Differential Fault Analysis of SHA3-224 and SHA3-256

Pei Luo* , Yunsi Fei*, Liwei Zhang†, and A. Adam Ding†
silenceluo@gmail.com, yfei@ece.neu.edu, mathliwei@gmail.com, a.ding@neu.edu
**Electrical & Computer Engineering Department, Northeastern University, Boston, MA 02115 USA*
†*Department of Mathematics, Northeastern University, Boston, MA 02115 USA*

***Abstract*—The security of SHA-3 against different kinds of attacks are of vital importance for crypto systems with SHA-3 as the security engine. In this paper, we look into the differential fault analysis of SHA-3, and this is the first work to conquer SHA3-224 and SHA3-256 using differential fault analysis. Comparing with one existing related work, we relax the fault models and make them realistic for different implementation architectures. We analyze fault propagation in SHA-3 under such single-byte fault models, and propose to use fault signatures at the observed output for analysis and secret retrieval. Results show that the proposed method can effectively identify the injected single-byte faults, and then recover the whole internal state of the input of last round χ operation (χ_i^{22}) for both SHA3-224 and SHA3-256.**

***Keywords*-SHA-3, Keccak, Security, Differential fault analysis**

I. Introduction

As the new secure hash standard (SHA-3), the security of Keccak against different attacks is of vital importance. Keccak algorithm is a family of sponge function based on the permutation function f, Keccak-f[r+c]. The parameter r means the bitrate and c means the capacity, and the internal state size is $1{,}600 = r + c$ bits for SHA-3. SHA-3 has four modes with different lengths of the digest, $d \in \{224, 256, 384, 512\}$ [1]. While there exists only one previous work of differential fault analysis (DFA) on SHA3-384 and SHA3-512 under single-bit fault model [2], SHA3-224 and SHA3-256 have not been attacked using DFA yet. In this paper, we extend DFA to SHA3-224 and SHA3-256, and also adopt more relaxing and realistic fault models.

DFA utilizes the dependency of the output faults on the internal intermediate variables to recover the secret. DFA is a powerful and efficient attack method, and has been used to break various cryptographic algorithms. It was first introduced to hack the Data Encryption Standard (DES) algorithm [3]. Later it was used to break the Advanced Encryption Standard (AES) [4] - only two pairs of correct and faulty ciphertexts with one fault injected are needed to break the AES-128. Many other ciphers have also been hacked by DFA, including CLEFIA [5], Mickey [6], [7] and Grain [8], [9].

This work has been published in FDTC'2016, please cite that version instead. Simulation code used in this paper will be provided online at http://tescase.coe.neu.edu/.

Some existing hash standards have been evaluated against DFA attacks, including SHA-1 [10], Streebog [11], MD5 [12] and GrøStl [13]. DFA can be used to retrieve the original message when hash functions are in general usage [12], [13]. When hash functions are used in the message authentication code (MAC) mode with a secret key, DFA also becomes a great threat and it can be used to recover the key, and then the attackers can generate forgery messages and MAC against authentication [10], [11], [13]. As Keccak has a very different design philosophy from previous crypto algorithms, previous attack methods on hash functions cannot be applied directly to SHA-3.

Previous works on SHA-3 mainly focus on side-channel power analysis and collisions attacks, etc [14]–[30]. The only existing DFA work on SHA-3 [2] is based on single-bit fault model, and targets two modes of SHA-3, SHA3-512 and SHA3-384. However, this fault model is overly simplified and unrealistic. Because many general fault injection methods, such as clock glitches and supply voltage variation, would affect a group of bits in intermediate states all together and it is almost impossible to precisely inject single-bit faults into the system without using sophisticated invasive fault injection methods, such as laser emission and ion beaming. Attacking the other two modes of SHA-3 is also much more challenging with less observable digest output.

In this paper, we propose DFA attack on SHA3-224 and SHA3-256, which have not been conquered using DFA yet, under more realistic and relaxed fault models - byte-level faults. Our approach includes generation of Fault Signature (FS) representing the propagation result in SHA-3 with various faults injected. It then uses the limited observable digest output to recover part of the input of the last round χ operation, χ_i^{23}, for attacks. We implement and simulate all the proposed methods and algorithms in C++ using randomly generated messages and faults. Results show that the proposed DFA can efficiently recover all 1,600 internal state bits for both SHA3-224 and SHA3-256.

The rest of this paper is organized as follows. In Section II, the preliminaries of SHA-3 will be given first, then the fault model used in this work will be presented. In Section III, we will introduce the use of fault signatures to represent the fault propagation in SHA-3, and the method to recover part of χ_i^{23} for attacks. Attack results based on the generated fault signatures and the recovered χ_i^{23} will be given. In

Section IV, we improve the attacks by constructing fault signatures at the output of last χ operation and present the attack results. We further improve the attacks by injecting faults into the last round input, and present the improved attack results. Finally, we conclude this paper in Section V.

II. Preliminaries of SHA-3 and Differential Fault Analysis

A. Preliminaries of Keccak Hash Function

The Keccak hash algorithm can work in different modes with variable length. Standardized by NIST, SHA-3 functions operate in modes of Keccak-f[1600, d] [1], where each internal state is 1600-bit organized in a 3-D array, as shown in Figure 1, and d is the capacity and also the output length at choice. Each state bit is addressed by three coordinates, denoted as $S(x, y, z),\ x, y \in \{0, 1, ..., 4\},\ z \in \{0, 1, ..., 63\}$. 2-D entities, plane, sheet and slice, and 1-D entities, lane, column and row, are also defined in Keccak and shown in Figure 1.

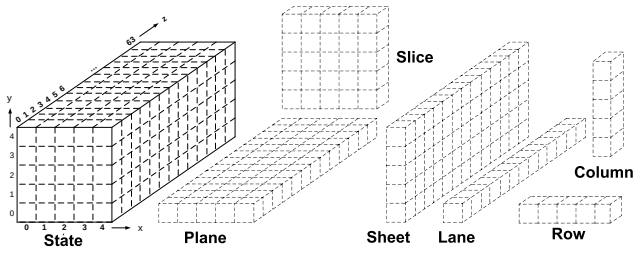

Figure 1: State data structures used in Keccak [31]

We also define vectors $X = [0 : 4]$, $Y = [0 : 4]$ and $Z = [0 : 63]$ to stand for multiple bits in one row, column, and lane, respectively. For example, we can denote the bottom plane of state S (320 bits) as $S(X, 0, Z)$. Note that coordinates x and y are modular of 5 while z is modular of 64.

Keccak relies on a Sponge architecture to iteratively absorb message inputs and squeeze out digests by a f permutation function. Each f function works on a state at a fixed length $b = r + c$. In the squeezing phase, the length of the output is configurable (a multiple of bitrate, r bits).

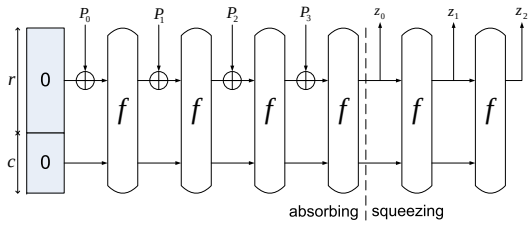

Figure 2: The sponge construction

The f function consists of 24 rounds for 1600-bit operations, where each round has five sequential steps:

$$S_{i+1} = \iota \circ \chi \circ \pi \circ \rho \circ \theta(S_i),\ i \in \{0, 1, \cdots, 23\} \quad (1)$$

in which S_0 is the initial input. Details of each step are described below:

$-$ θ is a linear operation which involves 11 input bits and outputs a single bit. Each output state bit is the XOR between the input state bit and two intermediate bits produced by its two neighbor columns. We denote the input to θ operation as θ_i while the output as θ_o, and the operation is given as follows:

$$\theta_o(x, y, z) = \theta_i(x, y, z) \oplus (\oplus_{y=0}^{4} \theta_i(x-1, y, z)) \oplus (\oplus_{y=0}^{4} \theta_i(x+1, y, z-1)). \quad (2)$$

$-$ ρ is a rotation over the state bits along z-axis (in lanes), and the shift amount of bits depends on the $(x,\ y)$ coordinates.

$-$ π is a rotation over the state bits within slices. Only the corner bit $(x = 0,\ y = 0)$ of the slice does not move. All other bits are permuted to other positions depending on their original coordinates. π can also be viewed as rotation among lanes.

$-$ χ is a non-linear step that contains mixed binary operations over state bits in rows. Each bit of the output state is the result of an XOR between the corresponding input state bit and its two neighboring bits along the x-axis (in a row):

$$\chi_o(x, y, z) = \chi_i(x, y, z) \oplus (\overline{\chi_i(x+1, y, z)} \cdot \chi_i(x+2, y, z)).$$

$-$ ι is a binary XOR with a round constant which is publicly known.

The SHA-3 family consists of four output lengths (d in Keccak-f[1600, d]), called SHA3-224, SHA3-256, SHA3-384, and SHA3-512 [1]. In this paper, we focus on SHA3-224 and SHA3-256, which have shorter output digests (less observable information) than SHA3-384 and SHA3-512, and therefore are more challenging to break by DFA.

For Keccak based crypto systems with the input message length smaller than the bitrate r, there will be only one f permutation function for squeezing and absorption. For such systems, if an internal state of the absorption phase (which is also squeezing phase here) is recovered, the original message and all the other internal states are recovered because the absorption algorithm is reversible [31]. We set our DFA target as recovering the entire internal state χ_i^{22} (1,600 bits) for SHA3-224 and SHA3-256. We annotate the last two rounds of SHA-3 operations and important intermediate states in Figure 3, and use the notations in the rest of this paper.

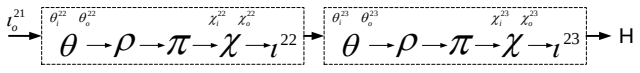

Figure 3: Notations for operations and intermediate states

The DFA takes the observed fault at the output (H), traces it back to certain intermediate state (comparison point) to

have the intermediate fault, which is then compared against all possible faults that are generated by propagation of different original faults injected at θ_i^{22}. In this paper, for basic attacks which will be presented in Section III, the fault injection point is θ_i^{22} and the comparison point is χ_i^{23}. For improved attacks in Section IV, another fault injection point θ_i^{23} and comparison point χ_o^{23} will be used to further improve the attack efficiency.

B. Fault Models in This Paper

In the previous work of DFA on SHA-3 [2], the fault model is single-bit. This is an unrealistic stringent model and not feasible to attain with general fault injection methods, including clock glitches and supply voltage variation. Faults injected by these methods tend to fall on multiple bits at one time, for example, on an 8-bit byte or a 32-bit word. Multiple concurrent bit faults will interfere with each other during operations in the hash algorithm, and considering individual independent single-bit faults only does not address these interactions. In this paper, we adopt relaxed and more realistic fault models for different implementations, and propose a generic fault propagation analysis method.

Different from DFA on block ciphers and stream ciphers [3]–[9], multiple faults are injected for the same input message in the attacks on hash functions. As different message may have different impact on the attack process, for all the simulation in this paper, we generate multiple random messages (i.e., 10^3 random messages) and attack each input message separately, and then combine their results to get an average number as the final simulation result. All the single-byte faults in this paper are randomly generated, with random value (1-255) and random positions (200 bytes). To illustrate our proposed method, we use the fault model of single-byte faults as example:

- The attacker can inject faults into one byte of the last two rounds input θ_i^{22} and θ_i^{23};
- The attacker has no control on either the position (which byte) or the value of the injected faults;
- The attacker can only observe the correct and faulty SHA-3 outputs, H and $H^{'}$, which are d bits for mode SHA3-d (where d is 224, 256, 384, and 512, for the four modes, respectively), instead of the entire 1,600 bits;
- The attacker can inject random faults for the same input message for multiple times.

For commonly used SHA-3 implementation examples, data structures are organized along each lane [32], [33]. Thus one byte is eight consecutive bits in one lane in this paper. We refer to the source code provided online [32] for all implementation and simulation in this paper. We note here that our attack method only requires the faults injected at θ_i^{22} to recover the whole internal state χ_i^{22}, and we propose to inject faults at θ_i^{23} to further improve the attack efficiency in this paper.

III. Basic Differential Fault Analysis of SHA3-224 and SHA3-256

Generally, because of confusion and diffusion properties in crypto operations, any bit flip at the input message will affect all the bits at the output under perfect randomness and the fault analysis would not work. For SHA-3, the path from the fault injection point (θ_i^{22}) to the observable output (H) is not very long - only two rounds of operations, and therefore different faults injected will cause different patterns at the differential output $\Delta H = H \oplus H^{'}$. We call such differential patterns as Fault Signature in this paper. We next discuss the observable information and the fault propagation process.

A. Observable Hash Digest

In [2], the comparison point is picked at θ_o^{23} for SHA3-384 and SHA3-512 to identify the single-bit fault injected. For SHA3-384 and SHA3-512, a whole plane of 320 bits ($y = 0$, the bottom plane) at the output H is observable. Because all the operations ρ, π, χ, and ι are reversible, the attacker can make use of this plane to recover 320 bits of χ_i^{23}:

$$\chi_i^{23}(x, 0, Z) = \chi^{-1} \circ (\iota^{23})^{-1}\big(H(y = 0)\big).$$

Observable five lanes of the bottom plane of H will be used by attackers to retrieve five lanes of $\theta_o^{23}(x, y, Z)$ ($x = y \in \{0, 1, 2, 3, 4\}$). By observing the original and faulty hash output, H and H', the attacker can calculate the corresponding five differential lanes of θ_o^{23} (not in the the same plane any more though, but on the diagonal of each slice):

$$\Delta\theta_o^{23}(x, y = x, Z) = \rho^{-1} \circ \pi^{-1}\Big(\Delta\chi_i^{23}(x, 0, Z)\Big). \quad (3)$$

While ρ and π operations just rotate state bits to different positions without changing their values, $\Delta\chi_i^{23}$ can be directly used instead, and the comparison point will be χ_i^{23} correspondingly. Thus, we need to construct fault signatures at χ_i^{23}, $FS_{\chi_i^{23}}$, for attacks in this paper.

In this paper, we focus on SHA3-224 and SHA3-256, which have not been targeted by DFA yet, and the method proposed in [2] cannot be applied directly to them because of limited number of observable digest bits. In Section III-C, we will show methods to recover part of χ_i^{23} in bottom plane, and illustrate how to use $\Delta\chi_i^{23}$ instead of $\Delta\theta_o^{23}$ as comparison point for DFA attacks in this work.

B. Fault Signature Generation

For the single-byte fault model, any internal state of Keccak-f[1600, d] is composed of 200 bytes ($0 \leq P < 200$), and the fault value (F, the fault dropped on one input state byte of the penultimate round) ranges from 1 to 255, where $F = 1$ means to flip the lowest bit of the corrupted byte while $F = 255$ means to flip all the eight bits. For any possible fault (F) at any one of the 200

positions (P), we denote the corresponding fault signature at χ_i^{23} as $FS_{\chi_i^{23}}[P][F]$. We note here that if without extra specification, all fault signatures are 1,600 bits, standing for the 1,600 differential bits of the state caused by the fault F injected at P.

For faults injected at θ_i^{22}, it will propagate to χ_i^{23} through the operations shown in Figure 3. We separate these operations into two categories:

- Operations that will not change bit values of fault signatures, including bit rotation operations ρ and π that only change the bit positions, and constant number addition operation ι.
- Operations that will change the bit values of fault signatures, which involve multiple bits to generate a single output bit, namely θ and χ. There is also difference between these two operations, θ is linear (only consisting of exclusive OR operations) while χ is non-linear (consisting of binary operations AND and NOT).

In the first kind of operations, for ρ and π, faults at the input will go through the operation (position permutation) directly to propagate to the output, i.e., $\Delta\rho_o = \rho(\Delta\rho_i)$ and $\Delta\pi_o = \pi(\Delta\pi_i)$. For operation ι, the fault does not change at all, i.e., $\Delta\iota_o = \Delta\iota_i$.

For the second kind of operations, one output bit is generated from multiple input bits. For θ^{22} operation, one single-bit fault $\Delta\theta_i^{22}(x, y, z)$ will propagate to 11 bits of θ_o^{22}, with their differential denoted as $\Delta\theta_o^{22}(x, y, z)$, $\Delta\theta_o^{22}(x+1, Y, z)$ and $\Delta\theta_o^{22}(x-1, Y, z+1)$, which are on three different sheets, respectively. For the single-byte fault model, all the faulty bits are in the same lane of θ_i^{22}. With θ operation, no θ_o^{22} bit will involve more than one faulty bit. Thus, for θ^{22}, we have $\Delta\theta_o^{22} = \theta(\Delta\theta_i^{22})$.

In this paper, we use a single-bit fault at $\theta_i^{22}(0, 0, 0)$ ($\Delta\theta_i^{22}(0, 0, 0) = 1$ while all other bits of $\Delta\theta_i^{22}$ are 0) as example to demonstrate the fault propagation in SHA-3, and use it to explain the construction of fault signatures. According to the above analysis of fault propagation through different operations, the single-bit fault will be diffused to 11 bits after θ^{22} operations, and then rotated into different lanes and rows through ρ and π. The fault signature $FS_{\chi_i^{22}}$ at the input of χ^{22} for this single-bit fault is shown in Figure 4.

It is direct forward to construct the fault signatures at χ_i^{22} because of the linear properties of θ, ρ and π operations. Each bit of $FS_{\chi_i^{22}}$ will be either 0 or 1, depending on the value and position of the injected faults only.

To construct the fault signature $FS_{\chi_i^{23}}$, we need to examine the fault propagation process of χ^{22} and θ^{23}. If we denote the fault propagation of χ^{22} as FP_χ, and the fault propagation of θ^{23} as FP_θ, the corresponding fault signature at χ_i^{23} can be denoted as follows (note that operation ι does not change the fault):

$$FS_{\chi_i^{23}} = \pi \circ \rho \circ FP_\theta \circ FP_\chi(\Delta\chi_i^{22}). \tag{4}$$

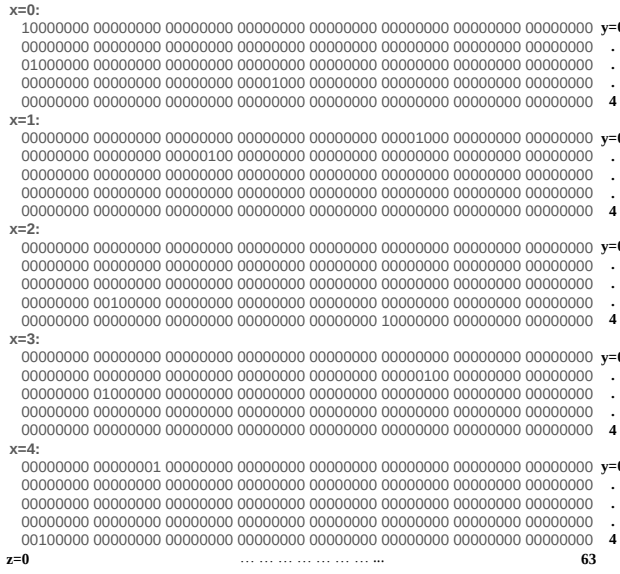

Figure 4: Fault signature at χ_i^{22} for the example single-bit fault injected

We next analyze fault propagation of χ^{22} and θ^{23}.

1) Fault Propagation in χ^{22}: χ is the only nonlinear operation in Keccak, and its bit-wise AND operation leaks information of its input state bits if fault(s) happen in χ_i. Under the single-bit fault model in [2], no more than one bit will be polluted in each row of χ_i^{22}, as also shown in Figure 4 for vectors $\Delta\chi_i^{22}(X, y, z)$. For the relaxed models used in this paper, multiple bits may be polluted in one row of χ_i^{22}. In this section, we present the general fault propagation of multi-bit faults in χ operation.

Denote five bits in one row of χ input as $\{a_i, b_i, c_i, d_i, e_i\}$, then five bits of corresponding χ_o output row can be denoted as $a_o = a_i \oplus (\bar{b_i} \cdot c_i)$, $b_o = b_i \oplus (\bar{c_i} \cdot d_i)$, $c_o = c_i \oplus (\bar{d_i} \cdot e_i)$, $d_o = d_i \oplus (\bar{e_i} \cdot a_i)$ and $e_o = e_i \oplus (\bar{a_i} \cdot b_i)$.

We take a_o as an example to demonstrate the fault propagation in χ operation. Bit a_o is affected by bits a_i, b_i and c_i:

1) With a single-bit fault on a_i ($\Delta a_i = 1$), $\Delta a_o = \Delta a_i = 1$.
2) With a single-bit fault on b_i ($\Delta b_i = 1$), $a'_o = a_i \oplus (\bar{b'_i} \cdot c_i)$, and then $\Delta a_o = \Delta b_i \cdot c_i = c_i$, which leaks the internal state c_i information.
3) With a single-bit fault on c_i ($\Delta c_i = 1$), $a'_o = a_i \oplus (\bar{b_i} \cdot c'_i)$, and then $\Delta a_o = (1 \oplus b_i) \cdot \Delta c_i = \overline{b_i}$.
4) With a two-bit fault on a_i and b_i ($\Delta a_i = \Delta b_i = 1$), $a'_o = a'_i \oplus (\bar{b'_i} \cdot c_i)$, and then $\Delta a_o = \overline{c_i}$.
5) With a two-bit fault on b_i and c_i ($\Delta b_i = \Delta c_i = 1$), $a'_o = a_i \oplus (\bar{b'_i} \cdot c'_i)$, and then $\Delta a_o = \Delta b_i \cdot c_i \oplus (1 \oplus b_i) \cdot \Delta c_i \oplus \Delta b_i \cdot \Delta c_i = b_i \oplus c_i$.
6) With a two-bit fault on a_i and c_i ($\Delta a_i = \Delta c_i = 1$), $a'_o = a'_i \oplus (\bar{b_i} \cdot c'_i)$, and then $\Delta a_o = \Delta a_i \oplus (1 \oplus b_i) \cdot \Delta c_i = b_i$.

Table I: Fault propagation of operation χ^{22}

Fault at χ input $\Delta\chi_i^{22}([x:x+2],y,z)$	Fault signature at χ output $FS_{\chi_o^{22}}(x,y,z)$
[1,0,0]	1
[0,1,0]	$\chi_i^{22}(x+2,y,z)$
[0,0,1]	$1 \oplus \chi_i^{22}(x+1,y,z)$
[1,1,0]	$1 \oplus \chi_i^{22}(x+2,y,z)$
[0,1,1]	$\chi_i^{22}(x+1,y,z) \oplus \chi_i^{22}(x+2,y,z)$
[1,0,1]	$\chi_i^{22}(x+1,y,z)$
[1,1,1]	$1 \oplus \chi_i^{22}(x+1,y,z) \oplus \chi_i^{22}(x+2,y,z)$

7) With a three-bit fault ($\Delta c_i = \Delta b_i = \Delta c_i = 1$), $a'_o = a'_i \oplus (\bar{b'_i} \cdot c'_i)$, and thus $\Delta a_o = \Delta a_i \oplus \Delta b_i \cdot c_i \oplus (1 \oplus b_i) \cdot \Delta c_i \oplus \Delta b_i \cdot \Delta c_i = \overline{b_i \oplus c_i}$.

In summary, we can denote the fault signature for bit $\chi_o^{22}(x,y,z)$ as in Table I. According to the above analysis, we present the whole fault patterns at χ_o^{22} as in Figure 5, in which $\Delta\chi_o^{22}(x,y,z)$ is denoted as $C(x,y,z)$ for simplicity, and the same single-bit fault $\Delta\theta_i^{22}(0,0,0) = 1$ example is assumed.

10000000 00000000 00000000 00000000 00000000 0000x000 00000000 00000000
00000000 00000000 00000x00 00000000 00000000 00000000 00000000 00000000
01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00x00000 00000000 00001000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 x0000000 00000000 00000000
$C(0,0,44) = \chi_i^{22}(2,0,44)$; $C(0,1,21) = \chi_i^{22}(2,1,21)$; $C(0,3,10) = 1 \oplus \chi_i^{22}(1,3,10)$; $C(0,4,40) = 1 \oplus \chi_i^{22}(1,4,40)$

00000000 00000000 00000000 00000000 00000000 00001000 00000000 00000000
00000000 00000000 00000100 00000000 00000000 00000x00 00000000 00000000
00000000 0x000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00x00000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 x0000000 00000000 00000000
$C(1,1,45) = 1 \oplus \chi_i^{22}(2,1,45)$; $C(1,2,9) = 1 \oplus \chi_i^{22}(2,2,9)$; $C(1,3,10) = \chi_i^{22}(3,3,10)$; $C(1,4,40) = \chi_i^{22}(3,4,40)$

00000000 0000000x 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000x00 00000000 00000000
00000000 0x000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00100000 00000000 00000000 00000000 00000000 00000000 00000000
00x00000 00000000 00000000 00000000 00000000 10000000 00000000 00000000
$C(2,0,15) = 1 \oplus \chi_i^{22}(3,0,15)$; $C(2,1,45) = \chi_i^{22}(4,1,45)$; $C(2,2,9) = \chi_i^{22}(4,2,9)$; $C(2,4,2) = 1 \oplus \chi_i^{22}(3,4,2)$

x0000000 0000000x 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000100 00000000 00000000
0x000000 01000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 0000x000 00000000 00000000 00000000 00000000
00x00000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
$C(3,0,0) = 1 \oplus \chi_i^{22}(4,0,0)$; $C(3,0,15) = \chi_i^{22}(0,0,15)$; $C(3,2,1) = 1 \oplus \chi_i^{22}(4,2,1)$;
$C(3,3,28) = 1 \oplus \chi_i^{22}(4,3,28)$; $C(3,4,2) = \chi_i^{22}(0,4,2)$
x0000000 00000001 00000000 00000000 00000000 0000x000 00000000 00000000
00000000 00000000 00000x00 00000000 00000000 00000000 00000000 00000000
0x000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 0000x000 00000000 00000000 00000000 00000000
00100000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
$C(4,0,0) = \chi_i^{22}(1,0,0)$; $C(4,0,44) = 1 \oplus \chi_i^{22}(0,0,44)$; $C(4,1,21) = 1 \oplus \chi_i^{22}(0,1,21)$;
$C(4,2,1) = \chi_i^{22}(1,2,1)$; $C(4,3,28) = \chi_i^{22}(1,3,28)$

Figure 5: Fault signature at the output of χ^{22}

In Figure 5, each differential bit $\Delta\chi_o^{22}(x,y,z)$ takes a value of '0', '1' or 'x', in which 1 (0) means this corresponding output bit flips (does not flip) with the specific fault injected, respectively, regardless of the internal states. However, 'x' at a bit position means that the corresponding $\Delta\chi_o^{22}$ bit value depends on some χ_i^{22} bit(s), and it can be '0' or '1'. For example, we denote $\Delta\chi_o^{22}(0,0,44)$ as 'x,' because $\Delta\chi_o^{22}(0,0,44) = \chi_i^{22}(2,0,44)$ under the fault injected ($\Delta\theta_i^{22}(0,0,0) = 1$), and $\chi_o^{22}(0,0,44)$ would flip if $\chi_i^{22}(2,0,44) = 1$, otherwise it remains unchanged if $\chi_i^{22}(2,0,44) = 0$. Thus, if the attacker has knowledge of $\Delta\chi_o^{22}(0,0,44)$ and the injected fault, he can construct the corresponding fault signature and then recover bit $\chi_i^{22}(2,0,44)$.

2) Fault Propagation in θ^{23}: Each bit of θ_o is the XOR of itself with two near columns. As shown in Section III-B1, each bit of $\Delta\chi_o^{23}$ can be denoted as 0, 1 or the XOR of χ_i^{22} bits. While θ involves only XOR operation for the 11 input bits, we can denote $\Delta\theta_o^{23}(x,y,z)$ as follows:

$$\begin{aligned}\Delta\theta_o^{23}(x,y,z) =& \Delta\theta_i^{23}(x,y,z) \oplus (\oplus_{y=0}^{4}\Delta\theta_i^{23}(x-1,y,z)) \\ &\oplus (\oplus_{y=0}^{4}\Delta\theta_i^{23}(x+1,y,z-1)).\end{aligned} \tag{5}$$

Thus the fault propagation function FP_θ can be denoted as follows:

$$FS_{\theta_o^{23}} = \theta(FS_{\chi_o^{22}}). \tag{6}$$

For each bit of $\Delta\theta_o^{23}$, some of the corresponding 11 $\Delta\theta_i^{23}$ bits may depend on the same χ_i^{22} bits, and therefore with the operation of XOR some dependencies will be eliminated. This is a key insight for our fault propagation analysis. For example, in the interleaved implementation [34], when fault $F = 65$ is injected at $P = 16$, $\Delta\theta_i^{23}(4,4,3) = \chi_i^{22}(0,4,3)$ and $\Delta\theta_i^{23}(3,4,3) = \chi_i^{22}(0,4,3)$. $\Delta\theta_o^{23}(4,4,3)$, which involves the two input bits $\theta_i^{23}(4,4,3)$ and $\theta_i^{23}(3,4,3)$, will not depend on $\chi_i^{22}(0,4,3)$ anymore because the dependencies get canceled out by XOR between the two input bits. Eventually, the fault signature at the θ^{23} output, $FS_{\theta_o^{23}}$, has a similar format as $FS_{\chi_o^{22}}$, with each bit being 0, 1, or an odd or even function (XOR) over some χ_i^{22} bits and constant one.

As $\Delta\chi_i^{23} = \pi \circ \rho(\Delta\theta_o^{23})$, it is easy for us to build the fault signature at χ_i^{23} with $FS_{\theta_o^{23}}$ constructed based on the above analysis, thus we show $FS_{\chi_i^{23}}$ directly here. We use the same example to show how the single-bit fault at

$\theta_i^{22}(0,0,0)$ propagates to χ_i^{23}. For SHA3-224 and SHA3-256, only partial bottom plane (less than 320 bits) of the output state H will be observable. Nevertheless Figure 6 presents the fault signature in the whole bottom plane of $FS_{\chi_i^{23}}$, in which we denote $\Delta\chi_i^{23}(x,0,z)$ as $E(x,z)$ for simplicity.

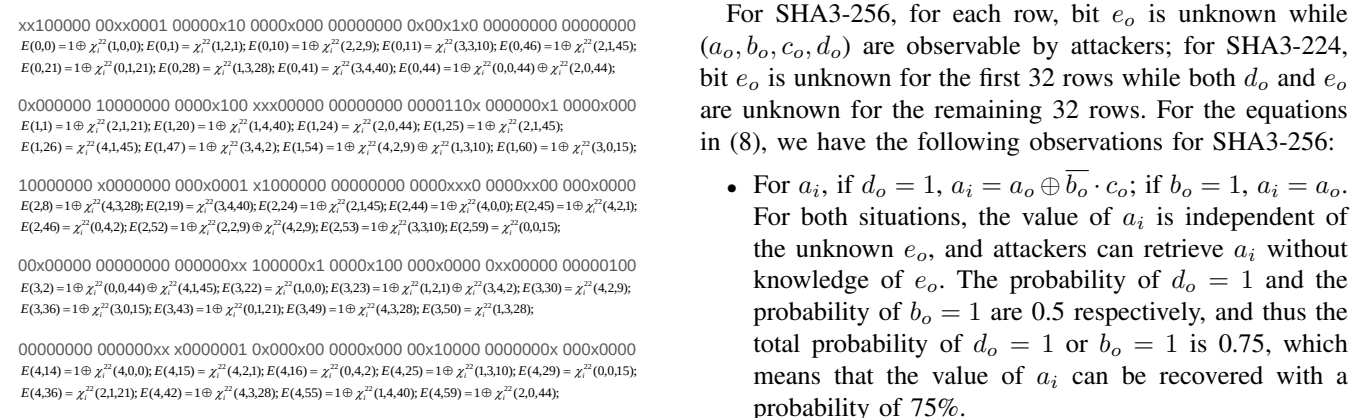

Figure 6: Fault signature at χ_i^{23} (Bottom plane)

With the observed bits of $\Delta\chi_i^{23}$ and the fault signatures, attackers can work on equations which involve only one bit of χ_i^{22} to recover the χ_i^{22} bits, and then plug them back into equations which involve more than one χ_i^{22} bit to recover the remaining χ_i^{22} bits. For example, as shown in Figure 6, with the single-bit fault injected at $\theta_i^{22}(0,0,0)$, attackers can use $FS_{\chi_i^{23}}(1,0,24)$ to recover $\chi_i^{22}(2,0,44)$ first. Then replace $\chi_i^{22}(2,0,44)$ in $FS_{\chi_i^{23}}(0,0,44)$ to recover $\chi_i^{22}(0,0,44)$.

C. χ_i^{23} Bits Recovery from the Observable Digest

For SHA3-224 and SHA3-256, only partial bottom plane of the hash output is observable, i.e., no more than four bits in each row of χ_o^{23} on the bottom plane are known. The χ operation is reversible, and therefore each χ_i bit can be expressed in below formula which involves all five bits of χ_o [31], [35]:

$$\chi_i(x,y,z) = \chi_o(x,y,z) \oplus \overline{\chi_o(x+1,y,z)} \cdot \big(\chi_o(x-1,y,z) \oplus \chi_o(x+2,y,z) \oplus \chi_o(x-1,y,z) \cdot \chi_o(x+3,y,z)\big). \quad (7)$$

Since not all the χ_o bits are know, the attacker cannot get the corresponding χ_i^{23} bits for SHA3-224 and SHA3-256 directly. In this section, we show that with limited information, part of χ_i^{23} on the bottom plan can still be recovered from the observable output.

1) Recover χ_i^{23} Bits in Theory: For simplicity, we use one row in χ^{23} operation as an example here. We express the input bits $(a_i, b_i, c_i, d_i, e_i)$ as functions over the output bits $(a_o, b_o, c_o, d_o, e_o)$ as:

$$\begin{cases} a_i = a_o \oplus \overline{b_o} \cdot \big(e_o \oplus c_o \oplus e_o \cdot d_o\big) \\ b_i = b_o \oplus \overline{c_o} \cdot \big(a_o \oplus d_o \oplus a_o \cdot e_o\big) \\ c_i = c_o \oplus \overline{d_o} \cdot \big(b_o \oplus e_o \oplus b_o \cdot a_o\big) \\ d_i = d_o \oplus \overline{e_o} \cdot \big(c_o \oplus a_o \oplus c_o \cdot b_o\big) \\ e_i = e_o \oplus \overline{a_o} \cdot \big(d_o \oplus b_o \oplus d_o \cdot c_o\big) \end{cases}. \quad (8)$$

For SHA3-256, for each row, bit e_o is unknown while (a_o, b_o, c_o, d_o) are observable by attackers; for SHA3-224, bit e_o is unknown for the first 32 rows while both d_o and e_o are unknown for the remaining 32 rows. For the equations in (8), we have the following observations for SHA3-256:

- For a_i, if $d_o = 1$, $a_i = a_o \oplus \overline{b_o} \cdot c_o$; if $b_o = 1$, $a_i = a_o$. For both situations, the value of a_i is independent of the unknown e_o, and attackers can retrieve a_i without knowledge of e_o. The probability of $d_o = 1$ and the probability of $b_o = 1$ are 0.5 respectively, and thus the total probability of $d_o = 1$ or $b_o = 1$ is 0.75, which means that the value of a_i can be recovered with a probability of 75%.
- For b_i, if $a_o = 0$, $b_i = b_o \oplus \overline{c_o} \cdot d_o$; if $c_o = 1$, $b_i = b_o$. Similarly, the probability of recovering b_i with unknown e_o is also 0.75.
- For c_i, if $d_o = 1$, $c_i = c_o$, thus the probability of recovering c_i is 0.5.
- For d_i, if $c_o \oplus a_o \oplus c_o \cdot b_o = 0$, $d_i = d_o$, thus the probability of recovering d_i is 0.5.
- The value of e_i always depends on e_o, thus the attackers cannot retrieve e_i without knowledge of e_o.

In conclusion, for SHA3-256, the attacker can recover the bits in the first and second lanes of the bottom plan of χ_i^{23} with 0.75 probability, and the bits in the third and fourth lane with 0.5 probability. In total, the attackers can recover 160 bits of χ_i^{23} theoretically. Similarly, for SHA3-224, attackers can use the same method to recover 112 bits of χ_i^{23} theoretically.

2) A Practical Method to Recover χ_i^{23} Bits: In the previous section, we analyze that the attacker can recover 160 bits of χ_i^{23} for SHA3-256 and 112 bits of χ_i^{23} for SHA3-224 theoretically. In this section, we present a practical method to recover χ_i^{23} bits which can be easily implemented.

We still use the row with input $(a_i, b_i, c_i, d_i, e_i)$ and output $(a_o, b_o, c_o, d_o, e_o)$ in SHA3-256 as an example here. While a_o, b_o, c_o, d_o are observable by attackers, e_o can only be either 0 or 1, then we can make assumptions of both situations and write them as $row_o^0 = \{a_o, b_o, c_o, d_o, 0\}$ and $row_o^1 = \{a_o, b_o, c_o, d_o, 1\}$. For both situations, we can calculate the input row_i^0, row_i^1 using χ inversion operation:

$$\begin{cases} \{a_i^0, b_i^0, c_i^0, d_i^0, e_i^0\} = \chi^{-1}(\{a_o, b_o, c_o, d_o, 0\}) \\ \{a_i^1, b_i^1, c_i^1, d_i^1, e_i^1\} = \chi^{-1}(\{a_o, b_o, c_o, d_o, 1\}) \end{cases}. \quad (9)$$

Take bit a_i as an example here, the value of a_i can only be a_i^0 or a_i^1:

1) If $a_i^0 = a_i^1$, then the value of a_i does not depend on the value of e_o and this is the correct recovered value for a_i;
2) If $a_i^0 \neq a_i^1$, then the value of a_i depends on the value of e_o, and attacker cannot recover a_i in this situation.

To verify the above algorithm, we implement both SHA3-224 and SHA3-256 in C++ and randomly generate 10^5 input messages for each of them. We use the proposed algorithm to recover the χ_i^{23} bits for both SHA3-224 and SHA3-256. Results show that the proposed algorithm can correctly recover 160.12 bits of χ_i^{23} for SHA3-256 and 111.84 bits of χ_i^{23} for SHA3-224 on average for these 10^5 trials, which are the same as the theoretical results given in the previous section.

Using the above method, the attacker can recover part of the χ_i^{23} bits in the bottom plane from the original digest H, and faulty $\chi_i'^{23}$ bits for faulty digest H'. Using the recovered $\chi_i^{23}(X,0,Z)$ and $\chi_i'^{23}(X,0,Z)$, the attacker can calculate the corresponding $\Delta\chi_i^{23}(X,0,Z)$ bits. Note that here the attacker can recover 160 (112) bits of both $\chi_i^{23}(X,0,Z)$ and $\chi_i'^{23}(X,0,Z)$ for SHA3-256 (SHA3-224) on average, but the recovered χ_i^{23} and $\chi_i'^{23}$ may have different locations, and therefore the attackers will recover fewer than 160 (112) bits of $\Delta\chi_i^{23}(X,0,Z)$ instead. The simulation results show that attacker can recover 136.42 bits of $\Delta\chi_i^{23}$ for SHA3-256, and 93.68 bits for SHA3-224 on average for 10^5 trials.

D. Injected Fault Identification and χ_i^{22} Bits Recovery

Using the previous algorithms, the attacker can construct fault signatures $FS_{\chi_i^{23}}$ for all injected faults, and recover some bits of $\Delta\chi_i^{23}(X,0,Z)$ from the observable output. In this section, we present the algorithms to use the above information to identify the injected faults and recover χ_i^{22} bits.

For the recovered $\Delta\chi_i^{23}(X,0,Z)$ bits, we separate them into two groups:

- $\Delta\chi_i^{23}.white$ contains the recovered bit positions (x,y,z) of $\Delta\chi_i^{23}$ with $\Delta\chi_i^{23}(x,y,z)=0$, which means the bits at these positions are not flipped;
- $\Delta\chi_i^{23}.black$ contains the recovered bit positions (x,y,z) of $\Delta\chi_i^{23}$ with $\Delta\chi_i^{23}(x,y,z)=1$, which means the bits at these positions are flipped.

For these recovered bit positions, we check the corresponding fault signatures at $FS_{\chi_i^{23}}[P][F](x,y,z)$ for fault F injected at position P at the penultimate round input. We can separate them into three groups:

- $FS_{\Delta\chi_i^{23}}[P][F].white$ contains the bit positions (x,y,z) with $\Delta\chi_i^{23}(x,y,z)$ recovered and $FS_{\chi_i^{23}}[P][F](x,y,z)=0$, i.e., the injected fault does not affect these output bits.
- $FS_{\chi_i^{23}}[P][F].black$ contains the bit positions (x,y,z) with $\Delta\chi_i^{23}(x,y,z)$ recovered and $FS_{\chi_i^{23}}[P][F](x,y,z)=1$, which are for sure to flip when the fault is injected.
- $FS_{\chi_i^{23}}[P][F].grey$ contains the bit positions (x,y,z) with $\Delta\chi_i^{23}(x,y,z)$ recovered and $FS_{\chi_i^{23}}[P][F](x,y,z)$ is a function dependent on some bits of χ_i^{22}, i.e., they can leak some internal state bits information.

For the correct fault F_0 injected at the correct position P_0, the following relations should hold:

- For bit in $FS_{\chi_i^{23}}[P_0][F_0].white$, this bit should not flip for sure, then this bit should be in $\Delta\chi_i^{23}.white$;
- For bit in $FS_{\chi_i^{23}}[P_0][F_0].black$, this bit should flip for sure, then this bit should be in $\Delta\chi_i^{23}.black$;
- For any bit in $FS_{\chi_i^{23}}[P_0][F_0].grey$, it can be in $\Delta\chi_i^{23}.white$ or $\Delta\chi_i^{23}.black$, depending on some internal state bits.

We can summarize the above relations as follows:

$$\begin{cases} FS_{\chi_i^{23}}[P][F].white \subseteq \Delta\chi_i^{23}.white \\ FS_{\chi_i^{23}}[P][F].black \subseteq \Delta\chi_i^{23}.black \\ \Delta\chi_i^{23}.white \subseteq \{FS_{\chi_i^{23}}[P][F].white \cup FS_{\chi_i^{23}}[P][F].grey\} \\ \Delta\chi_i^{23}.black \subseteq \{FS_{\chi_i^{23}}[P][F].black \cup FS_{\chi_i^{23}}[P][F].grey\} \end{cases} \tag{10}$$

By checking relationships in (10), attackers can exclude many positions and fault values. If only one position with one fault value satisfies these relationship, the injected fault is discovered. All the $FS_{\chi_i^{23}}[P_0][F_0].grey$ bits now are mapped to either zero (white) or one (black) in the observed differentials, and therefore the internal state bits can be recovered.

We simulate the injected fault identification algorithm for both SHA3-224 and SHA3-256. We randomly generate 10^4 input messages and randomly inject 10^3 single-byte faults into the penultimate round input θ_i^{22} for each message. Results show that for SHA3-256, with a probability of 66.61% the attacker can find a unique fault that satisfies the above relations. With the rest 33.39% probability, more than one faults satisfy the above relations and the attacker cannot precisely identify the injected fault. The results are shown in Table II.

Table II: Simulation results for SHA3-224 and SHA3-256 with fault injected at θ_i^{22}

	Number of Recovered		Probability of unique fault
	χ_i^{23}	$\Delta\chi_i^{23}$	
SHA3-224	111.84	93.68	30.67%
SHA3-256	160.12	136.42	66.61%

In this paper, we only make use of the injected faults which can be identified uniquely based on the proposed algorithm, and discard those faults that attacker has ambiguity - recovering more than one faults that satisfy the relationship in (10). We define such unique faults as effective faults, and a higher percentage of effective faults will make the attacks more efficient.

With the injected fault identified, including the fault value F and injected position P, we next show the results of

$$FS_{\chi_o^{23}}(x,y,z) = FS_{\chi_i^{23}}(x,y,z) \oplus FS_{\chi_i^{23}}(x+1,y,z) \cdot \chi_i^{23}(x+2,y,z) \oplus (1 \oplus \chi_i^{23}(x+1,y,z)) \cdot FS_{\chi_i^{23}}(x+2,y,z) \oplus FS_{\chi_i^{23}}(x+1,y,z) \cdot FS_{\chi_i^{23}}(x+2,y,z) \quad (11)$$

recovering χ_i^{22} bits in this section. Once the unique fault value at a certain position is identified, all the bits in the $FS_{\chi_i^{23}}$ are known to be zero or one. Using the method described in Section III-B2, we can recover all the χ_i^{22} bits based on the equations constructed on the x bits of $FS_{\chi_i^{23}}$.

We use similar simulation settings as previous section, and we simulate not only SHA3-224 and SHA3-256, but also compare their results with SHA3-384/512. The results for attacking four SHA-3 modes are shown in Figure 7, where the x-axis is the number of effective injected faults that attackers can identify a unique fault, while the y-axis is the total number of recovered χ_i^{22} bits. Results show that the proposed scheme can recover the χ_i^{22} bits, but the attack is much less efficient than the attack on SHA3-384/512. This is because much fewer bits of $FS_{\chi_i^{23}}$ and $\Delta\chi_i^{23}$ are available for SHA3-224 and SHA3-256 than SHA3-384/512. In next section, we propose effective methods to improve the proposed attacks.

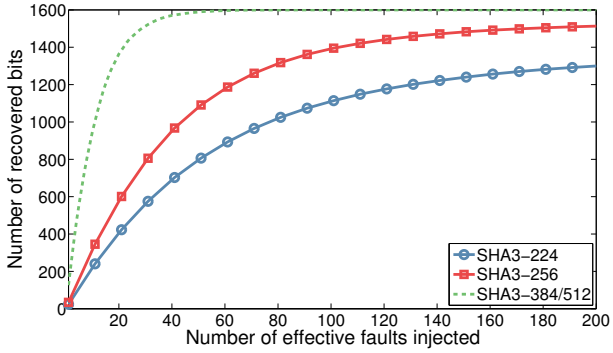

Figure 7: Number of recovered χ_i^{22} bits for different number of injected faults

IV. Improved Attacks

In previous section, we present how to use fault signatures at state χ_i^{23} to recover internal state χ_i^{22} bits for SHA3-224 and SHA3-256, with much less efficiency than SHA3-384/512. In this section, we propose two methods to improve the efficiency of the attacks:

- Propagate the faults further to generate fault signatures $FS_{\chi_o^{23}}$, and use them in addition to $FS_{\chi_i^{23}}$ to improve the attack. This method does not need any extra information (like state bits etc.) from the target system.
- Inject faults at the last round input θ_i^{23} to recover more bits of χ_i^{23} on the bottom plane, and thus to improve the number of available $\Delta\chi_i^{23}$ and $FS_{\chi_o^{23}}$ bits.

We note here that without using the second improvement method, the attacker can extract the whole internal state of χ_i^{22} by using the method proposed in Section III, and the attack efficiency can be further improved by applying the first improvement method mentioned above. We present the second improvement method here which requires to inject faults at the beginning of the last round input θ_i^{23} to show the vulnerability of SHA-3 based systems more thoroughly, and give attackers another choice to improve differential fault analysis. Next we present these two methods in detail in following sections.

A. Attacks Using both $FS_{\chi_i^{23}}$ and $FS_{\chi_o^{23}}$

Fault signature at χ_o^{23}, $FS_{\chi_o^{23}}$, can leak information not contained in $FS_{\chi_i^{23}}$. Combining $FS_{\chi_o^{23}}$ with $FS_{\chi_i^{23}}$, the attacker should be able to extract more information than using only $FS_{\chi_i^{23}}$.

1) Fault Signature $FS_{\chi_o^{23}}$ Generation: For any fault on the three bits a_i, b_i, and c_i, the χ^{23} operation propagates it onto the output bit, $a_o = a_i \oplus (\bar{b}_i \cdot c_i)$. Similar to the χ^{22} fault propagation, there are several types of possible faults on the three input bits. However, for any χ_i^{22}, it can only be 0 or 1 (independent of the internal state bits but only dependent on the fault) when faults are injected at the penultimate round input. While for any χ_i^{23} bit, it can be 0, 1 or x (as a function over certain χ_i^{22} bits).

While a_i, b_i and c_i can be all faulty, we can denote a'_o as $a'_i \oplus (\bar{b}'_i \cdot c'_i)$, then:

$$\Delta a_o = \Delta a_i \oplus \Delta b_i \cdot c \oplus (1 \oplus b_i) \cdot \Delta c_i \oplus \Delta b_i \cdot \Delta c_i. \quad (12)$$

Thus $FS_{\chi_o^{23}}(x,y,z)$ can be denoted as (11). Each bit of $FS_{\chi_i^{23}}$ can be denoted as 0, 1 or the operations of χ_i^{22} bits, thus $FS_{\chi_o^{23}}(x,y,z)$ can also be denoted as either 0, 1 or the operations of χ_i^{22} bits. There are some special case for the construction of Δa_o in (12). When there are two faulty bits:

1) If $\Delta c_i = 0$, $\Delta a_i \neq 0$ and $\Delta b_i \neq 0$, then $\Delta a_o = \Delta a_i \oplus \Delta b_i \cdot c_i$.
2) If $\Delta a_i = 0$, $\Delta b_i \neq 0$ and $\Delta c_i \neq 0$, then $\Delta a_o = \Delta b_i \cdot c_i \oplus (1 \oplus b_i) \cdot \Delta c_i \oplus \Delta b_i \cdot \Delta c_i$.
3) If $\Delta b_i = 0$, $\Delta a_i \neq 0$ and $\Delta c_i \neq 0$, then $\Delta a_o = \Delta a_i \oplus (1 \oplus b_i) \cdot \Delta c_i$.

If only one bit is faulty for a_i, b_i and c_i, the construction of Δa_i can be further simplified as follows:

1) If $\Delta b_i = \Delta c_i = 0$ and $\Delta a_i \neq 0$ ($\Delta a_i = 1$ or $\Delta a_i =$x), $\Delta a_o = \Delta a_i$, and the construction of Δa_o does not require knowledge of b_i and c_i.
2) If $\Delta a_i = \Delta c_i = 0$ and $\Delta b_i \neq 0$, $\Delta a_o = \Delta b_i \cdot c_i$, and the construction of Δa_o requires knowledge of c_i.

3) If $\Delta a_i = \Delta b_i = 0$ and $\Delta c_i \neq 0$, $\Delta a_o = (1 \oplus b_i) \cdot \Delta c_i$.

As demonstrated in Section III-C, we can recover some bits of χ_i^{23} using the observable digest H, thus we can construct fault signatures for some bits of χ_o^{23} based on the above analysis.

Use the same method as Section III-D, we can separate the constructed fault signature $FS_{\chi_o^{23}}[P][F]$ into three groups, definitely 0 (white), definitely 1 (black), or dependent on some input bits and/or input faults (grey). For SHA3-224, 224 bits of $\Delta\chi_o^{23}$ are available, 256 bits of $\Delta\chi_o^{23}$ are available for SHA3-256. We can use the same relationships for $\Delta\chi_i^{23}$ and $FS_{\chi_i^{23}}$ as shown in (10) to build relationships for $\Delta\chi_o^{23}$ and $FS_{\chi_o^{23}}$, and combine it with (10) to improve the effective fault identification rate.

2) Simulation Results: We run simulations for the improved attacks, and results show that the probability of identifying the unique injected faults rises significantly for both SHA3-224 and SHA3-256, from 30.67% to 49.12% for SHA3-224 and from 53.28% to 78.73% for SHA3-256.

We inject multiple random single-byte faults to extract all the $1,600$ χ_i^{22} bits for SHA3-224 and SHA3-256 using the proposed improved attacks, still comparing with the attack result of SHA3-384/512. The results are shown in Figure 8.

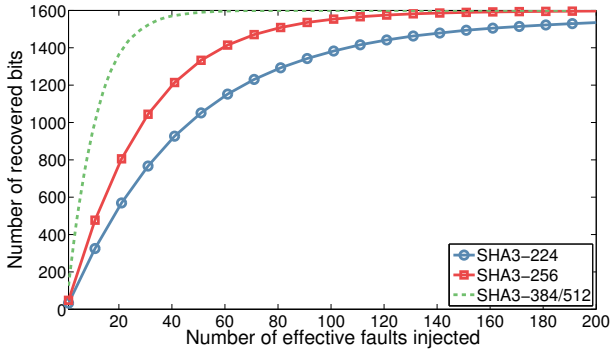

Figure 8: Number of recovered χ_i^{22} bits for different number of injected faults

Compared with the original attack and its results in Figure 7, the proposed method in this section improves the attack efficiency significantly for both SHA3-224 and SHA3-256. For SHA3-224, the attack method in Section III needs about 200 faults to recover $1,300$ bits of χ_i^{22}, while the improved method in this section only needs 82 faults. Similarly, for SHA3-256, the number of faults needed reduces from 200 to 80 to recover about $1,510$ bits of χ_i^{22}.

Note this improvement method does not need any extra information extracted from the target system, and generating fault signature $FS_{\chi_o^{23}}$ does not require much computation effort, making it suitable for real attacks.

B. Improved Attacks by Injecting Faults in θ_i^{23}

For the method proposed in Section IV-A, the unique fault identification rate and the number of recovered χ_i^{22} bits by using the same number of effective injected faults are still lower than attacks on SHA3-384/512. The reason lies in the fact that attackers can recover less number of χ_i^{23} and $\Delta\chi_i^{23}$ bits for SHA3-224 and SHA3-256 than SHA3-384/512. In this section, we propose to improve the attacks on SHA3-224 and SHA3-256 by injecting faults into the last round input to recover more χ_i^{23} bits from the correct hash digest, and thus to improve the number of available $\Delta\chi_i^{23}$ and $FS_{\chi_o^{23}}$ bits for attack.

1) Recovering more χ_i^{23} by Injecting Faults into θ_i^{23}: To recover χ_i^{23} bits by injecting faults at θ_i^{23}, we need to calculate the fault propagation from θ_i^{23} to χ_o^{23}. These faults will propagate through θ^{23}, ρ, π and χ operations. The fault propagation process is exactly the same as in the penultimate round (from θ_i^{22} to χ_o^{22}) as presented in Section III-B1. We denote the fault signature at χ_o^{23} for faults injected at θ_i^{23} as $FS'_{\chi_o^{23}}$ in this section.

Using the faults injected at θ_i^{23}, attackers can recover some bits of χ_i^{23} with a shorter distance between the hash digest and the comparison point (χ_o^{23}) for differential fault analysis. Note here that for SHA3-256 and the first 32 rows of SHA3-224 (with four bits out of five bits of each output row on the bottom plane known), if the attacker recovers one bit $\chi_i^{23}(x, 0, z)$ that has not been recovered using the algorithm in Section III-C, he can recover all the other unknown bits in this row. For example, we assume $a_i^0 \neq a_i^1$ in (9) and this bit has been recovered by injecting faults at θ_i^{23}, then the attackers can know which assumption of e_o is correct, and then recover all the five bits in this row. This method can be used for all 64 rows in the bottom plane of SHA3-256 and the first 32 rows of SHA3-224. In SHA3-224, for the remaining rows with two bits unknown, $\chi_i^{23}(X, 0, z)$, $32 \leq z < 63$, these two bits can only be recovered by injecting faults at θ_i^{23} separately.

To identify the injected faults, we use both fault signatures at χ_i^{23} and χ_o^{23}, denoted as $FS'_{\chi_i^{23}}$ and $FS'_{\chi_o^{23}}$. We randomly generate 10^3 messages, and for each message randomly inject 1000 faults at θ_i^{23} for attacks. For both SHA3-224 and SHA3-256, we can identify the correct fault injected at θ_i^{23} with about 20% probability. After identifying the correct fault injected at θ_i^{23}, we can recover all bits in the bottom plane of χ_i^{23}, and we present the results in Figure 9.

It shows that for SHA3-256, the attacker can recover about 244 bits of χ_i^{23} using only five effective faults, compared with 160 bits recovered when no faults injected at θ_i^{23}. We note here that the attacker does not need to recover all the bits of χ_i^{23} in the bottom plane for attacks, he can recover part of χ_i^{23} to improve the efficiency of attacking χ_i^{22}. We will show how the attacks on χ_i^{22} change with the number of χ_i^{23} bits recovered in next section.

2) Attack Results: With more fault-free χ_i^{23} bits, attackers can construct more bits of $\Delta\chi_i^{23}$ and $FS_{\chi_o^{23}}$ and use them for attacks. For simplicity, we make the following assumptions

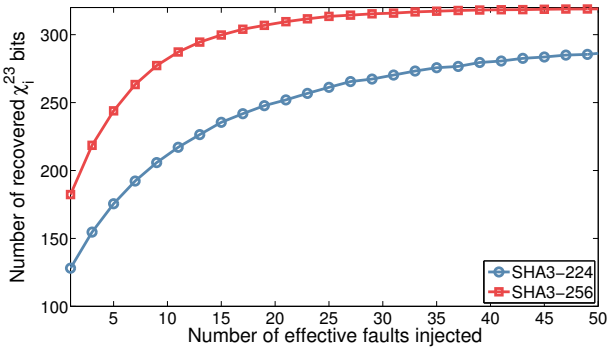

Figure 9: Reovery of χ_i^{23} bits by injecting faults at θ_i^{23}

for attackers:

- Attackers can recover part of χ_i^{23} bits using algorithm presented in Section III-C (with no fault injected at θ_i^{23}).
- Attackers can inject faults at θ_i^{23} to recover the remaining bits of χ_i^{23}, and we assume that the attackers can recover one row using the recovered bits in this row for all the rows of both SHA3-224 and SHA3-256 for simplicity.

We randomly generate 10^3 input message, for each message, we use the algorithm in Section III-C to recover part of χ_i^{23} bits first. For each message, we randomly recover from 0 to 64 rows of χ_i^{23} for 10^3 trials. In each trial, we inject 1000 random faults at θ_i^{22} to calculate the fault identification rate and show the results in Figure 10.

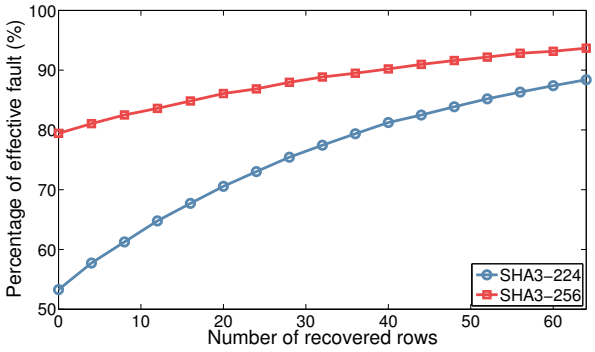

Figure 10: Percentage of unique faults identification with different number of χ_i^{23} rows recovered

Figure 10 shows that with more rows of χ_i^{23} recovered, attackers can identify the randomly injected faults at θ_i^{22} with higher rates. For example, for SHA3-224, the fault identification rate is 53.28% when no rows are recovered by injecting faults at θ_i^{23}, and it rises to 88.34% when all 64 rows (320 bits) are recovered. Similarly, for SHA3-256, this rate rises from 78.73% to 93.67%.

It means that by recovering extra χ_i^{23} bits, the faults injected at θ_i^{22} can be identified with much higher probability. Take SHA3-224 for an example, the attacker may need to inject about 375 faults at θ_i^{22} to get only 200 effective faults which can be uniquely identified, but he will need only to inject about 226 faults instead after he has knowledge of all the bits of θ_i^{23} in the bottom plane.

With knowledge of more χ_i^{23} bits, the attacker can build more equations like in Figure 6 with more $\Delta\chi_i^{23}$ and $FS_{\chi_o^{23}}$ bits, then attacker can recover more χ_i^{22} bits for each injected fault on average. To verify the assumption, we assume the attacker can recover from 0 to 64 rows of χ_i^{23}, and we run attacks on SHA3-224 and SHA3-256 to recover all the bits of χ_i^{22}. We present the attack results on SHA3-224 with different numbers of rows recovered in Figure 11.

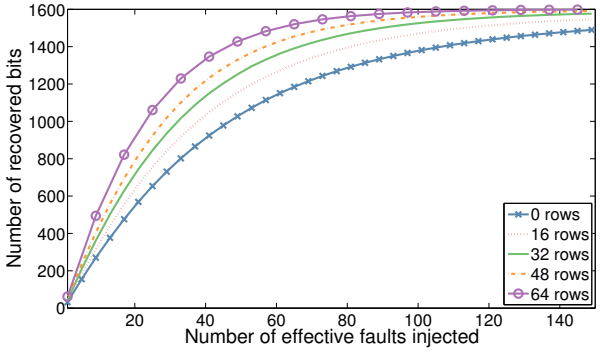

Figure 11: Number of recovered χ_i^{22} bits for different number of injected faults with a number of χ_i^{23} rows recovered

Figure 11 shows that attacker needs smaller number of effective faults to recover all the bits of χ_i^{22} if he has recovered more rows of χ_i^{23}. For example, if he has full knowledge of the bottom plane of χ_i^{23}, he can recover 1590 bits of χ_i^{22} using 110 random faults on average. For attacker who cannot inject fault into the last round input, using the improved method in Section IV-A1, he can recover about only $1,412$ bits using 110 injected faults. For SHA3-256, the results are similar, and we will not present the details here.

C. Discussions and Future Work

In this paper, we propose a method to use differential fault analysis to break SHA3-224 and SHA3-256, and then present two methods to improve the attacks. The first improved method in Section IV-A requires no extra knowledge of the target system, while the method proposed in Section IV-B requires to inject faults into θ_i^{23} to recover extra χ_i^{23} bits in the bottom plane.

Take SHA3-224 as an example here, the fault identification rate is about 49% if no extra rows recovered, and this rate rises to about 75% with about 30 extra rows recovered (needs about five effective faults, thus 25 faults in total at θ_i^{23}). Then if the attacker needs about 200 faults at θ_i^{22} to recover the whole state of χ_i^{22}, he needs to inject

about 408 faults (fault identification rate about 49%) without knowledge of extra rows of χ_i^{23}, and about 267 faults (fault identification rate about 75%) with knowledge of the extra rows of χ_i^{23}. In this case, injecting faults at θ_i^{23} to recover extra rows of χ_i^{23} will improve the efficiency of the attack significantly.

Actually, the proposed method used in the attacks of SHA3-224 and SHA3-256 can be applied to improve the attacks of SHA3-512. For SHA3-512, the digest size is 512 bits, and 192 bits will be observable on the plane $\chi_o^{23}(X, 1, Z)$. Thus using the method in Section III-C, some bits on the plane $\chi_i^{23}(X, 1, Z)$ can be recovered for attacks. Comparing with using only the 320 bits in the bottom plane of χ_i^{23}, this method can be used to improve the effective fault rate for the injected faults.

This work shows that DFA on SHA3-224 and SHA3-256 are more difficult than SHA3-384 and SHA3-512, while SHA3-224 is more difficult to conquer than SHA3-256, and this is different from their security level under other attack methods such as collision attacks [31]. Thus different kinds of attacks should be taken into consideration at the design stage of SHA-3 systems.

As the protection of SHA-3 against fault injection attacks has not been discussed thoroughly [36], [37], future work will include the protections of SHA-3 systems against fault injection attacks.

V. CONCLUSION

In this paper, we propose efficient methods to conquer SHA3-224 and SHA3-256 using differential fault analysis. Comparing with previous work, we extend the DFA on SHA-3 to SHA3-224 and SHA3-256 under relaxed fault model. Results show that the proposed methods in this paper can efficiently identify the randomly injected single-byte fault, and then use the recovered fault information to recover χ_i^{22} bits.

Acknowledgment: This work was supported in part by National Science Foundation under grants SaTC-1314655 and MRI-1337854.

REFERENCES

[1] N. F. Pub, "FIPS PUB 202. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," *Federal Information Processing Standards Publication*, 2015.

[2] N. Bagheri, N. Ghaedi, and S. Sanadhya, "Differential fault analysis of SHA-3," in *Progress in Cryptology – INDOCRYPT 2015*, 2015, pp. 253–269.

[3] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology - CRYPTO'97*, Aug. 1997, pp. 513–525.

[4] G. Piret and J.-J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and KHAZAD," in *Cryptographic Hardware & Embedded Systems*, Sept. 2003, pp. 77–88.

[5] H. Chen, W. Wu, and D. Feng, "Differential fault analysis on CLEFIA," in *Information and communications security*, 2007, pp. 284–295.

[6] S. Karmakar and D. R. Chowdhury, "Differential fault analysis of mickey-128 2.0," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on*, 2013, pp. 52–59.

[7] S. Banik and S. Maitra, "A differential fault attack on MICKEY 2.0," in *Cryptographic Hardware and Embedded Systems-CHES 2013*, pp. 215–232.

[8] 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*, 2012, pp. 122–139.

[9] P. Dey, A. Chakraborty, A. Adhikari, and D. Mukhopadhyay, "Improved practical differential fault analysis of Grain-128," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 459–464.

[10] L. Hemme and L. Hoffmann, "Differential fault analysis on the SHA1 compression function," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011 Workshop on*, Sept 2011, pp. 54–62.

[11] R. AlTawy and A. M. Youssef, *Information Security Practice and Experience: 11th International Conference, ISPEC 2015, Beijing, China, May 5-8, 2015, Proceedings*, 2015, ch. Differential Fault Analysis of Streebog, pp. 35–49.

[12] W. Li, Z. Tao, D. Gu, Y. Wang, Z. Liu, and Y. Liu, "Differential fault analysis on the MD5 compression function," *Journal of Computers*, no. 11, 2013.

[13] W. Fischer and C. A. Reuter, "Differential fault analysis on GrøStl," in *Proceedings of the 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, ser. FDTC '12, 2012, pp. 44–54.

[14] P. Luo, Y. Fei, X. Fang, A. Ding, M. Leeser, and D. Kaeli, "Power analysis attack on hardware implementation of MAC-Keccak on FPGAs," in *ReConFigurable Computing and FPGAs*, 2014.

[15] P. Luo, Y. Fei, X. Fang, A. Ding, D. Kaeli, and M. Leeser, "Side-channel analysis of MAC-Keccak hardware implementations," in *Hardware and Architectural Support for Security and Privacy*, 2015.

[16] M. Taha and P. Schaumont, "Side-channel analysis of MAC-Keccak," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, June 2013, pp. 125–130.

[17] I. Dinur, O. Dunkelman, and A. Shamir, "Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials," in *Fast Software Encryption*, S. Moriai, Ed., 2014, pp. 219–240.

[18] C. Boura and A. Canteaut, "A zero-sum property for the Keccak-f permutation with 18 rounds," in *2010 IEEE International Symposium on Information Theory*, June 2010, pp. 2488–2492.

[19] O. Benoît and T. Peyrin, "Side-channel analysis of six SHA-3 candidates," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, pp. 140–157.

[20] C. Boura and A. Canteaut, "Zero-sum distinguishers for iterated permutations and application to Keccak-f and Hamsi-256," in *Selected Areas in Cryptography*, 2010, pp. 1–17.

[21] C. Boura, A. Canteaut, and C. De Canniere, "Higher-order differential properties of Keccak and Luffa," in *Fast Software Encryption*, 2011, pp. 252–269.

[22] S. Das and W. Meier, "Differential biases in reduced-round Keccak," in *Progress in Cryptology–AFRICACRYPT 2014*, 2014, pp. 69–87.

[23] I. Dinur, O. Dunkelman, and A. Shamir, "New attacks on Keccak-224 and Keccak-256." in *FSE*, 2012, pp. 442–461.

[24] I. Dinur, O. Dunkelman, and A. Shamir, "Improved practical attacks on round-reduced Keccak," *Journal of cryptology*, no. 2, pp. 183–209, 2014.

[25] I. Dinur, P. Morawiecki, J. Pieprzyk, M. Srebrny, and M. Straus, "Cube attacks and cube-attack-like cryptanalysis on the round-reduced Keccak sponge function," in *Advances in Cryptology–EUROCRYPT 2015*, pp. 733–761.

[26] A. Duc, J. Guo, T. Peyrin, and L. Wei, "Unaligned rebound attack: application to Keccak," in *Fast Software Encryption*, 2012, pp. 402–421.

[27] J. Jean and I. Nikolić, "Internal differential boomerangs: practical analysis of the round-reduced Keccak-f permutation," in *Fast Software Encryption*, 2015, pp. 537–556.

[28] S. Kölbl, F. Mendel, T. Nad, and M. Schläffer, "Differential cryptanalysis of Keccak variants," in *Cryptography and Coding*, 2013, pp. 141–157.

[29] P. Morawiecki, J. Pieprzyk, and M. Srebrny, "Rotational cryptanalysis of round-reduced Keccak," in *Fast Software Encryption*, 2013, pp. 241–262.

[30] M. Naya-Plasencia, A. Röck, and W. Meier, "Practical analysis of reduced-round keccak," in *Progress in Cryptology–INDOCRYPT 2011*, 2011, pp. 236–254.

[31] G. Bertoni, J. Daemen, M. Peeters, and G. Assche, "The Keccak reference," *Submission to NIST (Round 3)*, January, 2011.

[32] "Reference and optimized code in C," http://keccak.noekeon.org/KeccakReferenceAndOptimized-3.2.zip.

[33] P. Pessl and M. Hutter, "Pushing the limits of SHA-3 hardware implementations to fit on RFID," in *Cryptographic Hardware and Embedded Systems - CHES 2013*, 2013, pp. 126–141.

[34] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, "Keccak implementation overview," *Report, STMicroelectronics, Antwerp, Belgium*, 2012.

[35] J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," Ph.D. dissertation, Doctoral Dissertation, March 1995, KU Leuven, 1995.

[36] P. Luo, C. Li, and Y. Fei, "Concurrent error detection for reliable SHA-3 design," in *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '16, 2016, pp. 39–44.

[37] S. Bayat-Sarmadi, M. Mozaffari-Kermani, and A. Reyhani-Masoleh, "Efficient and concurrent reliable realization of the secure cryptographic SHA-3 algorithm," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 7, pp. 1105–1109, 2014.