Noname manuscript No.
(will be inserted by the editor)

Differential Fault Analysis of SHA-3 under Relaxed Fault Models

Pei Luo · Yunsi Fei ✉ · Liwei Zhang · A. Adam Ding

Received: date / Accepted: date

Abstract Keccak-based algorithms such as Secure Hash Algorithm-3 (SHA-3) will be widely used in crypto systems, and evaluating their security against different kinds of attacks is vitally important. This paper presents an efficient differential fault analysis (DFA) method on all four modes of SHA-3 to recover an entire internal state, which leads to message recovery in the regular hashing mode and key retrieval in the message authentication code (MAC) mode. We adopt relaxed fault models in this paper, assuming the attacker can inject random single-byte faults into the penultimate round input of SHA-3. We also propose algorithms to find the lower bound on the number of fault injections needed to recover an entire internal state for the proposed attacks. Results show that on average the attacker needs about 120 random faults to recover an internal state, while he needs 17 faults at best if he has control of the faults injected. The proposed attack method is further extended for systems with input messages longer than the bitrate.

Keywords SHA-3 · Keccak · Differential fault analysis · Hardware security

Pei Luo
Department of Electrical and Computer Engineering
Northeastern University, Boston, USA
E-mail: silenceluo@gmail.com

Yunsi Fei ✉
Department of Electrical and Computer Engineering
Northeastern University, Boston, USA
E-mail: yfei@ece.neu.edu

Liwei Zhang
Department of Mathematics
Northeastern University, Boston, USA
E-mail: zhang.liw@husky.neu.edu

A. Adam Ding
Department of Mathematics
Northeastern University, Boston, USA
E-mail: a.ding@neu.edu

1 Introduction

Keccak, a family of sponge functions, can be used to build various security modules widely used in crypto systems, including hash function, symmetric cryptographic function, pseudo-random number generator and authenticated encryption [1]. The new SHA-3 standard [2] is based on Keccak. Two candidates for CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness), Ketje and Keyak, are also built upon Keccak. Therefore, the security of Keccak against different kinds of attacks is critical to security system design. In this paper, we use SHA-3 as an example for the security analysis of Keccak, and present an efficient differential fault analysis (DFA) method on all four modes of SHA-3 function.

DFA is a powerful and efficient attack method, which utilizes the dependency of the faulty output on the internal intermediate variables to recover the secret. DFA has been used to break many cryptographic algorithms. For example, it has been used to extract the secret key of symmetric ciphers. It was first introduced to conquer the Data Encryption Standard (DES) algorithm [3], and later it was used to break the Advanced Encryption Standard (AES) [4]. Many other ciphers have also been hacked by DFA, including CLEFIA [5], Mickey [6, 7] and Grain [8, 9].

Some existing hash standards have been evaluated against DFA attacks, including SHA-1 [10], Streebog [11], MD5 [12] and GrøStl [13]. For hash functions in general usage, DFA can be used to retrieve the original message. For example, it is found in [12] that on average 144 random faults are required for MD5 in order to discover the input message block. For GrøStl algorithm, on average 280 single-bit faults are needed to invert each compression step. When hash functions are used in the message authentication code (MAC) mode with a secret key, DFA also becomes an effective method to recover the key and then generate forgery messages and corresponding MACs against authentication. For example, in [10], the input of SHA-1, including the secret key and message, is fully extracted with about 1000 random faults. For Streebog, an average number of faults that varies between 338-1640 is required to recover the secret key for different MAC settings [11]. When GrøStl is used in a keyed hash function, about 300 faults are needed to retrieve the secret key [13].

Previous works on Keccak mainly focus on side-channel power analysis and cryptanalytic collisions attacks [14–21]. To the best of our knowledge, only one work about DFA on SHA-3 has been published before us [22]. In [22], a single-bit fault model is used , and only two modes of SHA-3 with longer digest length, namely SHA3-384 and SHA3-512, have been discussed. Injection of single-bit faults into crypto systems requires higher control precision and sometimes invasive methods like laser emission, which are costly and also less effective as the technology scales down. Typical non-invasive fault injection methods, such as clock glitches and supply voltage variation, are more general and would affect a group of bits in intermediate states all together, i.e., inducing byte-level faults. Our previous work [23] extends DFA to the other two modes of

SHA-3 with shorter digest length, SHA3-224 and SHA3-256, under a relaxed single-byte fault model. It analyzes fault propagation of SHA-3 under byte-level fault injection, and proposes two different ways to address the issue of short observable digests in SHA3-224 and SHA3-256.

This paper is an extension of our previous work [23]. In addition to the random fault model, we propose an optimization method with heuristics, so that the DFA can use the least chosen faults to recover the internal state. We also include more substantial discussion on extending the DFA to SHA-3 systems with input messages longer than the bitrate. The contributions of this work are as following:

- We introduce the concept of fault propagation into DFA of SHA-3, and this formal method will ensure the extendability of proposed DFA.
- We extend DFA on SHA-3 from single-bit fault model to more relaxed fault models, and from two SHA-3 functions with longer digest to all four of them. For example, we can break all four SHA-3 functions under single-byte fault model, and break SHA3-384 and SHA3-512 under 16-bit fault model in this work.
- An optimization method with heuristics is proposed, and this method can be used to optimize DFA so that the proposed method can use the fewest chosen faults to recover the internal state.
- Discussion about how variable key and message length will affect DFA is given.

We simulate all the proposed methods for all four modes of SHA-3. Results show that, for SHA3-384 and SHA3-512, about 120 random single-byte faults are needed to recover an entire internal state, while about 500 single-bit random faults are needed in previous work [22]. When the fault injection can be controlled, our simulation results of the heuristics show that only 17 selected single-byte faults and 129 selected single-bit faults are required to recover the internal state, under the two different fault models, respectively. Our attack method can break SHA3-224 and SHA3-256 as well, while the numbers of required random effective faults are about 250 and 150, respectively.

The rest of this paper is organized as follows. In Section 2, we present the basic knowledge of SHA-3 that will be used in this paper, and describe the fault models. In Section 3, we analyze the fault propagation process in SHA-3 and construct the fault signatures. In Section 4, we present the attack on SHA3-384 and SHA3-512 using the proposed fault signature method under the relaxed fault models. In Section 5, we extend the attack to SHA3-224 and SHA3-256, and show the method to further improve the attack efficiency. In Section 6, we propose a heuristic algorithm to improve the efficiency of the DFA with more control of the faults injected, and derive the lower bound on the number of fault injections needed for the proposed attacks. In Section 7, we discuss attacks on SHA-3 systems with the input message size larger than the bitrate, and the protection of SHA-3 systems against DFAs. Finally, we conclude the paper in Section 8.

2 Preliminaries of SHA-3 and Differential Fault Analysis

2.1 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] [2], where each internal state is 1600-bit organized in a 3-D array, as shown in Fig. 1, and d is 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 Fig. 1.

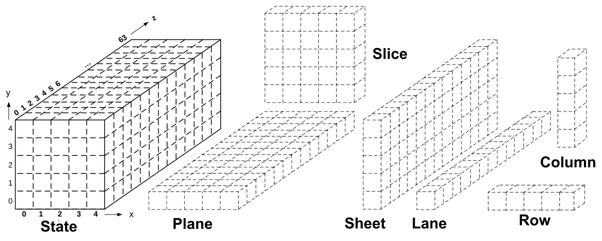


Fig. 1: State data structures used in Keccak [1]

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 digest by an f permutation function. The Sponge architecture is shown in Fig. 2, where r is called the bitrate and c is the capacity ($r + c$=1600). Here f_0 to f_5 are all the same f permutation function.

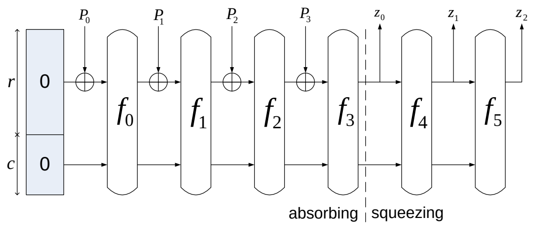


Fig. 2: The sponge construction

The number of f functions in the absorbing phase is determined by the message size, and the digest size (d in Keccak-f[1600, d]) will decide the number of f function in the squeezing phase together with bitrate r. SHA-3 family consists of four digest size ($d = 224, 256, 384, 512$), namely SHA3-224, SHA3-256, SHA3-384, and SHA3-512 [2]. We will discuss all four SHA-3 functions under relaxed fault models in this paper. Note here that for SHA3-d function, $c = 2d$ and $r + c = 1600$.

In this paper, we first simplify the setting by assuming that the length of the input message is smaller than the bitrate r, and as $r > d$ holds for all four modes of SHA-3 function, there will be only one f function involved for absorbing and squeezing. In Section 7, we will extend the attacks for input messages with variable length, in which multiple f functions will be involved.

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

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

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 neighbor columns. We denote the input of θ as θ_i while the output as θ_o, and the operation is given as follows:

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

– ρ is a rotation over the state bits along z-axis.

– π is a permutation over the state bits within slices.

– 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 successive bits along the x-axis:

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

The χ operation is reversible, each χ_i bit can be expressed in below formula which involves all five bits of χ_o in a row [1, 24]:

$$\begin{aligned}\chi_i(x, y, z) = \chi_o(x, y, z) \oplus \overline{\chi_o(x+1, y, z)} \cdot \big(\chi_o(x-1, y, z) \oplus \chi_o(x+2, y, z) \\ \oplus \chi_o(x-1, y, z) \cdot \chi_o(x+3, y, z)\big).\end{aligned} \tag{2}$$

– ι is a binary XOR with a round constant.

Besides the general hash mode, SHA-3 will be widely used for authentication, and MAC-Keccak is the recommended MAC function. It is designed to securely create a MAC by hashing the concatenation of the key and the message [25]:

$$MAC(M, K) = H(K||M). \tag{3}$$

When SHA-3 is used in general hash mode, the goal is to recover the input message; and when SHA-3 is used in MAC mode, the goal is to recover the authentication key.

2.2 Notations and Data Structure

We annotate the last two rounds of SHA-3 operations and important intermediate states in Fig. 3, and use these notations in the rest of this paper. We pick the penultimate round input (θ_i^{22}) as the fault injection point, and the target is to recover the whole internal state of χ_i^{22} (1600 bits) with access to the digest H at much shorter length, which is d-bit for SHA3-d function. The key insight of DFA is that the fault injection reveals some internal state bits through the differential digest output (the difference between the correct digest and faulty digest).

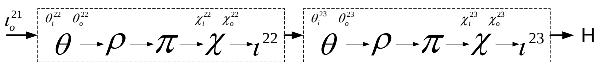

ι_o^{21}
θ_i^{22} θ_o^{22} χ_i^{22} χ_o^{22}
$\theta \to \rho \to \pi \to \chi \to \iota^{22}$
θ_i^{23} θ_o^{23} χ_i^{23} χ_o^{23}
$\theta \to \rho \to \pi \to \chi \to \iota^{23}$
H

Fig. 3: Notations for operations and intermediate states

For one f function, if the whole internal state χ_i^{22} is recovered, the input of SHA-3, S_0, can be recovered because all five operations in Keccak are reversible. We will extend the attacks to SHA-3 systems with multiple f functions in Section 7.1.

For commonly used SHA-3 implementations, data structures are organized along each lane [26,27]. There are two commonly used implementation methods for Keccak, non-interleaved and interleaved implementation [28]. One byte is eight consecutive bits in one lane for non-interleaved implementations. Bit interleaving is a technique to put the bits in even positions and odd positions of one lane into separate words (bytes) for more efficient implementation.

For interleaved and non-interleaved implementations, different bit organization structure in a byte or word will result in different fault propagation characteristics. As bit interleaving technique has been widely used in SHA-3 implementations in embedded cryptographic systems [26, 27], we will evaluate both of these two implementations in our work. We refer to the source code provided online [26] for all implementations and simulations in this paper.

2.3 The Fault Model Used in This Paper

Our DFA is based on byte-level faults. General fault injection methods, including clock glitches and supply voltage variations, will affect multiple bits in a data structure simultaneously. For example, one byte will be affected on 8-bit architectures, and one 16-bit word will be affected on 16-bit architectures.

Multiple concurrent bit faults will interfere with each other during operations in the hash algorithm, and considering only individual independent single-bit faults does not address these interactions. To illustrate our proposed method, we use the fault model of single-byte as example:

- The attacker can inject faults into one byte of the penultimate round input θ_i^{22};
- 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 digest outputs, H and $H^{'}$, which are d-bit instead of 1600 bits;
- The attacker can inject multiple random faults into the same input message for different execution runs.

Besides single-byte fault model, we will also check our method under single-word (16-bit) fault model. As the single-bit fault model used in [22] is just a special case of the single-byte fault model, we will also also check attack results under that model.

3 Fault Propagation and Fault Signature

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 (FS) in this paper. Different fault signatures can be used to identify the injected fault, and then recover the internal state bits.

Different from previous crypto systems like AES and DES, which are organized and operated in several modules (like 8-bit bytes), Keccak is organized at bit level and operations are performed on bits. For block ciphers like AES, the attacker can observe the differential output of 16 bytes to identify the injected fault positions and values. It has been shown in [22] that differential output of SHA-3 has much more complicated characteristics, and therefore DFA methods for previous block ciphers cannot be applied to Keccak based functions. In this section, we show the fault propagation process in SHA-3 and extract the fault signature for each possible injected fault.

3.1 Observable Hash Digest

For DFA, the first step is to select a comparison point (intermediate state of the algorithm), where information obtained from the differential outputs is used to match the various patterns of the fault propagation so as to identify the injected fault or recover secret message or key. In [22], 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 the bottom plane (320 bits) of χ_i^{23}:

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

and the differential, $\Delta\chi_i^{23}(X, 0, Z)$, can be derived. When selecting the comparison point at χ_i^{23}, we need to construct fault signatures at χ_i^{23}, $FS_{\chi_i^{23}}$, for attacks in this paper.

For SHA3-224 and SHA3-256, only a partial bottom plane of the output of the f function is observed, and therefore χ^{23} cannot be inverted directly, according to (2). For SHA3-224 and SHA3-256, we present corresponding attack methods in Section 5.

3.2 Fault Signature Generation

For 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, defined as the differential of the state byte of the penultimate round input) ranges from 1 to 255, where for each bit of F, 0 means the corresponding state bit does not change, and 1 means the state bit flips. For example, $F = 1$ means the lowest bit of the state byte flips. For any possible fault (F) at any one of the 200 positions (P), we denote the corresponding fault signature at χ_i^{23} as $FS_{\chi_i^{23}}[P][F]$. Without further 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 byte P of θ_i^{22}.

For faults injected at θ_i^{22}, it will propagate to χ_i^{23} through the operations shown in Fig. 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 XOR operations) while χ is non-linear (consisting of 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 and 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 the state bit and its two neighbor columns in different sheets. 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 an 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 Fig. 4.

```
x=0:
   10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  y=0
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00001000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   4
x=1:
   00000000 00000000 00000000 00000000 00000000 00001000 00000000 00000000  y=0
   00000000 00000000 00000100 00000000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   4
x=2:
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  y=0
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00000000 00100000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00000000 00000000 10000000 00000000 00000000   4
x=3:
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  y=0
   00000000 00000000 00000000 00000000 00000000 00000100 00000000 00000000   .
   00000000 01000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   4
x=4:
   00000000 00000001 00000000 00000000 00000000 00000000 00000000 00000000  y=0
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   .
   00100000 00000000 00000000 00000000 00000000 00000000 00000000 00000000   4
z=0                               ... ... ... ... ... ... ... ...                 63
```

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

Due to 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. As the fault keeps propagation, two important processes, χ^{22} and θ^{23}, will determine the fault signature $FS_{\chi_i^{23}}$. 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 propagation of the faults):

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

We next analyze fault propagation of χ^{22} and θ^{23} in following section respectively.

3.2.1 Fault Propagation in χ^{22} - FP_χ

χ is the only nonlinear operation in Keccak, and its AND operation leaks information of its input state bits with fault(s) on χ_i. Under the single-bit fault model in [22], no more than one bit will be polluted in each row of χ_i^{22}, as also shown in Fig. 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 $\Delta a_o = \Delta a_i \oplus (1 \oplus b_i) \cdot \Delta c_i = b_i$.
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 $\Delta a_o = \Delta a_i \oplus \Delta b_i \cdot c_i \oplus (1 \oplus b_i) \cdot \Delta c_i \oplus \Delta b_i \cdot \Delta c_i = \overline{b_i \oplus c_i}$.

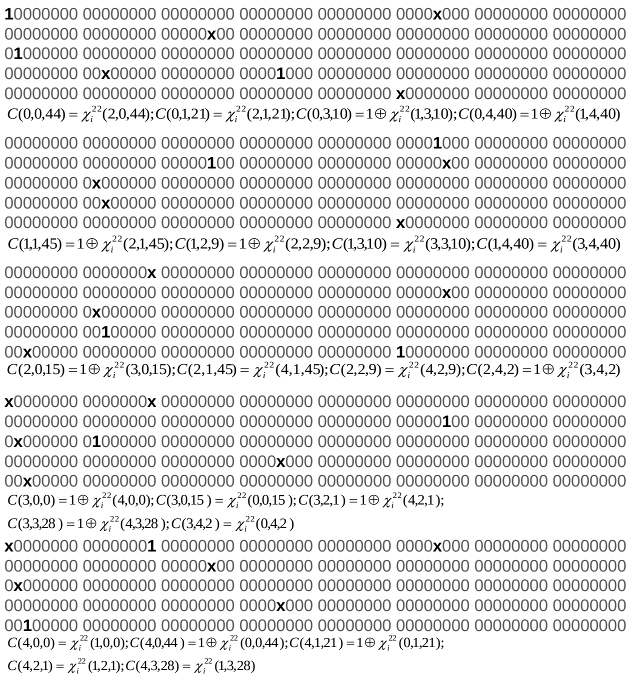
In summary, we can denote the fault signature for bit $\chi_o^{22}(x, y, z)$ as in TABLE 1.

According to the above analysis, we can see that the nonlinear χ operation may cause leakage of some χ_i^{22} bits in the differential χ^{22} output. We present the whole fault pattern at χ_o^{22} as in Fig. 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.

In Fig. 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

Table 1: Fault propagation of operation χ^{22}

Fault at χ input $\varDelta\chi_i^{22}([x:x{+}2],y,z)$	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]	$\overline{\chi_i^{22}(x+1,y,z)}$
[1,1,0]	$\overline{\chi_i^{22}(x+2,y,z)}$
[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]	$\overline{\chi_i^{22}(x+1,y,z)\oplus\chi_i^{22}(x+2,y,z)}$



```
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
```
$C(4,0,0)=\chi_i^{22}(1,0,0);C(4,0,44)=1\oplus\chi_i^{22}(0,0,44);C(4,1,21)=1\oplus\chi_i^{22}(0,1,21);$

$C(4,2,1)=\chi_i^{22}(1,2,1);C(4,3,28)=\chi_i^{22}(1,3,28)$

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

knowledge of $\varDelta\chi_o^{22}(0,0,44)$ and the injected fault, he can construct the corresponding fault signature and then recover the bit $\chi_i^{22}(2,0,44)$.

3.2.2 Fault Propagation in θ^{23} – FP_θ

Each bit of θ_o is the XOR result of 11 input bits: its corresponding input bit with two nearby input columns. 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:

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

Each bit of $FS_{\chi_o^{22}}$ can be denoted as 0, 1, or a function of certain χ_i^{22} bits. For each bit of $FS_{\theta_o^{23}}$, some of the corresponding 11 $FS_{\chi_o^{22}}$ bits may depend on the same χ_i^{22} bits, and therefore some dependencies will be eliminated with the operation of XOR. This is a key insight for our byte-level (multiple bit) fault propagation analysis. For example, in the interleaved implementation, 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 to build the fault signature at χ_i^{23} with $FS_{\theta_o^{23}}$ constructed from 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 Fig. 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.

With the observed bits of $\Delta\chi_i^{23}$ and the fault signatures, the attacker 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 Fig. 6, with the single-bit fault injected at $\theta_i^{22}(0, 0, 0)$, the attacker 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 $\chi_i^{22}(0, 0, 44)$.

In this section, we showed the fault propagation process in SHA-3, and analyzed the composition of fault signatures at χ_i^{23}. In next two sections, we will show how to use the constructed fault signatures to conquer all four modes of SHA-3.

xx100000 00**xx**000**1** 00000**x1**0 0000**x**000 00000000 0**x**00**x1x**0 00000000 00000000

$E(0,0)=1\oplus\chi_i^{22}(1,0,0)$; $E(0,1)=\chi_i^{22}(1,2,1)$; $E(0,10)=1\oplus\chi_i^{22}(2,2,9)$; $E(0,11)=\chi_i^{22}(3,3,10)$; $E(0,46)=1\oplus\chi_i^{22}(2,1,45)$;
$E(0,21)=1\oplus\chi_i^{22}(0,1,21)$; $E(0,28)=\chi_i^{22}(1,3,28)$; $E(0,41)=\chi_i^{22}(3,4,40)$; $E(0,44)=1\oplus\chi_i^{22}(0,0,44)\oplus\chi_i^{22}(2,0,44)$;

0**x**000000 **1**0000000 0000**x1**00 **xxx**00000 00000000 0000**11**0**x** 000000**x1** 0000**x**000

$E(1,1)=1\oplus\chi_i^{22}(2,1,21)$; $E(1,20)=1\oplus\chi_i^{22}(1,4,40)$; $E(1,24)=\chi_i^{22}(2,0,44)$; $E(1,25)=1\oplus\chi_i^{22}(2,1,45)$;
$E(1,26)=\chi_i^{22}(4,1,45)$; $E(1,47)=1\oplus\chi_i^{22}(3,4,2)$; $E(1,54)=1\oplus\chi_i^{22}(4,2,9)\oplus\chi_i^{22}(1,3,10)$; $E(1,60)=1\oplus\chi_i^{22}(3,0,15)$;

10000000 **x**0000000 000**x**000**1** **x1**000000 00000000 0000**xxx**0 0000**xx**00 000**x**0000

$E(2,8)=1\oplus\chi_i^{22}(4,3,28)$; $E(2,19)=\chi_i^{22}(3,4,40)$; $E(2,24)=1\oplus\chi_i^{22}(2,1,45)$; $E(2,44)=1\oplus\chi_i^{22}(4,0,0)$; $E(2,45)=1\oplus\chi_i^{22}(4,2,1)$;
$E(2,46)=\chi_i^{22}(0,4,2)$; $E(2,52)=1\oplus\chi_i^{22}(2,2,9)\oplus\chi_i^{22}(4,2,9)$; $E(2,53)=1\oplus\chi_i^{22}(3,3,10)$; $E(2,59)=\chi_i^{22}(0,0,15)$;

00**x**00000 00000000 000000**xx** **1**00000**x1** 0000**x1**00 000**x**0000 0**xx**00000 00000**1**00

$E(3,2)=1\oplus\chi_i^{22}(0,0,44)\oplus\chi_i^{22}(4,1,45)$; $E(3,22)=\chi_i^{22}(1,0,0)$; $E(3,23)=1\oplus\chi_i^{22}(1,2,1)\oplus\chi_i^{22}(3,4,2)$; $E(3,30)=\chi_i^{22}(4,2,9)$;
$E(3,36)=1\oplus\chi_i^{22}(3,0,15)$; $E(3,43)=1\oplus\chi_i^{22}(0,1,21)$; $E(3,49)=1\oplus\chi_i^{22}(4,3,28)$; $E(3,50)=\chi_i^{22}(1,3,28)$;

00000000 000000**xx** **x**000000**1** 0**x**000**x**00 0000**x**000 00**x1**0000 0000000**x** 000**x**0000

$E(4,14)=1\oplus\chi_i^{22}(4,0,0)$; $E(4,15)=\chi_i^{22}(4,2,1)$; $E(4,16)=\chi_i^{22}(0,4,2)$; $E(4,25)=1\oplus\chi_i^{22}(1,3,10)$; $E(4,29)=\chi_i^{22}(0,0,15)$;
$E(4,36)=\chi_i^{22}(2,1,21)$; $E(4,42)=1\oplus\chi_i^{22}(4,3,28)$; $E(4,55)=1\oplus\chi_i^{22}(1,4,40)$; $E(4,59)=1\oplus\chi_i^{22}(2,0,44)$;

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

4 Differential Fault Analysis of SHA3-384 and SHA3-512

In this section, we use the constructed fault signatures in the previous section to conquer SHA3-384 and SHA3-512. In Section 4.1, we will present the method to identify the injected fault, including the position P and value F, using the constructed fault signatures. Then we show how to recover some χ_i^{22} bits using the identified fault in Section 4.2.

4.1 Fault Position P and Value F Recovery

We separate the 320 observable $\Delta\chi_i^{23}$ bits (five lanes) into two groups:

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

We would like to use the observed $\Delta\chi_i^{23}$ to infer the fault injection position (at byte P_0) and the fault value (F_0). For any fault F at position P, the fault signature at the bottom plane of χ_i^{23} consists of five lanes, and we can separate the 320 bits of $FS_{\chi_i^{23}}[P][F](x,y,z)$ $(y=0)$ into three groups:

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

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

- For any bit in $FS_{\chi_i^{23}}[P_0][F_0].white$, this bit should be in $\Delta\chi_i^{23}.white$;
- For any bit in $FS_{\chi_i^{23}}[P_0][F_0].black$, 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 summarize the above relationships as following set relations:

$$\begin{cases} FS_{\chi_i^{23}}[P][F].white \subseteq \Delta\chi_i^{23}.white \\ FS_{\chi_i^{23}}[P][F].black \subseteq \Delta\chi_i^{23}.black \\ \Delta\chi_i^{23}.white \subseteq \{FS_{\chi_i^{23}}[P][F].white \cup FS_{\chi_i^{23}}[P][F].grey\} \\ \Delta\chi_i^{23}.black \subseteq \{FS_{\chi_i^{23}}[P][F].black \cup FS_{\chi_i^{23}}[P][F].grey\} \end{cases} . \tag{7}$$

By checking relationships in (7), the attacker can exclude many positions and fault values. If only one position with one fault value satisfies these relationships, 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 implement the attacks on both interleaved and non-interleaved versions of Keecak implementations in C++ [26], and run all the fault signature generation and mapping between the observed differential and the hypothesized signatures on a workstation, which consists of an Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz and 8GB memory. Results show that our one-time fault signature generation algorithm takes about 14.5s for offline execution, and it takes less than 0.3 ms to find the correct injected fault and recover the internal state bits from one fault.

For the non-interleaved version of SHA-3, using our algorithm, the attacker can find the unique injected fault with probability 99.13% under single-byte fault model. For the rest 0.87% probability, more than one faults satisfy the relationships in (7). For interleaved version, the attacker can find the unique fault with 100% probability under one single-byte fault injection. We define such unique faults as effective faults. As only effective faults are useful for identifying state bits, the higher percentage of the effective fault, the more efficient the attacks will be.

We also check our algorithms under other fault models. For the single-bit fault model used in [22], our algorithms find the unique faults with 100% probability. For the single-word (16-bit) fault model, the effective fault rate is about 40% for both interleaved and non-interleaved implementations. This is because for single-word faults, most bits of the final digest H (and internal state θ_o^{23}) will be polluted (more confusion), and therefore the difference between the signatures of two faults is less distinct.

As the results for non-interleaved and interleaved implementations are similar, and the methods for fault identification and χ_i^{22} bits recovery are the same for these two implementations, the rest of this paper presents only results on non-interleaved implementation, and focuses on the single-byte fault model.

4.2 χ_i^{22} Bits Recovery

Previous sections introduce the algorithms to derive fault signatures and use the signatures to infer the injected fault information. In this section, we describe the algorithms to recover bits of the internal state χ_i^{22}.

As demonstrated in Fig. 6 and Section 3.2.2, each bit of $FS_{\chi_i^{23}}$ may involve one or several bits of χ_i^{22}. Once the unique fault value at a certain position is identified, all the 'x' bits in the $FS_{\chi_i^{23}}$ are known to be zero or one. First, those bits that only depend on a single bit of χ_i^{22} are checked to recover the corresponding χ_i^{22} bits. Then these newly recovered χ_i^{22} bits are used in those signature bits that depend on multiple χ_i^{22} bits to recover other bits.

Note that for each single-bit fault injected at θ_i^{22}, 22 bits of χ_i^{22} can be recovered. With a multi-bit fault (n-bit) injected at θ_i^{22}, up to $22 * n$ bits can be recovered. However, the θ^{23} operation may cancel some χ_i^{22} bits by the XOR operation, and the number of χ_i^{22} bits that can be recovered by an n-bit fault is at most $22 * n$.

For fault $1 \leq F \leq 255$ on a byte, the distribution of the number of flipped θ_i^{22} bits (i.e., n) is shown in Fig. 7(a). We conduct an experiment to find the average number of bits recovered by each fault injected. We randomly generate 10^5 messages, and inject random faults at random positions, and count the recovered χ_i^{22} bits for each fault injected. The corresponding results are shown in Fig. 7(b), in which the x-axis is number of recovered χ_i^{22} bits, and the y-axis is the corresponding ratio of faults among all 255 faults.

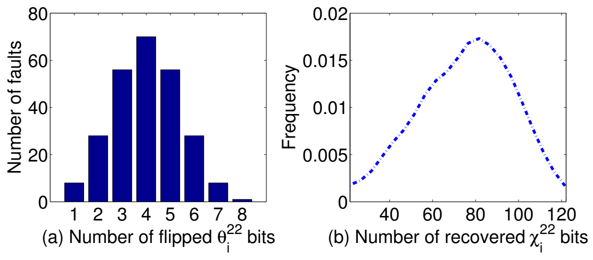


Fig. 7: Distribution of the number of recovered χ_i^{22} bits for single-byte faults

We define the average number of recovered bits for each injected fault as α, and the simulation results of Fig. 7(b) show that α is about 74.97 for randomly injected single-byte faults. We assume the l-th fault can recover ω_l new bits of χ_i^{22} that have not been recovered by the previous $l-1$ faults. We denote the total number of χ_i^{22} bits recovered by the first l injected faults as Ω_l:

$$\Omega_l = \sum_{j=1}^{l} \omega_j. \tag{8}$$

For the l-th fault, the previous $l-1$ faults have already recovered Ω_{l-1} bits of χ_i^{22}, thus the ratio of unrecovered bits can be denoted as $\frac{1600-\Omega_{l-1}}{1600}$. For random messages and randomly injected faults, we can assume that these $1600-\Omega_{l-1}$ bits are randomly distributed in the $1,600$-bit χ_i^{22} state. For SHA-3, we can simplify this problem by assuming that the probability for each bit to be recovered is equal. Thus, for the α bits of χ_i^{22} recovered by the l-th fault, the number of χ_i^{22} bits that have not been recovered by previous $l-1$ faults can be denoted as:

$$\omega_l = \frac{1600 - \Omega_{l-1}}{1600} \cdot \alpha \tag{9}$$

For the first injected fault, there will be no collision and thus $\omega_1 = \alpha$, which is 74.97 for randomly injected single-byte fault according to the results in Fig. 7. Thus we can plug this result into (8) and (9) to emulate the attack process and show the theoretical result in Fig. 8, where the x-axis is the number of random faults injected, and the y-axis is the corresponding total number of χ_i^{22} bits recovered. To simulate the attacks on SHA-3, we randomly generate 10^5 messages, and inject 200 random faults at random positions for each message. The experimental attack results are also shown in Fig. 8.

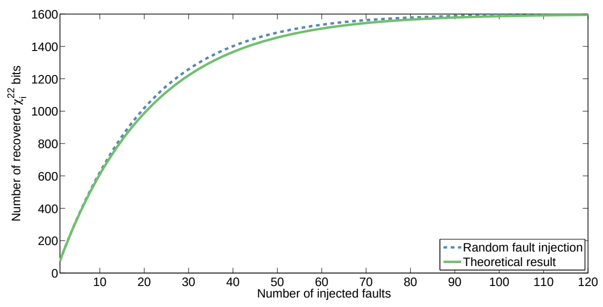


Fig. 8: Recover process

Fig. 8 shows that the theoretical result matches the experimental result very well. It also shows that by injecting random errors into θ_i^{22}, the attacker can recover all the $1,600$ bits of χ_i^{22} using about 120 random faults. Comparing with the attacks on SHA3-384/512 proposed in [22], our method needs much smaller number of faults to recover the whole internal state under relaxed fault models. We also test the theoretical model under random single-bit fault model used in [22], and the experimental result and theoretical result match very well, shown in Fig. 14 in Section 6.

5 Differential Fault Analysis of SHA3-224 and SHA3-256

Section 4 shows the method to identify the injected faults and to recover the internal state χ_i^{22} bits for SHA3-384 and SHA3-512. However, these methods cannot be applied to SHA3-224 and SHA3-256 directly, where the χ^{23} operation cannot be inverted to obtain the input rows with each output row only partially known. In this section, we show the method to recover χ_i^{23} bits first, and then use these intermediate state bits to recover χ_i^{22} of SHA3-224 and SHA3-256. We propose two improved attacks, presented in Section 5.2 and Section 5.3 correspondingly.

5.1 Basic Attacks of SHA3-224 and SHA3-256

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. In Section 5.1.1, we show that with limited information, part of χ_i^{23} on the bottom plane can still be recovered from the observable digest. Attack details using the recovered χ_i^{23} information will be presented in Section 5.1.2.

5.1.1 χ_i^{23} Bits Recovery from the Observable Digest

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 $(a_o, b_o, c_o, d_o, e_o)$ through χ^{-1} operation as:

$$\begin{cases} a_i = a_o \oplus \overline{b_o} \cdot \big(e_o \oplus c_o \oplus e_o \cdot d_o\big) \\ b_i = b_o \oplus \overline{c_o} \cdot \big(a_o \oplus d_o \oplus a_o \cdot e_o\big) \\ c_i = c_o \oplus \overline{d_o} \cdot \big(b_o \oplus e_o \oplus b_o \cdot a_o\big) \\ d_i = d_o \oplus \overline{e_o} \cdot \big(c_o \oplus a_o \oplus c_o \cdot b_o\big) \\ e_i = e_o \oplus \overline{a_o} \cdot \big(d_o \oplus b_o \oplus d_o \cdot c_o\big) \end{cases} . \tag{10}$$

For SHA3-256, for each row, bit e_o is unknown while (a_o, b_o, c_o, d_o) are observable by the attacker; 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 (10), we identify the cases where the input bits are independent of the unknown output bits, and 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 attacker 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 0.75.

- 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 $d_o = 1$, $c_i = c_o$, and the probability of recovering c_i is 0.5.
- For d_i, if $c_o \oplus a_o \oplus c_o \cdot b_o = 0$, $d_i = d_o$, and the probability of recovering d_i is 0.5.
- The value of e_i always depends on e_o, and the attacker 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 attacker can recover 160 bits of χ_i^{23} theoretically. Similarly, for SHA3-224, the attacker can use the same method to recover 112 bits of χ_i^{23} theoretically.

We propose a practical method to recover χ_i^{23} bits. For the same example shown in (10), while a_o, b_o, c_o, d_o are observable by the attacker, the unknown e_o can only be either 0 or 1, then we can make guesses 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 corresponding input row_i^0, row_i^1 using χ inversion operation:

$$\begin{cases} \{a_i^0, b_i^0, c_i^0, d_i^0, e_i^0\} = \chi^{-1}(\{a_o, b_o, c_o, d_o, 0\}) \\ \{a_i^1, b_i^1, c_i^1, d_i^1, e_i^1\} = \chi^{-1}(\{a_o, b_o, c_o, d_o, 1\}) \end{cases}. \tag{11}$$

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 without knowing e_i.

We implement this method for both SHA3-224 and SHA3-256 in C++, and randomly generate 10^5 input messages for each of them. 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 conform to 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 $\chi_i'^{23}(X, 0, Z)$ bits, the attacker can calculate the corresponding $\Delta\chi_i^{23}(X, 0, Z)$ bits. Note that for SHA3-256, the attacker can recover about 160 bits of both $\chi_i^{23}(X, 0, Z)$ and $\chi_i'^{23}(X, 0, Z)$ on average, but the recovered χ_i^{23} and $\chi_i'^{23}$ may have different locations, and therefore the attacker will recover fewer than 160 differential bits of $\Delta\chi_i^{23}(X, 0, Z)$. 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, with more details presented in Table 2.

5.1.2 Injected Fault Identification and χ_i^{22} Bits Recovery

Using the algorithms in Section 5.1.1, the attacker recovers some bits of $\Delta\chi_i^{23}(X, 0, Z)$ from the observable output, instead of 320 bits as in other two modes, SHA3-384 and SHA4-512. We use the same method presented before in Section 4.1 to match the recovered $\Delta\chi_i^{23}(X, 0, Z)$ bits against the fault signatures, and apply the same constraints of (7) to identify the injected fault. Results show that for SHA3-256, the attacker can uniquely identify the fault with a probability 66.61%. For SHA3-224, the effective fault ratio is 30.67% instead. The results are shown in Table 2.

Table 2: Simulation results for SHA3-224 and SHA3-256 with fault injected at θ_i^{22}

	Number of bits recovered		Effective fault ratio
	χ_i^{23}	$\Delta\chi_i^{23}$	
SHA3-224	111.84	93.68	30.67%
SHA3-256	160.12	136.42	66.61%
SHA3-384/512	320	320	99.13%

With the injected fault identified, we use the same method to recover the internal state bits of SHA3-224/256. Simulation results are in Fig. 9. It shows that attacks on SHA3-224 and SHA3-256 are much less efficient than attacks on the other two modes with the digest length higher than 320. The shorter the digest length, the fewer bits of $FS_{\chi_i^{23}}$ and $\Delta\chi_i^{23}$ are available for internal bits recovery, and therefore each effective fault injection recovers fewer bits of χ_i^{22}. Nevertheless, with more effective faults injected to recover groups of bits one by one, these two modes are also susceptible to DFA.

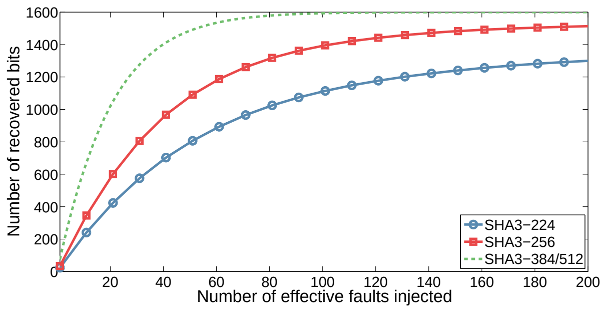


Fig. 9: Number of recovered χ_i^{22} bits for different number of effective faults

In next sections, we propose two methods to improve the fault analysis attacks of SHA3-224 and SHA3-256.

5.2 Improve the Attacks by Using both $FS_{\chi_i^{23}}$ and $FS_{\chi_o^{23}}$

Previous methods all select χ_i^{23} as the comparison point for fault signatures and observable differential digest. However, for SHA3-224 and SHA3-256 ($d =$ 224 and 256 respectively), fewer than d bits of $\Delta\chi_i^{23}$ are available to the attacker and this will cause information loss in attacks. We find that the fault signature at χ_o^{23}, $FS_{\chi_o^{23}}$, if used, can help to extract extra χ_i^{22} bits. This is because all d bits of $\Delta\chi_o^{23}$ are available for SHA3-d function, $FS_{\chi_o^{23}}$ can be used with $\Delta\chi_o^{23}$ to build constraints similar as in (7). We show in this section that with the introduction of $FS_{\chi_o^{23}}$ into attacks, both the effective fault ratio and the number of χ_i^{22} bits recovered by each effective fault increase.

We first present the construction of $FS_{\chi_o^{23}}$ in Section 5.2.1, and then give the results of fault identification and internal state recovery in Section 5.2.2.

5.2.1 Fault Signature $FS_{\chi_o^{23}}$ Construction

We have known that the fault propagation at χ_i^{23}, $FS_{\chi_i^{23}}$, are dependent on both the fault and internal state bits. Through one more step of χ^{23} operation on rows, $a_o = a_i \oplus (\bar{b_i} \cdot c_i)$, the fault propagation function is:

$$\Delta a_o = \Delta a_i \oplus \Delta b_i \cdot c_i \oplus (1 \oplus b_i) \cdot \Delta c_i \oplus \Delta b_i \cdot \Delta c_i. \tag{12}$$

Thus $FS_{\chi_o^{23}}(x, y, z)$ can be denoted as (13), which involves χ_i^{23} bits in addition to fault signature bits $FS_{\chi_i^{23}}$.

$$\begin{aligned} FS_{\chi_o^{23}}(x,y,z) = FS_{\chi_i^{23}}(x,y,z) &\oplus FS_{\chi_i^{23}}(x+1,y,z) \cdot \chi_i^{23}(x+2,y,z) \\ &\oplus (1 \oplus \chi_i^{23}(x+1,y,z)) \cdot FS_{\chi_i^{23}}(x+2,y,z) \\ &\oplus FS_{\chi_i^{23}}(x+1,y,z) \cdot FS_{\chi_i^{23}}(x+2,y,z) \end{aligned} \tag{13}$$

Bits in $FS_{\chi_o^{23}}$ can be 0, 1, or a function over χ_i^{22} and χ_i^{23} bits. When comparing $\Delta\chi_o^{23}$ (d-bits) against $FS_{\chi_o^{23}}$, more constraints are considered so as to help identify the fault and recover the internal bits. This improves the attacks in two ways. First, some faults that cannot be identified by the basic attack before can now be uniquely identified, i.e., the effective fault rate increases. Second, with a certain effective fault, more χ_i^{22} bits can be recovered because $\Delta\chi_o^{23}$ and $FS_{\chi_o^{23}}$ are used.

Note here we cannot use $\Delta\chi_o^{23}$ and $FS_{\chi_o^{23}}$ only for attacks while excluding using $\Delta\chi_i^{23}$ and $FS_{\chi_i^{23}}$, this is because only part of $FS_{\chi_o^{23}}$ bits (instead of d bits) are available. The construction of $FS_{\chi_o^{23}}$ requires knowledge of $\chi_i^{23}(x, y, z)$ bits which are only partially known. Thus attacker should combine information at χ_i^{23} ($\Delta\chi_i^{23}$, $FS_{\chi_i^{23}}$) and χ_o^{23} ($\Delta\chi_o^{23}$, $FS_{\chi_o^{23}}$) together for analysis.

5.2.2 Simulation Results

We construct the fault signature $FS_{\chi_o^{23}}$ and run simulations of the improved attacks. Simulation results show that the effective fault rate rises from 30.67% to 49.12% for SHA3-224, and from 53.28% to 78.73% for SHA3-256. The results of internal state recovery for SHA3-224 and SHA3-256 are shown in Fig. 10.

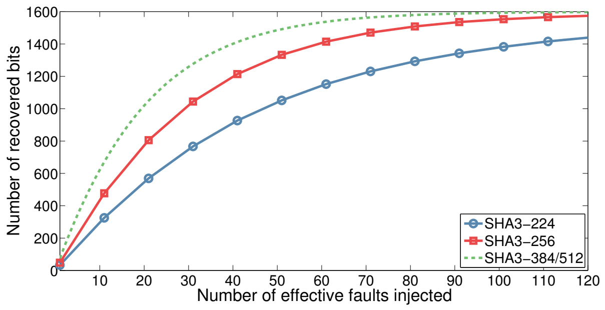


Fig. 10: Number of recovered χ_i^{22} bits for different number of effective faults

After involving $FS_{\chi_o^{23}}$ for attacks, the attack efficiency is improved significantly for both SHA3-224 and SHA3-256. For example, for SHA3-224, the results in Fig. 9 shows that the original attack method needs about 200 faults to recover $1,300$ bits of χ_i^{22}, while the improved method in this section needs only 82 faults to recover the same number of χ_i^{22} bits. It is worth noting that the proposed improved attack does not need any extra knowledge of the target platform, and the generation of fault signature $FS_{\chi_o^{23}}$ does not need much computation.

5.3 Further Improve the Attacks by Injecting Faults at θ_i^{23}

If the attacker can inject faults at multiple points, the attack will achieve higher efficiency. In previous sections, we assume that the attacker injects only faults at θ_i^{22}. In this section, we explore another fault injection point, the last round input θ_i^{23}, and use more faults to recover internal state bits more efficiently.

For the method proposed in Section 5.2, the effective fault ratio 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 the attacker can recover fewer bits of χ_i^{23} and $\Delta\chi_i^{23}$ for SHA3-224 and SHA3-256 than SHA3-384/512 (320 bits). To improve the attack efficiency, we propose to recover more χ_i^{23} bits first, by injecting faults at the last round input θ_i^{23} of SHA3-224 and SHA3-256.

We first present the details of recovering χ_i^{23} bits in Section 5.3.1, and then present the recovery of χ_i^{22} bits for this improved attack in Section 5.3.2.

5.3.1 Recovering More χ_i^{23} by Injecting Faults at θ_i^{23}

To recover χ_i^{23} bits by injecting faults at θ_i^{23} and comparing fault signature and differential fault at χ_o^{23}, we need to calculate the fault propagation from θ_i^{23} to χ_o^{23}. These faults will propagate through θ, 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 3.2.1. 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}, the attacker can recover some remaining bits of χ_i^{23} on the bottom plane that have not been recovered using the algorithm in Section 5.1.1. 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 and some input bits (not all the five) recovered), if the attacker recovers one bit $\chi_i^{23}(x, 0, z)$ that has not been recovered using the algorithm in Section 5.1.1, he can recover all the other unknown bits in this input row. For example, we assume $a_i^0 \neq a_i^1$ in (11) and this bit has been recovered by injecting faults at θ_i^{23}, then the attacker 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, the remaining rows ($\chi_i^{23}(X, 0, z)$, $32 \leq z < 63$) have two bits unknown, and these two bits can only be recovered by injecting faults at θ_i^{23} separately.

We use both fault signatures at χ_i^{23} ($FS^*_{\chi_i^{23}}$) and χ_o^{23} ($FS^*_{\chi_o^{23}}$) to identify the faults injected at θ_i^{23}. Results show that 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 faults injected at θ_i^{23}, we can recover all 320 bits in the bottom plane of χ_i^{23} with multiple faults, and we present the recovery process in Fig. 11.

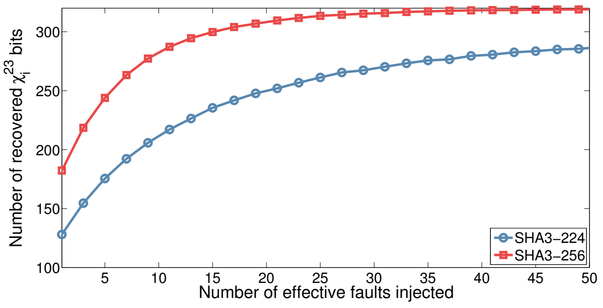


Fig. 11: Recovery of χ_i^{23} bits in the bottom plane by injecting faults at θ_i^{23}

Fig. 11 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 when no faults injected at θ_i^{23} (Table 2). Similar results for SHA3-224 are also presented in

Fig. 11. Note that the attacker does not need to recover all the bits of χ_i^{23} in the bottom plane, he can recover part of χ_i^{23} to improve the efficiency of recovering χ_i^{22}. We will show how the recovery of χ_i^{22} changes with the number of χ_i^{23} bits recovered in next section.

5.3.2 Simulation Results

With more χ_i^{23} bits recovered, the attacker can recover and 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 for the attacker:

– The attacker first recovers part of χ_i^{23} bits using algorithm presented in Section 5.1.1 (with no fault injected at θ_i^{23}).
– The attacker then injects faults at θ_i^{23} to recover the remaining bits of χ_i^{23}.

For simplicity, we assume the attacker can randomly recover from 0 to 64 rows of χ_i^{23} using the algorithm in Section 5.1.1. For each number of recovered rows, we inject random faults at θ_i^{22} to calculate the effective fault ratio and show the results in Fig. 12.

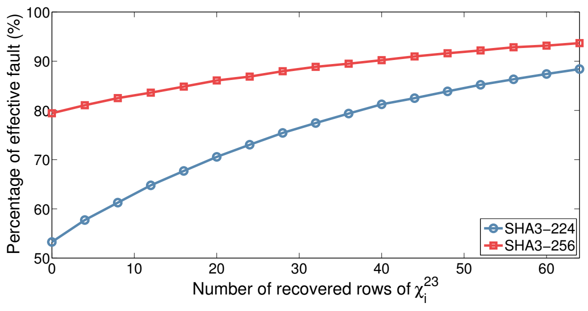


Fig. 12: Effective fault ratio with different number of χ_i^{23} rows recovered

Fig. 12 shows that with more rows of χ_i^{23} recovered, the attacker can identify the faults injected at θ_i^{22} with higher rate. For example, for SHA3-224, the effective fault ratio is 53.28% when only part of χ_i^{23} bottom plane has been recovered using the algorithm in Section 5.1.1, and this ratio rises to 88.34% when all 64 rows (320 bits) of χ_i^{23} bottom plane are recovered. Therefore, by recovering more bits of χ_i^{23} in the bottom plane, the effective fault ratio will increase for both SHA3-224 and SHA3-256.

With knowledge of more χ_i^{23} bits, the attacker can build more equations for χ_i^{22} bits like in Fig. 6, then the 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

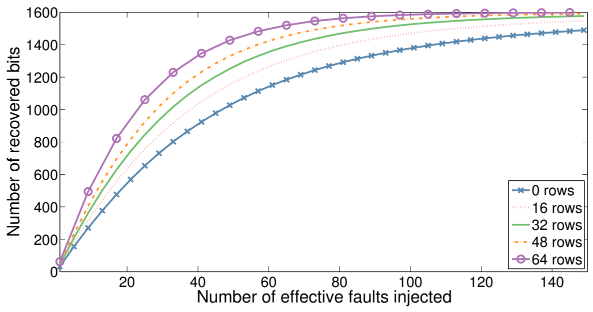


Fig. 13: Number of recovered χ_i^{22} bits for different number of effective faults with a number of χ_i^{23} rows recovered, SHA3-224

SHA3-224 with different numbers of rows recovered in Fig. 13. For SHA3-256, the results are similar, and we will not present the details here.

Fig. 13 shows that the 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 knowledge of the whole bottom plane of χ_i^{23}, he can recover 1590 bits of χ_i^{22} using 110 effective random faults on average. For attacker who cannot inject fault into the last round input, using the improved method in Section 5.2.1, he can only recover about $1,412$ bits (using 110 effective faults) instead.

In conclusion, by injecting faults at θ_i^{23} to recover more state bits of χ_i^{23}, the attacker can identify the faults injected at θ_i^{22} with a higher rate. Consequently, the attacker needs a smaller number of effective faults to recover the same number of χ_i^{22} bits.

As the improved attack method in Section 5.2 does not require extra knowledge of the target system, it should be applied to DFA directly, while the improvement proposed in this section can be applied if the attacker has the ability to inject extra faults at θ_i^{23}.

6 Optimized Attacks with Chosen Faults

In this section, we show an algorithm to optimize the attack if the injected faults (in terms of faulty byte location and fault value) can be controlled.

As shown in the previous sections, injecting an n-bit fault into θ_i^{22} can recover up to $22 * n$ bits of χ_i^{22}, and different faults may have overlapping χ_i^{22} bits. The most efficient attack method should avoid such overlap and use the smallest number of faults to recover the entire internal state. In this section, we show an optimized attack with the smallest number of faults into θ_i^{22} to recover all the bits of χ_i^{22}. This optimization is based on SHA3-384/512 for simplicity, and it can be easily extended to SHA3-224 and SHA3-256.

We formulate this problem into a set covering problem [29]. We assume the attacker can inject n different faults into θ_i^{22}, denote them as $F = \{f_1, f_2, \cdots, f_n\}$, and the bits to recover is a universe U for all the χ_i^{22} bits. For each fault f_i, it can be used to recover a subset of U and we denote it as U_i, where $U_1, U_2, \cdots \subseteq U$, and we denote the cost to inject fault f_i as c_i. The problem is to find a set of $I \subseteq \{1, 2, \cdots, n\}$ that minimize the total cost $\Sigma_{i \in I} c_i$ and $\cup_{i \in I} U_i = U$. The set covering problem is an NP-hard problem. We adopt a greedy heuristic to find solutions, given in Algorithm 1.

Algorithm 1 Optimization of the fault injection attacks

```
Input: Element set $U$, subset set $U = \{U_1, U_2, \cdots, U_n\}$ and their costs
Output: Set cover C with the minimum cost
 1: $C \leftarrow \varnothing$
 2: while $C \neq U$ do
 3:     for all $i = 1$ to n do
 4:         if $U_i \in C$ then
 5:             $\alpha_i = 0$;
 6:         else
 7:             $\alpha_i = \frac{|U_i - C|}{c_i}$
 8:         end if
 9:     end for
10:     Choose $k$ s.t. $\alpha_k = MAX(\{\alpha_i\})$ ;
11:     $C \leftarrow C \cup U_k$;
12: end while
```

Each time, the algorithm chooses a set that gives the highest gain (number of new bits), and we assume that the cost for each set/fault injection is the same (all $c_i = 1$). We use this algorithm to find the "best" fault injection solutions under our byte-level fault model, and the result is shown in Fig. 14(a). It demonstrates that for DFA with byte-level faults, the attacker needs to inject at least 17 faults to recover all the $1,600$ bits of χ_i^{22}, while random fault injection attacks require about 120 faults to retrieve all the bits for SHA3-384/512.

We also use the method and algorithm to analyze the DFA on SHA-3 under single-bit fault model used in [22]. Three curves are presented in Fig. 14(b): the random fault injection experimental result, the random fault injection theoretical prediction, and the optimized attack using the greedy heuristic. Under the single-bit fault model, each fault injected at θ_i^{22} can be used to recover 22 bits of χ_i^{22}, and thus $\alpha = 22$ for the theoretical result. Fig. 14(b) shows that the theoretical result matches the simulation result for random fault injection very well. It also shows that the attacker needs to inject at least 129 single-bit selected faults to recover all the $1,600$ bits of χ_i^{22}, while he needs about 500 single-bit random faults to recover the state.

The optimized attack we present in this section is the lower bound of the proposed differential fault attacks. For SHA3-224 and SHA3-256, similar method can be used to find the lower bound of the attacks.

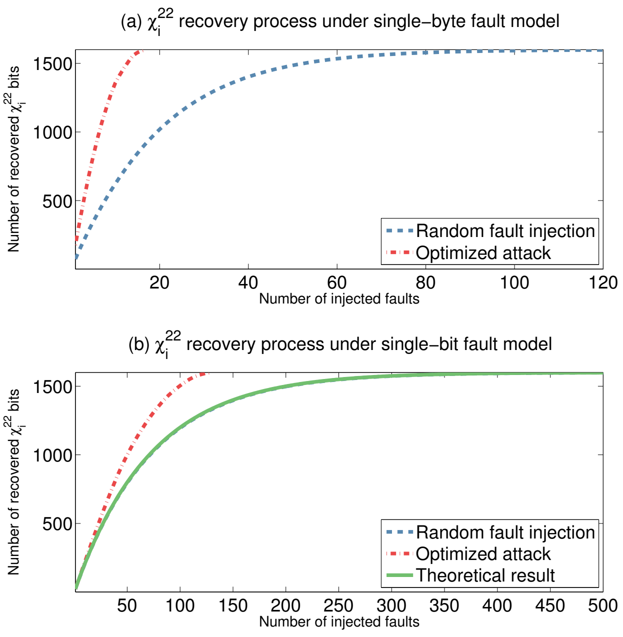


Fig. 14: Optimized fault injection attacks at byte level

7 Discussions

7.1 SHA-3 Systems with Long Input Message

In previous sections, we simplify the analysis by assuming that only one f function is involved for absorbing and squeezing. In this section, we extend the proposed attack by assuming that the size of the input message can be larger than the bitrate r. Meanwhile, we assume that the digest size can be larger than r, instead of just d bits for SHA3-d function. Therefore, multiple f functions may be involved for absorbing and squeezing.

First of all, we assume that the digest size is still d-bit for SHA3-d function:

- There are n $(n \geq 1)$ f functions $(f_0 \cdots f_{n-1})$ involved for absorbing;
- There are m $(m \geq 0)$ f functions $(f_n \cdots f_{n+m-1})$ involved for squeezing, as $r > d$ for all four SHA3-d functions, no extra f function will be involved for squeezing $(m = 0)$;
- The attacker has control of the input message part of $P_0 \cdots P_{n-1}$ in MAC mode, while he has no access to the messages when SHA-3 is used in hash mode;
- The attacker can observe the digest $z_0 \cdots z_m$ in both hash and MAC mode.

For MAC-Keccak system, the attacker can inject faults into the penultimate round input of the last permutation f_{n-1}, and use the proposed method in this paper to recover the input of f_{n-1}, $f_{n-1}(in)$. Combining with P_{n-1}, the attacker can recover $f_{n-2}(in)$. There are two possible situations for the key length l_k here:

– If $l_k \leq r$, the attacker can iteratively recover P_0 which contains the key used for MAC, and therefore recover the secret key.
– If $l_k > r$, for example, all the bits of P_0 and part of P_1 bits are key bits, then the attacker can recover $f_1(in)$, which is

$$f_1(in) = f(P_0||0^c) \oplus (P_1||0^c), \tag{14}$$

then the attacker will be unable to recover the key bits contained in P_0 and P_1 directly. The attacker needs to make assumption of the key bits in P_1 so as to recover the key bits in P_0. The difficulty increases rapidly with the number of the key bits in P_1.

As the key size in MAC system is usually much smaller than r (for example, 128 bits or 256 bits), the proposed DFA is a great threat for SHA-3 based MAC systems. We propose to increase the length of MAC key for higher resillence against fault injection attacks.

For SHA-3 system in general hash mode, the attacker will have no access to the input message $P_0 \cdots P_{n-1}$. Therefore, after recovering $f_{n-1}(in)$, he will be unable to further recover $f_{n-2}(out)$ without knowledge of P_{n-1}. Thus the attacker will be unable to recover the original input message in hash mode directly if the message length is greater than bitrate r.

For modified SHA-3 system which allows the digest size larger than d, there may be extra f functions involved for squeezing, and the digest comes from multiple z_i. As all the z_i are observable, z_0 will be used to attack f_{n-1}. The length of z_0 will be r-bit, which is 1,152, 1,088, 832 and 576-bit for SHA3-224, SHA3-256, SHA3-384 and SHA3-512 respectively. More bits than original (224, 256, 384 and 512-bit) are available, and the attacks become easier. In conclusion, the proposed DFA method is still applicable for SHA-3 systems which involve extra f functions for squeezing.

7.2 Countermeasure against Differential Fault Analysis

As unprotected SHA-3 systems are vulnerable to DFAs, countermeasures should be added into SHA-3 systems to improve their resilience against DFA.

One method is to implement detection of system disturbance which are used to inject faults. For example, a clock glitch and power supply disturbance detection module [30,31] can be inserted into SHA-3 implementations to detect fault injections.

Another kind of widely used countermeasures relies on some redundancy to check the integrity of the intermediate results, such that errors caused by injected faults will be detected. For example, parity checking codes are used to detect the injected faults in SHA-3 [32]. Similarly, another copy of modified Keccak implementation can be introduced into the system for error detection [33,34]. Such schemes can detect errors in the system caused by injected faults, and thus the attacker will have no access to the faulty results to conduct DFA.

8 Conclusion and Future Work

In this paper, we propose efficient DFA methods for all four modes of SHA-3 functions under relaxed single-byte fault models. Results show that our method can effectively identify the injected faults, and then recover the corresponding internal state bits for all four SHA-3 functions. Meanwhile, we also present the lower bound of the proposed attacks and extend the attacks to systems with input message longer than bitrate r.

The proposed method can possibly be applied to other relaxed fault models. Under the word (16-bit) fault model, the effective fault ratio is much lower for SHA3-384/512, but the attacks can still work, while it cannot identify any effective fault for SHA3-224 and SHA3-256. Our future work will address attacks on SHA3-224 and SHA3-256 under more relaxed 16-bit and 32-bit fault models. Another line of future work would include more efficient and effective methods to protect SHA-3 platforms against fault injection attacks.

Acknowledgements This work was supported in part by National Science Foundation under grants SaTC-1314655 and MRI-1337854. Simulation code used in this paper is available at http://tescase.coe.neu.edu/.

References

1. G. Bertoni, J. Daemen, M. Peeters, and G. Assche, “The Keccak reference,” *Submission to NIST (Round 3)*, January, 2011.
2. N. F. Pub, “FIPS PUB 202. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,” *Federal Information Processing Standards Publication*, 2015.
3. E. Biham and A. Shamir, “Differential fault analysis of secret key cryptosystems,” in *Advances in Cryptology – CRYPTO*, 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 *5th International Wkshp on Cryptographic Hardware and Embedded Systems, Cologne, Germany, September 8–10*, 2003, pp. 77–88.
5. H. Chen, W. Wu, and D. Feng, “Differential fault analysis on CLEFIA,” in *9th International Conference on Information and Communications Security, Zhengzhou, China, December 12-15*, 2007, pp. 284–295.

6. S. Karmakar and D. R. Chowdhury, "Differential fault analysis of MICKEY-128 2.0," in *Wkshp on Fault Diagnosis and Tolerance in Cryptography*, Aug 2013, pp. 52–59.
7. S. Banik and S. Maitra, "A differential fault attack on MICKEY 2.0," in *15th International Wkshp on Cryptographic Hardware and Embedded Systems, Santa Barbara, CA, USA, August 20-23*, 2013, pp. 215–232.
8. S. Banik, S. Maitra, and S. Sarkar, "A differential fault attack on the Grain family of stream ciphers," in *14th International Wkshp on Cryptographic Hardware and Embedded Systems, Leuven, Belgium, September 9-12*, 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*, 2015, pp. 459–464.
10. L. Hemme and L. Hoffmann, "Differential fault analysis on the SHA1 compression function," in *Wkshp on Fault Diagnosis and Tolerance in Cryptography*, Sept. 2011, pp. 54–62.
11. R. Altawy and A. M. Youssef, "Differential fault analysis of Streebog," in *11th International Conference on Information Security Practice and Experience, Beijing, China, May 5-8*, 2015, 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 *Wkshp on Fault Diagnosis and Tolerance in Cryptography*, Sept. 2012, pp. 44–54.
14. C. Boura and A. Canteaut, "A zero-sum property for the KECCAK-f permutation with 18 rounds," in *IEEE International Symposium on Information Theory*, June 2010, pp. 2488–2492.
15. S. Das and W. Meier, "Differential biases in reduced-round keccak," in *Progress in Cryptology – AFRICACRYPT 2014: 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30*, 2014, pp. 69–87.
16. I. Dinur, O. Dunkelman, and A. Shamir, "Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials," in *20th International Workshop on Fast Software Encryption, Singapore, March 11-13*, 2013, pp. 219–240.
17. 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: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30*, 2015, pp. 733–761.
18. P. Luo, Y. Fei, X. Fang, A. A. Ding, D. R. Kaeli, and M. Leeser, "Side-channel analysis of MAC-Keccak hardware implementations," in *Proceedings of the Fourth Wkshp on Hardware and Architectural Support for Security and Privacy*, June 2015.

19. P. Morawiecki, J. Pieprzyk, and M. Srebrny, "Rotational cryptanalysis of round-reduced keccak," in *20th International Wkshp on Fast Software Encryption, Singapore, March 11-13*, 2013, pp. 241–262.
20. M. Naya-Plasencia, A. Rck, and W. Meier, "Practical analysis of reduced-round Keccak," in *Progress in Cryptology – INDOCRYPT 2011: 12th International Conference on Cryptology in India, Chennai, India, December 11-14*, 2011, pp. 236–254.
21. M. Taha and P. Schaumont, "Side-channel analysis of MAC-Keccak," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, June 2013, pp. 125–130.
22. N. Bagheri, N. Ghaedi, and S. Sanadhya, "Differential fault analysis of SHA-3," in *Progress in Cryptology – INDOCRYPT 2015: 16th International Conference on Cryptology in India, Bangalore, India, December 6-9*, 2015, pp. 253–269.
23. P. Luo, Y. Fei, L. Zhang, and A. Ding, "Differential fault analysis of SHA3-224 and SHA3-256," in *Thirteenth Wkshp on Fault Diagnosis and Tolerance in Cryptography*, Aug. 2016.
24. J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," Ph.D. dissertation, Doctoral Dissertation, March 1995, KU Leuven, 1995.
25. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Cryptographic sponge functions," *Submission to NIST (Round 3)*, 2011.
26. "Reference and optimized code in C," http://keccak.noekeon.org/KeccakReferenceAndOptimized-3.2.zip.
27. P. Pessl and M. Hutter, "Pushing the limits of SHA-3 hardware implementations to fit on RFID," in *15th International Wkshp on Cryptographic Hardware and Embedded Systems, Santa Barbara, CA, USA, August 20-23*, 2013, pp. 126–141.
28. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, "Keccak implementation overview," *Report, STMicroelectronics, Antwerp, Belgium*, 2012.
29. M. Karpinski and A. Zelikovsky, "Approximating dense cases of covering problems," in *DIMACS Wkshp on Network Design: Connectivity and Facilites Location*, 1998, pp. 169–178.
30. K. A. Bowman, C. Tokunaga, J. W. Tschanz, A. Raychowdhury, M. M. Khellah, B. M. Geuskens, S.-L. L. Lu, P. A. Aseron, T. Karnik, and V. K. De, "All-digital circuit-level dynamic variation monitor for silicon debug and adaptive clock control," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 9, pp. 2017–2025, 2011.
31. P. Luo, C. Luo, and Y. Fei, "System clock and power supply cross-checking for glitch detection," Cryptology ePrint Archive, Report 2016/968, 2016.
32. P. Luo, C. Li, and Y. Fei, "Concurrent error detection for reliable SHA-3 design," in *26th edition on Great Lakes Symposium on VLSI*, May 2016, pp. 39–44.
33. P. Luo, L. Zhang, Y. Fei, and A. A. Ding, "An improvement of both security and reliability for Keccak implementations on smart card," Cryptology

ePrint Archive, Report 2016/214, 2016.

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