Algebraic Fault Analysis of SHA-3

Pei Luo*, Konstantinos Athanasiou†, Yunsi Fei*, Thomas Wahl†
silenceluo@coe.neu.edu, konathan@ccs.neu.edu, yfei@ece.neu.edu, wahl@ccs.neu.edu
*Electrical & Computer Engineering Department, Northeastern University, Boston, MA 02115 USA
†College of Computer and Information Science, Northeastern University, Boston, MA 02115 USA

***Abstract*—This paper presents an efficient algebraic fault analysis on all four modes of SHA-3 under relaxed fault models. This is the first work to apply algebraic techniques on fault analysis of SHA-3. Results show that algebraic fault analysis on SHA-3 is very efficient and effective due to the clear algebraic properties of Keccak operations. Comparing with previous work on differential fault analysis of SHA-3, algebraic fault analysis can identify the injected faults with much higher rates, and recover an entire internal state of the penultimate round with much fewer fault injections.**

I. Introduction

Keccak is a family of sponge functions, and has been selected as the new Secure Hash Algorithm (SHA-3) standard. Therefore, Keccak based security modules, including hash function, symmetric cryptographic function, pseudo random number generator, and authenticated encryption, will be widely used in future crypto systems [1]–[3]. For example, Keccak-based variations, Ketje and Keyak, have been chosen as candidates in the third round for CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness). The vulnerability of SHA-3 to various attacks has to be thoroughly examined. In this paper, we focus on fault attacks, and propose an efficient algebraic fault analysis (AFA) method on all the four modes of SHA-3.

Differential fault analysis (DFA) is a powerful 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 symmetric ciphers, e.g., the Data Encryption Standard (DES) algorithm [4], and Advanced Encryption Standard (AES) [5]. Many other ciphers have also been hacked through DFA, including CLEFIA [6], Mickey [7] and Grain [8].

Some existing hash standards have also been evaluated against DFA attacks, including SHA-1 [9], Streebog [10], MD5 [11] and Grøstl [12], used in either message authentication code (MAC) mode, or general hash mode. DFA aims at recovering the original input message of general hash mode functions, and extracting the secret key of MAC mode functions. Besides DFA, algebraic technique has also been applied to improve the efficiency of fault attacks, and a new kind of attacks called algebraic fault analysis (AFA) has been proposed [13]–[17]. AFA translates the problem of fault analysis into a boolean satisfiability problem (abbreviated as SAT), and then relies on SAT solvers to find the solution for the formulated problem. With efficient SAT solvers, the complex fault propagation analysis in traditional DFA is avoided.

There exist only two papers on DFA of SHA-3 [18], [19]. They differ in fault models and target modes of Keccak algorithm. In [18], they inject single-bit faults at the penultimate round input of SHA3-384 and SHA3-512, two modes of SHA3 with longer digest outputs, and they can recover an entire internal state. In [19], the authors relax the fault model to byte-level faults, and they apply the attack onto the other two modes, SHA3-224 and SHA3-256. Their results show that much smaller number of faults are needed to recover the whole internal state than the work in [18]. These two works show that Keccak based systems, like SHA-3, are susceptible to DFAs.

In this paper, we extend fault attacks on SHA-3 by introducing algebraic techniques into the attacks. Our work shows that clear and simple algebraic properties of Keccak make algebraic methods very suitable for analyzing SHA-3. Based on the problem formulation, the SAT solver used in this paper needs only several seconds to find the injected fault and then recover some χ_i^{22} bits. Meanwhile, AFA makes use of both the correct and faulty digest output, H and H', instead of just their differential ΔH as used in DFA. With more information, AFA is more effective than DFA in terms of fault identification and internal state bits recovery. Comparing with the previous attacks in [18] and [19], AFA method proposed in this paper has the following advantages:

- It does not involve the complex fault propagation analysis required for DFA, and can be highly automatic.
- It achieves higher effectiveness (with more bits recovered by each fault injection) and efficiency (fewer fault injections).

The rest of this paper is organized as follows. In Section II, we introduce the algorithm of SHA-3, and then present the fault models used in this paper. In Section III, we present the algebraic analysis method to recover χ_i^{22} bits. In Section IV, the AFA attack results are given in detail. In Section V, we extend the proposed attacks to SHA-3 systems with longer input message, and discuss countermeasures against the proposed attacks. Finally, we conclude this paper in Section VI.

II. Preliminaries of SHA-3 and Fault Model

A. Preliminaries of SHA-3

Standardized by NIST, SHA-3 functions operate in modes of Keccak-f[1600, d] [1], where each internal state is 1600-bit organized in a 3-D array ($5 \times 5 \times 64$, i.e., 5 bits in each row, 5 bits in each column and 64 bits in each lane), and d is the capacity and also the output length at choice. The SHA-3 family includes four output lengths, called SHA3-224, SHA3-256, SHA3-384, and SHA3-512 [1]. Keccak relies on a Sponge architecture to iteratively absorb message inputs and squeeze out digests by a f permutation function. Each f function works on a state at a fixed length $b = r + c$.

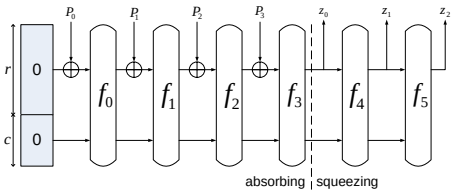

Fig. 1: The sponge construction

In this paper, we simplify the setting of SHA-3 by assuming that only one f function is involved for absorbing and squeezing. Then we extend the proposed attack to systems with longer input messages which involves multiple f functions. The attack goal is to recover the authentication key when SHA-3 is used in MAC mode, or to recover the input message when SHA-3 is used in hash mode. The f function consists of 24 rounds for 1600-bit operations, and each round has five sequential steps:

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

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

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

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

$-\ \rho$ is a rotation over the state bits along z-axis (in lanes).

$-\ \pi$ is a permutation over the state bits within slices.

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

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

$-\ \iota$ is a binary XOR with a round constant.

All the above operations are reversible [2]. Thus if an internal state is recovered in SHA-3, the original message and all the other internal states can be recovered. We set our target as recovering the entire internal state of χ_i^{22} (1,600 bits). We annotate the last two rounds of SHA-3 operations and important intermediate states in Fig. 2, and use these notations in the rest of this paper.

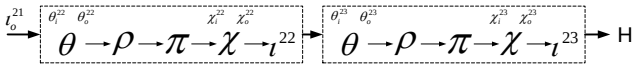

Fig. 2: Notations for operations and intermediate states

In this paper, the fault injection point is θ_i^{22}, and the attackers can only observe the clean and faulty digest, H and H', with length d for Keccak-f[1600, d].

B. Fault Models in This Paper

For DFA on block ciphers and stream ciphers, the goal is to recover the key, and therefore operations on multiple inputs under the same fault injection are performed. In fault analysis on SHA-3 hash mode, the goal is to recover the input message, and multiple faults are injected into the system with the same input message. In this paper, we generate multiple random messages and attack each message separately, and present their average results for performance and effectiveness evaluation.

Two different fault models have been used in previous work, single-bit [18] vs. single-byte [19]. In this paper, we adopt the same byte-level fault model as [19], and also extend our AFA to different platforms with more relaxed fault models, for example, single 16-bit word fault model for 16-bit architectures. We first use the random single-byte fault model to demonstrate the AFA on SHA-3 in this paper:

- The attacker can inject faults into one byte of the penultimate round input θ_i^{22} for 8-bit architectures;
- The attacker has no control on either the position (which byte) or the value of the injected faults;
- The attacker can observe only the correct and faulty SHA-3 digest, H and H', which are d bits for SHA3-d function;
- The attacker can inject random faults for the same input message multiple times.

All the single-byte faults in this paper are randomly generated, with any value (1-255) at any position (200 bytes). For commonly used SHA-3 implementation, data structures are organized along each lane [1], [2]. Thus one byte is eight consecutive bits in one lane in this paper.

In this paper, we use C++ API of CryptoMiniSat for the SAT problem formulation and solving [20]. All the simulations are run on a Ubuntu 14.04.3 system, with an Intel i7-2600 CPU and 8 GB memory.

III. Algebraic Fault Analysis on SHA-3

A. Overview of Algebraic Fault Analysis on SHA-3

The key idea of AFA is to transform the internal state recovery problem into a satisfiability problem, representing the target algorithms and operations using Boolean equations, and then use a SAT solver to find solutions for the variables which contain secret information.

First, we build a set of equations for the hash function for both correct and faulty executions. With the input of the penultimate round input θ_i^{22}, the correct hash digest is:

$$H = \iota^{23} \circ \chi \circ \pi \circ \rho \circ \theta \circ \iota^{22} \circ \chi \circ \pi \circ \rho \circ \theta(\theta_i^{22}). \quad (3)$$

Denote the injected fault as $\Delta\theta_i^{22}$, then the faulty digest is:

$$H' = \iota^{23} \circ \chi \circ \pi \circ \rho \circ \theta \circ \iota^{22} \circ \chi \circ \pi \circ \rho \circ \theta(\theta_i^{22} \oplus \Delta\theta_i^{22}). \quad (4)$$

Both H in (3) and H' in (4) have d bits for SHA3-d function. For DFA in [18] and [19], the attacker uses the differential of H and H' to retrieve $\Delta\chi_i^{23}$, the differential of χ_i^{23}, for attack. In this paper, with AFA we are able to make use of both H and H' directly.

We first build the equation set for (3) and (4) in Section III-B1, and then construct constraints for the injected fault $\Delta\theta_i^{22}$ in Section III-B2. We show the recovery of χ_i^{22} bits using a SAT solver in Section III-C, and then improve the recovery process by identifying the injected fault first in Section III-D.

B. SAT Problem Formulation

In this section, we represent the SHA-3 algorithm in a set of equations and build constraints for the injected fault.

1) Construction of the Equation Set for Keccak: Thanks to the simple algebraic operations of Keccak, the construction of equations of SHA-3 is straightforward. We use $1,600$ single-bit variables to denote the input θ_i^{22}, and another $1,600$ variables to denote the differential input (fault injected) $\Delta\theta_i^{22}$. Then we build equations for the operations in the last two rounds of SHA-3 for both the correct and faulty hashing processes based on (3) and (4). Finally additional d equations can be used to represent the observed value of digest H and H', respectively.

For the θ step, the only operation is XOR of state bits, which can be easily expressed in CryptoMiniSat. We introduce 320 variables to denote the θ compression results, and then use another $1,600$ bits to denote the θ operation outputs. Both ρ and π are bit permutation operations, which can be simply denoted in SAT. The step χ involves XOR, NOT and AND operations, and can be simplified as:

$$\chi_o(x,y,z) = \chi_i(x,y,z) \oplus \chi_i(x+2,y,z) \\ \oplus [\chi_i(x+1,y,z) \cdot \chi_i(x+2,y,z)].$$

We introduce another $1,600$ variables $\chi_{and}(x,y,z)$ to denote $\chi_i(x+1,y,z) \cdot \chi_i(x+2,y,z)$. Then χ operation can be represented as XOR of three variables. ι can be denoted using XOR operation as it is a constant number addition operation.

For the last two rounds of SHA-3 for both correct and faulty hashing, we use $39,360$ variables and $52,160$ equations in total. Note here some optimizations can be applied to improve the efficiency of algebraic analysis by reducing the number of variables and equations. For example, ρ and π can be combined to form one new step, and ι can be combined into χ to save variables and equations [21]. Even without any optimization, the AFA method proposed in this paper needs only seconds to recover the χ_i^{22} bits, which will be described in detail in the following sections.

2) Constraints for the Injected Fault: The fault model we adopt will be represented as constraints in the SAT problem. Under the single-byte fault model, we use $D = d_1, d_2, \cdots, d_{200}$ to denote $\Delta\theta_i^{22}$, which is 200 bytes, and d_i ($1 \le i \le 200$) is one byte of $\Delta\theta_i^{22}$. We use d_i^j to represent the j^{th} bit of d_i, in which $j \in \{1,2,\cdots 8\}$. A one-bit variable c_i is introduced to represent whether the i^{th} byte of θ_i^{22} is corrupted:

$$c_i = d_i^1 \vee d_i^2 \vee \cdots \vee d_i^8. \tag{5}$$

We have the following observations:

1) One and only one byte out of 200 bytes of D has non-zero value, because only one byte of θ_i^{22} is corrupted.
2) If d_i is corrupted, $c_i = 1$, otherwise $c_i = 0$.

Then we have the following constraints which will make sure that one and only one byte of θ_i^{22} is corrupted:

$$\begin{cases} c_1 \vee c_2 \cdots \vee c_{200} = 1 \\ \bar{c_i} \vee \bar{c_j} = 1, 1 \le i < j \le 200 \end{cases}. \tag{6}$$

C. Recover χ_i^{22} Bits by the SAT Solver Directly

For the SAT problem formulated in Section III-B, an SAT solver can be used to find solutions for all the variables. As the internal states are interdependent and the f function is reversible, hacking the hash algorithm only needs to recover one internal state. With one fault injected (which results in H'), only the bits uniquely recovered are useful and defined as solution in our scheme. The effectiveness (number of bits recovered by one fault injection) may vary for different states. We target recovering χ_i^{22} in AFA, because its effectiveness may be the highest, which is determined by the hashing algorithm itself. Both previous DFA works on SHA-3 also target to recover χ_i^{22} bits [18], [19]. Since a single fault can only recover some bits, our AFA method has to inject multiple faults so as to recover the entire χ_i^{22} state.

We need an efficient method to find the χ_i^{22} bits that can be recovered uniquely under a certain fault injection. For the original SAT problem (the equation set and the constraints), we run the SAT solver to find the first solution (1600-bit χ_i^{22} state) and then prune it to reduce to unique bits, which will be much fewer than 1600. Note the SAT solver also outputs the corresponding solution for other variables, like the fault and other states. We do not use them in further search as they are not our attack target. We then convert the first solution into blocking constraints and run SAT solver again (i.e., the new solution should not be equal to the first solution). For the second solution found, we compare it with the first solution. On some bit positions they have different values, and on others they have the same values, which will possibly be the unique bits recovered by the specific fault. We then convert these common bits into constraints for the next round search (to find a solution that differs in one or multiple of these common bits). Iteratively, each search will reduce the number of potential unique bits a little until the SAT solver cannot output any new solution, and we will end up with a number of χ_i^{22} bits uniquely recovered from this fault injection. We can view that with each round of SAT solver, more specific constraints are added into the SAT problem to help the solver converge onto a unique solution. The more specific the constraints, the faster the convergence speed of the iterative SAT solver runs.

For example, we generate a random message for SHA3-512 and inject a fault 0x5A (01011010_2, four bits flipped) at the first byte of θ_i^{22}. Using the above method we can recover 80 bits of χ_i^{22} after 41 iterations of running the SAT solver. However, only 21 bits of χ_o^{22} and zero bits of θ_i^{22} can be recovered using the above method. With more bits recovered for each fault injection, there will be less number of faults required to recover the whole state of χ_i^{22}, i.e., achieving higher efficiency for the AFA.

D. Improving the Recovery of χ_i^{22} bits

In the previous section, the SAT solver starts from searching for a 1600-bit solution for χ_i^{22}, without identifying the injected fault. It has been shown in [19] that with the knowledge of $\Delta\chi_i^{22}$ (derived from the fault $\Delta\theta_i^{22}$), the attacker can know which χ_i^{22} bits may be leaked. In this section, we propose another method that can improve the recovery of χ_i^{22} bits, by finding the injected fault first and then using the fault information to search for the unique bits more efficiently.

1) Identify the Injected Fault: As mentioned in [19], more than one fault may satisfy the differential output constraints in DFA, because limited number of $\Delta\chi_i^{23}$ bits are derived from the observable digest, H and H'. In our AFA scheme, we are using more information, H and H' themselves rather than the differential, to find the fault. Therefore we may have better chance identifying the fault. However, the non-uniqueness issue still exists for AFA due to the limited digest length (224 to 512, rather than 1600), i.e., some faults cannot be uniquely identified. We define those faults uniquely identified as effective faults, and only effective faults can be used to recover the internal state bits in this section.

Similar to the algorithm given in the previous section, we run the SAT solver twice to identify unique faults. The first solution (8-bit fault value and position) from the SAT solver is converted into additional constraints for the second search. If the second search finds another solution (fault), they both are not effective, and this fault injection is of no use with the uncertainty. Otherwise, the solution from the first-round search is effective and indeed the injected fault, and can be used to recover the internal state bits. With this search algorithm, for single-byte faults, our results show that it only needs about 1.7 seconds to 3.0 seconds to identify the injected fault for four SHA-3 functions. For larger digest size d, the proposed algorithm needs less time for fault identification. This is because for larger d, more information of H and H' is available and such information will be translated to constraints that help to narrow the search space and thus decrease the searching time. More results of fault identification will be given in Section IV.

2) Recover χ_i^{22} Bits with the Effective Fault Information: With the effective fault identified, hopefully we can recover some state bits of χ_i^{22} more efficiently by knowing what are these bit positions.

After the injected fault $\Delta\theta_i^{22}$ identified, attackers can find how the bits of χ_i^{22} are affected by the fault, i.e., derive the differential $\Delta\chi_i^{22}$: $\Delta\chi_i^{22} = \pi \circ \rho \circ \theta(\Delta\theta_i^{22})$. It has been shown that the bit-wise nonlinear AND operation of step χ may leak information of the input bits of χ at the output differential [18], [19]. For each fault, only part of χ_i^{22} bits can be recovered uniquely. We use three bits $\Delta\chi_i^{22}([x : x+2]$ in a row as an example to demonstrate which χ_i^{22} bit(s) can be leaked and present them in Table I [19].

TABLE I: Information leakage for different $\Delta\chi_i^{22}([x : x+2], y, z)$

$\Delta\chi_i^{22}([x : x+2], y, z)$	Information leakage
[1,0,0]	$\varnothing$
[0,1,0]	$\chi_i^{22}(x+2, y, z)$
[0,0,1]	$\chi_i^{22}(x+1, y, z)$
[1,1,0]	$\chi_i^{22}(x+2, y, z)$
[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]	$\chi_i^{22}(x+1, y, z) \oplus \chi_i^{22}(x+2, y, z)$

By checking the entire 1600-bit $\Delta\chi_i^{22}$, we identify all the χ_i^{22} bits that may be recovered for the given identified fault. Meanwhile, from Table I, we can see that sometimes the attacker can only recover the XOR of two χ_i^{22} bits, instead of the value of χ_i^{22} bits directly. It has also been shown in [19] that the later θ^{23} operation may cancel out the effect of some χ_i^{22} bits before they propagate to the final digest. Thus we still need a pruning algorithm similar to the one presented in Section III-C to reduce the number of unique χ_i^{22} bits recovered iteratively.

We run the same fault injection experiment as Section III-C: inject a fault 0x5A on the first byte of θ_i^{22} for the same random message. The improved method finds that only 88 bits of of χ_i^{22} may possibly be recovered, and run the SAT solver to look for a solution for these 88 bits. The improved method needs only 3 iterations to find all the 80 χ_i^{22} bits which can be uniquely defined, using only 0.28 seconds. By comparison, the method in Section III-C needs more than 10 seconds to recover these 80 χ_i^{22} bits with 41 iterations.

For real attacks, we propose to use a hybrid of these two methods. We first try to identify the injected fault. If the fault is effective, we use the method in Section III-D2 to recover the χ_i^{22} bits; otherwise, we use the method in Section III-C.

IV. Algebraic Fault Analysis Results

In this section, we present the simulation results of the proposed AFA attacks on SHA-3. For fault injection attacks on crypto systems, we evaluate them in several aspects:

- The impact of the fault model used for attacks. In this work, we test the proposed method under both a single-byte fault model and a single-word (16-bit) fault model.
- The effectiveness of identifying the injected fault. We will present the ratio of effective faults in detail in Section IV-A.
- The efficiency of the attack. We will report the total number of effective faults to recover the entire state and the total time, which will be presented in Section IV-B.

We note here that as DFA can use only effective faults for attacks, we use only effective faults in AFA to better compare the efficiency of AFA and DFA.

A. Fault Identification Results

Using the method proposed in Section III-D1, an attacker can identify the injected fault for some fault injection. Considering the total number of possible faults is 51,000 (200 positions with each possible 255 fault values), we can find the ratio of effective faults. Our simulation results show that AFA has much higher effective fault ratio than DFA, presented in Table II. It shows that under a single-byte fault model, the effective fault ratio of AFA is almost 100% for all four SHA-3 modes, while the ratio of DFA is very low for the two shorter digest modes presented in [19].

TABLE II: Results of effective fault ratio under two fault models

	8-bit		16-bit	
	DFA	AFA	DFA	AFA
SHA3-224	30.67%	96.85%	0.00%	0.00%
SHA3-256	66.61%	99.89%	0.00%	0.30%
SHA3-384	99.13%	100%	40.05%	99.99%
SHA3-512		100%		100%

We extend the proposed AFA to 16-bit architecture, by injecting single 16-bit word fault into θ_i^{22}. The simulation

results for such fault model are shown in the right columns of Table II. It shows that AFA still achieves close to 100% effective fault ratio for SHA3-384 SHA3-512, while the ratio of DFA is much lower. For SHA3-224 and SHA3-256, the observable digest is limited, and the footprints of different faults will no longer vary significantly, thus neither AFA nor DFA can identify the injected fault.

B. Internal State χ_i^{22} Recovery Results

To find the effectiveness of AFA, i.e., the average number of state bits recovered by each fault injected, we conduct an experiment by generating 10^5 messages and injecting random single-byte faults at θ_i^{22} for each message. We count the recovered χ_i^{22} bits for each fault value. For the proposed methods in Section III-C and Section III-D, the number of χ_i^{22} bits that can be recovered for different number of flipped θ_i^{22} is shown in Fig. 3(a), and the distribution of number of χ_i^{22} bits that can be recovered is shown in Fig. 3(b). The more bit flipped at the input of the penultimate round, the more state bits of θ_i^{22} can be recovered. For random faults, on average 80 state bits can be recovered for each fault.

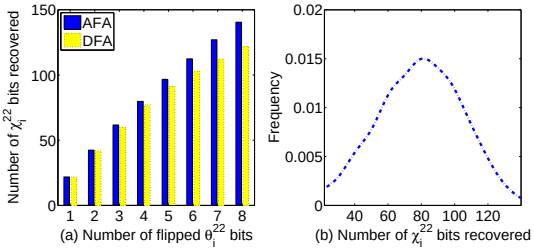

Fig. 3: Number of χ_i^{22} bits recovered for each fault

Attackers can inject multiple faults for the same input message and run SHA-3 algorithm several times to recover the whole internal state of χ_i^{22}. To evaluate the efficiency of the proposed AFA method, we randomly generate 1000 messages, and for each message, we randomly inject multiple faults to recover the whole internal state χ_i^{22}. We present the χ_i^{22} state recovery processes for SHA3-224 and SHA3-256, under single-byte fault model in Fig. 4 and Fig. 5, by both our AFA and DFA. It shows that with the same number of effective faults, AFA can recover more χ_i^{22} bits than DFA, and therefore it requires much less number of fault injections to recover the entire 1600 bit state. We note here that we do not present the DFA results on SHA3-384/512, because DFA results are very near to the AFA results, though less efficient. Overall, AFA is much more efficient than DFA.

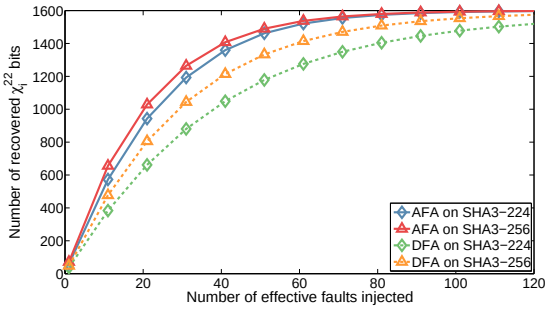

Fig. 4: χ_i^{22} recovery process for algebraic fault analysis

The difference in efficiency stems from the different usage of information. DFA makes use of the information $\Delta\chi_i^{23}$ while AFA uses both H and H', which contain much more information than $\Delta\chi_i^{23}$. What's more, for SHA3-384 and SHA3-512, only 320 bits of $\Delta\chi_i^{23}$ is available and used by DFA, while there are 384 bits and 512 bits of digest (H and H') available for AFA, respectively. For SHA3-224 and SHA3-256, the number of available $\Delta\chi_i^{23}$ bit is 112 and 160 for DFA respectively, while the number of available digest bits are 224 and 256 for AFA.

C. AFA Attacks Under A More Relaxed Fault Model

AS shown in Table II, the proposed AFA scheme still has very high effective fault ratio under the single-word fault model for SHA3-384 and SHA3-512. Here we present the results of our AFA on SHA3-384 and SHA3-512 under single-word fault model in Fig. 5. It shows that the AFA needs much smaller number of effective faults to recover the whole internal state under the random single-word fault model than under the single-byte fault model. This is because single-word fault flips more bits than single-byte fault on average, and thus more bits of χ_i^{22} can be recovered for each identified fault.

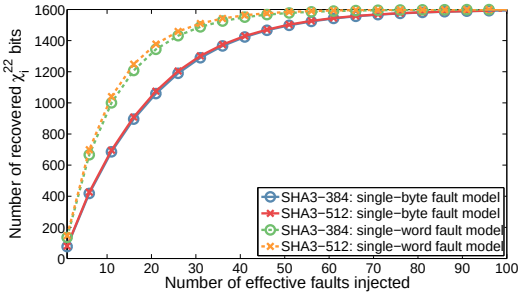

Fig. 5: AFA results under single-word (16-bit) fault model

As shown in Table II, the effective fault ratio for SHA3-224 and SHA3-256 are almost zero under 16-bit fault model. Limited information makes the method in Section III-C ineffective for SHA3-224 and SHA3-256 either. Attacks on SHA3-224 and SHA3-256 under relaxed fault models will be future work. Meanwhile, we also tried the proposed attacks under 32-bit fault model and the SAT solver needs long time to identify the fault and recover the internal state bits. Optimization to improve the efficiency will also be future work.

V. Discussion and Future Work

In previous sections, we simplify the target SHA-3 system by assuming that only one f function is involved for absorbing the message and squeezing, which means that the input message length is shorter than the bitrate r. In Section V-A, we extend fault attacks to SHA-3 systems with input message with variable length. Then we will discuss future work and countermeasures in Section V-B.

A. Fault Attacks on SHA-3 Systems with Long Input Message

In this section, we first extend the proposed attack to scenarios where the size of the input message is larger than bitrate r, and there are n ($n \geq 1$) f functions ($f_0 \cdots f_{n-1}$) involved for absorbing. We assume that the digest size is still d-bit for SHA3-d function (refer to Fig. 1). For all the four

modes of SHA-3, the bitrate r is larger than d, and therefore there is no need of f function for squeezing.

For MAC-Keccak, the attacker has control (knowledge) of the input messages $P_0, \cdots, P_{n-1}$ except for the key bits. The adversary can first attack the last f function, f_{n-1}, by injecting faults into its penultimate round input. Using the proposed method in this paper, the input $f_{n-1}(in)$ will be recovered. Combining with P_{n-1}, the attacker can recover $f_{n-2}(in)$. There are two possible situations for the key length l_k:

- 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 $f_1(in) = f(P_0||0^c) \oplus (P_1||0^c)$. Thus 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 and then try to recover the key bits in P_0, then the difficulty increases rapidly with the number of the key bits in P_1.

For SHA-3 system in general hash mode, the attacker has 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 digest size larger than d, there may be extra f functions involved for squeezing. Then the size of z_0 will be larger than d (up to r-bit), which means more information (larger size H and H') available to attackers. Thus the attacker can recover more bits of $\Delta\theta_i^{23}$ directly, which will make the attack much easier. Conclusively, the proposed fault analysis method is still applicable for SHA-3 systems which involve extra f functions for squeezing.

B. Future Work and Countermeasures

Both this work and [19] show that AFA/DFA cannot be applied for the attacks of SHA3-224 and SHA3-256 under more relaxed single-word fault model, and effective attacks on SHA3-224/SHA3-256 under more relaxed fault models should be devised in future work.

Protection can be added into Keccak based systems to protect them against fault injection attacks. For example, error detection codes can be added to detect injected faults, and this has been discussed in [21]. It has been shown that utilizing the algebraic properties makes the protection of Keccak efficient. Meanwhile, extra modules can be added into crypto systems to detect the disturbance used to inject faults [22]. More effective countermeasures against fault attacks will be future work.

VI. Conclusion

In this paper, we present efficient algebraic fault analysis attacks on SHA-3. We show that AFA is very suitable for the attacks of SHA-3 because of the clear algebraic properties of Keccak operations. The analysis and simulation results show that our method has much higher effective fault ratio than previous attacks, and it requires much fewer faults to recover the whole internal state of χ_i^{22}.

Acknowledgment: This paper is originally published in DATE'17, and supported in part by National Science Foundation under grants SaTC-1314655 and MRI-1337854. Simulation code in this paper is available at http://tescase.coe.neu.edu.

References

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

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

[3] W. Cai and F. Shi, "2.4 GHz heterodyne receiver for healthcare application," *International Journal of Pharmacy and Pharmaceutical Sciences*, vol. 8, no. 6, pp. 162–165, 2016.

[4] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology – CRYPTO'97*.

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

[6] H. Chen, W. Wu, and D. Feng, "Differential fault analysis on CLEFIA," in *Information and Communications Security*, 2007.

[7] S. Karmakar and D. R. Chowdhury, "Differential fault analysis of MICKEY-128 2.0," in *WkShp on Fault Diagnosis & Tolerance in Cryptography*, 2013.

[8] P. Dey, A. Chakraborty, A. Adhikari, and D. Mukhopadhyay, "Improved practical differential fault analysis of Grain-128," in *Proc. Design, Automation & Test in Europe*, 2015.

[9] L. Hemme and L. Hoffmann, "Differential fault analysis on the SHA1 compression function," in *WkShp on Fault Diagnosis & Tolerance in Cryptography*, Sept. 2011.

[10] R. Altawy and A. M. Youssef, "Differential fault analysis of Streebog," in *Int. Conf. on Information Security Practice & Experience*, 2015.

[11] 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.

[12] W. Fischer and C. A. Reuter, "Differential fault analysis on Grøstl," in *FDTC'2012*.

[13] N. T. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations," in *Advances in Cryptology – ASIACRYPT 2002*.

[14] P. Jovanovic, M. Kreuzer, and I. Polian, "An algebraic fault attack on the LED block cipher." *IACR Cryptology ePrint Archive*, 2012.

[15] F. Zhang, X. Zhao, S. Guo, T. Wang, and Z. Shi, "Improved algebraic fault analysis: A case study on Piccolo and applications to other lightweight block ciphers," in *Constructive Side – Channel Analysis and Secure Design*, 2013.

[16] X. Zhao, F. Zhang, S. Guo, T. Wang, Z. Shi, H. Liu, and K. Ji, "MDASCA: an enhanced algebraic side-channel attack for error tolerance and new leakage model exploitation," in *Constructive Side-Channel Analysis and Secure Design*, 2012.

[17] X. Zhao, S. Guo, F. Zhang, Z. Shi, C. Ma, and T. Wang, "Improving and evaluating differential fault analysis on LED with algebraic techniques," in *FDTC'2013*.

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

[19] P. Luo, Y. Fei, L. Zhang, and A. Ding, "Differential fault analysis of SHA3-224 and SHA3-256," in *FDTC 2016*.

[20] M. Soos, K. Nohl, and C. Castelluccia, "Extending SAT solvers to cryptographic problems," in *Theory and Applications of Satisfiability Testing-SAT 2009*.

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

[22] 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, 2011.