The result obtained for this type of faults are summarized in Figure 8. Note that this algorithm can be executed in a couple of minutes.

For faults occurring on bits s_{i+p_0}, s_{i+p_1}, s_{i+p_2}, ... where $p_j - p_{j-1} > 1$, we use Algorithm "RecoverNFSR" to retrieve the NFSR state at time $t = i + p_0 + 32$. As the distance increases, the NFSR has to be formally clocked an increasing number of times before the fault enters the NFSR state. As a result, the degree of the equations obtained from the keystream differences increases because of the nonlinear feedback. The number of linear equations obtained from each new fault will be smaller than what we obtained with consecutive faults (see Figure 9). The worst situation (which occurs when the faults are at the beginning of the LFSR state) and the better one (when the faults are at the end of the state) are shown. To minimize the number of faults needed, we stop when the number of linear equations is greater than 97 and find the remaining bits with an exhaustive search. For example, the whole state can be recovered when the space between two faults is smaller than 5, but with a space of 30, exhaustive search can not be done in practice.

Space between faults	# of faults min/max	corresponding # of independent linear equations
2	4/6	108/99
5	3/9	103/102
10	3/12	106/79
20	5/8	104/59
30	4/5	45/79

Fig. 9. Numbers of equations from non consecutive faults

From the analysis of this phase, we see that the faults used in Phase 3 to recover the LFSR state are sufficient to recover the NFSR state with a couple of minutes of computation.

VIII. Attack Description: Phase 5

Given the internal state of Grain-128 at time t, $S_t = \{s_t, \ldots, s_{t+127}, b_t, \ldots, b_{t+127}\}$, we show that we can compute previous internal states. The computation of S_{t-1} consists in computing s_{t-1} and b_{t-1}. During initialization, the output of h is fed back and xored with the input both to the LFSR and to the NFSR. We first compute the value of the output function which is denoted by z_{t-1}:

$$\begin{aligned} z_{t-1} \quad = \quad & h(b_{t+11}, s_{t+7}, s_{t+12}, s_{t+19}, b_{t+94}, s_{t+41}, s_{t+59}, \\ & s_{t+78}, s_{t+94}) \oplus s_{t+92} \oplus b_{t+1} \oplus b_{t+14} \oplus b_{t+35} \\ & \oplus b_{t+44}, \oplus b_{t+63} \oplus b_{t+72} \oplus b_{t+88}. \end{aligned}$$

Next, the value of s_{t-1} and of b_{t-1} can be computed as follows:

$$s_{t-1} = z_{t-1} \oplus s_{t+127} \oplus s_{t+6} \oplus s_{t+37} \oplus s_{t+69} \oplus s_{t+80} \oplus s_{t+95},$$

and

$$\begin{aligned} b_{t-1} = {} & z_{t-1} \oplus s_{t-1} \oplus b_{t+127} \oplus b_{t+25} \oplus b_{t+55} \oplus b_{t+90} \\ & \oplus b_{t+95} \oplus b_{t+2}b_{t+66} \oplus b_{t+10}b_{t+12} \oplus b_{t+16}b_{t+17} \\ & \oplus b_{t+26}b_{t+58} \oplus b_{t+39}b_{t+47} \oplus b_{t+60}b_{t+64} \\ & \oplus b_{t+67}b_{t+83}. \end{aligned}$$

Knowing how to compute S_{t-1} from S_t, it is straightforward to compute S_{t-i} for any index $i \geq 0$ and thus to recover the secret key K.

IX. Countermeasures

Up to our knowledge, there is no fault attack that targets the NFSR. Thus, protecting Grain-128 against our fault attack amounts to protect the LFSR part only. This observation makes Grain-128 more suited to fault-tolerant hardware implementations than other stream ciphers which, in the general case, must be protected on their entire internal memory space.

A simple countermeasure consists in duplicating the LFSR in a mirror LFSR and to synchronously update it at each clock signal. A comparator checks that the contents of both LFSR's remain identical over time. In case of mismatch, the circuitry triggers a killing event, *e.g.,* the stream cipher halts, thereby preventing any form of observation by the attacker. One can also think of combining several mirror LFSR's together *e.g.,* the main LFSR is replaced with the XOR of 3 mirror LFSR's.

A more intricate countermeasure consists in adding linear redundancy in the LFSR's internal state. In this case, the update operation of the extended LFSR is not a feedback anymore but must be replaced with some more general linear transformation, and the number of faults that the device can detect is inherently limited by the detection capability of the underlying linear code.

X. Conclusion

We have proposed a fault attack on Grain-128. With an average number of 24 consecutive faults in the LFSR state, we can recover the secret key within a couple of minutes of off-line computation. We also propose some realistic countermeasures which protect Grain-128 at low extra cost.

References

[1] F. Armknecht and W. Meier. Fault Attacks on Combiners with Memory. In *SAC 2005*, volume 3897 of *LNCS*, pages 36–50. Springer, 2005.

[2] R.E. Atani, W. Meier, S. Mirzakuchaki, and S.E. Atani. Design and Implementation of DPA Resistive Grain-128 Stream Cipher Based on SABL Logic. *IJCCC*, III:100–110, 2008.

[3] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Crypto '97*, volume 1294 of *LNCS*, pages 513–525. Springer, 1997.

[4] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Eurocrypt '97*, volume 1233 of *LNCS*, pages 37–51. Springer, 1997.

[5] ECRYPT. eSTREAM: ECRYPT Stream Cipher Project. cf. http://www.ecrypt.eu.org/stream/.

[6] M. Hell, T. Johansson, A. Maximov, and W. Meier. A Stream Cipher Proposal: Grain-128. *IT, IEEE International Symposium on*, pages 1614–1618, 2006.

[7] M. Hell, T. Johansson, A. Maximov, and W. Meier. Grain - a stream cipher for constrained environments. *IJWMC, Spec. Iss. on Security of Computer Network and Mobile Systems*, 2006.

[8] J.J. Hoch and A. Shamir. Fault Analysis of Stream Ciphers. In *CHES 2004*, volume 3156 of *LNCS*, pages 240–253. Springer, 2004.

[9] Sergei P. Skorobogatov. Optically enhanced position-locked power analysis. In *CHES 2006*, volume 4249 of *LNCS*, pages 61–75. Springer, 2006.

[10] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In *CHES 2002*, volume 2523 of *LNCS*, pages 2–12. Springer, 2002.