Differential Fault Analysis of MICKEY-128 2.0

SANDIP KARMAKAR AND
DIPANWITA ROY CHOWDHURY,

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR, WB, INDIA

Outline of the Talk

Introduction

Brief Review
- MICKEY-128 2.0
- Fault Attack

Fault Analysis of MICKEY-128 2.0
- Assumptions of the Fault Model
- Determining Fault Location
- Determining R bits
- Determining S bits

Improvements

Conclusions

Introduction

MICKEY-128 2.0 [15] is one of the three hardware based ciphers enlisted in the eStream[1, 3] portfolio.

Two fault based attacks [16,17] are the only known weaknesses of MICKEY-128 2.0.

[16] targets Mickey-128 and breaks the system with 640 faults with a similar fault model as used here.

While [17] targets Mickey v1 and breaks the system with $2^{16.7}$ faults, with a different fault model than used here.

We break Mickey-128 2.0, the strongest version of the family with only 480 faults and few minutes of computation.

The fault model used is used widely in literature[4,5,7,9,11,12,16].

Brief Review

Fault Attack

Fault attacks are one of the most efficient side channel attacks known till date.

In this kind of attack, faults are injected during cipher operations.

The attacker then analyzes the fault free and faulty cipher-texts or key-streams to deduce partial or full value of the secret key.

The literature shows that both the block ciphers ([5], [6]) and stream ciphers [4,5,7,9,11,12,16] are vulnerable against fault attack.

Methods like clock glitch, laser shots ([13], [14]) have been shown to be a practical way of inducing faults in a crypto-system.

Specification of Mickey-128 2.0

Mickey-128 2.0 consists of two non-linear registers, R and S of 160 bits each, i.e., R0, … R159 and S0, … S159.

A key of 128 bits and an IV up to 128 bits is used in the cipher.

The key, IV is initialized using algorithm INIT() described later.

The key stream is generated following procedure KeyStream(), which in turn clocks the registers and generates key stream per cycle following equation, z=R0+S0.

A detailed description may be found in [15].

Specification of Mickey-128 2.0

Mickey-128 2.0 works using the following algorithms,

Algorithm 1 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.

Specification of Mickey-128 2.0

Algorithm 2 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.

Specification of Mickey-128 2.0

Algorithm 3 CLOCK_KG(MIXING, INPUT_BIT)

```
CONTROL_BIT_R = s_54 + r_106
CONTROL_BIT_S = s_106 + r_53
if (MIXING == 1) then
   CLOCK_R(INPUT_BIT+s_80, 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 4 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 127) do
   CLOCK_KG(TRUE, k_i)
end for
for (i=0 to 159) do
   CLOCK_KG(TRUE, 0)
end for
```

Algorithm 5 KeyStream()

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

Proposed Fault Attack

Fault Analysis Model

The attacker is able to induce faults at random positions of the R or S registers of the Mickey-128 2.0 implementation (hardware or software).

The fault affects exactly one bit of register at any cycle of operation.

A fault to an register bit can be reproduced at any cycle of operation, once, it is created.

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.

The attacker can reset the implementation to its original state.

The attacker can run the implementation with different IV, without changing the key.

Overview of the Attack

The following steps are carried out in the attack:

- Determining Fault Location
- Determining R bits
- Determining S bits

Determining fault position

The output bit differential is given by,

$$\Delta(z(i)) \quad = \quad \Delta(r_0(i)) + \Delta(s_0(i))$$

The differential at k-th bit of R is given by,

$$\begin{aligned}\Delta(r_k(i)) \quad = \quad & (k > 0)\Delta(r_{k-1}(i-1)) \\ & +(k \in RTAPS)\Delta(r_{159}(i-1) \\ & +\Delta(s_{54}(i-1))).r_k(i-1) \\ & +\Delta(r_{106}(i-1).r_k(i-1) \\ & +(s_{54}(i-1) + r_{106}(i-1)).\Delta(r_k(i-1))\end{aligned}$$

Determining fault position

The k-th bit of S has differential,

$$
\begin{aligned}
\Delta(s_k(i)) \quad = \quad & (k>0)\Delta(s_{k-1}(i-1)) \\
& +(0<k<159)[\Delta(s_k(i-1)).s_{k+1}(i-1) \\
& +\Delta(s_{k+1}(i-1)s_k(i-1) \\
& +COMP0_i\Delta(s_{k+1}(i-1)) \\
& +COMP1_i\Delta(s_k(i-1))] \\
& +\Delta(s_{159}(i-1)FB0_i \\
& +(FB0_i+FB1_i).\Delta(s_{159}(i-1))s_{106}(i-1) \\
& +(FB0_i+FB1_i).s_{159}(i-1)\Delta(s_{106}(i-1)) \\
& +(FB0_i+FB1_i).\Delta(s_{159}(i-1))r_{53}(i-1) \\
& +(FB0_i+FB1_i) \\
& .\Delta(r_{53}(i-1))s_{159}(i-1) \qquad (5)
\end{aligned}
$$

Determining fault position

So, a bit flip at cycle 0, producing output difference immediately, implies fault occurred at R0 or S0.

Faults following above fault at locations R106 or S54 are identified as,

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

This works since, after a fault at R0 or S0, a second fault means,

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

Determining fault position

We also need to identify faults at S159, this is described in the algorithm,

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

Determining the R Bits

S54(i)+R106(i): From the differential at R0, we can see that fault at R0(i-1) gives the value of S54(i)+R106(i)=D(z(i)).

R0(i): From differential of z, we can see that a fault at S54(i-1) or R106(i-1) gives the value of R0(i) = D(z(i)).

R159(i): This value can be obtained from. as others are known.

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

Other values can be obtained recursively from,

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

where, a=S54(i-1)+R106(i-1) and b=R159(i-1).

Determining the R Bits

Number of faults required:

Register Bit	Required Values	Known	Unknown	#Faults
$r_0(i)$	-	-	-	1
$r_{159}(i)$	$r_0(i+1)$, $r_0(i)$, $s_{54}(i) + r_{106}(i)$	$r_0(i)$	$r_0(i+1)$, $s_{54}(i) + r_{106}(i)$	2
$r_{158}(i)$	$r_{159}(i+1)$, $r_{159}(i)$, $s_{54}(i) + r_{106}(i)$	$r_{159}(i)$, $r_{159}(i)$, $s_{54}(i) + r_{106}(i)$	$r_{159}(i+1)$	2
$r_k(i)$	$r_{k+1}(i+1)$, $r_{k+1}(i)$, $r_{159}(i)$, $s_{54}(i) + r_{106}(i)$	$r_{k+1}(i)$, $r_{159}(i)$, $s_{54}(i) + r_{106}(i)$	$r_{k+1}(i+1)$	2 $(k = 157, \ldots, 1)$

Determining the S Bits

S106(i-1)+R5(i-1): A fault at S159(i) gives, S106(i-1)+R5(i-1)=D(z(i+1))+1

S0(i): S0(i)=z(i)+R0(i), R0(i) being known and z(0) obtainable S0(i) can be determined.

S159(i): S159(i)=S0(i+1), which is obtained earlier.

S158(i) etc.: Determined from,

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

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

Determining S Bits

Number of faults required:

Register Bit	Required Values	Known	Unknown	#Faults
$s_0(i)$	-	-	-	-
$s_{159}(i)$	-	-	-	1
$s_{158}(i)$	$s_{159}(i+1)$, $s_{159}(i)$, $(s_{106}(i-1)+r_{53}(i-1))$	$s_{159}(i+1)$, $s_{159}(i)$	$(s_{106}(i-1)$ $+r_{53}(i-1))$	1
$s_k(i)$	$s_{k+2}(i+1)$, $s_{k+1}(i)$, $s_k(i)$, $s_{159}(i)$, $(s_{106}(i-1)+r_{53}(i-1))$	$s_{k+1}(i)$, $s_k(i)$, $s_{159}(i)$, $(s_{106}(i-1)+r_{53}(i-1))$	$s_{k+2}(i+1)$	1 $(k=157,\ldots,1)$

Complexity of the Attack

480 faults are needed to obtain full state of the cipher. 320 faults are required to determine R register and 160 faults are required to determine S register.

480 pairs of faulty/fault-free keystreams are required to break the system.

The time complexity of the attack in computation is negligible.

Improvements

The attack can be improved by including other linear/nonlinear equations to reduce requirements of number of faults at the cost of more computation.

Faults affecting multiple R/S register bits per fault injection might reduce attack complexity.

Conclusions

In this work, we have demonstrated that a much less number of faults in Mickey-128 2.0 may break the system compared to its earlier versions.

The experimental result shows that 480 faults and 480 pair of faulty/fault-free keystreams will break the system.

The computational complexity of our attack is negligible.

Hence, in few minutes the attack may be mounted.

The fault model used in the paper is widely used in contemporary literature.

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] 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 InternationalWorkshop on, 0:7–14, 2009.

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

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

[7] Itai Dinur and Adi Shamir. Breaking Grain-128 with DynamicCube Attacks. FSE 2011: 167–187.

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

References

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

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

[11] Aleksandar Kircanski and Amr M. Youssef. Differential Fault Analysis of Rabbit. SELECTED AREAS IN CRYPTOGRAPHY,

5867/2009:197–214, 2009.

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

[13] Sergei P. Skorobogatov. Optically Enhanced Position locked Power Analysis. CHES 2006, 4249 of LNCS:61–75, 2006.

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

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

[16] ZHANG Peng,HU Yupu,ZHANG Tao. Fault attack of MICKEY-128. Electronic Science and Technology 2011,24(6) 80-, 2011

[17] Subhadeep Banik and Subhamoy Maitra. A Differential Fault Attack on MICKEY 2.0. Cryptology ePrint Archive, Report 2013/029, 2013

Questions

Please mail your questions to:

sandip1kk@gmail.com

or

drc@cse.iitkgp.ernet.in

Thank You