Differential Fault Analysis of MICKEY Family of Stream Ciphers

Sandip Karmakar, and Dipanwita Roy Chowdhury, *Member, IEEE,*
Department of Computer Science and Engineering,
Indian Institute of Technology, Kharagpur, WB, India.
e-mail: {sandip.karmakar, drc}@cse.iitkgp.ernet.in.

***Abstract*—This paper presents differential fault analysis of the MICKEY family of stream ciphers, one of the winners of eStream project. The current attacks are of the best performance among all the attacks against MICKEY ciphers reported till date. The number of faults required with respect to state size is about 1.5 times the state size. We obtain linear equations to determine state bits. The fault model required is reasonable. The fault model is further relaxed without reproducing the faults and allowing multiple bit faults. In this scenario, more faults are required when reproduction is not allowed whereas, it has been shown that the number of faults remains same for multiple bit faults.**

***Index Terms*—MICKEY-128 2.0, MICKEY v1, MICKEY 2.0, MICKEY-128, Side Channel Attack, Fault Attack**

I. Introduction

Stream ciphers have gained popularity in recent years in resource constrained environments. Considerable amount of effort in contemporary research have been given to stream ciphers. eStream [1] project started in 2005 introduced a number of stream ciphers in hardware and software efficient environments. The aim was to standardize stream ciphers. Over the years the eStream ciphers have undergone extensive crypanalytic efforts. After comparing strengths, throughput and few other essential criteria the winners were declared. The winners of the eStream project [3] were 7 candidates, 3 in hardware category and 4 in software category. Researchers have widely analyzed strengths of the 7 ciphers in vast cryptnalytic environments.

Very large number of cryptanalysis of ciphers are studied by the research community in recent years. Algebraic or statistical analysis mostly target the mathematical strength of the ciphers. Side channel attack of stream ciphers is another class of analysis of strength of ciphers, which includes, power analysis, fault analysis and timing analysis etc. Side channel attacks target implementations of ciphers. Implementations leak a number of side channel information, such as, power consumption, electromagnetic profile, faulty output etc. A side channel attacker analyzes these side channel information to break a cryptosystem. In recent literature, a large number of work on side channel attacks are published. Power analysis and fault analysis are most explored types of side channel attack. Most of the ciphers in the eStream portfolio are susceptible to side-channel-attacks ([13], [14], [15], [2]).

Fault attacks (or Differential Fault Attack(DFA)) are vastly explored in recent years and are efficient form of side channel attacks. In this scenario, attacker induces faults during cipher operations. The fault-free and faulty ciphertexts/keystreams are then analyzed to deduce partial or full value of the secret key. One of the main advantages of fault attacks is that it is practical like most side channel attacks and once required fault model is established the system may be broken in at most a few hours. Fault attacks have been shown to be successful against both block ciphers ([9], [10]) and stream ciphers ([13], [14], [15]). Recent literature shows that stream ciphers are extremely vulnerable to fault attacks([13], [14], [15]). Practical methods of fault induction are also studied in contemporary literature. Clock glitch, laser shots ([17], [18]) etc. are efficiently and accurately applicable in fault induction.

Fault attacks are successfully applied against almost all the eStream winners like Trivium, Grain, Mickey [19] from the hardware category and against Rabbit and Sosemnauk from the software category. Fault attack is reported against Mickey-128 [20] and Mickey 2.0 in [21]. In the paper [20], the mentioned faults and fault-free/faulty keystreams to break MICKEY-128 following initialization are reported as, 640 and 960 respectively. While [21] takes $2^{16.7}$ faults to break the MICKEY 2.0. A preliminary version of this paper appeared in [22]. In this paper, we present a fault attack on MICKEY-128 2.0, which is the latest version of MICKEY and no known result of fault attack is present against it. We show that this version can be attacked using 480 random single-bit faults and 480 pairs of faulty and fault-free keystreams. We show that our attack can easily be applied to all versions of the MICKEY family namely, MICKEY v1, MICKEY 2.0, MICKEY-128 and MICKEY-128 2.0 with maximum number of 480 faults.

This paper is organized as follows. Following this introduction we describe the design of MICKEY stream ciphers in section 2. The fault attack model of our attack is presented in section 3. In section 4 we have described detail procedure of our attack against all the ciphers of MICKEY family. Section 5 relaxes assumptions of the fault model. Finally, section 6 concludes the paper.

II. Background

In this section, we briefly discuss specification of the MICKEY family of stream ciphers. Throughout the paper, + refers to modulo 2 addition.

A. Specification of the MICKEY Stream Ciphers

The cipher family MICKEY, which stands for, Mutual Irregular Clocking KEYstream generator is one of the winners of the eStream cipher contest. A number of MICKEY ciphers are introduced to the eStream project over its duration, MICKEY v1 being the first and MICKEY-128 2.0 being the final cipher. Here, we briefly discuss the specification of the MICKEY ciphers. A detailed description may be found in [5] (MICKEY v1), [6] (MICKEY 2.0), [7] (MICKEY-128) and [19](MICKEY-128 2.0).

The structure of MICKEY consists of two registers, R and S. For different ciphers this registers are of different lengths. For MICKEY v1 the length, s is 80 for both the registers, while the lengths are 100, 128 and 160 for MICKEY 2.0, MICKEY-128 and MICKEY-128 2.0 respectively. Although the design concept of all the versions are same but they differ in the size of the state registers, s and so on security of the ciphers.

The initialization algorithm takes v bits of key and IV_LENGTH number of IV bits to initialize the system. The initialization algorithm is described in algorithm 1. The cipher takes, $v = 128/80$ bit key $k_0, \ldots, k_{127}/k_0, \ldots, k_{79}$ and IV_LENGTH number of IV bits, $v_0, \ldots, v_{IV\_LENGTH-1}$, where, $0 \leq IV\_LENGTH \leq 128/0 \leq IV\_LENGTH \leq 80$. MICKEY v1 and MICKEY 2.0 takes 80 bits of key, while MICKEY-128 and MICKEY-128 2.0 takes 128 bits of key.

Algorithm 1 INIT()

```
Initialize $R$ and $S$ registers are loaded with zeros.
for (i=0 to IV_LENGTH-1) do
    CLOCK_KG(TRUE, $iv_i$)
end for
for (i=0 to v-1) do
    CLOCK_KG(TRUE, $k_i$)
end for
for (i=0 to s-1) do
    CLOCK_KG(TRUE, 0)
end for
```

The overall system is clocked through CLOCK_KG function. Its description is given in algorithm 2.

The registers R and S are clocked following procedures CLOCK_R and CLOCK_S respectively, which are called from procedure CLOCK_KG (algorithm 2). The different versions of ciphers takes different parameters and executes CLOCK_R and CLOCK_S which clocks R and S registers respectively. For different versions the algorithms are described in the appendix. The update of each bits of R and S (r_i and s_i) is derived in closed form in comments of the algorithms.

Finally, the keystream generation is performed by algorithm 3 for all the ciphers.

Hence, the update of R and S registers following initialization can be written as,

1. **MICKEY-128 2.0**

$$\begin{aligned}
r_k(i) &= (k > 0)r_{k-1}(i-1) + (k \in RTAPS)r_{159}(i-1) \\
&\quad +(s_{54}(i-1) + r_{106}(i-1))r_k(i-1) \\
s_k(i) &= (k > 0)s_{k-1}(i-1) + (0 < k < 159) \\
&\quad [s_k(i-1)s_{k+1}(i-1) \\
&\quad +COMP0_k s_{k+1}(i-1) + COMP1_k s_k(i-1) \\
&\quad +COMP0_k COMP1_k] + s_{159}(i-1)FB0_k \\
&\quad +s_{159}s_{106}(FB0_k + FB1_k) \\
&\quad +s_{159}(i-1)r_{53}(i-1)(FB0_k + FB1_k) \\
z(i) &= r_0(i) + s_0(i)
\end{aligned}$$

Algorithm 2 CLOCK_KG(MIXING, INPUT_BIT)

```
//MICKEY-128 2.0
CONTROL_BIT_R = $s_{54} + r_{106}$
CONTROL_BIT_S = $s_{106} + r_{53}$
//MICKEY v1
CONTROL_BIT_R = $s_{27} + r_{53}$
CONTROL_BIT_S = $s_{53} + r_{26}$
//MICKEY-128
CONTROL_BIT_R = $s_{43} + r_{85}$
CONTROL_BIT_S = $s_{85} + r_{42}$
//MICKEY 2.0
CONTROL_BIT_R = $s_{34} + r_{67}$
CONTROL_BIT_S = $s_{67} + r_{33}$
if (MIXING == 1) then
    //MICKEY-128 2.0
    CLOCK_R(INPUT_BIT+$s_{80}$, CONTROL_BIT_R).
    //MICKEY v1
    CLOCK_R(INPUT_BIT+$s_{40}$, CONTROL_BIT_R).
    //MICKEY-128
    CLOCK_R(INPUT_BIT+$s_{64}$, CONTROL_BIT_R).
    //MICKEY 2.0
    CLOCK_R(INPUT_BIT+$s_{50}$, CONTROL_BIT_R).
    CLOCK_S(INPUT_BIT, CONTROL_BIT_S).
else
    CLOCK_R(INPUT_BIT, CONTROL_BIT_R).
    CLOCK_S(INPUT_BIT, CONTROL_BIT_S).
end if
```

Algorithm 3 KeyStream()

```
keystream = $r_0$ + $s_0$
CLOCK_KG(FALSE, 0)
```

2. **MICKEY v1**

$$\begin{aligned}
r_k(i) &= (k > 0)r_{k-1}(i-1) + (k \in RTAPS).r_{79}(i-1) \\
&\quad +(s_{27}(i-1) + r_{53}(i-1)).r_k(i-1) \\
s_k(i) &= (k > 0)s_{k-1}(i-1) + (0 < k < 79) \\
&\quad [s_k(i-1)s_{k+1}(i-1) \\
&\quad +COMP0_k s_{k+1}(i-1) + COMP1_k s_k(i-1) \\
&\quad +COMP0_k COMP1_k] + s_{79}(i-1)FB0_k \\
&\quad +s_{79}s_{53}(FB0_k + FB1_k) \\
&\quad +s_{79}(i-1)r_{26}(i-1)(FB0_k + FB1_k) \\
z(i) &= r_0(i) + s_0(i)
\end{aligned}$$

3. **MICKEY-128**

$$\begin{aligned}
r_k(i) &= (k > 0)r_{k-1}(i-1) + (k \in RTAPS)r_{127}(i-1) \\
&\quad +(s_{43}(i-1) + r_{85}(i-1))r_k(i-1) \\
s_k(i) &= (k > 0)s_{k-1}(i-1) + (0 < k < 127) \\
&\quad [s_k(i-1)s_{k+1}(i-1)
\end{aligned}$$

$$\begin{aligned} &+COMP0_k s_{k+1}(i-1) + COMP1_k s_k(i-1) \\ &+COMP0_k COMP1_k] + s_{127}(i-1)FB0_k \\ &+s_{127}s_{85}(FB0_k + FB1_k) \\ &+s_{127}(i-1)r_{42}(i-1)(FB0_k + FB1_k) \\ z(i) &= r_0(i) + s_0(i) \end{aligned}$$

4. **MICKEY 2.0**

$$\begin{aligned} r_k(i) &= (k>0)r_{k-1}(i-1) + (k \in RTAPS)r_{99}(i-1) \\ &\quad +(s_{34}(i-1) + r_{67}(i-1))r_k(i-1) \\ s_k(i) &= (k>0)s_{k-1}(i-1) + (0<k<99) \\ &\quad [s_k(i-1)s_{k+1}(i-1) \\ &\quad +COMP0_k s_{k+1}(i-1) + COMP1_k s_k(i-1) \\ &\quad +COMP0_k COMP1_k] + s_{99}(i-1)FB0_k \\ &\quad +s_{99}s_{67}(FB0_k + FB1_k) \\ &\quad +s_{99}(i-1)r_{33}(i-1)(FB0_k + FB1_k) \\ z(i) &= r_0(i) + s_0(i) \end{aligned}$$

III. Differential Fault Attack Model

In this section, we describe the fault analysis model assumed in our attack.

Our fault model is same as used in most of the fault attacks against stream ciphers ([8], [13], [14], [15], [16]). Hence, the fault is a bit-flip and it occurs in the internal state registers. Single bit faults are injected and the difference of the faulty and fault-free keystreams are exploited to analyze the system. We assume the following controllability in order to perform this attack.

1) The attacker is able to inject faults at $random$ positions of the internal state bits of the MICKEY implementation (hardware or software). Hence, exact fault position is not needed to be known beforehand.
2) The fault affects *exactly one* bit of the internal register at any cycle of operation. So, the fault flips *exactly one bit* of the internal register of MICKEY-128 2.0 implementation. Although, flipping exactly one bit of the internal register seems a strong assumption, but can be achieved practically by triggering laser shots through the I/O signal for hardware implementations ([17], [18]).
3) A fault to an internal register bit can be reproduced at any cycle of operation, once, it is created.
4) The attacker is able to determine and control the cycles of operation of the implementation, i.e., the timing of the implementation is under control of the attacker. This is easy for externally clocked systems.
5) The attacker can reset the implementation to its original state.

In other words, a key is embedded into the system. Adversary induces single bit faults at $random$ positions. The adversary first detects position of the fault. Then determines equations involving internal states from faulty and fault-free keystreams. If necessary the adversary reproduces the fault at the same location possibly at different clock cycles. The current cycle of operation is known to the attacker and can be controlled.

IV. Fault Analysis of MICKEY

In this section, we present our proposed fault attack on MICKEY. The attack is described fully for MICKEY-128 2.0. It has been shown that how the same attack is applied to other versions of MICKEY like MICKEY v1, MICKEY 2.0 and MICKEY-128 with changing parameters and relevant modifications. The actual attack is discussed in two subsections namely the determination of internal state of the cipher (i) R register and (ii) the S register. In both the cases attacks are mounted in the following two phases:

- Determine position of the fault.
- Obtain and Solve equations involving internal state bits.

In the first phase, we consider single bit random faults. Then, we use related single bit faults to determine faults at certain locations. Related single bit fault is different from multiple faults in the sense that two single bit faults are induced in different cycles at two fixed locations of two different internal states instead of inducing two faults in one cycle.

In the second phase, we retain our focus on linear equations only. The required number of equations can therefore be easily reduced by including higher degree equations. It is seen that the linear equations obtained are in single variable in a number of equations, thus are directly solvable. Obtained equations involve state bits at different iterations like $r_0(t)$ at t^{th} cycle, $r_0(t+10)$ etc. Following state update operation in MICKEY, we are able to obtain state bits of the cipher at a base point, say, at T^{th} clock cycle. Once, the state of the cipher at T is known we are able to predict keystream of the cipher at any iteration $t > T$, thus breaking the system. Note that the attack is performed $after$ the initialization phase.

A. Approach

In this attack, we consider single bit faults only, either random or related single faults. Let, $\Delta(X)$ denote the difference between faulty (X^f) and fault-free(X) values of X, (which can be a register bit or the keystream bit), i.e., $\Delta(X) = X + X^f$. Hence, the standard difference rules are applicable, i.e., for a fault at X,

- $\Delta(X) = 1$,
- $\Delta(Y) = 0$, Y is independent of X,
- $\Delta(Y+Z) = \Delta(Y) + \Delta(Z)$,
- $\Delta(YZ) = \Delta(Y)Z + Y\Delta(Z)$.

Throughout the paper we have used this terminology with corresponding fault locations mentioned in context.

We rewrite in closed form the update function of the state bits of R and S register of MICKEY-128 2.0. The k^{th} bit of the R register, the k^{th} bit of the S register $(0 \le k \le 159)$ and the output bit z at iteration i are given by,

$$\begin{aligned} r_k(i) &= (k>0)r_{k-1}(i-1) + (k \in RTAPS)r_{159}(i-1) \\ &\quad +(s_{54}(i-1) + r_{106}(i-1))r_k(i-1) \qquad (1) \\ s_k(i) &= (k>0)s_{k-1}(i-1) \\ &\quad +(0<k<159)[s_k(i-1)s_{k+1}(i-1) \\ &\quad +COMP0_k s_{k+1}(i-1) + COMP1_k s_k(i-1) \\ &\quad +COMP0_k COMP1_k] + s_{159}(i-1)FB0_k \end{aligned}$$

$$
\begin{aligned}
&+s_{159}s_{106}(FB0_k + FB1_k) \\
&+s_{159}(i-1)r_{53}(i-1)(FB0_k + FB1_k) \quad (2)\\
z(i) \;=\;& r_0(i) + s_0(i) \quad (3)
\end{aligned}
$$

First of all, let's note that the faults at r_k and s_k, $\Delta(r_k(i))$ and $\Delta(s_k(i))$ are determined by the following equations,

$$
\begin{aligned}
\Delta(r_k(i)) \;=\;& (k > 0)\Delta(r_{k-1}(i-1)) \\
&+(k \in RTAPS)\Delta(r_{159}(i-1) \\
&+\Delta(s_{54}(i-1))).r_k(i-1) \\
&+\Delta(r_{106}(i-1).r_k(i-1) \\
&+(s_{54}(i-1) + r_{106}(i-1)).\Delta(r_k(i-1)) \quad (4)\\
\Delta(s_k(i)) \;=\;& (k > 0)\Delta(s_{k-1}(i-1)) \\
&+(0 < k < 159)[\Delta(s_k(i-1)).s_{k+1}(i-1) \\
&+\Delta(s_{k+1}(i-1)s_k(i-1) \\
&+COMP0_i\Delta(s_{k+1}(i-1)) \\
&+COMP1_i\Delta(s_k(i-1))] \\
&+\Delta(s_{159}(i-1)FB0_i \\
&+(FB0_i + FB1_i).\Delta(s_{159}(i-1))s_{106}(i-1) \\
&+(FB0_i + FB1_i).s_{159}(i-1)\Delta(s_{106}(i-1)) \\
&+(FB0_i + FB1_i).\Delta(s_{159}(i-1))r_{53}(i-1) \\
&+(FB0_i + FB1_i) \\
&.\Delta(r_{53}(i-1))s_{159}(i-1) \quad (5)\\
\Delta(z(i)) \;=\;& \Delta(r_0(i)) + \Delta(s_0(i)) \quad (6)
\end{aligned}
$$

Our target is to obtain linear equations involving R and S bits from the above set of equations. Since, $\Delta(z(i)) = \Delta(r_0(i)) + \Delta(s_0(i))$ (equation 6) and we are considering single bit faults, $\Delta(r_0(i))$ and $\Delta(s_0(i))$ should be considered separately.

- $\Delta(r_0(i))$: Equations 4 and 5 imply, faults at s_{54}, r_{106}, r_0 and (s_{54} or r_{106}) can produce linear equations, with values of, r_0, r_0, $(s_{54}(i-1) + r_{106})$ and r_0 found out respectively at appropriate cycles. Faults at other locations will lead to non-linear equations.
- $\Delta(s_0(i))$: In this case, equations 4 and 5 imply, faults at s_k, s_{k+1}, s_{159}, s_{106} and r_{53} can lead to linear equations. However, faults at s_k or s_{k+1} is not immediately reflected in $\Delta(z)$, hence, faults at s_{159} and (s_{106} or r_{53}) are only useful. Fault at s_{159} will provide the value of, $(s_{106}+r_{53})$, while, fault at (s_{106} or r_{53}) will provide the value of s_{159} at appropriate cycles. Faults at other locations will involve equations with non-linear terms.

With these noted fault locations and target equations, we proceed to derive values of R and S. This procedures are discussed in the following sections.

Note that due to the similarity of the update equations analogus observations can be made for MICKEY v1, MICKEY-128 and MICKEY 2.0.

B. Determining R Register

In this subsection, we describe the methodology to determine register bits of R at iteration T, following initialization. We discuss the two phases of the attack with relevant algorithms and experimental results in the following two sections. We first determine the position of the fault. Then obtain information about the internal state bits of the cipher analyzing the output differences of the keystreams. Then, we solve the obtained linear equations to infer state bits of R register of the cipher at a base point T. Finally, we count the number of faults required to determine entire R register.

1) Determining Position of the Fault: The determination of fault-locations is done during a preprocessing phase. The output bit is dependent on r_0 and s_0. When a fault is induced in R register, through the presence of $r_{k-1}(i-1)$ term in the update of $r_k(i)$ (equation 1) it propagates to the next bit in R register during the following cycle and so on. Hence, it follows that after $(160-f)$ iterations we will get a faulty keystream if and only if position f was originally faulted for $f > 106$, for obvious reasons, this explains table I. Note that this is because $r_0(i)$ is dependent on $r_{159}(i-1)$ (equation 1). However, the fault-difference may not be a constant i.e. bit-flip, due to the presence of $(s_{54}(i-1)+r_{106}(i-1))r_k(i-1)$ term (equations 1, 4).

Note that it is possible to determine fault locations determinable by single (unrelated) fault, F_s^s by observing equations 4 and 6. *It turns out that we need only use fault at location r_0 for our subsequent use, hence, from equations 4 and 6, it follows that for a fault at r_0, immediately (i.e., at cycle 0), a output bit-flip is obtained.* However, this may be due to a fault at s_0 also. The distinguisher (distinguishing beween r_0 and s_0 as fault location) is reported in algorithm 5. Note that this means that our preprocessing phase is very short in terms of time required.

A similar argument applies to faults at locations $f > 106$ in R register. Following equations 4 and 6, the obtained equations of the output-difference can be obtained. Other terms of equation 4 becomes 0, only $(s_{54}(i-1)+r_{106}(i-1))$ remain for faults at $r_k(i-1)$. It is tabulated in table I.

Faults at bit r_{106} or s_{54}: For our analysis, equation 4 shows that faults in locations r_{106} or s_{54} need to be determined. However, we are not able to determine faults at r_{106} or s_{54} using single random faults. We apply two related faults; first fault at r_0 and then at (r_{106} or s_{54}. *Note that this follows the fault model.* We induce a random fault and determine if it is in location r_0 or s_0 (algorithm 4). If r_0 or s_0 can once be faulted, it may be reproduced later. Then we fault randomly again in the faulty or normal state register (following no fault and a fault at (r_0 or s_0)). Then we check if the fault was in (r_{106} or s_{54}) following an algorithm devised later (algorithm 5).

Fault propagations in R and S are determined by equations 4 and 5, respectively. Hence, $\Delta(z) = 1$ at cycle 0 following a fault induction at cycle 0, holds if and only if fault is induced in r_0 or s_0. The fault propagation at r_0 and s_0 are respectively given by (from equations 4 and 5),

$$
\begin{aligned}
\Delta(r_0(i)) \;=\;& \Delta(r_{159}(i-1) + \Delta(s_{54}).r_0(i-1) \\
&+\Delta(r_{106}(i-1).r_0(i-1) \\
&+(s_{54}(i-1) + r_{106}(i-1)) \\
&.\Delta(r_0(i-1)) \quad (7)
\end{aligned}
$$

TABLE I
FAULT LOCATION VS. $\Delta(z) = 1$ VS. EQUATION LHS

Location (F_r)	Cycle for $\Delta(z) = 1$	Output Equation LHS (next cycle)
r_{160-i}	i (i¿106)	(s_{54}(i)+r_{106}(i))
s_0	0	(s_1(0))

$$\Delta(s_0(i)) \quad = \quad \Delta(s_{159}(i-1)) \tag{8}$$

So, following fault at r_0 or s_0, the fault propagation to next cycle is determined by ($\Delta(r_0(i)) = 1$, other differentials being 0),

$$\begin{aligned} \Delta\Delta(r_0(i+1)) &= \Delta(s_{54}(i) + r_{106}(i)) \\ \Delta\Delta(s_0(i+1)) &= 0 \end{aligned}$$

This gives us a way to distinguish between faults at r_0 and s_0. A fault at r_0 and (r_{106} or s_{54}) in cycle 0 will definitely yield $\Delta\Delta(z(1)) = 1$, while a fault at s_0 and any other location will set $\Delta\Delta(z(1))$ to 0. *Therefore, following this methodology we are able to determine single fault at r_0 and related faults at r_0 and r_{106} or s_{54}.* The methodology is described in algorithms 4 and 5. This preprocessing phase requires 160+159=319 faults on average.

Algorithm 4 Location-R0orS0()

```
Induce single bit fault in the registers.
//$z(0)$ is fault-free output at cycle=0.
//$z^f(0)$ is faulty output at cycle=0
**if** $z(0) + z^f(0) = 1$ **then**
    $r_0$ or $s_0$ is faulty.
**end if**
```

Algorithm 5 Locations-R106orS54()

```
Fault $r_0$ or $s_0$ following algorithm 4.
//By assumption, we can reproduce faults at any location.
Compute $\Delta(z(1)) = z(1) + z^f(1)$.
Fault $r_0$ or $s_0$.
Fault any other location.
Compute $\Delta(z^{fr}(1)) = z(1) + z^{fr}(1)$.
**if** $\Delta(z(1)) + \Delta(z^{fr}(1)) = 1$ **then**
    $r_0$ and ($r_{106}$ or $s_{54}$) is faulty.
**end if**
```

In a similar fashion we need to obtain fault-positions r_0/s_0 and s_{27} or r_{53} for MICKEY v1, r_0/s_0 and s_{43} or r_{85} for MICKEY-128 and r_0/s_0 and s_{34} or r_{67} for MICKEY-2.0. Clearly algorithms similar to 4 and 5 can be devised to determine those positions of faults at those locations.

2) Obtain and Solve Equations to Determine internal R state bits: The second phase of the attack targets obtaining equations from the identifiable fault locations. Suppose, faults at r_k is identifiable using table I. We can form a fault trace table for faults at location r_k and running the cipher for c cycles following the fault induction. If the fault creates a linear (degree 1 involving r_k and s_k bits) fault at $\Delta(z)$ at any cycle c', we have one equation for solving the internal state bits of the cipher. The only relations involved in state bits of MICKEY-128 2.0 determinable from random single faults are listed in table I.

Now we explore the single fault induction at different cycles. From table I, evidently from faults at r_k we find LHS (left hand side) of the form (s_{54}(i)+r_{106}(i)), which can be obtained by inducing faults at cycle (i-1) at location r_0. *That is one fault at r_0 (identifiable through algorithms 1 and 2) at cycle (i-1) will give the adversary the value of,*

$$(s_{54}(i) + r_{106}(i))$$

for all values of i. This process requires 1 fault.

Now, we make use of faults at location r_{106} or s_{54}. Following algorithm 5, we are able to induce and determine faults at (r_{106} or s_{54}). Now, from equation 7, a single bit fault at (r_{106} or s_{54}) at cycle $(i-1)$ yields ($\Delta(r_{159}(i-1)$=0),

$$\Delta(z(i)) \quad = \quad r_0(i-1)$$

This gives the attacker the exact value stored in r_0 at cycle $(i-1)$ by observing difference between fault-free and faulty keystreams at cycle i. That is, $r_0(i)$ is known from this procedure, by a single fault.

Now, let's look at the update relation of r_k given in equation 1. For $k = 0$, this becomes,

$$\begin{aligned} r_0(i) &= r_{159}(i-1) \\ &\quad +(s_{54}(i-1) + r_{106}(i-1))r_0(i-1) \\ or, r_{159}(i-1) &= r_0(i) \\ &\quad +(s_{54}(i-1) + r_{106}(i-1))r_0(i-1) \end{aligned} \tag{9}$$

Since, $(s_{54}(i-1) + r_{106}(i-1))$ and $r_0(i-1), r_0(i)$ can be obtained as described earlier, $r_{159}(i-1)$ becomes known from above equation. It requires 2 extra faults once $r_0(i)$ is known.

For all other values of $k > 0$, first it can be noted that as $(s_{54}(i-1) + r_{106}(i-1))$ (say, a) and $r_{159}(i-1)$ are known, (say, b),

$$\begin{aligned} r_k(i) &= (k > 0)r_{k-1}(i-1) \\ &\quad +(k \in RTAPS).b \\ &\quad +a.r_k(i-1) \\ or, r_{k-1}(i-1) &= r_k(i) + (k \in RTAPS).b \\ &\quad +a.r_k(i-1) \end{aligned}$$

Now going backwards from r_{159} to r_1, we see from the above equation that as $r_{159}(i)$ can be obtained for all i, $r_{158}(i)$ can be found out for all i (requiring 2 faults to determine $r_{159}(i-1)$), from which $r_{157}(i)$ can be obtained and so on till $r_1(i)$ at each step 2 extra faults are required. Hence, the values of rest of the register bits of R can be determined.

Again in a similar manner the above described process may be adapted to other versions of MICKEY, requiring proportionate number of faults and solution of similar equations.

3) Number of Faults Required: The required number of faults in order to determine values of all register bits of R is 320. The number is counted as follows. To determine $r_0(i)$ we need a single fault at ($r_{106}(i)$ or $s_{54}(i)$). To obtain the value of $r_{159}(i)$ we need values of $r_0(i)$ and $r_0(i+1)$, which requires another extra fault at ($r_{106}(i+1)$ or $s_{54}(i+1)$). Through a fault at $r_0(i-1)$ we get value of $(r_{106}(i-1)+s_{54}(i-1))$. So, $r_{159}(i)$ becomes known for two extra faults. $r_{158}(i)$ requires values of $r_{159}(i)$ and $r_{159}(i-1)$. $r_{159}(i)$ is known and obtaining $r_{159}(i-1)$ requires two extra faults. So, $r_{158}(i)$ can be found out with two more faults. Similarly, to obtain $r_k(i)$ we need values of $r_{(k+1)\%160}(i)$, $r_{(k+1)\%160}(i+1)$ and $r_{159}(i)$. A similar argument shows that for each k and i two extra faults are required. Table II tabulates this requirement.

Similarly, MICKEY v1 requires 160 faults, MICKEY-128 requires 256 faults and MICKEY-2.0 requires 200 faults to determine all register bits of R. The whole process is summurized for all versions of MICKEY in table II.

C. Determining S Register

In this section, we determine values of the S register bits. Note that at this point we have known $r_0(i)$, therefore, observing $z(i)$, we directly obtain $s_0(i) = z(i)+r_0(i)$. Hence, values not known at this point are, $s_1(i), \ldots, s_{159}(i)$. The update of S register is given by equation 2. It directly gives, $s_0(i) = s_{159}(i-1)$ as, $FB0_0 = 1$ and $FB1_0 = 1$. Hence, $s_{159}(i)$ is obtained from the value of $s_0(i+1)$, which requires a fault at ($r_{106}(i+1)$ or $s_{54}(i+1)$).

For s_{159}, equation 2 becomes,

$$\begin{aligned} s_{159}(i) &= s_{158}(i-1) + s_{159}(i-1)(FB0_{159} \\ &\quad +(s_{106}(i-1)+r_{53}(i-1)) \\ &\quad (FB0_{159}+FB1_{159})) \\ &= s_{158}(i-1) + s_{159}(i-1) \\ &\quad (1+(s_{106}(i-1)+r_{53}(i-1))) \end{aligned}$$

As $(s_{106}(i-1)+r_{53}(i-1))$ is not known we are not able to proceed further. We need to introduce faults in S register to determine that value. We describe the process of determining rest of the values of S register in the following two subsections.

1) Determining Fault Location: This is a preprocessing phase. We start by looking at the equations propagating faults to r_0 and s_0. These are given by,

$$\begin{aligned} \Delta(r_0(i)) &= \Delta(r_{159}(i-1) + \Delta(s_{54}).r_0(i-1) \\ &\quad +\Delta(r_{106}(i-1).r_0(i-1) \\ &\quad +(s_{54}(i-1)+r_{106}(i-1)) \\ &\quad .\Delta(r_0(i-1)) \\ \Delta(s_0(i)) &= \Delta(s_{159}(i-1)) \end{aligned}$$

As, $\Delta(z(i)) = \Delta(r_0(i)) + \Delta(s_0(i))$, $\Delta(z(i)) = 1$ if and only if, $r_{159}(i-1)$ or $s_{159}(i-1)$ is faulted (as an identity). It can be noted that at this point we can find out, $r_0(i)$, $r_{159}(i)$, $s_0(i)$ and $s_{159}(i)$ for all i. Hence, it can be easily found out, which register-bit is faulted by observing these values and the output bit $z(i)$. That is the adversary is able to fault $s_{159}(i)$ using algorithm 6. This preprocessing phase requires 160 faults on an average.

Algorithm 6 Location-S159()

```
Induce single bit fault in the registers.
//$z(i)$ is fault-free output at cycle=i.
//$z^f(i)$ is faulty output at cycle=i.
if $z(1)+z^f(1)=1$ then
    $r_{159}(0)$ or $s_{159}(0)$ is faulty.
    Check if $r_{159}(0)$ is faulted.
    Find value of $c = r_0^f(1)$ in this faulty system,
    through method described earlier.
    Find value of $d = r_0(1)$ in the fault-free system,
    using values known earlier of $R$ register.
    if $c == d$ then
        $s_{159}$ is faulted.
    end if
end if
```

We will not need no other locations for faults to deduce S register-bits.

Similarly we need only know positions of faults at s_{79}, s_{127} and s_{99} for MICKEY v1, MICKEY-128 and MICKEY-2.0 respectively. Algorithms similar to algorithm 6 may be devised to this end.

The method of retrieving values of rest of the bits is described in the following section.

2) Obtain and Solve Equations to Determine internal S state bits: The S register-bit $s_{159}(i)$ is updated as,

$$\begin{aligned} s_{159}(i) &= s_{158}(i-1) + s_{159}(i-1) \\ &\quad (1+(s_{106}(i-1)+r_{53}(i-1))) \end{aligned}$$

Therefore, it follows following a bit-flip at s_{159} at cycle, $i-1$,

$$\begin{aligned} \Delta(z_0(i+1)) &= \Delta(s_0(i+1)) \\ &= \Delta(s_{159}(i)) \\ &= (1+(s_{106}(i-1)+r_{53}(i-1))) \end{aligned}$$

Hence, a fault at $s_{159}(i-1)$ at cycle (i-1) gives the value of $(1+(s_{106}(i-1)+r_{53}(i-1))) = \Delta(z(i+1))$. Similarly, for other versions analogous linear equations may be obtained. Therefore, $(s_{106}(i-1)+r_{53}(i-1))$ can be found out for all i. To sum up, at this point we are able to find $s_0(i)$, $s_{159}(i)$ and $(s_{106}(i-1)+r_{53}(i-1))$ for all i. Hence, $s_{158}(i)$ is known using one extra fault at $s_{159}(i-1)$,

$$\begin{aligned} s_{158}(i-1) &= s_{159}(i) + s_{159}(i-1) \\ &\quad (1+(s_{106}(i-1)+r_{53}(i-1))) \end{aligned}$$

For other values of k, $s_k(i)$ is given in equation 2. Beginning with $k = 158$ and going backwards till $k = 1$, we see equation

TABLE II
FAULTS REQUIRED TO DETERMINE R REGISTER BITS

Register Bit	Required Values	Known	Unknown	#Faults/ Loc.
MICKEY-128 2.0				
$r_0(i)$	-	-	-	1 / (r_{106} or s_{54})
$r_{159}(i)$	$r_0(i+1)$, $r_0(i)$, $s_{54}(i)+r_{106}(i)$	$r_0(i)$	$r_0(i+1)$, $s_{54}(i)+r_{106}(i)$	2/ (r_{106} or s_{54}) , r_0
$r_k(i)$	$r_{k+1}(i+1)$, $r_{k+1}(i)$, $r_{159}(i)$, $s_{54}(i)+r_{106}(i)$	$r_{k+1}(i)$, $r_{159}(i)$, $s_{54}(i)+r_{106}(i)$	$r_{k+1}(i+1)$	2 ($k=158,\ldots,1$) / r_0, (r_{106} or s_{54})
MICKEY v1				
$r_0(i)$	-	-	-	1, $s_{27} or r_{53}$
$r_{79}(i)$	$r_0(i+1)$, $r_0(i)$, $s_{27}(i)+r_{53}(i)$	$r_0(i)$	$r_0(i+1)$, $s_{27}(i)+r_{53}(i)$	2 /r_0, s_{27} or r_{53}
$r_k(i)$	$r_{k+1}(i+1)$, $r_{k+1}(i)$, $r_{79}(i)$, $s_{27}(i)+r_{53}(i)$	$r_{k+1}(i)$, $r_{79}(i)$, $s_{27}(i)+r_{53}(i)$	$r_{k+1}(i+1)$	2 ($k=78,\ldots,1$) /r_0, s_{27} or r_{53}
MICKEY-128				
$r_0(i)$	-	-	-	1/s_{43} or r_{85}
$r_{127}(i)$	$r_0(i+1)$, $r_0(i)$, $s_{43}(i)+r_{85}(i)$	$r_0(i)$	$r_0(i+1)$, $s_{43}(i)+r_{85}(i)$	2 /r_0, s_{27} or r_{53}
$r_k(i)$	$r_{k+1}(i+1)$, $r_{k+1}(i)$, $r_{127}(i)$, $s_{43}(i)+r_{85}(i)$	$r_{k+1}(i)$, $r_{127}(i)$, $s_{43}(i)+r_{85}(i)$	$r_{k+1}(i+1)$	2 ($k=126,\ldots,1$) /r_0, s_{43} or r_{85}
MICKEY 2.0				
$r_0(i)$	-	-	-	1/$s_{34}(i)+r_{67}(i)$
$r_{99}(i)$	$r_0(i+1)$, $r_0(i)$, $s_{34}(i)+r_{67}(i)$	$r_0(i)$	$r_0(i+1)$, $s_{34}(i)+r_{67}(i)$	2 /r_0, s_{34} or r_{67}
$r_k(i)$	$r_{k+1}(i+1)$, $r_{k+1}(i)$, $r_{99}(i)$, $s_{34}(i)+r_{67}(i)$	$r_{k+1}(i)$, $r_{99}(i)$, $s_{34}(i)+r_{67}(i)$	$r_{k+1}(i+1)$	2 ($k=99,\ldots,1$) /r_0, s_{34} or r_{67}

2 can be rewritten as,

$$\begin{aligned} s_{k-1}(i-1) &= s_k(i)+[s_k(i-1)s_{k+1}(i-1) \\ &\quad +COMP0_k s_{k+1}(i-1) \\ &\quad +COMP1_k s_k(i-1) \\ &\quad +COMP0_k COMP1_k] \\ &\quad +s_{159}(i-1)(FB0_k \\ &\quad +(s_{106}(i-1) \\ &\quad +r_{53}(i-1))(FB0_k+FB1_k)) \end{aligned}$$

It can be noted that all values on the RHS (Right Hand Side) of the relation can be obtained by methods described earlier in the range ($0 < k < 159$) in a recursive manner, with $(s_{106}(i-1)+r_{53}(i-1))$ being known for all i; in a similar manner as the R register. It follows, all the bits of S register can be known. This process requires one extra fault to determine $s_k(i)$ recursively.

An analogous recursive equation gives values of S registers for MICKEY v1, MICKEY-128 and MICKEY-2.0.

3) Number of Faults Required: The total number of faults required to get all bits of the S register is 159. s_0 and s_{159} were known without any faults. To determine $s_{158}(i-1)$ we needed a fault in $s_{159}(i)$. For $s_{157}(i-1)$, $s_{158}(i)$ needs to be known. This requires one fault at $s_{159}(i+1)$. Similarly, for $k = 156,\ldots,1$, one extra fault is needed for each k. Thus the total number of faults required comes to be, 158. Similarly, MICKEY v1 requires 80 faults, MICKEY-128 requires 128 faults and MICKEY 2.0 requires 100 faults approximately. Table III tabulates this requirement. This table also summurizes the process of fault-attack on other vesrions of MICKEY.

D. Performance

In summary, about 480 faults and 480 faulty and fault-free keystream pairs are required to break MICKEY-128 2.0, 240 faults and that many faulty and fault-free key streams are required for MICKEY v1, 384 faults and that many faulty and fault-free key streams are required for MICKEY-128, 300 faults and that many faulty and fault-free key streams are required for MICKEY-2.0. It is seen that faults are to be induced in very few of the register-bits. This localization of faults implies small preprocessing overhead in determining fault locations, only algorithms 4, 5 and 6 or similar ones need to be executed in preprocessing phase of determining fault locations. However, faults need to be induced at a wide-span of cycles of operation of the cipher. Hence, the adversary will heavily use the controllability of clocking of the system. Altogether, the system may be broken in few hours for a given

TABLE III
FAULTS REQUIRED TO DETERMINE S REGISTER BITS

Register Bit	Required Values	Known	Unknown	#Faults/ Loc.
MICKEY-128 2.0				
$s_0(i)$	-	-	-	0
$s_{159}(i)$	-	-	-	1/r_0
$s_{158}(i)$	$s_{159}(i+1)$, $s_{159}(i)$, $(s_{106}(i-1)+r_{53}(i-1))$	$s_{159}(i+1)$, $s_{159}(i)$	$(s_{106}(i-1)$ $+r_{53}(i-1))$	1 /s_{159}
$s_k(i)$	$s_{k+2}(i+1)$, $s_{k+1}(i)$, $s_k(i)$, $s_{159}(i)$, $(s_{106}(i-1)+r_{53}(i-1))$	$s_{k+1}(i)$, $s_k(i)$, $s_{159}(i)$, $(s_{106}(i-1)+r_{53}(i-1))$	$s_{k+2}(i+1)$ $(k=157,\ldots,1)$	1 /s_{159}
MICKEY v1				
$s_0(i)$	-	-	-	-
$s_{79}(i)$	-	-	-	1 /r_0
$s_{78}(i)$	$s_{79}(i+1)$, $s_{79}(i)$, $(s_{53}(i-1)+r_{27}(i-1))$	$s_{79}(i+1)$, $s_{79}(i)$	$(s_{53}(i-1)$ $+r_{27}(i-1))$	1 /s_{79}
$s_k(i)$	$s_{k+2}(i+1)$, $s_{k+1}(i)$, $s_k(i)$, $s_{79}(i)$, $(s_{53}(i-1)+r_{27}(i-1))$	$s_{k+1}(i)$, $s_k(i)$, $s_{79}(i)$, $(s_{53}(i-1)+r_{27}(i-1))$	$s_{k+2}(i+1)$ $(k=77,\ldots,1)$	1 /s_{79}
MICKEY-128				
$s_0(i)$	-	-	-	-
$s_{127}(i)$	-	-	-	1/r_0
$s_{126}(i)$	$s_{127}(i+1)$, $s_{127}(i)$, $(s_{85}(i-1)+r_{42}(i-1))$	$s_{127}(i+1)$, $s_{127}(i)$	$(s_{85}(i-1)$ $+r_{42}(i-1))$	1 /s_{127}
$s_k(i)$	$s_{k+2}(i+1)$, $s_{k+1}(i)$, $s_k(i)$, $s_{127}(i)$, $(s_{85}(i-1)+r_{42}(i-1))$	$s_{k+1}(i)$, $s_k(i)$, $s_{79}(i)$, $(s_{85}(i-1)+r_{42}(i-1))$	$s_{k+2}(i+1)$ $(k=125,\ldots,1)$	1 /s_{127}
MICKEY 2.0				
$s_0(i)$	-	-	-	-
$s_{99}(i)$	-	-	-	1/r_0
$s_{98}(i)$	$s_{99}(i+1)$, $s_{99}(i)$, $(s_{85}(i-1)+r_{42}(i-1))$	$s_{99}(i+1)$, $s_{99}(i)$	$(s_{67}(i-1)$ $+r_{33}(i-1))$	1 /s_{99}
$s_k(i)$	$s_{k+2}(i+1)$, $s_{k+1}(i)$, $s_k(i)$, $s_{99}(i)$, $(s_{67}(i-1)+r_{33}(i-1))$	$s_{k+1}(i)$, $s_k(i)$, $s_{99}(i)$, $(s_{67}(i-1)+r_{33}(i-1))$	$s_{k+2}(i+1)$ $(k=97,\ldots,1)$	1 /s_{99}

implementation when our analysis model is usable.

In table IV we compare known fault attacks against MICKEY with the present work. It can be seen that present work improves on the number of faults required against the earlier work [20] with comparable key size, while it is much efficient than [21] with lower state and/or key size. The paper [21], uses random faults and not reproduction of those faults at same location. It requires huge number of faults, $2^{16.7}$ or about 100000 faults.

V. MODEL RELAXATION

The fault model required to mount the attack on MICKEY ciphers needs,

- Reproduction of faults at *specific* locations once a fault can be created at that location.
- Fault/Bit-flip is created at exactly one register bit of the cipher.

Although the above requirements can be implemented, we investigate whether it can be improved further. Hence, we proceed to relax our fault model by removing the above assumptions, then the method can easily be implemented and the model is at per with [21]. It will be seen that the first requirement is extremely important for reducing complexity of our attack, while the second requirement turns out not to be much of an obstruction.

A. Relaxing Reproduction of Faults

We assume that faults may be created in any location of the 320 bit register of MICKEY-128 2.0. Faults are created at random locations. Locations of the faults are not known beforehand. The created faults amounts to bit-flip at the location of the fault. We do not require the fault to be reproduced later. In this case faults are random in the 320 positions, with a specific location being faulted has probability $1/320$. Faults are distributed at the 320 locations with replacement

TABLE IV
COMPARISON OF FAULT ATTACKS AGAINST MICKEY

Attack	*Version*	*Fault Model*	*State Size*	*#Faults*	*#Key-stream*
[20]	MICKEY-128	Similar	256	640	960
[21]	MICKEY 2.0	Different	200	$2^{16.7}$	$2^{17.7}$
This Work	MICKEY-128 2.0	-	320	480	960
This Work	MICKEY v1	-	160	240	480
This Work	MICKEY-128	-	256	384	768
This Work	MICKEY 2.0	-	200	300	600

and independently, meaning that a location faulted may be faulted with same probability at any following cycle in the fault induction process.

We have described our attack in the previous sections. In order to perform our attack we needed faults at r_0, (r_{106} or s_{54}) and s_{159} in 160 different cycles. If faults can not be reproduced, let us calculate the expected number of faults required per cycle to fault r_0. Here, success probability is $p = 1/320$, hence, failure probability is, $q = 1 - 1/320$. The probability of success after n trials is, pq^{n-1}. Hence, the distribution is Geometric and the expected number of trials for fault at r_0 is, Σnpq^{n-1}=$1/p = 320$. Similar is the argument for faults at r_{106}, s_{54} and s_{159}. That is, expected number of faults required per round is, $3 * 320 = 960$. Therefore, the attack requires 160×960 or about 160000 faults and that many pairs of faulty and fault-free keystreams. This number even though large is still implementable. Also note that it is slightly better than [21] since for MICKEY 2.0 the required number of faults come as, $3 * 200 * 100 = 60000$.

Clearly the number of faults required to break the system is extremely large. It is about $1000/3$ times the fault required in the model which needed fault reproduction. It follows that fault reproduction is very important in fault attack in terms of complexity of the attack.

B. Relaxing Single Bit-flip

We now allow a single fault affecting multiple bits. Note that we *allow* reproduction of faults in this scenario. A straightforward calculation of expected number of faults as described in the earlier subsection gives an estimate of number of faults required without reproduction.

For the attack to succeed we needed to identify faults at locations, r_0, (r_{106} or s_{54}) and s_{159}. Let's see how identifying this positions are affected by multiple-bit faults. Ideally a fault will affect consecutive bits of the state register of the system, say, 16 bits. We will see how this helps in our attack in a few lines.

- r_0: If multiple bits are faulted (bit-flipped) and r_0 is among the faulted bits, we have, $\Delta(z_0) = 1$. Even if other locations are faulted the effect does not propagate to $\Delta(z_0)$. However, this is true even if s_0 is faulted. In case both r_0 and s_0 are faulted, $\Delta(z_0) = 0$. So, if and only if, multiple bits are faulted and either r_0 or s_0 is among the faulted bits, $\Delta(z_0) = 1$. Hence, we need to distinguish between faults at r_0 and s_0. Earlier we distinguished by related faults at r_{106} or s_{54}. We see if this still works for multiple bit faults. After a fault at r_0 at cycle $i - 1$, the output difference is given by,
$\Delta(z_1) = \Delta(r_{106} + s_{54})$
Therefore, the identity $\Delta(z_1) = 1$ is attained if and only if location r_{106} or s_{54} is faulted. Clearly, faults at other locations does not affect it, as $\Delta(z_1)$ does not depend on other bits. Therefore the method described earlier for single-bit faults, applies here unchanged.
- r_{106} or s_{54}: Above step depicts how to determine fault at location r_0. It also describes how to determine faults at location r_{106} or s_{54}. This follows from algorithms 4 and 5.
- s_{159}: Following a single multi-bit fault at the internal state, the output difference, $\Delta(z_1) = \Delta(s_{159}) = 1$ if and only if s_{159} or r_{159} is faulted. Again due to independence of other variables fault at this location can be determined by the method described earlier(algorithm 6). Multiple bit faults does not affect the algorithm as well.

The fault locations being identified, during determination of state bits, same method as before applies with certain restrictions.

- When r_0 and other locations are faulted, (r_{106} and s_{54}) should not be faulted. As following equation 4, if only r_0 (with other bits) and not (r_{106} and s_{54}) is faulted we in a straightforward manner get the value of (r_{106} + s_{54}). From equation 4 it also becomes evident that r_k and r_{k-1} should not be faulted while determining value of r_{k-1}. Hence, we require algorithm to determine if r_k is faulted. The algorithm for distinguishing these locations is simple. The register bits form R, which affects the output bit z are, r_{53} and r_{106}. Hence, a fault from r_k to any of those two bits when propagated should give a different in value faulty keystream. So, let $m = minpositive\{53 - k, 106 - k, 160 - k\}$, where minpositive is the minimum positive value among set elements. Then after m cycle of operation following fault induction there should be faulty keystream. However, for a large number of different IVs it will show a difference with very high probability for faults at r_k. Therefore this process will require resetting the device applying different IV and reproducing the faults. We use this method to determine fault locations r_k. Note that it is still possible that r_k is not faulted even if mth cycle shows a different faulty keystream. But if there is a fault at r_k in i th cycle there will be a difference in m th cycle. The process is described in algorithm 7.
- When r_{106} or s_{54} are faulted (and not both, among other faults), r_0 and r_{159} should not be faulted. Then form

Algorithm 7 Location-$R_k()$

```
Induce single multiple bit fault in the registers.
$m = minpositive\{53 - k, 106 - k\}$.
//$z(i)$ is fault-free output at cycle=i.
//$z^f(i)$ is faulty output at cycle=i.
if $z(m) + z^f(m) = 1$ then
    $r_k$ or $s_k$ is faulty.
end if
Try this condition for large number of times.
```

equation 4 again we get the value of $r_k(i-1)$. Even if r_{159} is faulted we get an affine equation. From equation 4 it also becomes evident that r_k and r_{k-1} should not be faulted while determining value of r_{k-1}.

- When s_{159} is faulted (among other bits), s_{106} and r_{53} should not be faulted. As from equation 5, the dependence shows that this gives us the value of (s_{106} + r_{53}). From equation 5 it becomes evident that s_{k-1}, s_k and s_{k+1} should not be faulted, while determining value of s_{k-1}. Hence, we need to determine faults at those locations by some algorithm. A similar algorithm as 7 can be devised for this purpose.

Note that r_0, r_{106}, s_{54} and s_{159} being non-consecutive up to 16 bits, it is not hard to assume the exclusiveness of faults while for others the process is easy to test. Note that we don't need the exact values of faulty keystreams when there is fault at r_k or s_k. Thus the procedures described above applies.

Therefore multiple bits does not affect our algorithms. Hence, total number of multi-bit faults required to determine the full internal state of MICKEY-128 2.0 remains 480. However there is an overhead of multiple resetting and IV-ing of the device. This is clearly an advantage on much relaxed fault model. Clearly, other ciphers of MICKEY family adapts to this relaxation. It follows that the design of MICKEY ciphers is very vulnerable to multi-bit fault attack. The reason is that very few register bits are involved in faults and their exploitation only gives equations to determine internal state.

VI. Conclusion

In this paper, we have performed a fault analysis of the MICKEY family of stream ciphers. The attack can be mounted practically with small number of faults. Finally, this paper gives a demonstration of the process of analyzing ciphers through related faults, which can possibly be used against other stream/block ciphers to simplify or mount fault attacks.

References

[1] The eStream project. *"http://www.ecrypt.eu.org/stream/"*.

[2] Mukesh Agrawal, Sandip Karmakar, Dhiman Saha, and Debdeep Mukhopadhayay. Scan Based Side Channel Attacks on Stream Ciphers and their Counter-measures. *Progress in Cryptology - INDOCRYPT 2008*, 5365/2008:226–238, 2008.

[3] Steve Babbage, Christophe De Canniere, Anne Canteaut, Carlos Cid, Henri Gilbert, Thomas Johansson, Matthew Parker, Bart Preneel, Vincent Rijmen, and Matthew Robshaw. The eStream Portfolio. *"http://www.ecrypt.eu.org/stream/portfolio.pdf"*.

[4] Steve Babbage and Matthew Dodd. The stream cipher MICKEY-128 2.0. *eSTREAM, ECRYPT Stream Cipher Project*, 2006

[5] Steve Babbage and Matthew Dodd. The stream cipher MICKEY v1. *eSTREAM, ECRYPT Stream Cipher Project*, 2006

[6] Steve Babbage and Matthew Dodd. The stream cipher MICKEY 2.0. *eSTREAM, ECRYPT Stream Cipher Project*, 2006

[7] Steve Babbage and Matthew Dodd. The stream cipher MICKEY-128. *eSTREAM, ECRYPT Stream Cipher Project*, 2006

[8] Alexandre Berzati, Cecile Canovas, Guilhem Castagnos, Blandine Debraize, Louis Goubin, Aline Gouget, Pascal Paillier, and Stephanie Salgado. Fault Analysis of Grain-128. *Hardware-Oriented Security and Trust, IEEE International Workshop on*, 0:7–14, 2009.

[9] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. *Crypto 97*, 1294 of LNCS:513–525, 1997.

[10] Johannes Blomer and Jean-Pierre Seifert. Fault Based Cryptanalysis of the Advanced Encryption Standard. *Financial Cryptography*, 2742:162–181, 2003.

[11] Itai Dinur and Adi Shamir. Breaking Grain-128 with Dynamic Cube Attacks. *FSE 2011: 167–187.*

[12] Martin Hell, Thomas Johansson, and Willi Meier. A Stream Cipher Proposal: Grain-128. *eSTREAM, ECRYPT Stream Cipher Project*, 2006.

[13] Jonathan J. Hoch and Adi Shamir. Fault Analysis of Stream Ciphers. *CHES 2004*, 3156/2004 of LNCS:1–20, 2004.

[14] Michal Hojsk and Bohuslav Rudolf. Differential Fault Analysis of Trivium. *FAST SOFTWARE ENCRYPTION*, 5086/2008 of LNCS:158–172, 2008.

[15] Aleksandar Kircanski and Amr M. Youssef. Differential Fault Analysis of Rabbit. *SELECTED AREAS IN CRYPTOGRAPHY*, 5867/2009:197–214, 2009.

[16] Sandip Karmakar and Dipanwita Roy Chowdhury. Fault analysis of Grain-128 by targeting NFSR. *AFRICACRYPT'11*, 298–315, 2011.

[17] Sergei P. Skorobogatov. Optically Enhanced Positionlocked Power Analysis. *CHES 2006*, 4249 of LNCS:61–75, 2006.

[18] Sergei P. Skorobogatov and Ross J. Anderson. Optical Fault Induction Attacks. *CHES 2002*, 2523 of LNCS:2–12, 2002.

[19] Steve Babbage and Matthew Dodd. The stream cipher MICKEY 2.0. *eSTREAM, ECRYPT Stream Cipher Project*, 2006

[20] ZHANG Peng1,HU Yupu1,ZHANG Tao. Fault attack of MICKEY-128. *Electronic Science and Technology 2011, 24(6) 80-*, 2011

[21] Subhadeep Banik and Subhamoy Maitra. A Differential Fault Attack on MICKEY 2.0. *CHES*, 2013

[22] Sandip Karmakar and Dipanwita Roy Chowdhury. Differential Fault Analysis of MICKEY-128 2.0. *FDTC 2013*, 2013

Appendix

In the appendix we describe the CLOCK_R and CLOCK_S algorithms for different versions of the MICKEY cipher.

- MICKEY v1: The CLOCK_R and CLOCK_S algorithms are described in algorithms, algorithm 8, algorithm 9.
- MICKEY 2.0: The CLOCK_R and CLOCK_S algorithms are described in algorithms, algorithm 10, algorithm 11.
- MICKEY-128: The CLOCK_R and CLOCK_S algorithms are described in algorithms, algorithm 12, algorithm 13.
- MICKEY-128 2.0: The CLOCK_R and CLOCK_S algorithms are described in algorithms, algorithm 14, algorithm 15.

$FB0_i$, $FB1_i$, $COMP0_i$, $COMP1_i$, $RTAPS$ are constant Boolean values, the exact values of these constants for different i can be found in [5] (MICKEY v1), [6] (MICKEY 2.0), [7] (MICKEY-128) and [19](MICKEY-128 2.0).

Algorithm 8 CLOCK_R(INPUT_BIT_R, CONTROL_BIT_R)

```
Let, $r_0, \dots, r_{79}$ denote states of $R$ register.
FEEDBACK_BIT = $r_{79}$ + INPUT_BIT_R.
//FEEDBACK_BIT = $r_{79}$ during key generation.
for (i = 1 to 79) do
  $r'_i = r_{i-1}$
  //$r'_i = (i > 0)r_{i-1}$
end for
$r'_0 = 0$.
for (i=0 to 79) do
  if ($i \in RTAPS$) then
    $r'_i = r'_i + FEEDBACK\_BIT$
    //$r'_i = (i > 0)r_{i-1} + (i \in RTAPS).r_{79}$
  end if
end for
for (i=0 to 79) do
  if (CONTROL_BIT_R == 1) then
    $r'_i = r'_i + r_i$
    //CONTROL_BIT_R=$s_{27} + r_{53}$
    //$r'_i = (i > 0)r_{i-1} + (i \in RTAPS).r_{79} + (s_{27} + r_{53}).r_i$
  end if
end for
$r'_0, \dots, r'_{79}$ represents updated $R$ register.
```

Algorithm 9 CLOCK_S(INPUT_BIT_S, CONTROL_BIT_S)

```
Let, $s_0, \dots, s_{79}$ denote states of $S$ register.
FEEDBACK_BIT = $s_{79}$ + INPUT_BIT_S.
//FEEDBACK_BIT = $s_{79}$ during key generation.
for (i = 1 to 78) do
  $s'_i = s_{i-1} + ((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
end for
$s'_0 = 0$.
$s'_{79} = s_{78}$.
//$s'_i = (i > 0)s_{i-1} + (0 < i < 79)((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
for (i=0 to 79) do
  if (CONTROL_BIT_S == 1) then
    //CONTROL_BIT_S=$s_{53} + r_{26}$ during key generation.
    $s''_i = s'_i + (FB0_i.FEEDBACK\_BIT)$
  else
    $s''_i = s'_i + (FB1_i.FEEDBACK\_BIT)$
  end if
end for
//$s''_i = (i > 0)s_{i-1} + (0 < i < 79)((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
//+$(s_{53} + r_{26} + 1).FB0_i.s_{79}$ + $(s_{53} + r_{26}).FB1_i.s_{79}$
$s''_0, \dots, s''_{79}$ represents updated $S$ register.
```

Algorithm 10 CLOCK_R(INPUT_BIT_R, CONTROL_BIT_R)

```
Let, $r_0, \dots, r_{99}$ denote states of $R$ register.
FEEDBACK_BIT = $r_{99}$ + INPUT_BIT_R.
//FEEDBACK_BIT = $r_{99}$ during key generation.
for (i = 1 to 99) do
  $r'_i = r_{i-1}$
  //$r'_i = (i > 0)r_{i-1}$
end for
$r'_0 = 0$.
for (i=0 to 99) do
  if ($i \in RTAPS$) then
    $r'_i = r'_i + FEEDBACK\_BIT$
    //$r'_i = (i > 0)r_{i-1} + (i \in RTAPS).r_{99}$
  end if
end for
for (i=0 to 99) do
  if (CONTROL_BIT_R == 1) then
    $r'_i = r'_i + r_i$
    //CONTROL_BIT_R=$s_{34} + r_{67}$
    //$r'_i = (i > 0)r_{i-1} + (i \in RTAPS).r_{99} + (s_{34} + r_{67}).r_i$
  end if
end for
$r'_0, \dots, r'_{99}$ represents updated $R$ register.
```

Algorithm 11 CLOCK_S(INPUT_BIT_S, CONTROL_BIT_S)

```
Let, $s_0, \dots, s_{99}$ denote states of $S$ register.
FEEDBACK_BIT = $s_{99}$ + INPUT_BIT_S.
//FEEDBACK_BIT = $s_{99}$ during key generation.
for (i = 1 to 98) do
  $s'_i = s_{i-1} + ((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
end for
$s'_0 = 0$.
$s'_{99} = s_{98}$.
//$s'_i = (i > 0)s_{i-1} + (0 < i < 99)((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
for (i=0 to 99) do
  if (CONTROL_BIT_S == 1) then
    //CONTROL_BIT_S=$s_{67} + r_{33}$ during key generation.
    $s''_i = s'_i + (FB0_i.FEEDBACK\_BIT)$
  else
    $s''_i = s'_i + (FB1_i.FEEDBACK\_BIT)$
  end if
end for
//$s''_i = (i > 0)s_{i-1} + (0 < i < 99)((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
//+$(s_{67} + r_{33} + 1).FB0_i.s_{127}$ + $(s_{67} + r_{33}).FB1_i.s_{127}$
$s''_0, \dots, s''_{99}$ represents updated $S$ register.
```

Algorithm 12 CLOCK_R(INPUT_BIT_R, CONTROL_BIT_R)

```
Let, $r_0, \ldots, r_{127}$ denote states of $R$ register.
FEEDBACK_BIT = $r_{127}$ + INPUT_BIT_R.
//FEEDBACK_BIT = $r_{127}$ during key generation.
for (i = 1 to 127) do
  $r'_i = r_{i-1}$
  //$r'_i = (i > 0)r_{i-1}$
end for
$r'_0 = 0$.
for (i=0 to 127) do
  if ($i \in RTAPS$) then
    $r'_i = r'_i + FEEDBACK\_BIT$
    //$r'_i = (i > 0)r_{i-1} + (i \in RTAPS).r_{79}$
  end if
end for
for (i=0 to 127) do
  if (CONTROL_BIT_R == 1) then
    $r'_i = r'_i + r_i$
    //CONTROL_BIT_R=$s_{43} + r_{85}$
    //$r'_i = (i > 0)r_{i-1} + (i \in RTAPS).r_{127} + (s_{43} + r_{85}).r_i$
  end if
end for
$r'_0, \ldots, r'_{127}$ represents updated $R$ register.
```

Algorithm 13 CLOCK_S(INPUT_BIT_S, CONTROL_BIT_S)

```
Let, $s_0, \ldots, s_{127}$ denote states of $S$ register.
FEEDBACK_BIT = $s_{127}$ + INPUT_BIT_S.
//FEEDBACK_BIT = $s_{127}$ during key generation.
for (i = 1 to 126) do
  $s'_i = s_{i-1} + ((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
end for
$s'_0 = 0$.
$s'_{127} = s_{126}$.
//$s'_i = (i > 0)s_{i-1} + (0 < i < 127)((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
for (i=0 to 127) do
  if (CONTROL_BIT_S == 1) then
    //CONTROL_BIT_S=$s_{85} + r_{42}$ during key generation.
    $s''_i = s'_i + (FB0_i.FEEDBACK\_BIT)$
  else
    $s''_i = s'_i + (FB1_i.FEEDBACK\_BIT)$
  end if
end for
//$s''_i = (i > 0)s_{i-1} + (0 < i < 127)((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
//+$(s_{85} + r_{42} + 1).FB0_i.s_{127}$ + $(s_{85} + r_{42}).FB1_i.s_{127}$
$s''_0, \ldots, s''_{127}$ represents updated $S$ register.
```

Algorithm 14 CLOCK_R(INPUT_BIT_R, CONTROL_BIT_R)

```
Let, $r_0, \ldots, r_{159}$ denote states of $R$ register.
FEEDBACK_BIT = $r_{159}$ + INPUT_BIT_R.
//FEEDBACK_BIT = $r_{159}$ during key generation.
for (i = 1 to 159) do
  $r'_i = r_{i-1}$
  //$r'_i = (i > 0)r_{i-1}$
end for
$r'_0 = 0$.
for (i=0 to 159) do
  if ($i \in RTAPS$) then
    $r'_i = r'_i + FEEDBACK\_BIT$
    //$r'_i = (i > 0)r_{i-1} + (i \in RTAPS).r_{159}$
  end if
end for
for (i=0 to 159) do
  if (CONTROL_BIT_R == 1) then
    $r'_i = r'_i + r_i$
    //CONTROL_BIT_R=$s_{54} + r_{106}$
    //$r'_i = (i > 0)r_{i-1} + (i \in RTAPS).r_{159} + (s_{54} + r_{106}).r_i$
  end if
end for
$r'_0, \ldots, r'_{159}$ represents updated $R$ register.
```

Algorithm 15 CLOCK_S(INPUT_BIT_S, CONTROL_BIT_S)

```
Let, $s_0, \ldots, s_{159}$ denote states of $S$ register.
FEEDBACK_BIT = $s_{159}$ + INPUT_BIT_S.
//FEEDBACK_BIT = $s_{159}$ during key generation.
for (i = 1 to 158) do
  $s'_i = s_{i-1} + ((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
end for
$s'_0 = 0$.
$s'_{159} = s_{158}$.
//$s'_i = (i > 0)s_{i-1} + (0 < i < 159)((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
for (i=0 to 159) do
  if (CONTROL_BIT_S == 1) then
    //CONTROL_BIT_S=$s_{106} + r_{53}$ during key generation.
    $s''_i = s'_i + (FB0_i.FEEDBACK\_BIT)$
  else
    $s''_i = s'_i + (FB1_i.FEEDBACK\_BIT)$
  end if
end for
//$s''_i = (i > 0)s_{i-1} + (0 < i < 159)((s_i + COMP0_i)(s_{i+1} + COMP1_i))$
//+$(s_{106} + r_{53} + 1).FB0_i.s_{159}$ + $(s_{106} + r_{53}).FB1_i.s_{159}$
$s''_0, \ldots, s''_{159}$ represents updated $S$ register.
```