Differential Fault Analysis of SHA-3

Nasour Bagheri[1,2(✉)], Navid Ghaedi[1], and Somitra Kumar Sanadhya[3]

[1] Electrical Engineering Department,
Shahid Rajaee Teacher Training University, Tehran, Iran
na.ghaedi@gmail.com
[2] The School of Computer Science,
Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
Nbagheri@srttu.edu
[3] IIIT-Delhi, Delhi, India
somitra@iiitd.ac.in

Abstract. In this paper we present the first differential fault analysis (DFA) of SHA-3. This attack can recover the internal state of two versions of SHA-3 (namely, SHA3-512 and SHA3-384) and can be used to forge MAC's which are using these versions of SHA-3. Assuming that the attacker can inject a random single bit fault on the intermediate state of the hash computation, and given the output of the SHA-3 version for a correct message and 80 faulty messages, we can extract 1592 out of the 1600 bits of the compression function's internal state. To the best of our knowledge, this is the first public analysis of SHA-3 against DFA. Although our results do not compromise any security claim of SHA-3, it shows the feasibility of DFA on this scheme and possibly other Sponge based MACs and increases our understanding of SHA-3.

Keywords: SHA-3 · Keccak · Differential fault analysis

1 Introduction

The new SHA-3 standard [23] is adapted from the Keccak hash function [9]. Keccak hash function is a family of sponge based hash function [8] with *Keccak-f*$[r+c]$ as the primitive. The parameters r and c are the bit rate and the capacity and determine the width of the *Keccak-f* permutation. In the case of SHA-3, the internal state size is $b = 1600$ bits while the output size $n \in \{224, 256, 384, 512\}$ bits. In this paper, we concentrate on the standard Keccak versions submitted to the SHA-3 competition and denote them by SHA3-224, SHA3-256, SHA3-384 and SHA3-512, depending on the output size n.

Keccak has received significant attention of the cryptography community, both during and after the SHA-3 competition. Some of the prominent works analyzing Keccak are [3,7,11–13,15,17–21,29,33–36]. However, these numerous analysis have not compromised any security claim of Keccak and there exists a big gap between the number of rounds practically broken and the number of rounds of Keccak suggested by the designers.

A. Biryukov and V. Goyal (Eds.): INDOCRYPT 2015, LNCS 9462, pp. 253–269, 2015.
DOI: 10.1007/978-3-319-26617-6_14

Differential Fault Analysis (DFA), first introduced by Biham and Shamir in [10], derives information about the secret key from the physical implementation of the cipher by examining the differences between a fault-free encryption and several faulty ones. A fault may be injected by introducing an external impact on the processing device by means of voltage variation, glitch, laser, etc. However, neither the fault location nor the bit-value at the fault location may be known to the attacker. This attack model has been successfully applied to several block ciphers, stream ciphers, hash functions and authenticated encryption functions, where the attacks on DES [27,39], AES [1,22,25,26,31,32,37,41], Grain [5,16], Mickey [4,30], SHA-1 compression function [28], Grøstl [24], Streebog [2] and APE [40], CLOC and SILC authenticated encryption schemes [14] are examples. In the case of the hash functions, DFA make sense when the hash function is used in a message authenticated mode such as secret IV-MAC [38], HMAC or NMAC [6]. In such applications, DFA could be used to recover the secret key or perform forgery against the MAC. Since using a hash function is common in constructing a secure MAC scheme, it is important to investigate the security of a hash function against the DFA attack. However, to the best of our knowledge, there is no public report on DFA on SHA-3 or Keccak.

On the other hand, Dinur et al. [20] recently have analyzed the keyed modes of Keccak sponge function where it is used to generate bit stream for stream cipher or it is used as a building block for message authentication codes(MACs) and authenticated encryption (AE) schemes. Motiviated by that work, where the internal state of Keccak permutation is extracted using cube attack, in this paper the internal state of Keccak permutation is reconstructed with fault injection. Therefore, this attack could be used to recover the secret key or do forgery against MACs based on Keccak. It also maybe applicable to recover the secret key and the initial value of stream cipher based on Keccak if the attack scenario change to known plaintext attack. In addition, in the keyed version of SHA-3, if it is used with nonce and there is a restriction on the repeating the nonce, then the given attack does not work.

1.1 Contribution

We present differential fault analysis on two versions of SHA-3, namely SHA3-512 and SHA3-384. The attack model follows the approach used already in DFA against Grøstl [24], Streebog [2] and CLOC & SILC authenticated encryption schemes [14] where the adversary injects a single bit fault in a random position of the internal state. The presented attack can recover the complete internal state of SHA-3 given the first 320 least significant bits of its output (called a plane in Keccak) for the correct message and enough faulty messages. Since the output of SHA3-384 and SHA3-512 includes a complete plane, we apply our attack to these variants of SHA-3. We then present a theoretical bound on the number of detected bits of the Keccak state after injecting a single bit fault on N messages and compare the same with simulation results. Our theoretical analysis shows that injecting 80 randomly distributed single bit faults on internal state are enough to recover 1592 bits out of 1600 bits of the internal state.

1.2 Paper Organization

The rest of the paper is organized as follows: in Sect. 2, the notations used in the paper are presented, and in Sect. 2.2, the Keccak hash function is briefly described. In Sect. 3, we present our differential fault analysis on SHA3-384 and SHA3-512. In Sect. 4, we present theoretical bounds and simulation results for state bit recovery. Finally, we conclude the paper in Sect. 5.

2 Notations and Preliminaries

2.1 Notations

Throughout the paper, we use the following notations:

- $A^i(x, y, z)$: denotes a single bit of state at the beginning of the i^{th} round.
- $H(M)$: denotes the output of SHA-3.
- $plane_{H(m)}(0)$: denotes 320 consecutive bits of output of SHA-3 that could be used to reconstruct $plane(0)$ of the state.
- $plane_{H(m)}(0)'$: denotes 320 consecutive bits of faulty output of SHA-3 that could be used to reconstruct $plane(0)$ of the state.
- θ^i: denotes the θ step at i^{th} round.
- χ^i: denotes the χ step at i^{th} round.
- π^i: denotes the π step at i^{th} round.
- ρ^i: denotes the ρ step at i^{th} round.
- ι^i: denotes the ι step at i^{th} round.
- B: denotes the state before χ step in the penultimate round and B^l_b denotes the b^{th} bit of the l^{th} lane of B.
- C: denotes the state after θ step in the last round and C^l_b denotes the b^{th} bit of the l^{th} lane of C.

2.2 Description of Keccak Hash function

Keccak [9] is a family of hash functions and the winner of the SHA-3 competition. Some of its variants were adapted as SHA-3 [23]. It is a sponge based hash function based on Keccak-$f[b, n_r]$ permutation. Figure 1 illustrates the sponge construction, based on permutation $f : \{0,1\}^r \times \{0,1\}^c \rightarrow \{0,1\}^r \times \{0,1\}^c$. In the sponge construction, r is the bitrate and called *rate* and c is the security parameter and called *capacity*. Larger r provides higher speed and larger c provides better security. The state size of the hash function is determined by $b = c + r$. In Keccak $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ and for the case of SHA-3, the state size is 1600 bits.

In Keccak-$f[b, n_r]$, b and n_r denote the state size and number of the rounds of the permutation respectively. The SHA-3 standard uses Keccak-f[1600,24] permutation from [9]. Depending on the output length n, 4 versions of SHA-3 use $c = 2n$ for $n \in \{224, 256, 384, 512\}$, and are called SHA3-224, SHA3-256 etc.

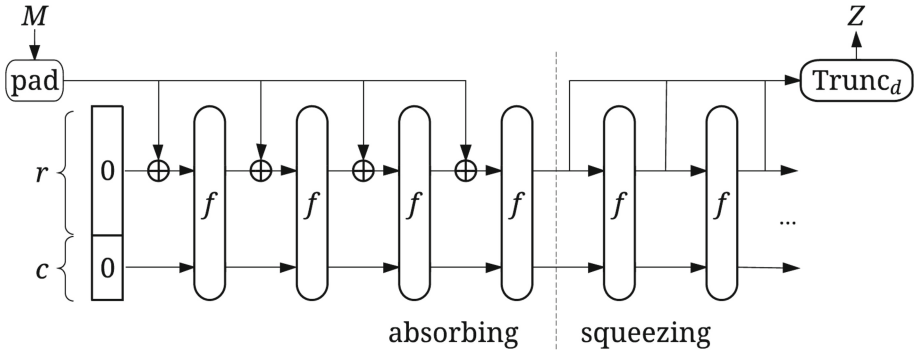


Fig. 1. The sponge construction based on permutation f [23].

Keccak-f[1600,24] has 24 rounds, indexed from 0 to 23, and each round performs 5 consecutive permutations on the state, denoted by θ, ρ, π, χ, and ι. The only non-linear permutation is χ and the only round dependent permutation is ι. The input to these permutations is constructed as a 3-dimensional array (x, y, z) where the dimensions of x, y and z are 5, 5 and 64 respectively. Denoting the Array by A, each bit of the array can be described as $A(x, y, z)$, where $0 \leq x \leq 4$, $0 \leq y \leq 4$ and $0 \leq z \leq 63$. The initial state of Keccak-f[1600,24] (i.e. before the application of the 0^th round) is denoted by $A^0(.,.,.)$ and the output of Keccak-f[1600,24] after the 23^{rd} round is denoted by $A^{24}(.,.,.)$.

Keccak-f[1600,24] state can be defined in different parts. This naming convention is helpful in describing Keccak-f[1600,24]:

- A row is a set of 5 bits with constant y and z coordinates, i.e. $A(*, y, z)$.
- A column is a set of 5 bits with constant x and z coordinates, i.e. $A(x, *, z)$.
- A lane is a set of 64 bits with constant x and y coordinates, i.e. $A(x, y, *)$.
- A sheet is a set of 320 bits with constant x coordinate, i.e. $A(x, *, *)$.
- A plane is a set of 320 bits with constant y coordinate, i.e. $A(*, y, *)$.
- A slice is a set of 25 bits with constant z coordinate, i.e. $A(*, *, z)$.

In this paper, each lane of the state is specified according to Table 1.

Table 1. Lane numbering, each square represents a lane in the state.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Next we briefly describe the 5 permutations used in each round of SHA-3. **θ.** As the first step in each round, the role of θ is to XOR each bit $A(x, y, z)$ with

bits in column $(x-1,*,z)$ and $(x+1,*,z-1)$. Hence, θ can be represented as follows:

$$\theta : A(x,y,z) \leftarrow A(x,y,z) + \sum_{y'=0}^{4} A(x-1,y',z) + \sum_{y'=0}^{4} A(x+1,y',z-1).$$

ρ. In this step, the bits are rotated in their lane by $T(x,y)$ positions, which is a predefined offset value for each lane. Table 2 shows these offset values.

Table 2. Offset values of ρ for Keccak-p[1600,24]

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	25	39	3	10	43
$y=1$	55	20	36	46	6
$y=0$	28	27	0	1	62
$y=4$	56	14	18	2	61
$y=3$	21	8	41	45	15

π. Permutation π is used to rearrange the positions of the lanes in the array. Each lane in position $(x,y,*)$ is moved to the new position $(x',y',*)$, where $x'=y$ and $y'=(x+3y)$ mode 5.

χ. This is the only non-linear permutaion in each round. A row in position $(*,y,z)$ is processed and replaced by a new row by this permutation. Each input bit affects 3 bits at the output of χ as described below.

$$A(x',y',z') = A(x,y,z) \oplus (\overline{A(x+1,y,z)} \ \& \ A(x+2,y,z)),$$

where $\bar{x} = x \oplus 1$ and & is the bitwise AND operator.

ι. The final step in each round is the application of ι, which is the only round dependent step. In the i^{th} round of this step, a round dependent value $RC(i)$ is XOR'ed with $Lane(0)$. The values of $RC(i)$ can be found in [9]. However, they do not impact our attack.

3 DFA Attack on SHA3-384, and SHA3-512

In this section, we show how to obtain the bits of internal state that do not appear in the output of SHA3-384, and SHA3-512. We assume a single bit fault is injected at the beginning of penultimate round of Keccak-p[1600,24]. In the rest of the paper, for simplicity we denote Keccak-p[1600,24] permutation by Keccak-p. We use the following observations on χ function in the our DFA attack on SHA-3 variants.

Observation 1. Suppose a single bit fault is injected in the input of χ and we are given the difference of the correct and the faulty output of χ. In this case, it is easy to extract two bits of input of χ given the differential output for χ. This is due to the fact that bitwise AND operation in χ leaks information of its input, if a single bit of its input was corrupted.

Let a single bit fault be injected in position (f, i, j) in the input to χ then it leads to a single bit difference in the output of χ with probability '1'. Moreover, it also leads to a single bit difference in $(f-1, i, j)$ if $A(f+1, i, j) = 1$' and a single bit difference in $(f-2, i, j)$ if $A(f-1, i, j) = 0$. It can be seen that if we have differential output of χ, we can extract two bits of its input.

Observation 2. Given any input bit of the χ function, we receive some linear equations of the input bits at the output of χ. Denoting the input and output of χ by $x_0, \ldots, x_4$ and $y_0, \ldots, y_4$ respectively, given the input bit x_i it appears at the output as $y_{i-2} = x_{i-2} \oplus (\overline{x}_{i-1} \& x_i)$, $y_{i-1} = x_{i-1} \oplus (\overline{x}_i \& x_{i+1})$ and $y_i = x_i \oplus (\overline{x}_{i+1} \& x_{i+2})$. It is clear that given x_i we achieve two linear equations of the inputs of χ at its output, see Table 3. In Table 4, we represent a case where all output bits are a linear function of the unknown inputs or they are constant.

Table 3. The relation between input bits (x_i) and output bits (y_i) of χ assuming that x_4 is known and the rest of the inputs are unknown. Here, NL is used to denote non-linear function. (In the linear function, the output depends only on the XOR of its inputs. Otherwise the function is non- linear.)

x_0	x_1	x_2	x_3	x_4	y_0	y_1	y_2	y_3	y_4
x_0	x_1	x_2	x_3	0	NL	NL	x_2	$x_3 \oplus x_0$	NL
x_0	x_1	x_2	x_3	1	NL	NL	$x_2 \oplus \overline{x}_3$	x_3	NL

Table 4. The relation between input's bits (x_i) and output's bits (y_i) of χ assuming that x_0 and x_2 are unknown and the rest of the inputs are known.

x_0	x_1	x_2	x_3	x_4	y_0	y_1	y_2	y_3	y_4
x_0	0	x_2	0	0	$x_0 \oplus x_2$	0	x_2	x_0	0
x_0	0	x_2	0	1	$x_0 \oplus x_2$	0	$\overline{x}_2$	0	1
x_0	0	x_2	1	0	$x_0 \oplus x_2$	$\overline{x}_2$	x_2	$\overline{x}_0$	0
x_0	0	x_2	1	1	$x_0 \oplus x_2$	$\overline{x}_2$	x_2	1	1
x_0	1	x_2	0	0	x_0	1	x_2	x_0	$\overline{x}_0$
x_0	1	x_2	0	1	x_0	1	$\overline{x}_2$	0	$\overline{x}_0$
x_0	1	x_2	1	0	x_0	$\overline{x}_2$	x_2	$\overline{x}_0$	$\overline{x}_0$
x_0	1	x_2	1	1	x_0	$\overline{x}_2$	x_2	1	$\overline{x}_0$

Observation 3. Assuming that we know some input bits of the χ function, the output bits will be non-linear if and only if two consecutive input bits of χ are unknown. This observation directly comes from the fact that for any output bit we have $y_i = x_i \oplus (\overline{x}_{i+1} \& x_{i+2})$. In this equation the only no-linear part is the AND operation and, assuming that we know some input bits, it remains non-linear if and only if both x_{i+1} and x_{i+2} are unknown. Table 3 represents an example of the case where some of the output bits are non-linear function of unknown bits and some are linear.

Next, we explain how this single bit propagates in one round.

– **Propagation of Active Bit in One Round.** Assume that we have injected a single bit fault on $A^{22}(x, y, z)$, which is the input of penultimate round of the Keccak-p. Hence, we have $\Delta A^{22}(x, y, z) = 1$ and $\Delta A^{22}(x', y', z') = 0$, for any $(x', y', z') \neq (x, y, z)$. On the other hand, this active bit affects 11 bits after θ^{22} in the following positions and converts them to active:

$$(x, y, z), (x-1, 0, z+1), (x-1, 1, z+1), (x-1, 2, z+1), (x-1, 3, z+1), (x-1, 4, z+1),$$
$$(x+1, 0, z), (x+1, 1, z), (x+1, 2, z), (x+1, 3, z), (x+1, 4, z),$$

The rest of the bits would remain inactive after the θ^{22} function. It is clear from offset values of ρ in Table 2 that, excluding the case where $x = 1$, ρ^{22} function moves active bits to different slices. Hence, after ρ^{22}, we expect to receive 11 slices with only one active bit in each slice or 9 slices with only one active bit in each slice and $slice(z+1)$ with two active bits in the positions $(0, 0, z+1)$ and $(1, 0, z+1)$. The π^{22} function changes the location of the active bits in their slices. Hence, the number of active bits after the π^{22} function remains unaffected. On the other hand, if we have active bits in $(0, 0, z+1)$ and $(1, 0, z+1)$ after ρ^{22}, the π^{22} function will move these active bits to $(0, 0, z+1)$ and $(0, 2, z+1)$. Hence, up to the end of the π^{22} function, we have 11 rows each having only one active bit and the rest of the rows having no active bits. Next we apply χ^{22} to the internal state. It is clear that if any row has no active bit then it does not generate any active bit after χ^{22}. Hence, the number of active bits for those rows that have no active bits remains zero after χ^{22}. On other hand, if $Row(i, j)$ includes single active bit in position (f, i, j), then after χ^{22} it leads to an active bit in (f, i, j) with probability '1', leads to an active bit in $(f-1, i, j)$ if $A(f+1, i, j) = 1$ and leads to an active bit in $(f-2, i, j)$ if $A(f-1, i, j) = 0$. Finally, the ι^{22} function keeps the number of active bits in the state unaffected and produces $A^{23}(x, y, z)$.

According to Observation 1, these 11 active bits, that are located in different rows at beginning of χ^{22}, on the differential inputs of χ^{22} leak 22 bits of internal state. It must be noted that these 22 bits of internal state before χ^{22} are distinct.

Thanks to θ^{23}, the θ function of the last round, any of those 22 bits of internal state affects each plane of internal state after θ^{23} and its differential output. Then

```
Differential state at the beginning of round 22 after injection of a single bit fault
|--------------1|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|

Differential output after theta in round 22:
|--------------1|--------------1|----------------|----------------|--------------2|
|----------------|--------------1|----------------|----------------|--------------2|
|----------------|--------------1|----------------|----------------|--------------2|
|----------------|--------------1|----------------|----------------|--------------2|
|----------------|--------------1|----------------|----------------|--------------2|

Differential output after rho in round 22:
|--------------1|--------------2|----------------|----------------|-------1-------|
|----------------|----1----------|----------------|----------------|---------2-----|
|----------------|------------4--|----------------|----------------|-----1---------|
|----------------|----2----------|----------------|----------------|------------2--|
|----------------|--------------4|----------------|----------------|-----------8---|
Differential output after pi in round 22:
|--------------1|----1----------|----------------|----------------|-----------8---|
|----------------|---------2-----|----------------|----2-----------|----------------|
|--------------2|----------------|----------------|------------2--|----------------|
|-------1-------|----------------|------------4--|----------------|----------------|
|----------------|----------------|-----1---------|----------------|--------------4|
Differential output after chi in round 22:
|----x----------1|----1----------|-----------x---|-----------x--x|----x-------8--x|
|---------x-----|----x-----2-----|----x----------|----2----------|---------x-----|
|--------------2|------------x--|------------x--|------------2-x|--------------x|
|-------1----x--|------------x--|------------4--|-------x-------|-------x-------|
|-----x---------|-----x---------|-----1--------x|--------------x|--------------4|

The state is described as a matrix of 5 × 5 lanes of 64 bits, ordered from left to right, where each lane is
given in hexadecimal using little-endian format.
x represents a bit of χ input ($B_b^l$) according to observation 1. We stress that the differential characteristic
has probability 1.
```

Fig. 2. Injection a single bit fault and it's differential path

the internal state goes through ρ^{23}, π^{23}, χ^{23} and ι^{23} to produce the final output. In Fig. 2, a differential characteristic is shown under the assumption that a single bit fault is injected on $A^{22}(0,0,0)$.

The output of SHA3-384 and SHA3-512 ($H(M)$) include one complete plane. Given that plane of the output of $H(M)$, it is possible to invert ι^{23}, χ^{23}, π^{23} and ρ^{23} for this plane. Hence, it is possible to compute $(\rho^{23})^{-1}((\pi^{23})^{-1}((\chi^{23})^{-1}((\iota^{23})^{-1}(plane_{H(M)}(0)))))$. Following this fact, given the output of SHA3-384 (or SHA3-512) it is possible to invert a fraction of output that are included in $plane(0)$ and determine the internal state of $lane(0)$, $lane(6)$, $lane(12)$, $lane(18)$ and $lane(24)$ after θ^{23}. These lanes comes from $sheet(0)$, $sheet(1)$, $sheet(2)$, $sheet(3)$ and $sheet(4)$ respectively. In addition, any active bit in $sheet(i)$ affects all the lanes in $sheet(i-1)$ and $sheet(i+1)$. Hence, we can be sure that any of these 22 target bits appear linearly in $lane(0)$, $lane(6)$, $lane(12)$, $lane(18)$ and $lane(24)$ after θ^{23}. Suppose a single bit fault is injected on $A^{22}(0,0,0)$, the equations of these lanes are shown as follow:

$$C^0 = 0000000000000000B_{45}^6 1C_{44}^0 00B_{40}^{21} 000000000000B_{28}^{19} 000001000B_{10}^{16} B_9^{11} 00000001B_1^{14} C_0^0$$
$$C_0^0 = 1 \oplus B_0^4, C_{44}^0 = B_{44}^4 \oplus B_{44}^0$$
$$C^6 = 0000000000000000B_{45}^7 B_{45}^6 B_{44}^0 001B_{40}^{20} 000000000001000000B_{21}^5 0000B_{15}^2 00001C_{10}^6 000000B_2^{22} 011$$
$$C_{10}^6 = B_{10}^{15} \oplus B_9^{12}$$
$$C^{12} = 0000000000000001B_{45}^6 1000B_{40}^{21} 0000000000B_{28}^{18} 000000010000B_{15}^3 00000B_{10}^{16} B_9^{11} 00000B_2^{23} B_1^{13} B_0^3 0$$
$$C^{18} = 0000000000000000C_{45}^{18} 000001000000000B_{28}^{19} B_{28}^{18} 00000B_{21}^9 000001B_{15}^2 00001B_9^{12} 000001C_2^{18} B_0^4 0$$
$$C_2^{18} = B_2^{22} \oplus B_1^{14}, C_{45}^{18} = B_{45}^6 \oplus B_{44}^4$$
$$C^{24} = 0000000000000000C_{45}^{24} 000B_{40}^{20} 000000000001B_{28}^{18} 00000B_{21}^5 000000B_{15}^3 000B_{10}^{15} 01000000C_2^{24} C_1^{24} B_0^3$$
$$C_1^{24} = 1 \oplus B_1^{13}, C_2^{24} = B_2^{23} \oplus B_2^{22} \oplus B_1^{14}, C_{45}^{24} = 1 \oplus B_{44}^0$$

Each lane is represented in 64 bits. Obviously, the above equations are constructed under the assumption that a single bit fault is injected on $A^{22}(0,0,0)$. On the other hand, the attacker needs to know the position of single bit fault to obtain these 22 bits of internal state. Next we describe the technique to know the fault location itself.

– **Determining the Fault Position:** Given a faulty output and a correct output of SHA3-384 (or SHA3-512) (say $H(M)$), it is possible to compute $(\chi^{23})^{-1}((\iota^{23})^{-1}(plane_{H(M)}(0)))$. In fact, the $plane(0)$ of differential output of π^{23} can be computed as follows:

$$(\chi^{23})^{-1}((\iota^{23})^{-1}(plane_{H(M)}(0))) \oplus (\chi^{23})^{-1}((\iota^{23})^{-1}(plane_{H(M)}(0)'))). \quad (1)$$

Again suppose a single bit fault is injected on $A^{22}(0,0,0)$, the $plane(0)$ of differential output of π^{23} is shown as follow (Fig. 3):

Any bit in the above differential output could be 1, 0 or x. When the value of a bit is 1 or 0 then the difference in that position is deterministic. However, any bit that is marked as x could appear as 1 or 0 in the real differential output. It can be shown that the attacker can find the position of the fault by using this deterministic differential bits. We describe this method in Algorithm 1. Given the inverted $plane(0)$ of the output of the correct and faulty messages, based on a single bit fault on $A^{22}(x,y,z)$, Algorithm 1 can be used to determine the position of the fault.

We also verified this algorithm in simulations. For any single fault in $A^{22}(x,y,z)$, the algorithm returned the position of the fault correctly. In the off-line phase of the algorithm, for any possible positions of fault in $A^{22}(x,y,z)$ (i.e. all 1600 positions), related differential outputs up to the end of π^{23} are generated and $plane(0)$ of the state at the output of π^{23} is stored in a table $\mathcal{T}$. In Appendix A, the distribution of fault on $plane(0)$ after π^{23}, for injecting a single fault in any bits of $slice(0)$ are presented. Any other positions of fault is just a simple rotation of one of these patterns.

Since there is no collision between deterministic bits of stored values in $\mathcal{T}$, our Algorithm 1 will return the position of fault correctly, if the fault happened at the input of θ^{22}. Our simulations verify this claim and the success probability of the given algorithm to return the correct position of the injected single bit fault is '1'.

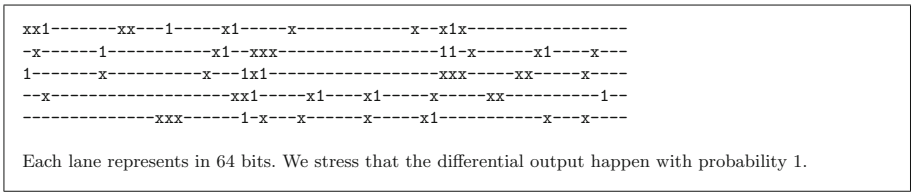


Fig. 3. The $plane(0)$ of differential output of π^{23}

Algorithm 1. Determining the position of a single bit fault, injected at the beginning of θ^{22}

off-line phase: For any possible position of injecting a single bit fault in $A^{22}(x, y, z)$, calculate related differential characteristics up to the output of π^{23} and store $plane(0)$ of the differential output of π^{23} in table $\mathcal{T}$.

on-line phase: Given a faulty output($plane_{H(M)}(0)'$)):

S.1 Compute the expected value of the difference of $plane(0)$ at the output of π^{23}, using Eq. 1.

S.2 For any entry in $\mathcal{T}$, compare it with the computed value in step **S.1** based on the deterministic bits only.

S.3 If there is a match between the computed value in step **S.1** and an entry in $\mathcal{T}$, return the related position as the fault location; otherwise return unknown.

4 Theoretical Bound and Simulation Result

Following the discussion in Sect. 3, for any single bit fault injection, we can determine the position of fault uniquely and determine 22 bits of internal state. Given that we have inverted 5 lanes from output, the total number of required independent equations are $1600 - 5 \times 64 = 1280$. However, we have no control on the position of faults and therefore the extracted bits. Hence, we consider the upper-bound of required independent equations as 1600. The lower bound on the number of faulty messages is $\frac{1600}{22} \cong 73$ if all the faults produce independent equations. However, this lower bound is unlikely to be achieved in reality because of the possible injection of the fault in same locations or the overlap between extracted bits of the state by different position of faults. To provide a bound of the number of detected bits after injecting a single bit fault on N messages, one can argue that after the first fault one can extract 22 bits of internal state. On injecting the second fault, if it is in the same location as the first one then it does not provide any new information, otherwise it is possible to extract the information of any bit that is not extracted with the first fault. The same argument can be stated for other faults. Hence, a raw bound of the number of detected bits after injecting a single bit fault on N messages, denoted by $\mathcal{S}^N$, are as follows:

$$\mathcal{S}^N = \sum_{1}^{N} \mathcal{X}_i, \tag{2}$$

where $\mathcal{X}_1 = 22$ and $\mathcal{X}_i = \left(1 - \frac{i-1}{1600}\right)\left(22 \times \frac{1600-\mathcal{X}_{i-1}}{1600}\right)$. We evaluated this bound by simulations as well. The results of our simulations are shown in Fig. 4. For the experiments, for any value of N, we injected faults randomly in N positions of $A^{22}(x, y, z)$ and counted the number of extracted bits of internal state after these faults. We also repeated this simulation 100 times for any value of N and counted the average. The simulation results match the given theoretical bound.

Extracting all bits of the internal state only by injecting single bit faults requires few faults, because the number of the extracted bits decreases when N is increased. For example, based on the given theoretical bound, to extract 1480 bits of the internal state, we need 200 faulty messages. However, after extracting a part of the internal state, given that the only non-linear layer in Keccak-p round function is χ we can use Observation 2 to extract some of the unknown bits, without injecting extra faults. On the other hand, we know that the expected number of the unknown bits after injecting a single bit fault on N messages is $1600 - \mathcal{S}^N$. Hence, for any bit at the input of χ^{23} to be unknown, the probability is determined as follows, denoted by $Pr^N(Uk)$:

$$Pr^N(Uk) = \frac{1600 - \mathcal{S}^N}{1600}.$$

Based on Observation 3, we have non-linear bit at the output of χ^{22} if there are at least two consecutive unknown bits. We know that there are 1600 possibilities for two consecutive bits in the internal state, and any two consecutive bits are both unknown with probability $(Pr^N(Uk))^2$. Therefore the probability of any bit of the output of χ^{22} to be a non-linear function of the unknown bits is $(Pr^N(Uk))^2$ which we denote by $Pr^N(NL)$. Later, θ^{23} combines the state bits linearly. However, if any of bits that are XOR'ed together to generate a single bit of the internal state after θ^{23} is non-linear, the output bit will also be non-linear. Hence, we can state that the probability of any bit at the output of θ^{23} to be linear, denoted by $Pr^N(L)$, is as follows:

$$Pr^N(L) = (1 - Pr^N(NL))^{11}.$$

Therefore, the expected number of linear equations in the 320 bits that are extracted from $(\rho^{23})^{-1}((\pi^{23})^{-1}((\chi^{23})^{-1}((\iota^{23})^{-1}(plane_{H(M)}(0)))))$ are as follows, denoted by $\#L^N$:

$$\#L^N \cong 320 \times (1 - Pr^N(NL))^{11}.$$

Hence, we can rewrite $\mathcal{X}_i$ as follows:

$$\mathcal{X}_i = \left(1 - \frac{i-1}{1600}\right)\left((22 + \#L^{N-1}) \times \frac{1600 - \mathcal{X}_{i-1}}{1600}\right). \tag{3}$$

We simulated the stated theoretical bound $\mathcal{S}^N$ of the number of the extracted bits after injecting a single bit fault on N messages. The results of our experiments are shown in Fig. 4. It is clear from this figure that to recover the complete internal state of the last permutation of Keccak-p at the input of χ^{22}, we need at most 80 faulty messages and the related correct message. To be more precise, given 80 messages with single bit faults, we can retrieve 1592 out of 1600 input bits of χ^{22}. The remaining 8 bits can be found by exhaustive search. Given the internal state of the permutation, it would be straightforward to recover the secret key or forge MAC of any desired message.

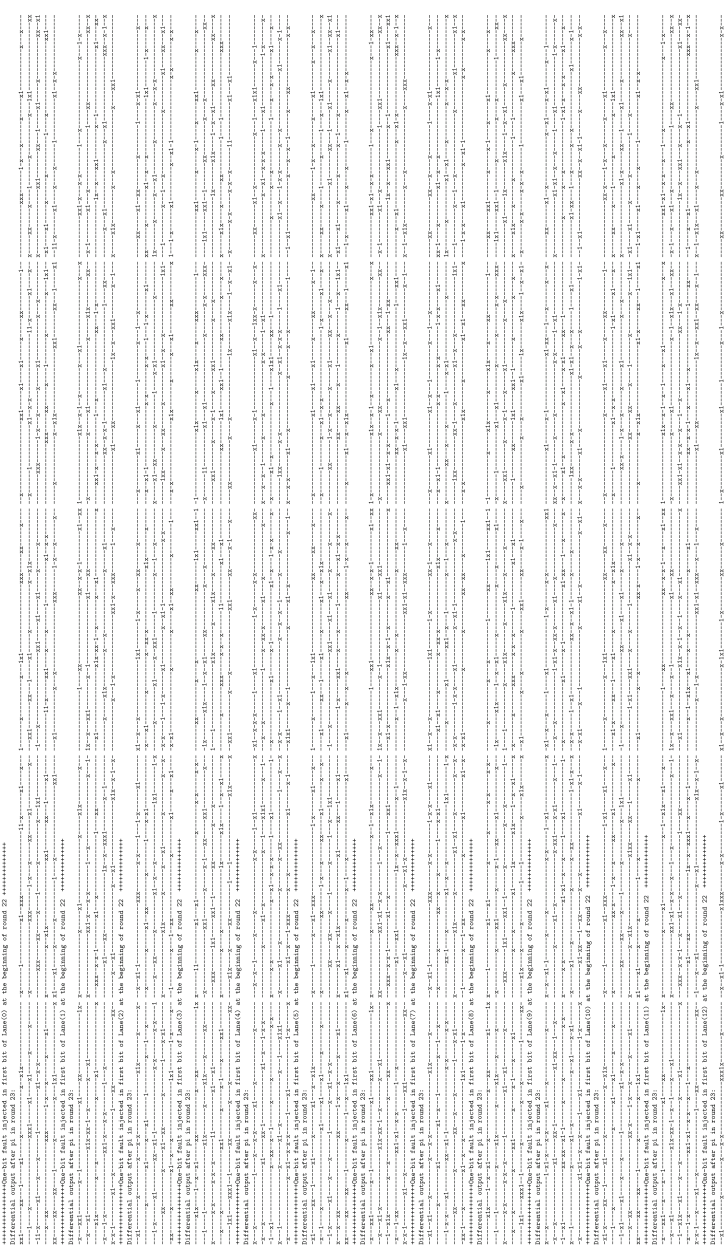
Fig. 4. The number of extracted bits of internal state after injecting a single bit fault on N messages. Here, Theory 1 and Theory 2 are the theoretical bounds that are given by Eqs. 2 and 3 respectively and Simulation is number of the extracted bits after simulating the attack without using information from Observations 2 and 3 which matches Theory 1.

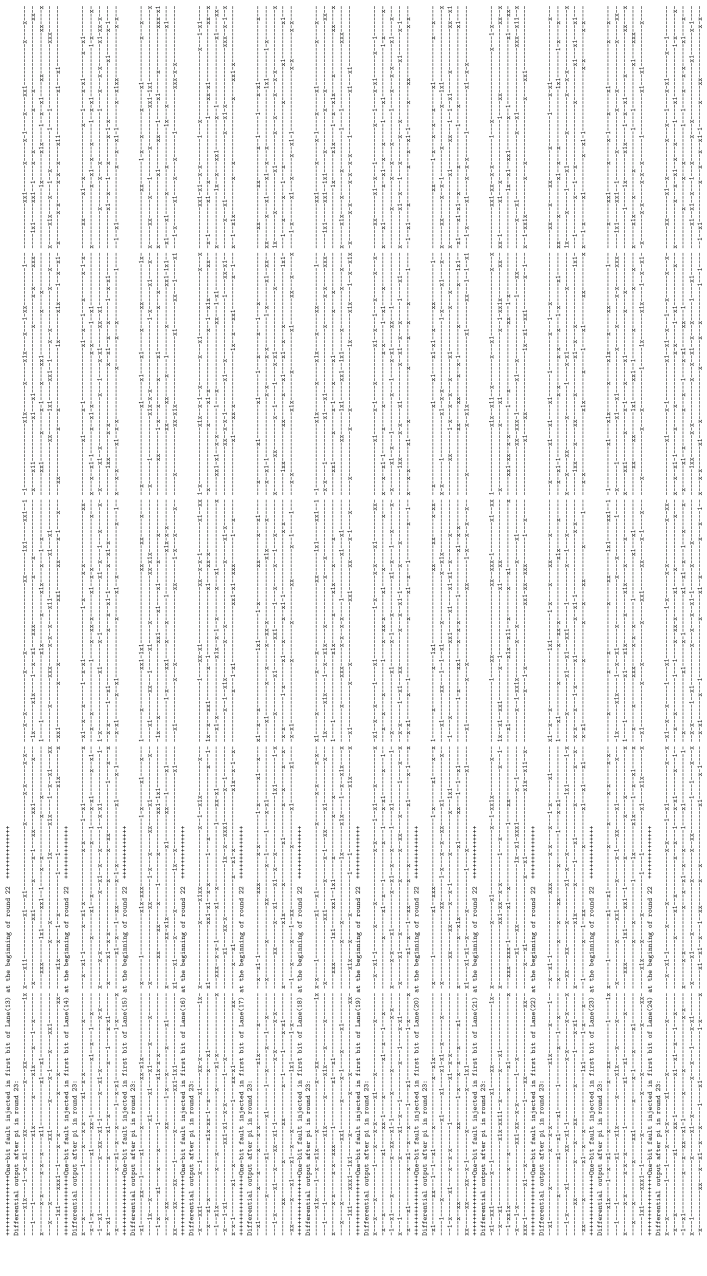
5 Conclusions

In this work, we investigated the security of SHA-3 against DFA. The main idea was to extract the capacity of the sponge construction given a correct output and some related faulty outputs. Our study shows that by injecting around a random single bit faults on 80 messages, one can obtain the internal state of the compression function. Given that the primitives of SHA-3 and other sponge based schemes are permutations, it would be possible to do state/key recovery on such sponge based MACs/AE. This approach can also be applicable to perform DFA on sponge based CAESAR candidates, which is an on going work. Another direction to continue this study is to use some relaxation on the fault injection model (both the value of fault and its location) which is the subject for future works.

A Appendix

We stress that all differential output happen with probability 1. The state is described as a matrix of 5×5 lanes of 64 bits, ordered from right to left, where each lane is given in bit using the little-endian format.
`x` represents a linear equation of χ input (B).





References

1. Ali, S., Mukhopadhyay, D., Tunstall, M.: Differential fault analysis of AES: towards reaching its limits. J. Crypt. Eng. **3**(2), 73–97 (2013)
2. AlTawy, R., Youssef, A.M.: Differential fault analysis of streebog. In: Lopez, J., Wu, Y. (eds.) ISPEC 2015. LNCS, vol. 9065, pp. 35–49. Springer, Heidelberg (2015)

3. Aumasson, J.-P., Meier, W.: Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. Rump session of Cryptographic Hardware and Embedded Systems-CHES 2009, p. 67 (2009)
4. Banik, S., Maitra, S.: A differential fault attack on MICKEY 2.0. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 215–232. Springer, Heidelberg (2013)
5. Banik, S., Maitra, S., Sarkar, S.: A differential fault attack on the grain family of stream ciphers. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 122–139. Springer, Heidelberg (2012)
6. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
7. Benoît, O., Peyrin, T.: Side-channel analysis of six SHA-3 candidates. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 140–157. Springer, Heidelberg (2010)
8. Bertoni, G., Daemen, J., Peeters, M.: Cryptographic sponge functions. Report, STMicroelectronics, Antwerp, Belgium, January 2011
9. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak SHA-3 submission (2009)
10. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
11. Boura, C., Canteaut, A.: A zero-sum property for the Keccak-f permutation with 18 rounds. In: ISIT 2010s, pp. 2488–2492. IEEE (2010)
12. Boura, C., Canteaut, A.: Zero-sum distinguishers for iterated permutations and application to KECCAK-*f* and Hamsi-256. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 1–17. Springer, Heidelberg (2011)
13. Boura, C., Canteaut, A., De Cannière, C.: Higher-order differential properties of KECCAK and *Luffa*. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 252–269. Springer, Heidelberg (2011)
14. Chakraborti, A., Chang, D., Nandi, M.: Fault based forgeries on CLOC and SILC. In: Latincrypt, LNCS. Springer (2015). https://groups.google.com/forum/#!topic/crypto-competitions/_qxORmqcSrY
15. Das, S., Meier, W.: Differential biases in reduced-round Keccak. In: Pointcheval, D., Vergnaud, D. (eds.) AFRICACRYPT. LNCS, vol. 8469, pp. 69–87. Springer, Heidelberg (2014)
16. Dey, P., Chakraborty, A., Adhikari, A., Mukhopadhyay, D.: Improved practical differential fault analysis of Grain-128. DATE **2015**, 459–464 (2015)
17. Dinur, I., Dunkelman, O., Shamir, A.: New attacks on Keccak-224 and Keccak-256. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 442–461. Springer, Heidelberg (2012)
18. Dinur, I., Dunkelman, O., Shamir, A.: Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 219–240. Springer, Heidelberg (2014)
19. Dinur, I., Dunkelman, O., Shamir, A.: Improved practical attacks on round-reduced Keccak. J. Crypt. **27**(2), 183–209 (2014)
20. Dinur, I., Morawiecki, P., Pieprzyk, J., Srebrny, M., Straus, M.: Cube attacks and cube-attack-like cryptanalysis on the round-reduced Keccak sponge function. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 733–761. Springer, Heidelberg (2015)

21. Duc, A., Guo, J., Peyrin, T., Wei, L.: Unaligned rebound attack: application to Keccak. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 402–421. Springer, Heidelberg (2012)
22. Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on A.E.S. In: Zhou, J., Yung, M., Han, Y. (eds.) ACNS 2003. LNCS, vol. 2846, pp. 293–306. Springer, Heidelberg (2003)
23. FIPS-202. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. National Institute for Standards and Technology, pub-NIST, May 2014
24. Fischer, W., Reuter, C.A.: Differential fault analysis on Grøstl. In: Bertoni, G., Gierlichs, B. (eds.) FDTC 2012, pp. 44–54. IEEE Computer Society (2012)
25. Giraud, C.: DFA on AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) AES 2005. LNCS, vol. 3373, pp. 27–41. Springer, Heidelberg (2005)
26. Giraud, C., Thillard, A.: Piret and quisquater's DFA on AES revisited. IACR Cryptology ePrint Archive, 2010:440 (2010)
27. Hemme, L.: A differential fault attack against early rounds of (Triple-)DES. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 254–267. Springer, Heidelberg (2004)
28. Hemme, L., Hoffmann, L.: Differential fault analysis on the SHA1 compression function. In: Breveglieri, L., Guilley, S., Koren, I., Naccache, D., Takahashi, J. (eds.) FDTC 2011, pp. 54–62. IEEE Computer Society (2011)
29. Jean, J., Nikolic, I.: Internal differential boomerangs: practical analysis of the round-reduced Keccak-f permutation. IACR Cryptology ePrint Archive 2015:244 (2015)
30. Karmakar, S., Chowdhury, D.R.: Differential fault analysis of MICKEY-128 2.0. In: Fischer, W., Schmidt, J. (eds.) FDTC 2013, pp. 52–59. IEEE Computer Society (2013)
31. Kim, C.H.: Differential fault analysis of AES: toward reducing number of faults. Inf. Sci. **199**, 43–57 (2012)
32. Kim, C.H., Quisquater, J.-J.: New differential fault analysis on AES key schedule: two faults are enough. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 48–60. Springer, Heidelberg (2008)
33. Kölbl, S., Mendel, F., Nad, T., Schläffer, M.: Differential cryptanalysis of Keccak variants. In: Stam, M. (ed.) IMACC 2013. LNCS, vol. 8308, pp. 141–157. Springer, Heidelberg (2013)
34. Luo, P., Fei, Y., Fang, X., Ding, A.A., Kaeli, D.R., Leeser, M.: Side-channel analysis of MAC-Keccak hardware implementations. Cryptology ePrint Archive, Report 2015/411 (2015). http://eprint.iacr.org/
35. Morawiecki, P., Pieprzyk, J., Srebrny, M.: Rotational cryptanalysis of round-reduced Keccak. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 241–262. Springer, Heidelberg (2014)
36. Naya-Plasencia, M., Röck, A., Meier, W.: Practical analysis of reduced-round KECCAK. In: Bernstein, D.J., Chatterjee, S. (eds.) INDOCRYPT 2011. LNCS, vol. 7107, pp. 236–254. Springer, Heidelberg (2011)
37. Piret, G., Quisquater, J.-J.: A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 77–88. Springer, Heidelberg (2003)
38. Preneel, B., van Oorschot, P.C.: On the security of iterated message authentication codes. IEEE Trans. Inf. Theory **45**(1), 188–199 (1999)
39. Rivain, M.: Differential fault analysis on DES middle rounds. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 457–469. Springer, Heidelberg (2009)

40. Saha, D., Kuila, S., Chowdhury, D.R.: EscApe: diagonal fault analysis of APE. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 197–216. Springer, Heidelberg (2014)
41. Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential fault analysis of the advanced encryption standard using a single fault. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 224–233. Springer, Heidelberg (2011)