Multi-Bit Differential Fault Analysis of Grain-128 with Very Weak Assumptions

Prakash Dey[1], Abhishek Chakraborty[2],
Avishek Adhikari[1] and Debdeep Mukhopadhyay[2]

[1] Department of Pure Mathematics
University of Calcutta, Kolkata-700019, India.
[2] Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur-721302, India.
pdprakashdey@gmail.com, abhishek.chakraborty@cse.iitkgp.ernet.in,
avishek.adh@gmail.com, debdeep.mukhopadhyay@gmail.com

Abstract. Very few differential fault attacks (DFA) were reported on *Grain-128* so far. In this paper we present a generic attack strategy that allows the adversary to challenge the cipher under different multi-bit fault models with faults at a targeted keystream generation round even if bit arrangement of the actual cipher device is unknown. Also unique identification of fault locations is not necessary. To the best of our knowledge, this paper assumes the weakest adversarial power ever considered in the open literature for DFA on *Grain-128* and develops the most realistic attack strategy so far on *Grain-128*. In particular, when a random area within $k \in \{1, 2, 3, 4, 5\}$ neighbourhood bits can only be disturbed by a single fault injection at the first keystream generation round (k-neighbourhood bit fault), without knowing the locations or the exact number of bits the injected fault has altered, our attack strategy always breaks the cipher with 5 faults. In a weaker setup even if bit arrangement of the cipher device is unknown, bad-faults (at the first keystream generation round) are rejected with probabilities 0.999993, 0.999979, 0.999963, 0.999946 and 0.999921 assuming that the adversary will use only 1, 2, 3, 4 and 5 neighbourhood bit faults respectively for *key-IV* recovery.

Keywords: Stream Cipher, Differential Fault Attack, Multi-Bit Fault, SAT Solver.

1 Introduction

Stream ciphers are generally fast algorithms to generate pseudo-random bits. They are used to encrypt data coming in a stream. Stream ciphers are best for the cases when the amount of data is either unknown, or continuous - such as network streams. *Grain-128* is a hardware based stream cipher and previous works (such as [1],[2], [3], [5], [6], [8]) show that it is vulnerable to DFA under various assumptions.

In the DFA model, faults are injected into the internal state of the cipher and from the difference of the normal and the faulty keystream, information about the internal state is partially or completely deduced. Faults can be injected in a register by under-powering and power spikes, clock glitches, temperature attacks, optical attacks, electromagnetic (EM) fault injection, etc. The first three methods do not require expensive equipment, but their effect can hardly be focused to a particular part of the device. On the other hand, optical and EM methods can affect a very restricted area, but require a more complex setup.

In the DFA reported on *Grain* family of ciphers so far the adversarial power has evolved from somewhat impractical and unrealistic to something that has some practical sense but with the hunch that *only single bit faults were considered in all of them* to recover the internal state of the cipher. [8] proposed a DFA on *Grain* family of ciphers with reasonable assumptions. The work also explored double and triple bit faults but rejected the cases in favour of single bit faults. Also attack strategy of [8] fails in case fault locations are not uniquely identified.

In this paper we present a generic attack strategy that allows the adversary to challenge *Grain-128* under different multi-bit fault models with faults at a targeted keystream generation round even if bit arrangement of the actual cipher device is unknown. Also unique identification of fault locations is not necessary. To the best of our knowledge, this paper assumes the weakest adversarial power ever considered in the open literature for DFA on *Grain-128* and develops the most realistic attack strategy so far on *Grain-128*.

In particular, when a random area within $k \in \{1, 2, 3, 4, 5\}$ neighbourhood bits can only be disturbed by a single fault injection at the first keystream generation round (k-neighbourhood bit fault), without knowing the locations or the exact number of bits the injected fault has altered, our attack strategy always breaks the cipher with 5 faults. In a weaker setup even if bit arrangement of the cipher device is unknown, bad-faults (at the first keystream generation round) are rejected with probabilities 0.999993, 0.999979, 0.999963, 0.999946 and 0.999921 assuming that the adversary will use only 1, 2, 3, 4 and 5 neighbourhood bit faults respectively for *key-IV* recovery.

2 Description of *Grain-128*

The *Grain-128* cipher has three stages namely, Key Loading Algorithm (KLA), Key Scheduling Algorithm (KSA) and Pseudo-Random keystream Generation Algorithm (PRGA). Secret *key* (128 bit) is loaded into the NFSR, 96 bit *IV* is loaded into the first 96 LFSR positions, the last 32 bits of the LFSR are filled with ones in the KLA. After the KLA, the KSA (256 clock) is used to pseudo-randomise the internal state without producing any keystream. Instead the output bit is fed back and XOR-ed with the input, both to the NFSR and to the LFSR. After the completion of the KSA, in the PRGA the output bit is no longer XOR-ed to the NFSR or to the LFSR but it is used as the Pseudo-Random keystream bit. During PRGA rounds, the NFSR and LFSR are updated normally. Full description of *Grain-128* is available in [4]. Here we present the version of Grain-128 with index starting from 1 instead of usual 0.

The internal state IS_i of Grain-128 consists of two feedback shift registers X (non-linear) and Y (linear) with inner states $X_i = (x_i, \ldots, x_{i+127})$ and $Y_i = (y_i, \ldots, y_{i+127})$ respectively at the beginning of the PRGA round i (≥ 1),

$$IS_i = (\underbrace{x_i, \ldots, x_{i+127}}_{X_i} \mid \underbrace{y_i, \ldots, y_{i+127}}_{Y_i}).$$

The secret *key* $(k_1, \ldots, k_{128})$ and *IV* $(IV_1, \ldots, IV_{96})$ are used to initialize the inner state as follws:

$$(\underbrace{k_1, \ldots, k_{128}}_{X} \mid \underbrace{IV_1, \ldots, IV_{96}, 1, \ldots, 1}_{Y}).$$

The keystream bit z_i and the new inner state bits x_{i+128}, y_{i+128} of Grain-128 registers X, Y are generated respectively as follows:

$z_i = h(X_i, Y_i)$, $x_{i+128} = u(X_i, Y_i)$, $y_{i+128} = v(Y_i)$ where,

$$h(X_i, Y_i) = x_{i+2} + x_{i+15} + x_{i+36} + x_{i+45} + x_{i+64} + x_{i+73} + x_{i+89} + y_{i+93} + x_{i+12}x_{i+95}y_{i+95} + x_{i+12}y_{i+8} + y_{i+13}y_{i+20} + x_{i+95}y_{i+42} + y_{i+60}y_{i+79},$$

$$u(X_i, Y_i) = y_i + x_i + x_{i+26} + x_{i+56} + x_{i+91} + x_{i+96} + x_{i+3}x_{i+67} + x_{i+11}x_{i+13} + x_{i+17}x_{i+18} + x_{i+27}x_{i+59} + x_{i+40}x_{i+48} + x_{i+61}x_{i+65} + x_{i+68}x_{i+84},$$

$$v(Y_i) = y_i + y_{i+7} + y_{i+38} + y_{i+70} + y_{i+81} + y_{i+96}.$$

3 Proposed Attack on *Grain-128*: Attack Model, Tools and Definitions

This paper assumes that the actual cipher device can be re-keyed with the same *key-IV* before each fault injection trial.

For any two integers a and b, with $a \leq b$, we denote the set $\{x : x \text{ is an integer with } a \leq x \leq b\}$ simply by $[a, b]$. Also if $V = (V_1, \ldots, V_p)$ is a vector of length p, we denote V_i simply by $V(i)$ for all $i \in [1, p]$. We shall also use the following notations:

a. For any integer i, $\emptyset + i = \emptyset$ ($\emptyset$ being the empty set).
b. For any set S of integers and for any integer i, $S + i = \{s + i : s \in S\}$.
c. For any set S if $s \in S$ implies that s is a set of integers then for any integer i, $S + i = \{s + i : s \in S\}$.

1. Fault Location. If a fault injection trial flips exactly the bits at the r distinct positions given by $\phi = \{\phi_1, \ldots, \phi_r\}$ of the internal state, only at the PRGA round t, then the set ϕ will be called a *fault position* and the ordered pair (ϕ, t) will be called a *fault location* or simply a *fault.*

Remark. A fault model is a set of properties of an injected fault that is used to characterize an attack. In this paper we consider faults at a single PRGA round. Let Γ be the set of all possible fault positions corresponding to a fault model Σ. We shall represent Σ simply by Γ.

2. The XOR Differential Keystream. Let $IS_i^{key,IV}$ be the internal state of the cipher at PRGA round i $(i \geq 1)$. Let us consider a fault (ϕ, t). Let $IS_i^{key,IV,\phi,t}$ be the faulty internal state and let $z_i^{key,IV,\phi,t}$ be the faulty output key bit at that PRGA round i. Then $d_i^{key,IV,\phi,t} = z_i^{key,IV,\phi,t} + z_i^{key,IV}$ is the XOR difference of the normal (fault free) keystream bit $z_i^{key,IV}$ from the faulty one $z_i^{key,IV,\phi,t}$ at the PRGA round i. For given n we denote, $d^{key,IV,\phi,t,n} = (d_1^{key,IV,\phi,t}, \dots, d_n^{key,IV,\phi,t})$. We shall analyse the XOR differential keystream $d^{key,IV,\phi,t,n}$ for DFA.

Remark. We shall drop the *key, IV* superscript when there is no need to emphasise them. Following the simplified notation, $d^{key,IV,\phi,t,n}$ becomes $d^{\phi,t,n} = (d_1^{\phi,t}, \dots, d_n^{\phi,t})$. One should note that each $d_i^{\phi,t}$ may be thought of as a function of the *Key-IV* pair. Since the cipher device is re-keyed before each fault injection, after the fault injection, at the fault injection PRGA round t we have, $IS_t^{\phi,t}(e) = IS_t(e) + 1, \ \forall e \in \phi$ and $IS_t^{\phi,t}(e) = IS_t(e), \ \forall e \in [1, 256] \setminus \phi$.

Since we are considering the XOR differential keystream for finitely many PRGA rounds, it may happen that XOR differential keystreams corresponding to two fault locations match exactly with each other.

3. Signature of Fault Locations. After a fault (ϕ, t) is injected in the PRGA round t, *we shall study the l PRGA rounds $t, \dots, t+l-1$*. We consider the XOR differential keystream $d^{\phi,t,n} = (d_1^{\phi,t}, \dots, d_n^{\phi,t})$ where $n = t + l - 1$. It should be noted that $d_1^{\phi,t} = \dots = d_{t-1}^{\phi,t} = 0$ as the fault is injected at the PRGA round t. We now treat *key-IV* as variables and each $d_i^{\phi,t}$ as a function of the *Key-IV* pair.

For certain (ϕ, t) there may be some special values in $d^{\phi,t,n}$ such that

(A) $d_i^{\phi,t} = b$, $b \in \{0, 1\}$ irrespective of *Key-IV*.

(B) $d_i^{\phi,t}, d_j^{\phi,t} \notin \{0, 1\}, i \neq j$, but $d_i^{\phi,t} = d_j^{\phi,t}$ happens deterministically irrespective of *Key-IV*.

(C) $d_i^{\phi,t}, d_j^{\phi,t} \notin \{0, 1\}, i \neq j$, but $d_i^{\phi,t} = d_j^{\phi,t} + 1$ happens deterministically irrespective of *Key-IV*.

For each fault location (ϕ, t) the signature [8], $sig_{\phi,t}$ of the fault (ϕ, t) is defined to be the 4-tuple $sig_{\phi,t} = (sig_{\phi,t}^1, \ sig_{\phi,t}^0, \ sig_{\phi,t}^=, \ sig_{\phi,t}^{\neq})$ as explained below, where each of $sig_{\phi,t}^e, \ e \in \{1, 0, =, \neq\}$ will be called a component of $sig_{\phi,t}$.

We define,

$sig_{\phi,t}^b = \{i \in [t, n] : d_i^{\phi,t} = b\}, \ \ b \in \{0, 1\}$.

$sig_{\phi,t}^= = \{\{i_1, \dots, i_p\} : i_1, \dots, i_p \in [t, n], \ d_{i_1}^{\phi,t} = \dots = d_{i_p}^{\phi,t} \notin \{0, 1\}$ and $\exists$ no $i_r \in [t, n]$ other than $i_1, \dots, i_p$ such that $d_{i_r}^{\phi,t} = d_{i_1}^{\phi,t}\}$.

$sig_{\phi,t}^{\neq} = \{\{i, j\} : i, j \in [t, n], d_i^{\phi,t} + d_j^{\phi,t} = 1$ and $d_i^{\phi,t}, d_j^{\phi,t} \notin \{0, 1\}\}$.

If $d_i^{\phi,t} = b, \ b \in \{0, 1\}$ holds irrespective of *Key-IV*, we shall say that: "all the XOR differential keystreams are fixed to b at the position i under the fault (ϕ, t)". One should note that, **(A)** $sig_{\phi,t}^b$ is the set of the positions (PRGA rounds) where the XOR differential keystreams are fixed to b under the fault (ϕ, t), **(B)** $sig_{\phi,t}^=$ gives sets of PRGA rounds where the XOR differential keystreams are deterministically equal (but not fixed) irrespective of *Key-IV* under the fault (ϕ, t) and **(C)** $sig_{\phi,t}^{\neq}$ gives pairs of PRGA rounds where the XOR differential keystreams are deterministically different (but not fixed) irrespective of *Key-IV* under the fault (ϕ, t).

Remark. The signatures are *Key-IV* independent and depend only on fault locations and the cipher design. Since signature of faults are constructed for finitely many PRGA rounds it may happen that (1) for some fault locations some signature component becomes completely void and (2) signatures of two fault locations match exactly with each other.

4. We now use the following notation:
For any integer i, $sig_{\phi,t} + i = (sig_{\phi,t}^1 + i, sig_{\phi,t}^0 + i, sig_{\phi,t}^= + i, sig_{\phi,t}^{\neq} + i)$.
The next theorem shows that *"for a fixed fault position ϕ we do not need to compute signatures for all fault locations (ϕ, t)"*.

Theorem 1. *For any fault location (ϕ, t), $\ sig_{\phi,t} = sig_{\phi,1} + t - 1$.*

Proof. Let fault be injected at the PRGA round 1 at position ϕ and $(s_1, s_2, \dots, s_{256})$ be the corresponding internal state (IS) where each of s_j is a variable.

We now assume that the XOR differential keystream bit (XOR DKB) $d_i^{\phi,1} = 1$. Then this happens (at the PRGA round i) independent of the internal state (IS) at the PRGA round 1.

PRGA round	Fault Free Internal State	XOR DKB	PRGA round	Fault Free Internal State	XOR DKB
1	IS at PRGA round 1	$d_1^{\phi,1}$	t	IS at PRGA round t	$d_t^{\phi,t}$
2	IS at PRGA round 2	$d_2^{\phi,1}$	$t+1$	IS at PRGA round $t+1$	$d_{t+1}^{\phi,t}$
3	IS at PRGA round 3	$d_3^{\phi,1}$	$t+2$	IS at PRGA round $t+2$	$d_{t+2}^{\phi,t}$
i	IS at PRGA round i	$d_i^{\phi,1}=1$	$t+i-1$	IS at PRGA round $t+i-1$	$d_{t+i-1}^{\phi,t}=1$

Thus $d_i^{\phi,1}=1 \Leftrightarrow d_{t+i-1}^{\phi,t}=1$ and hence $i \in sig_{\phi,1}^1 \Leftrightarrow t+i-1 \in sig_{\phi,t}^1$.
This shows that $sig_{\phi,t}^1 = sig_{\phi,1}^1 + t - 1$.
With similar arguments the theorem follows.

Remark. (1) In consequence of the above theorem it can be said that if fault is injected in the same position then a pattern is generated from the fault injection PRGA round, in the XOR differential keystream, all previous keystream bits being 0's. **(2)** For any fault (ϕ, t) if $t = 1$, we shall drop the subscript 't' form its signature and signature components. Thus with this simplified notation $sig_{\phi,1} = (sig_{\phi,1}^1, sig_{\phi,1}^0, sig_{\phi,1}^=, sig_{\phi,1}^{\neq})$ becomes $sig_\phi = (sig_\phi^1, sig_\phi^0, sig_\phi^=, sig_\phi^{\neq})$ and in this case sig_ϕ will be called the signature of the fault position ϕ.

We now present methods for computing the signature components of sig_ϕ.

4 Signature Generation

Let us consider a fault position ϕ. We generate sig_ϕ^1, sig_ϕ^0 using the probabilistic algorithm *GenSig*10 (Generate Signature One and Zero) and $sig_\phi^=$, $sig_\phi^{\neq}$ using the deterministic algorithm *GenSigSym* as described below. This increases the fault identification and bad-fault rejection success probabilities. The deterministic algorithm *GenSigSym* has implementation limitations. However, the probabilistic algorithm *GenSig*10 is much more efficient.

```
Algorithm 1. GenSig10($\phi$, $\Omega$, Number of PRGA round = $L_1$)
1. For $\Omega$ number of distinct uniformly random independent Key-IV pair :
2.       Generate XOR differential keystreams upto round $L_1$ under the fault $(\phi, 1)$
3. Find positions (PRGA rounds) at which all the generated XOR differential
   keystreams are fixed
4. If $b \in \{0, 1\}$ is at a fixed position : then add the position to $sig_\phi^b$
5. return : $sig_\phi^1$, $sig_\phi^0$.
```

```
Algorithm 2. GenSigSym($\phi$, Number of PRGA round = $L_2$)
1. Define 256 symbolic variables over GF(2) and initialise the inner state with
   this symbolic variables. This will represent the inner state at the beginning
   of the PRGA round 1.
2. Compute symbolically the XOR differential keystream $d^{\phi,1,L_2}$.
3. Observe the XOR differential keystream and compute $sig_\phi^=$ and $sig_\phi^{\neq}$.
```

Let i be an actual fixed position (PRGA round) for $b \in \{0, 1\}$ under the fault $(\phi, 1)$. Then the algorithm *GenSig*10 will surely append i to sig_ϕ^b. But if i is not a fixed position, then Pr(The algorithm appends i to sig_ϕ^b) $= 1/2^\Omega$ provided we assume that 0 and 1 are equally probable at the position i (since i is not a fixed position) and the XOR differential bit generated at the position i for each *Key-IV* pair are independent (*Key-IV* pairs are distinct uniformly random and independent). Therefore taking large value of Ω it can be guaranteed that the algorithm generates correct signatures with very small failure probability. e.g., simply taking $\Omega = 1000$ we have $1/2^\Omega = 10^{-\Omega log_{10}2} \approx 10^{-301}$ which is practically negligible.

Remark. The algorithm *GenSig*10 is generic in nature and is capable of coping with any computationally feasible fault model Γ. It taps statistical weakness of the cipher under DFA.

5 Fault Location Determination

Let Γ be a fault model and the adversary has computed sig_ϕ for all $\phi \in \Gamma$. Adversary is fully confident that any injected fault (position) will be in Γ.

In this stage the adversary actually injects a fault into the cipher device and compares the XOR differential keystream with pre-computed signatures of all possible faults in order to identify the fault location.
We define, $n_1 = max(\bigcup_{\phi\in\Gamma}(sig^1_\phi \bigcup sig^0_\phi))$, $n_2 = max(\bigcup_{\phi\in\Gamma}(\bigcup_{A\in sig^=_\phi \bigcup sig^{\neq}_\phi} A))$ and $end(\Gamma) = max(n_1, n_2)$.
1. Obtain the fault-free keystream. In this stage we need a XOR differential keystream of length $n = end(\Gamma) + T - 1$ in order to match it with all possible pre-computed signatures, if fault is injected at the known PRGA round T.
2. Let a fault be injected at an unknown position ψ at the known PRGA round T. Compute the faulty keystream and obtain $d^{\psi,T,n} = (d^{\psi,T}_1, \dots, d^{\psi,T}_n)$.
3. Define, $support^b = \{i \in [1,n] : d^{\psi,T}_i = b\}$, $\forall\ b \in \{0,1\}$.
4. $pf = allPossibleFaults\_known(\Gamma, T,\ d^{\psi,T,n})$ as described in Algorithm 3.

```
Algorithm 3. $allPossibleFaults\_known(\Gamma, t,\ d^{\psi,t,n})$
1. $pf = \emptyset$
2. for all $\phi \in \Gamma$ :
3.     if $isaPossibleFault(\phi, t, d^{\psi,t,n}) == True$ : // Algorithm 4
4.         $pf = pf \bigcup \{(\phi, t)\}$
5. return $pf$
```

```
Algorithm 4. $isaPossibleFault(\phi, t,\ d^{\psi,t,n})$
output : "True" if $(\phi, t)$ is a possible fault location else "False".
1. $sig_{\phi,t} = sig_{\phi,1} + t - 1$
2. if $sig^1_{\phi,t} \subseteq support^1$ :
3.     if $sig^0_{\phi,t} \subseteq support^0$ :
4.         if $sig^=_{\phi,t} == \emptyset$ :
5.             if $sig^{\neq}_{\phi,t} == \emptyset$ :
6.                 return $True$
7.             else :
8.                 if $\forall \{i,j\} \in sig^{\neq}_{\phi,t} \Rightarrow d^{\psi,t}_i + d^{\psi,t}_j = 1$ :
9.                     return $True$
10.                else : return $False$
11.        else :
12.            if $\forall A \in sig^=_{\phi,t}$ and $\forall i,j \in A, i < j \Rightarrow d^{\psi,t}_i = d^{\psi,t}_j$ :
13.                if $sig^{\neq}_{\phi,t} == \emptyset$ :
14.                    return $True$
15.                else :
16.                    if $\forall \{i,j\} \in sig^{\neq}_{\phi,t} \Rightarrow d^{\psi,t}_i + d^{\psi,t}_j = 1$ :
17.                        return $True$
18.                    else : return $False$
19.            else : return $False$
20.    else : return $False$
21. else : return $False$
```

The basic idea is to check whether the pre-computed pattern (signature) of a fault location occurs in the XOR differential keystream. If the pattern due to a fault location (ϕ, t) occurs in the XOR differential keystream, then it is a possible fault location, otherwise we reject it. It should be noted that from the construction it immediately follows that the actual fault location $(\psi, T) \in pf$. Now if pf is singleton then, (ψ, T) is uniquely determined. When pf is not singleton we do not need to reject the case as a failure. We shall address the issue in the next section.

6 Recovering the Internal State of the Cipher

The adversary wishes to recover the internal state of the cipher at the PGRA round t. In the online phase, the adversary inject faults to the cipher device at PRGA round t, re-keying each time, for m times. Let Q_j be the set of possible faults returned by the fault location determination algorithm at the j-th fault injection trial, $z^j = (z^j_1, \dots, z^j_n)$ being the corresponding faulty keystream of length n $(> t)$, $\forall j \in [1,m]$. We now consider the Cartesian product set $Q = Q_1 \times \cdots \times Q_m$. Then one of the elements, say α of Q corresponds to the actual m injected faults.
The adversary starts with the following information:

1. m fault locations, given by $\beta = ((\phi_1, t), \ldots, (\phi_m, t)) \in Q$.
2. normal (fault free) keystream $z = (z_1, \ldots, z_n)$ of length n.
3. m faulty keystreams $z^1, \ldots, z^m$ each of length n.

The adversary either surely knows or guess that the faulty keystream z^j occurred due to the fault (ϕ_j, t) based on the cardinality of Q_j, $j \in [1, m]$.

6.1 Generating Polynomial Equations

We use procedure similar to [8] in order to obtain a system of polynomial equations, modifying the *fault injection strategy* in order to cope with multi-bit faults at any targeted round.

Let the fault free internal state at the PRGA round i $(\geq t)$ be $IS_i = (X_i, Y_i)$ where $X_i = (x_i, \ldots, x_{i+127})$ and $Y_i = (y_i, \ldots, y_{i+127})$, the internal state at PRGA round t being $IS_t = (x_t, \ldots, x_{t+127}, y_t, \ldots, y_{t+127})$. We treat each x_i and y_i as variables and consider the PRGA rounds $t, \ldots, n$. Corresponding to each key-stream bit z_i, we introduce two new variables x_{i+128}, y_{i+128} $(i \geq t)$ and obtain the following three equations: $z_i = h(X_i, Y_i)$, $x_{i+128} = u(X_i, Y_i)$, $y_{i+128} = v(Y_i)$. Thus we have in total $2N + 256$ variables and $3N$ equations, where $N = n - t + 1$.

Let us now consider the fault (ϕ_j, t). Since the cipher device is re-keyed before each fault injection, after the fault injection, if the faulty internal state at PRGA round i be IS_i^j then at the targeted fault injection PRGA round t we have, $IS_t^j(e) = IS_t(e) + 1$, $\forall e \in \phi_j$ and $IS_t^j(e) = IS_t(e)$, $\forall e \in [1, 256] \setminus \phi_j$. Again corresponding to each key-stream bit z_i, we introduce two new variables x_{i+128}^j, y_{i+128}^j $(i \geq t)$ and obtain three more equations. In this case we have additional $2N$ variables and $3N$ equations.

Thus if we consider m faults, after these many re-keyings, the total number of variables is $2(m+1)N+256$ and the total number of equations is $3(m+1)N$.

Now the system of polynomial equations are simply passed on to the SAT solver in sage for extracting solution for the variables $x_t, \ldots, x_{t+127}, y_t, \ldots, y_{t+127}$ and to mean this we shall use the phrase that "$(\beta, t, n, z, z^1, \ldots, z^m)$ *are passed on to the SAT solver*".

6.2 Recovering the Internal State with SAT Solver

Now we pass $(\beta, t, n, z, z^1, \ldots, z^m)$ to the SAT solver, for each element β of Q. Multiple solutions may be obtained. Solution from the element α, if returned, will surely correspond to the actual internal state of the cipher at the PRGA round t. Assuming each returned solution as a possible internal state at the PRGA round t, we simply use "Guess and Determine Strategy" [7] to detect the correct internal state. If we have a match, the internal state together with the actual fault locations (not needed any more) will be recovered.

If the cardinality of Q and SAT solving time for m faults are low then the internal state can be recovered in reasonable time with 100% success. Otherwise we have to re-key the cipher device for more fault injection trials.

Remark. (1) During this phase a limit should be placed on the running time of the SAT solver. **(2)** When β does not correspond to the actual m injected fault, the polynomial equations may be unsatisfiable and in this case the SAT solver may terminate quickly throwing an error message.

7 Alien Fault Model

Authors of [8] showed that if the adversary is confident that an injected fault is atmost a triple bit fault and if she restrict herself to single bit faults only for internal state recovery then with high probability she can detect and reject exactly the cases of multiple bit (double and triple bit) faults.

In this paper, we generalise this notion. If Γ is a fault model such that the adversary is capable of computing signatures for all fault positions in Γ then, Γ will be called a *native fault model* and faults corresponding to the native fault model Γ will be called *native faults*.

Let us consider a fault model Σ. Adversarial power guarantees that any injected fault will be in Σ but Σ is too large and the adversary simply cannot compute the signatures for all fault positions in Σ. In this case she chooses a computationally feasible proper subset Γ of Σ and computes signatures for all fault positions in Γ. Since signatures corresponding to the fault positions in $\Sigma \setminus \Gamma$ are not computed, corresponding faults if occur simply cannot be identified. But the Adversary wants to detect such a case and reject the faulty keystream. The faults in $\Sigma \setminus \Gamma$ will be called 'alien faults' (bad-faults) and Σ will be called an ‘alien fault model’ with respect to Γ. If Σ is the set of all possible fault positions (due to faults affecting only the single targeted PRGA round), then in particular the alien faults in $\Sigma \setminus \Gamma$ will be called ‘absolute alien faults’. If Γ is such that, **(1)** native faults in Γ can be identified with very high probability and **(2)** absolute alien faults can be detected (and hence can be rejected) with very high probability, then attack is possible on a cipher device with completely unknown bit-arrangement.

The adversary may use a number of strategy in choosing Γ. She may choose the fault model Γ depending on her knowledge or best guess on the cipher device. She may choose those fault positions which are more likely to occur by experimenting on similar cipher device if she has that privilege. If that is not possible she chooses more relaxed absolute alien fault model where any possible fault can occur but she will only use the *native faults* to recover the internal state of the cipher, rejecting *alien faults* with some probability. Success in a chosen native fault model is not guaranteed but is purely experiment based. One should note that different fault models might crucially affect the capabilities and the complexities of the attack strategy. The adversary is now very weak and will require more re-keying of the cipher device. Depending on the cipher device there may be some fault models that produce the best or the worst results.

Fault Location Determination in Alien Fault Model
The adversary will use the same fault location determination algorithm to identify or to detect a fault. When considering an alien fault model, error will occur if the injected fault is alien but the fault location determination algorithm identifies it as a native fault. Thus error occurs if possible faults pf returned by the fault location determination algorithm is non-empty when injected fault is alien and $pf = \emptyset$ will surely imply that an alien fault is injected.

Recovering Internal State in This Case
Internal state recovery process in this case will be the same as described in Section 6. Depending on the chosen native faults, since the fault location determination algorithm is probabilistic, it may happen that the fault location determination algorithm returns an erroneous fault (fault was alien but wrongfully identified as native). In this case the adversary may have to use the “Guess and Determine” strategy to recover the internal state of the cipher device by matching fault free keystream with the original one and re-keying the cipher device again for more fault injection trials if necessary.

8 Experimental Results

Attack strategy in this paper is generic. In particular we demonstrate the attack strategy for the scenario, denoted by the symbol nbdMBF^k, in which randomly chosen atmost k consecutive location (k-neighbourhood bit fault) can be disturbed by a single fault injection without knowing the locations or the exact number of bits the injected fault has altered. In this paper we consider the popular convention of treating $IS(e), IS(e+1), IS(e+2), \ldots, IS(e+p-1)$ as p neighbouring bits (IS representing the internal state of the cipher) but in real life the arrangement may not follow this pattern. However this does not affect our analysis.

For each fault position ϕ we used the algorithm $GenSig10(\phi, \Omega, L_1)$ to generate sig^1_ϕ and sig^0_ϕ by taking $L_1 = 1000$, $\Omega = 1000$ while $GenSigSym(\phi, L_2)$ was used to generate $sig^=_\phi$ and $sig^{\neq}_\phi$ by taking $L_2 = 150$. Here we present experimental results for (comparing) $k = 1, 2, 3, 4, 5$. The case for $k = 1$ results in the same native fault model considered in [8].

Abbreviations: (POS, Probability of Success), (NOE, Number of Experiments), (NOT, Number of Timed out cases).

Our Arsenal
1. One standalone desktop PC with AMD 4.0 GHz FX-8350 processor and 32 GB RAM, referred to as AMD.
2. A Beowulf Cluster of 20 desktop PC each with 2.60 GHz Intel Pentium E5300 Dual-core processor and 4 GB RAM connected via LAN and setupped for Distributed Parallel Computing, referred to as BEOWULF.
3. Ubuntu 12.04 LTS Operating System.
4. Mathematical software sage-6.1.1.
5. SAT solver Cryptominisat-2.9.6 installed in sage.
The BEOWULF cluster (all available cores) was used to (1) generate sig^1_ϕ and sig^0_ϕ, (2) compute the success rates (probabilities) and (3) SAT solving where as the standalone AMD (only a single core) was

used to generate $sig_{\phi}^{=}$ and $sig_{\phi}^{\neq}$.

All the experiments were performed assuming PRGA round 1 as the targeted round.

Probability of Identifying a Random Native Fault Location

For each column of the following table, we considered a set of 2^{20} experiments.

Grain-128					
k	1	2	3	4	5
POS	1.0	1.0	0.997974	0.958305	0.927482
Avg Fault	1.0	1.0	1.000051	1.040921	1.107610

Explanation: For $k = 5$, POS (Avg Fault) = 0.927482 (1.107610) means that the fault location determination algorithm had uniquely identified actual fault locations with a probability of 0.927482 and average number of faults returned by the fault location determination algorithm in 2^{20} experiments is 1.107610 which is very low. For $k = 3, 4$ the average number of faults returned by the fault location determination algorithm are even lower. For $k = 1, 2$ the success rates are 100% in uniquely identifying actual fault locations.

SAT Solving: Results

Behaviour of SAT solvers and the time to return a solution could hardly be predicted. For each SAT solving trial (with cutting number 4) we first generated an inner state (by choosing key-IV randomly) and m random faults uniformly and independently. For each SAT solving trial (compiled codes were not used) we allocated a time limit of 4 hours. If the SAT solver does not self terminate within that allocated time we agree to terminate it forcefully and mark the case as a TIMEOUT (which may have resulted in a success if enough time was permitted). Very few such case occurred for $m = 4$ during experimentation while the case for $m = 5$ always resulted in a success. For each row of the following table, we performed SAT experiments independently.

Grain-128					Time in Seconds if SUCCESS		
k	m	N	NOE	NOT	MinTime	MaxTime	AvgTime
1	4	256	160	5	9.80	12301.04	650.77
2	4	256	160	4	9.38	10803.83	811.91
3	4	256	160	7	10.40	14071.91	426.38
4	4	256	160	6	10.25	14378.43	679.92
5	4	256	160	8	8.88	10294.13	419.93
1	5	256	1500	0	5.18	1923.07	52.61
2	5	256	1500	0	4.86	3110.49	52.76
3	5	256	1500	0	6.06	1909.46	52.75
4	5	256	1500	0	6.38	1383.41	52.90
5	5	256	1500	0	3.66	2250.79	52.52

Probability of Rejecting Absolute Alien Faults

For each column of the following table, we considered a set of 2^{20} experiments.

Grain-128					
k	1	2	3	4	5
POS	0.999993	0.999979	0.999963	0.999946	0.999921

Thus absolute alien faults are rejected with very high probability. Experimental results show that for these cases it is not required to know the actual bit arrangement of the cipher device since native faults are identified with very high probability.

9 Conclusion

This paper considers multi-bit faults for *key-IV* recovery. Attack strategy in this paper is generic and allows the adversary to challange the cipher under different fault models with faults at a targeted PRGA round. For k-neighbourhood bit fault ($k \in \{1, 2, 3, 4, 5\}$) at the PRGA round 1, not more than 5 faults always breaks the cipher. Also the bit arrangement of the cipher device may be unknown. The algorithm $GenSig10$ is slower than D-GRAIN [8] but it can cope with any multi-bit fault. Thus allowing the adversary to challenge any multi-bit fault model. The validity of the randomised algorithm $GenSig10$ is further justified by the experiments when the probabilities were calculated. In all such experiments the fault detection algorithm never rejected an injected native fault.

References

1. Banik, S., Maitra, S., and Sarkar, S. A differential fault attack on the grain family of stream ciphers. In *Cryptographic Hardware and Embedded Systems–CHES 2012*. Springer, 2012, pp. 122–139.
2. Banik, S., Maitra, S., and Sarkar, S. A differential fault attack on the grain family under reasonable assumptions. In *Progress in Cryptology-INDOCRYPT 2012*. Springer, 2012, pp. 191–208.
3. Berzati, A., Canovas, C., Castagnos, G., Debraize, B., Goubin, L., Gouget, A., Paillier, P., and Salgado, S. Fault analysis of grain-128. In *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on* (2009), IEEE, pp. 7–14.
4. Hell, M., Johansson, T., Maximov, A., and Meier, W. A stream cipher proposal: Grain-128. `http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain128_p3.pdf`.
5. Karmakar, S., and Chowdhury, D. R. Fault analysis of grain-128 by targeting nfsr. In *Progress in Cryptology–AFRICACRYPT 2011*. Springer, 2011, pp. 298–315.
6. Karmakar, S., and Chowdhury, D. R. Fault analysis of grain family of stream ciphers. *IACR Cryptology ePrint Archive 2014* (2014), 261.
7. Rohani, N., Noferesti, Z., Mohajeri, J., and Aref, M. R. Guess and determine attack on trivium family. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on* (2010), IEEE, pp. 785–790.
8. Sarkar, S., Banik, S., and Maitra, S. Differential fault attack against grain family with very few faults and minimal assumptions. *IACR Cryptology ePrint Archive 2013* (2013), 494.