Improved Practical Differential Fault Analysis of Grain-128

Prakash Dey[1], Abhishek Chakraborty[2], Avishek Adhikari[3] and Debdeep Mukhopadhyay[4]
[1, 3]*Department of Pure Mathematics, University of Calcutta, Kolkata-700019, India.*
[2, 4]*Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur-721302, India.*
Email: pdprakashdey@gmail.com, abhishek.chakraborty@cse.iitkgp.ernet.in,
avishek.adh@gmail.com, debdeep.mukhopadhyay@gmail.com

***Abstract*— Differential Fault Attacks (DFA) on stream ciphers have been an active field of research. However, their practical realizations have not been reported in the public literature. Hence, the assumptions on the fault models made in the context of DFA for stream ciphers have not been studied. Furthermore, there have been few efforts reported on the popular stream cipher candidate, Grain-128. We consider a simple low-cost fault injection set-up, using clock glitches and show that in stream ciphers the critical path of the circuit affects few bit positions (the feedback bit for the Shift Registers in the stream ciphers). Thus the fault is often localized to single bit position, and because of the absence of required faulty ciphers makes existing theoretical DFAs invalid. In order to create multiple instance of faults, we use clock glitches to induce the fault, and then use the shifting property of the internal registers of Grain to create multiple instances of contiguously located faults. In parallel, we also develop a more relaxed DFA for Grain-128, to show that when the fault is k neighbourhood bits, $k \in \{1, .., 5\}$, the attack is successful to retrieve the key without knowing the locations or exact number of bits flipped by the internal fault. We also devise a technique for rejecting the bad faults with high probabilities, i.e., when the faults are not in the contiguous location as required in the attack. Combining the above attacks we demonstrate using a simple set-up via clock glitches that such faults can be practically obtained and analysed using the proposed attack algorithm to retrieve the key.**

I. Introduction

Stream Ciphers are fast ciphering algorithms to generate pseudrandom bits. Like conventional cryptosystems, these ciphers have also been subjected to the powerful Differential Fault Analysis ([1], [2], [3], [4], [5]). Faults can be injected in a register, which stores data or state in a hardware implementation using several techniques: ranging from low cost methods like power spiked, clock glitches to costly methods like optical injections via lasers, etc. Although a very powerful attack model, the practicality of a DFA depends on the cost of the set-up and also on the type of faults which actually occur in a practical set-up.

In this context, this paper addresses the fact that although several research work on DFA on stream ciphers have been performed, they have not been supported with real life experiments, as performed on other category of ciphers, like the Advanced Encryption Standard (AES). This lack of support with real experiments, have resulted in the absence of information on which fault models are practical and occurs in real life. In this paper, we consider the eSTREAM finalist name Grain-128, and consider a low-cost fault injection technique using clock glitches to develop a practical DFA on a standard stream cipher. We observe several interesting observations, namely the faults because of the conventional clock glitches are only single-bit in nature and also occur in the feedback path of the Feedback Shift Registers in the stream cipher, as they lie in the critical path of the circuit. This shows that the existing DFAs on Grain-128 will not be feasible, as they require at least 3 faults, which are so rare that they are not distinct. This motivates us to modify the fault injection technique, combining the effect of the clock glitch and also the shift in the registers to propagate the faults to multiple locations in contiguous positions. In parallel, we also revisit the DFA algorithm on Grain-128, relaxing the fault model to faults which are k-consecutive, ie. occur in k consecutive locations. We develop a method using SAT solvers and show that the solver is able to determine the fault locations, and subsequently the key when $k \in \{1, ..., 5\}$. Our attack is made more practical because we develop a rejection method, so that we are able to determine those cases where the faults are not of the above category and discard them. Finally, we demonstrate using the faults which occur in an actual experiment using the clock glitch based methods, and our solution method how the attack is performed in a real life setting. To the best of our knowledge, this is the first reported work on a real-life practical DFA on a standard stream cipher, namely Grain-128.

The paper is organized as follows: In Section II, we briefly describe the cipher. Then in Section III we describe the attack model, tools and definitions. After that in Section IV and V we discuss the fault location determination procedure. Section VI deals with key-IV recovery procedure. In Section VII we describe our experimental results. Section VIII concludes the paper.

II. A Brief Description of *Grain-128*

Full description of *Grain-128* is available in [6]. 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 $Y_i = (y_i, \ldots, y_{i+127})$

[2, 4]*The authors would like to thank Secured Embedded Architecture Laboratory, Indian Institute of Technology Kharagpur.*
[3]*The author is thankful to the Centre of Excellence in Cryptology of Indian Statistical Institute.*

respectively at the beginning of the Pseudo-Random keystream Generation Algorithm (PRGA) round i (≥ 1),

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

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

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

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,

$h(X_i, Y_i) = x_{i+2} + x_{i+15} + x_{i+36} + x_{i+45} + x_{i+64} + x_{i+73} + x_{i+89} + y_{i+93} + x_{i+12}x_{i+95}y_{i+95} + x_{i+12}y_{i+8} + y_{i+13}y_{i+20} + x_{i+95}y_{i+42} + y_{i+60}y_{i+79}$,
$u(X_i, Y_i) = y_i + x_i + x_{i+26} + x_{i+56} + x_{i+91} + x_{i+96} + x_{i+3}x_{i+67} + x_{i+11}x_{i+13} + x_{i+17}x_{i+18} + x_{i+27}x_{i+59} + x_{i+40}x_{i+48} + x_{i+61}x_{i+65} + x_{i+68}x_{i+84}$,
$v(Y_i) = y_i + y_{i+7} + y_{i+38} + y_{i+70} + y_{i+81} + y_{i+96}$.

III. Proposed Theoretical Attack: 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. In this paper we consider faults at a single PRGA round or multi-round faults equivalent to a fault 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 be the internal state of the cipher at PRGA round i ($i \geq 1$). Let us consider a fault (ϕ, t). Let $IS_i^{\phi,t}$ be the faulty internal state and let $z_i^{\phi,t}$ be the faulty output key bit at that PRGA round i. Then $d_i^{\phi,t} = z_i^{\phi,t} + z_i$ is the XOR difference of the normal (fault free) keystream bit z_i from the faulty one $z_i^{\phi,t}$ at the PRGA round i. For given n we denote, $d^{\phi,t,n} = (d_1^{\phi,t}, \ldots, d_n^{\phi,t})$.

Remark. 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 $IS_t^{\phi,t}(e) = IS_t(e),\ \forall e \in [1, 256] \setminus \phi$.

3. Signature of Fault Locations. After a fault (ϕ, t) is injected in the PRGA round t, *we shall study the l PRGA rounds* $t, \ldots, t + l - 1$. We consider the XOR differential keystream $d^{\phi,t,n} = (d_1^{\phi,t}, \ldots, d_n^{\phi,t})$ where $n = t + l - 1$. It should be noted that $d_1^{\phi,t} = \cdots = d_{t-1}^{\phi,t} = 0$ as the fault is injected at the PRGA round t. We now treat *key-IV* as variables and each $d_i^{\phi,t} : \text{GF(2)}^{128+96} \to \text{GF(2)}$ as a function of the *Key (128 bit) - IV (96 bit)* pair. For simplicity we write $d_i^{\phi,t} \notin \{0, 1\}$, to mean that $d_i^{\phi,t}$ is not a constant function.

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

We define,

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

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

$sig^{\neq}_{\phi,t} = \{\{i, j\} : i, j \in [t, n], d_i^{\phi,t} + d_j^{\phi,t} = 1$ 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) $sig^b_{\phi,t}$ 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) $sig^{\neq}_{\phi,t}$ 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, $sig_{\phi,t} + i = (sig^1_{\phi,t} + i, sig^0_{\phi,t} + i, sig^{=}_{\phi,t} + i, sig^{\neq}_{\phi,t} + i)$.

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

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

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

Thus $d_i^{\phi,1} = 1 \Leftrightarrow d_{t+i-1}^{\phi,t} = 1$ and hence $i \in sig^1_{\phi,1} \Leftrightarrow t + i - 1 \in sig^1_{\phi,t}$. This shows that $sig^1_{\phi,t} = sig^1_{\phi,1} + t - 1$.

With similar arguments the theorem follows.

Remark. (1) In consequence of the above theorem it can be said that if a 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 $sig_{\phi,1} = (sig^1_{\phi,1}, sig^0_{\phi,1}, sig^{=}_{\phi,1}, sig^{\neq}_{\phi,1})$ becomes $sig_\phi = (sig^1_\phi, sig^0_\phi, sig^{=}_\phi, sig^{\neq}_\phi)$ 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.

IV. Signature Generation

Let us consider a fault position ϕ. We generate sig^1_ϕ, sig^0_ϕ using the probabilistic algorithm $GenSig10$ and $sig^{=}_\phi$, $sig^{\neq}_\phi$ using the deterministic algorithm $GenSigSym$ as described below. The deterministic algorithm $GenSigSym$ has implementation limitations. However, the probabilistic algorithm $GenSig10$ is much more efficient.

Algorithm 1. $GenSig10(\phi, \Omega, L_1)$

1. For Ω number of distinct uniformly random independent *Key-IV* pair : 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^b_ϕ
5. **return** : sig^1_ϕ, sig^0_ϕ.

Algorithm 2. $GenSigSym(\phi, L_2)$

// L_2 is the number of PRGA rounds used by the algorithm.

1. Define 256 symbolic variables over GF(2) and initialise the inner state with these 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 $sig^{=}_\phi$, $sig^{\neq}_\phi$.

Let i be an actual fixed position (PRGA round) for $b \in \{0, 1\}$ under the fault $(\phi, 1)$. Then the algorithm $GenSig10$ will surely append i to sig^b_ϕ. But if i is not a fixed position, then Pr(The algorithm appends i to sig^b_ϕ) $= 1/2^\Omega$ 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 $GenSig10$ 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.

V. 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 an XOR differential keystream of length $n = end(\Gamma) + T - 1$ 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 $d^{\psi,T,n} = (d_1^{\psi,T}, \ldots, d_n^{\psi,T})$.

3. Define, $support^b = \{i \in [1, n] : d_i^{\psi,T} = b\}$, $\forall\ b \in \{0, 1\}$.

4. $pf = allPossibleFaults\_known(\Gamma, T,\ d^{\psi,T,n})$ as described in Algorithm 3.

Algorithm 3. $allPossibleFaults\_known(\Gamma, t,\ d^{\psi,t,n})$

```
pf = ∅
for all φ ∈ Γ :
    if isaPossibleFault(φ, t, d^{ψ,t,n}) == True : // Algorithm 4
        pf = pf ⋃ {(φ, t)}
return pf
```

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 a 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.

VI. 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, $z^j = (z^j_1, \ldots, z^j_n)$ 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.

Algorithm 4. $isaPossibleFault(\phi, t, d^{\psi,t,n})$
output : "True" if (ϕ, t) is a possible fault location else "False".

```
1. sig_{φ,t} = sig_{φ,1} + t − 1
2. if sig^1_{φ,t} ⊆ support^1 :
3.     if sig^0_{φ,t} ⊆ support^0 :
4.         if sig^=_{φ,t} == ∅ :
5.             if sig^≠_{φ,t} == ∅ :
6.                 return True
7.             else :
8.                 if ∀{i, j} ∈ sig^≠_{φ,t} ⇒ d_i^{ψ,t} + d_j^{ψ,t} = 1 :
9.                     return True
10.                else : return False
11.        else :
12.            if ∀A ∈ sig^=_{φ,t} and ∀i, j ∈ A ⇒ d_i^{ψ,t} = d_j^{ψ,t} :
13.                if sig^≠_{φ,t} == ∅ :
14.                    return True
15.                else :
16.                    if ∀{i, j} ∈ sig^≠_{φ,t} ⇒ d_i^{ψ,t} + d_j^{ψ,t} = 1 :
17.                        return True
18.                    else : return False
19.            else : return False
20.    else : return False
21. else : return False
```

The adversary starts with the following information:

1. m fault locations $\beta = ((\phi_1, t), \ldots, (\phi_m, t)) \in Q$.
2. 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, $j \in [1, m]$.

A. Generating Polynomial Equations

We use procedure similar to [5] 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 $(i \geq t)$ respectively for NLFSR and LFSR 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*".

B. 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.

VII. Experimental Results

A. Faults generated from hardware implementation

1) Fault Injection Setup: In this section, we present the experimental results of fault injection by *clock glitching* on a hardware implementation of *Grain-128*. To the best of our knowledge, we report for the very first time the actual chip results for a fault attack on any stream cipher. An input clock was provided to a *Grain-128* Spartan-3A (XC3S400A) FPGA implementation from an external function generator. A fast clock of 20 times the frequency compared to the input slow clock was used to introduce a clock glitch at a fixed PRGA round. The fast clock was derived from the slow clock using a Xilinx Digital Clock Manager (DCM) module in the design and the states of the registers were monitored using Chipscope Pro 12.3 analyzer. We obtained the correct ciphertext (corresponding to the fault free *Grain-128* internal state) for input slower clock frequencies up to 7 MHz or fast clock frequencies up to 140 MHz. We gradually increased the input slower clock frequency in steps of 0.1 MHz and captured the corresponding states of the registers at each step. The number of attempts to inject a fault at each step was 1024. In **Fig. 1**, we plot the nature of the induced faults with respect to the frequency of the fast clock for different *key-IV* pairs.

The faults observed were all single bit ones and in each case the faults affected only the 128^{th} bit of the NFSR ($bitNLFSR_{128}$) due to clock glitch introduced set up time violations. This is because the critical path of *Grain-128* is through the NLFSR feedback [8]. From the experimental results we conclude that the single bit faults (if injected) were biased at a particular bit position irrespective of the initial register states. However, the frequency of occurrences of single bit faults varied for different *key-IV* pair initializations of the cipher. The reason for this variation may be due to the data dependent nature of fault sensitivity [9].

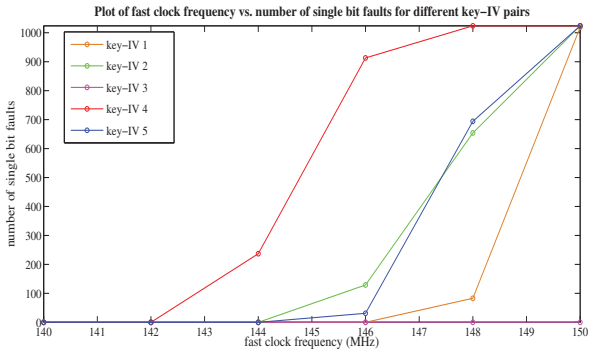

Fig. 1: Plot of fast clock frequency vs. number of single bit faults for different *key-IV* pairs.

2) Introducing multi-bit faults: The reported DFAs on Grain 128 requires at least 3 distinct single bit faults [5]. However, as we see using fault induction techniques like clock glitches, only a single bit is affected (namely due to the critical path), and thus we do not get the required number of faulty outputs to perform the analysis. One may employ more costly injection methods, however in such case also there are chances of the faults being spread to multiple bits. We try to tweak the fault induction technique using clock glitches to obtain distinct faults which can be exploited by the attack developed in this paper. The experimental results of our clock glitch set up revealed that targeting a particular PRGA round leads to only single bit faults at $bitNLFSR_{128}$ with a very high bias. Therefore, in order to induce multi-bit faults in cipher's internal state within k-neighbourhood bits, an adversary can introduce clock glitches in k consecutive PRGA rounds provided the targeted bits lie between the two bounding tap positions of the shift register. According to this modified multi-round fault injection technique, if a fault occurred in the first targeted PRGA round at $bitNLFSR_{128}$, it will propagate (shifted) to $bitNLFSR_{127}$ in the next PRGA round and so on. Therefore, there is a chance of fault injection up to k adjacent bits in the final targeted PRGA round (i.e. the k-th round). In Table 1, we present the experimental results of such multi-round fault injection (considering $k = 5$) for a randomly chosen *key-IV* pair.

B. *DFA of* Grain-128*: Based on simulations*

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 at most 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.

Faults in the Model nbdMBF^k. let Γ_k be set of all possible fault positions in this case. In the case when exactly r bits are flipped, the fault positions are of the form $i, i+j_1, i+j_2, \ldots, i+j_{r-1}$ where $i \in [1, 256 - j_{r-1}]$ and $(j_1, j_2, \ldots, j_{r-1})$ follows lexicographic ordering (increasing) without repetition of length $r-1$ in the range $[1, k-1]$. e.g., when $k = 4, r = 3$ possible (j_1, j_2) are $(1,2), (1,3), (2,3)$ and hence the fault positions are of the form $(i, i+1, i+2)$, $(i, i+1, i+3)$, $(i, i+2, i+3)$.

i	$i+1$	$i+2$	$i+3$
*	*	*	
*	*		*
*		*	*

i	$i+1$	$i+2$	$i+3$
i	$i+1$	$i+2$	
i	$i+1$		$i+3$
i		$i+2$	$i+3$

j_1	j_2
1	2
1	3
2	3

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 fault model considered in [5].

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. SAT solver Cryptominisat - 2.9.6 installed in SAGE - 6.1.1.

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 $sig^{\neq}_\phi$.

All simulation based experiments were performed assuming PRGA round 1 as the target round.

Prob. of Identifying a Random 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

TABLE I. *Grain-128* FAULT DISTRIBUTION PATTERN VS. FAST CLOCK FREQUENCY.

Fast Clock Frequency (MHz)	No fault	Single bit fault	Multi-bit fault
130	1024	0	0
140	1024	0	0
150	1024	0	0
160	1024	0	0
170	1024	0	0
180	1024	0	0
190	0	$bitNLFSR_{128}$ (1024)	0
200	0	$bitNLFSR_{125}$ (2)	$bitNLFSR_{128}$ (1022), $bitNLFSR_{125}$ (1022)
210	0	0	$bitNLFSR_{127}$ (1024), $bitNLFSR_{126}$ (1024), $bitNLFSR_{125}$ (1024)
220	0	0	$bitNLFSR_{127}$ (1024), $bitNLFSR_{126}$ (1024), $bitNLFSR_{125}$ (1024), $bitNLFSR_{124}$ (1024)

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.

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

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

SAT Solving Based on Clock Glitch Induced Faults

To demonstrate the feasibility of our attack approach, we took '0000ffffffffffffffffffffffffffff' and 'ffff2c480000ffffffff0000' as key and IV respectively. We targeted PRGA round 17. We performed 5 consecutive clock glitch fault injection trials at PRGA rounds 12, 13, 14, 15 and 16 but not at the PRGA round 17. Based on hardware experiments, due to shifting of NLFSR bits, the possible faults are $(\{127\}, 17)$, $(\{124\}, 17)$, $(\{127, 124\}, 17)$, $(\{126, 125, 124\}, 17)$,$(\{126, 125, 124, 123\}, 17)$.
We passed these faults to the SAT solver as explained in Section VI. The internal state '94b7841c7bff0ca4765cd312e25c51de269e886808dde1a2faed 03df8fed13a0' at PRGA round 17 was successfully recovered in 527.83 seconds with $N = 256$.

Probability of Rejecting Bad-Faults

In this paper any possible fault other than a k-neighbourhood fault at a fixed PRGA round (that disturbs only one PRGA round) is considered as a bad-fault. In this paper we show that arbitrary bad-faults can be rejected in favour of k-neighbourhood bit fault. For each column of the following table, we considered a set of 2^{20} experiments (at PRGA round 1).

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

Thus bad-faults at PRGA round 1 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 faults are identified with high probability.

VIII. CONCLUSION

This paper considers multi-bit faults for *key-IV* recovery. Attack strategy in this paper is generic and allows the adversary to challenge 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. This is the very first time the actual chip results for a fault attack on any stream cipher is produced. Also clock glitch is a very low cost technique to introduce faults. The algorithm $GenSig10$ is slower than D-GRAIN [5] but it can cope with any multi-bit fault. Thus allowing the adversary to challenge any multi-bit fault model. The validity of the randomized 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 k-neighbourhood bit fault ($k \in \{1, 2, 3, 4, 5\}$).

REFERENCES

[1] A. Berzati, C. Canovas, G. Castagnos, B. Debraize, L. Goubin, A. Gouget, P. Paillier, and S. Salgado, "Fault Analysis of GRAIN-128," in *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on*. IEEE, 2009, pp. 7–14.

[2] S. Banik, S. Maitra, and S. Sarkar, "A Differential Fault Attack on the Grain Family of Stream Ciphers," in *Cryptographic Hardware and Embedded Systems–CHES 2012*. Springer, 2012, pp. 122–139.

[3] ——, "A Differential Fault Attack on the Grain Family under Reasonable Assumptions," in *Progress in Cryptology-INDOCRYPT 2012*. Springer, 2012, pp. 191–208.

[4] S. Karmakar and D. R. Chowdhury, "Fault analysis of Grain-128 by targeting NFSR," in *Progress in Cryptology–AFRICACRYPT 2011*. Springer, 2011, pp. 298–315.

[5] S. Sarkar, S. Banik, and S. Maitra, "Differential Fault Attack against Grain family with very few faults and minimal assumptions," *IACR Cryptology ePrint Archive*, vol. 2013, p. 494, 2013.

[6] M. Hell, T. Johansson, A. Maximov, and W. Meier, "A Stream Cipher Proposal: Grain-128," http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain128_p3.pdf.

[7] N. Rohani, Z. Noferesti, J. Mohajeri, and M. R. Aref, "Guess and Determine Attack on Trivium Family," in *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*. IEEE, 2010, pp. 785–790.

[8] S. S. Mansouri and E. Dubrova, "An Improved Hardware Implementation of the Grain Stream Cipher," in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*. IEEE, 2010, pp. 433–440.

[9] Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta, "Fault Sensitivity Analysis," in *Cryptographic Hardware and Embedded Systems, CHES 2010*. Springer, 2010, pp. 320–334.