The ISC Int'l Journal of Information Security

ISeCure

July 2019, Volume 11, Number 2 (pp. 129–143)

http://www.isecure-journal.org

Relaxed Differential Fault Analysis of SHA-3

S.Ehsan Hosiny Nezhad [1], Masoumeh Safkhani [2], and Nasour Bagheri [1,3,*]

[1] *Department of Electrical Engineering, Shahid Rajaee Teacher Training University, Tehran, Iran*
[2] *Department of Computer Engineering, Shahid Rajaee Teacher Training University, Tehran, Iran*
[3] *School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran*

A R T I C L E I N F O.

Article history:
Received: 4 May 2019
Revised: 8 July 2019
Accepted: 22 July 2019
Published Online: 31 July 2019

Keywords:
Hash Function, SHA-3, Differential Fault Analysis, State Recovery, Differential Equations, Algebraic Resolution, Multiple Byte Fault.

Abstract

In this paper, we propose a new method of differential fault analysis of SHA-3 which is based on the differential relations of the algorithm. Employing those differential relations in the fault analysis of SHA-3 gives new features to the proposed attacks, e.g., the high probability of fault detection and the possibility of re-checking initial faults and the possibility to recover internal state with 22-53 faults. We also present two improvements for the above attack which are using differential relations in reverse direction to improve that attack results and using the algebraic relations of the algorithm to provide a second way to recover the internal state of SHA-3. Consequently, we show that with 5-8 faults on average, SHA-3's internal state can be fully recovered.

1 Introduction

SHA-3 is a secure hash algorithms standard by NIST at 2015 [1, 2]. SHA-3's applications can be Massage Authentication Codes (MACs), with being a stream cipher or a quasi-random generator second in line. As the new standard, in near future SHA-3 will be used more and more in cryptographic and security systems and protocols, making its security a critical factor in these systems. This fact has motivated extensive studies of this primitive in recent years, e.g. [3–12]. Cryptographic systems are sensitive to random/intentional errors caused by environmental factors such as temperature, X-rays or adversarial attacks such as fault attacks. For a cryptographic system, random faults generate false results which make them unreliable. An attacker can also apply temporary faults to a cryptography system and analyze the results to get the key or system's sensitive data. These attacks are known as fault analysis attacks, among them is differential fault analysis (DFA), but not limited to it and other type/concepts of attack was introduced such as blind fault attack, fault sensitivity analysis, statistical fault attack and differential fault intensity analysis attacks. However, in this paper, we concentrate on DFA attack. DFA was first induced in [13] to recover secret key of DES or an unknown function and then this technique has been applied to Triple-DES [14], AES [15], SKINNY [16], SHA-1 [17], Grϕstl [18] and many more algorithms.

So far, several successful instances for fault analysis of SHA-3 have been proposed in related literature [19–22]. In 2015, the first DFA on SHA-3 was proposed by Bagheri et al. [19]. They proposed DFA for two members of the SHA-3 family, i.e., SHA-3-384 and SHA-3-512, with injection fault on one-bit of the state at a time, as the DFA model. In 2016, Luo et al. [20] targeted all four hash members of the SHA-3, with 1-byte faults. They later extended their work [21], where a SAT solver is used to improve the fault detection process. However, the basic idea of fault detection remaining the same: setting 0, 1 and x (unknown) as possible stands for differences. More recently, a new instance of SHA-3's fault attacks [22] was published

* Corresponding author.

Email addresses: `ehsan.hosinynezhad@yahoo.com`, `Safkhani@srttu.edu`, `NBagheri@srttu.edu`

ISSN: 2008-2045 © 2019 ISC. All rights reserved.

which should be considered as an extension over [21]'s authors. Same as the previous one, an SAT solver is used but with recovering SHA-3's state bits as the primary goal. With this change, usage of 1-byte fault for all four SHA-3 members, 2-byte fault for SHA-3-256 and 4-byte fault for SHA-3-512 become possible. In addition, with injecting fault to rounds 23 and 24, the 2-byte fault can also be used for SHA-3-224.

To improve the fault analysis, we use the differential relations of the ciphers components as the fault signature. Using these differential relations in our DFA gives the features to our analysis:

- The ability to use 1 to 6-byte faults in the attack,
- The process can be applied to all four members of SHA-3,
- Our work only needs fault injection to a single round
- Since our attack is self-improving, rechecking the faulty outputs that were not fully used is possible,
- SHA-3's internal state full-recovery, given 5 to 8 faulty hash/Tag, is possible.

The rest of the paper is structured as follows: Section 3 gives a brief description of how SHA-3's components work. The concept of differential relations of SHA-3 is introduced in Section 4. In Section 5, the method of recovering SHA-3's internal state bits with fault analysis is explained. Our proposed methods to improve the attack along with simulation results are presented in Section 6. Finally, Section 7 concludes the paper.

2 Notation

Throughout the paper, we use the notations represented in Table 1.

Table 1. Notations used in this paper

Notation	
$\theta, \rho, \pi, \chi, \iota$	SHA-3's inner functions
A_n	Output of inner function "A" at round n
X_n	A bit of χ_{23}'s input (used in equations)
$A(x, y, z)$	A bit of inner state "A"
Δi_k / Δo_k	Difference of bit "k" of Input / Output
V_i / V_o	A row of χ_{24}'s Input / Output

3 SHA-3 Hash Function

A cryptographic hash/MAC function takes an arbitrary string as the input and produces a constant length output. The hash/MAC function's input is called "message", and its output is called "hash/TAG" or message digest. Hashes are often used as a unique digest for the message. SHA-3 is the latest standard of secure hash algorithms which is based on Keccak [2], winner of the SHA-3 competition [23], which also supports MAC applications. In the following, a brief description, of how it works, is presented.

SHA-3 family is composed of four hash functions and two extendable-length output functions (EXOs). The four SHA-3 hash functions are SHA-3-224, SHA-3-256, SHA-3-384, and SHA-3-512, where x in SHA-3-x denotes the output length. All members of SHA-3 have a common data absorbing structure called "SPONGE" structure [1], which is illustrated in Figure 1.

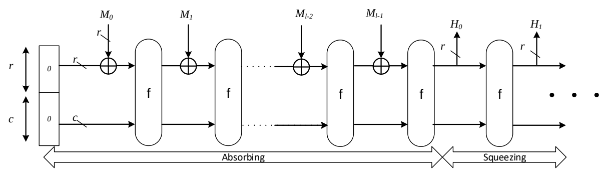

Figure 1. SPONGE structure [1]

SPONGE structure uses a permutation f as the main building block. This permutation, f, has a constant length input and output, which are called internal state or simply "state". The first r bits of it are called the rate bits, and the other $c = b - r$ bits are called capacity bits [1]. To generate a hash/Tag for a message, that message needs to be divided into r-bit blocks with $1\{0\}^*1$ padding. Then each block will be combined with rate bits of the state followed by the application of compression function. After absorbing message blocks and after that the squeezing calculation finishes, first d-bits of state are presented as the output. In the following equations, "A" represents the state.

All members of the SHA-3 family use the same internal functions (sub-permutations): $\theta, \rho, \pi, \chi, \iota$. Application of all five inner functions, as follows, is known as a round:

$$Round_i = \iota_i \circ \chi \circ \pi \circ \rho \circ \theta(A) \tag{1}$$

The calculations of the internal functions are described in the following. In SHA-3, the permutation function (Keccak-f) is composed of 24 rounds of inner functions on a 1600-bit state:

$$Keccak\text{-}f : for\ i = 0 \quad to \quad 23\ Round_i(A)\ end \tag{2}$$

In the internal functions of SHA-3, the state is represented by a three-dimensional matrix with 1600 cells as bits. In Figure 2, the internal state, and its sub-strings are specified. For example, a sub-string of internal states whose bits have the same component y and z is called the row (y, z).

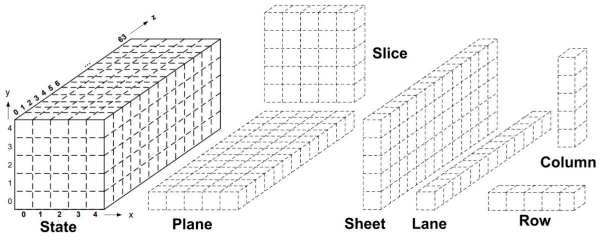

Figure 2. SHA-3's state and it's substrings [20]

The first internal function, θ, combines state bits in y-direction as:

$$\theta : A'(x, y, z) = A(x, y, z) \oplus \sum_{i=0}^{5} A(x-1, i, z) \oplus \sum_{i=0}^{5} A(x+1, i, z-1) \quad (3)$$

The next round-function is ρ, which rotates state bits in z-direction. The number of rotations are pre-defined offsets which can be seen in Table 2.

Table 2. Offsets of ρ

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	22	39	3	10	43
$y=1$	55	20	36	46	6
$y=0$	28	72	0	1	62
$y=4$	56	14	20	2	61
$y=3$	21	8	41	45	15

The third round-function, π, rotates state bits in x and y direction. (x', y') in the following equation are the updated value of these components after π's application:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} \quad (4)$$

The round-function χ is the only nonlinear function on each round which updates a bit with the help of the next two bits along the x-direction. Figure 3 illustrates the application of χ on a row of a state.

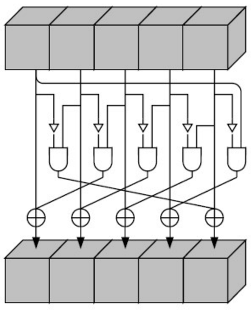

Figure 3. A row's update in χ [1]

The last of the internal functions, ι, updates state by combining a 64-bit array, dependent on the round number, with the first 64-bit of the state. These arrays originally are produced by an LFSR. However, their values do not effect on our analyses here.

4 Differential relations in SHA-3

The differential relations in this paper are a set of binary functions which can determine the value of difference in any bit of state, given the value of state(s) before it. To calculate the differential relations of SHA-3, first, we need to know the differential relations of the operators employed in it. In SHA-3, four basic operators are used: AND, XOR, NOT, and Shift. In the following, differential relations of these operators are presented:

AND: Given k and k' are produced by multiplying respectively (a, b) and (a', b'), we would have:

$$\{k, k'\} = \{a \cdot b, a' \cdot b'\} \quad (5)$$

Now, to calculate the output difference, we do following operation:

$$\Delta k = k \oplus k' = (a \cdot b) \oplus (a' \cdot b') = \Delta a \times \Delta b \quad (6)$$

$$\Delta k = (a \cdot b) \oplus ((a \oplus \Delta a) \cdot (b \oplus \Delta b)) \quad (7)$$

$$\Delta k = a \cdot b \oplus ((a \cdot b \oplus a \cdot \Delta b) \oplus (\Delta a \cdot b \oplus \Delta a \cdot \Delta b)) \quad (8)$$

$$\Delta k = a \cdot \Delta b \oplus \Delta a \cdot b \oplus \Delta a \cdot \Delta b =: \Delta a \times \Delta b \quad (9)$$

Equation (9) shows that the output difference of multiplication of two bits is dependent on the initial and differential value of its input bits. Also, if we know that one of input difference is '0', the output difference will only depend on that input bit's initial value. The same argument is valid for known initial values. Hence, to determine the output difference, the adversary does not necessarily need the full knowledge of input.

XOR: Suppose that k and k' are produced by applying XOR to (a, b) and (a', b') respectively i.e.:

$$k = a \oplus b\ ,\ k' = a' \oplus b' \tag{10}$$

So, the output difference will be as follows:

$$\Delta k = k \oplus k' = (a \oplus b) \oplus (a' \oplus b') = \Delta a \oplus \Delta b \tag{11}$$

Therefore, the difference in output only depends on differences of inputs and can be calculated without knowing the initial value of input bits.

NOT: An inverter logical effect is same as XOR with '1'. If we use equation (11) with $\Delta b = 0$, it is proven that the input difference is not changed by NOT operation.

Shift: The Shift operators in SHA-3 are circular rotations, and they behave the same for differences.

With the help of the differential equations expressed above, we can obtain differential relations of internal functions of SHA-3.

In function θ, each bit is combined with XORs of its two adjacent columns. This means that differential outputs can be calculated with the same method as normal θ, just with differential bits as input. ρ and π only rotate state bits, therefore they will also be the same to the differential input.

Calculation of the differential relations in χ can be divided into two steps: calculating differential outputs of ANDs and then calculating the total differential output. Figure 4 illustrates these two steps for a row of state. In this Figure and the following equations, $X1-X5$ represent the initial values of the row's input.

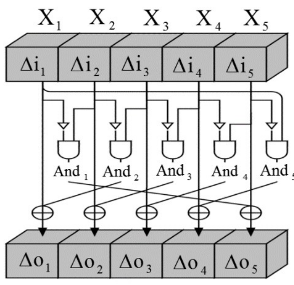

Figure 4. Calculation of differential relations in χ [1]

The first step will be as equation (9) :

$$AND_n = \Delta i_n \times \Delta i_{n+1} = X_n \cdot \Delta i_{n+1} \oplus \Delta i_n \cdot X_{n+1} \oplus \Delta i_n \cdot \Delta i_{n+1} \tag{12}$$

and the output difference will be as equation (11) :

$$\Delta o_n = \Delta i_n \oplus AND_{n+1} \tag{13}$$

We used equation (12) and (13) to calculate output differential relation of some examples of input differences, in Table 3.

With a closer look, we can see that in some of the rows, only a portion of initial values, $(X1, ..., X5)$, are presented as a variable in output relations. In another word, χ's output differences are dependent on input difference and a portion of initial bits.

The last function ι, only inverts some bits on lane 0. Therefore, it does not change its differential input.

The used approach: In this section, we described a method to find the differential relations and our approach to use them in a fault analysis attack. If we consider the injected fault's effects on the algorithm as a differential input, differential relations can express how they are going to be distributed between the injection and compression point. For example, if there is no χ function between them, output's difference can be calculated by only knowing the input difference.

In this paper, differential relations of 1.5 round (1round plus θ, ρ and π of the next round) are used as the resources of attack, for two reasons: first, they only have one χ function between them and given that χ is the only non-linear inner function in SHA-3, we would have a lower algebraic degree in our relations. Second, each plain carry all of the useful differential information in its differential relations. This is due to the second θ's application. This feature is useful since in a hash/MAC function only a small number of bits are available as output and having many bits which carry secret information make it more likely to discover the secret parameter. Appendix of the paper represents the distribution of a 16 bit fault with fault injection starting at 16th bit of the first lane, in 1.5 rounds.

5 Differential Fault Analysis of SHA-3

In this section, the process of recovering SHA-3's internal state using the fault-free and faulty generated hashes will be presented. We use a relax fault model with the following assumptions:

- The attacker can inject faults to the input of 23^{rd} round of the last compression function.
- The faults can occur in 1 to 6 continuous bytes.
- During the attack, the message remains fixed.

Figure 5 illustrates a simplified model for fault's

Table 3. Some examples of Differential Relations in χ

Input Differences					Output Differential relations				
i_1	i_2	i_3	i_4	i_5	O_1	O_2	O_3	O_4	O_5
ΔS_1	0	0	0	0	ΔS_1	0	0	$\bar{X}_5 \cdot \Delta S_1$	$X_2 \cdot \Delta S_1$
ΔS_1	ΔS_2	0	0	0	$X_3 \cdot \Delta S_2 \oplus \Delta S_1$	ΔS_2	0	$\bar{X}_5 \cdot \Delta S_1$	$X_2 \cdot \Delta S_1 \oplus \bar{X}_1 \cdot \Delta S_2 \oplus \Delta S_1 \cdot \Delta S_2$
ΔS_1	0	ΔS_3	0	0	$\bar{X}_2 \cdot \Delta S_3 \oplus \Delta S_1$	$X_4 \cdot \Delta S_3$	ΔS_3	$\bar{X}_5 \cdot \Delta S_1$	$X_2 \cdot \Delta S_1$
ΔS_1	ΔS_2	ΔS_3	0	0	$X_3 \cdot \Delta S_2 \oplus \bar{X}_2 \cdot \Delta S_3 \oplus \Delta S_2 \cdot \Delta S_3 \oplus \Delta S_1$	$X_4 \cdot \Delta S_3 \oplus \Delta S_2$	ΔS_3	$\bar{X}_5 \cdot \Delta S_1$	$X_2 \cdot \Delta S_1 \oplus \bar{X}_1 \cdot \Delta S_2 \oplus \Delta S_1 \cdot \Delta S_2$

distribution from the injection point to the output-hash/Tag. To use a faulty hash/Tag value, the following three processes must be performed:

(1) Half-Round Reversing
(2) Fault Detection
(3) State Bits Recovering

The rest of this section is aimed to explain how these processes work and show their results. Also, an overall view of these processes, that are required to analyze a faulty hash/Tag, is presented in Figure 6.

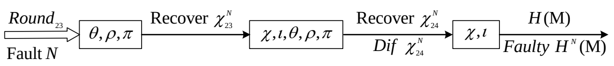

Figure 5. The used model to describe injected fault's effects

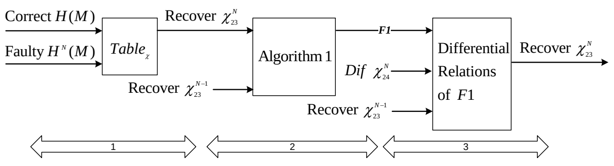

Figure 6. Differential fault analysis of SHA-3

5.1 Half-round reverse

In Section 4, a method of calculating differential relations of internal functions of SHA-3 was expressed. According to the overview of attack, faults should be injected into the input of 23rd round, meaning that the faulty state goes through 23rd and 24th round. Therefore, their differential relations will have three sets of variables: injected fault, χ_{23} and χ_{24}. If we calculate these differential relations and try to use them to determine the injected fault's value, the results will not be very interesting due to the high complexity in relations.

One way to counter the complexity of relations is to eliminate differential relation's dependency on χ_{24}. Of course, χ_{24}'s bits are dependent on χ_{23} bits though inner functions, and each of its bits can be expressed on with equations only consisting of χ_{23}'s bits. At the first glance, it seems using the algebraic relation between χ_{23} and χ_{24} can be used to eliminate differential relation's dependency to χ_{24} and solve the complexity problem. However, if we do this, each of χ_{24}'s bits will be replaced by a relation contains 33 bits of χ_{23} which make the relations more complex than before.

Another way to reduce the complexity is to retrieve differences at χ_{24}'s input using fault-free and faulty hashes. Since these differences did not pass through χ_{24}, they do not depend on its input. We call this process **half-round reverse** which will be described in the following.

Generated hash/Tag in SHA-3 is a part of the state after the application of round 24. To find χ_{24} input bits, one must first have input bits of the last applied function ι_{24}. In ι, a constant value is combined with the first lane of the state, so one can obtain the input of this function by re-applying the same constant value to its output.

Next function in the way is χ_{24}. Based on the functionality of χ, each output row only depends on the same row of input. If all bits of an output row are known, that row's input can be simply retrieved by a one-row I-O table. However, if some of the row's output bits are unknown, it is not possible to retrieve its input in the same way. In this case, if we check χ's I-O table with known bits, instead of one answer, there could be several possible answers for the row's input. Since there is no way to extract the correct answer, only bits that have constant value through all possible answers can be given as definitive input. In the following, a toy example of this method is presented:

Example: Suppose we are reversing a row with 4 known bits, V_o, in χ function . In this case, we will have:

$$\chi^{-1}(V_o = 1001x) = \{(V_{i_1} = 10000), (V_{i_2} = 11011)\} \quad (14)$$

where χ^{-1} denotes the I/O lookup table with 32 lines, its i^{th} row contains t such that $\chi(x) = i$. The given output, V_o, is used as a filter for table lines. Comparing V_o with each line, if there is any mismatch between the line's bits and the known bits of V_o that line will be removed from possible row's input. The remaining lines will be presented as the initial results, for our example there are two rows ($2^{5-4} = 2$).

In the next step, each bit is checked to see whether it remains fixed over all input candidates:

$$\chi^{-1}(\{(V_{i_1} = 10000), (V_{i_2} = 11011)\}) = (V_i = 1x0xx) \quad (15)$$

So, for the given V_o, two input bits are retrievable.

If the expressed process applies to all rows of output hash, a portion of χ_{24}'s input can be recovered. If a bit is known in both correct and faulty hash/Tag, input difference on that bit can also be retrieved. Table 4 shows the average number of correct and differential bits recovered at the input of χ_{24} function on SHA-3.

Table 4. Results of half-round reverse on SHA-3

Function	Fault-free bits	Differential bits
SHA-3-224	114.15	67.5
SHA-3-256	159.5	102.2
SHA-3-384	320	320
SHA-3-512	382.8	347.4

In the following, the method of using these differences to identify injected fault will be described.

5.2 Fault Detection

When an attacker injects fault to an algorithm, by default its position and value are unknown and the process of finding them from the available information is called fault detection. Using the differential relations of Section 4 and the differences that are obtained from the previous section, we propose Algorithm 1 to find all valid position-values for fault.

In this algorithm, first, we choose a position for the fault. Then fault's bits are set to 0 or 1, one after another. In each step, differential relations are checked for a contradiction between calculable table relation's value and real differences. If a contradiction exists, it would be taken to effect at the next fault bit setting. If all faulty bits are set without any contradictions, that fault is considered as a valid fault.

Algorithm 1 Fault Detection Algorithm

```
Offline:
1:  for each IP(Injection Point) in State do
2:      Set $F_{size}$ bits as faulty
3:      Calculate and store resulting differential relations
4:  end for
Online:
1:  for each IP in State do
2:      Load (Equs , IP)
3:      Update (Equs, $Recover_{Xi23}$)
4:      N=0, $V_{remain}$ = true, Wrong = fasle
5:      while $V_{remain}$ do
6:          New_I (I, N, Wrong)
7:          if N $\geq F_{size}$ then
8:              Find_X (Equs, I, $Dif_o$ )
9:          else if N $\leq$ -1 then
10:             $V_{remain}$ = fasle
11:         end if
12:         Chek_Equ (Equs, I , $Dif_o$, Wrong)
13:     end while
14: end for
```

If Algorithm 1 finds only one position-value for the injected fault, the fault is considered uniquely detected. The results of unique fault detection on SHA-3 hash functions are presented in Figure 7. In this figure, X-vector is the known-state-percent which increases as the attack goes on. We can use this to improve the attack, by reducing attack model. For example, the attacker can inject 1-byte fault with 87% detection chance on SHA-3-224 for start and after recovering at least 560 bits, or 35% of state bits, inject 2-byte fault with 78% detection chance and after that adversary uses 32-bit fault to recover the rest of the state. A same argument can be presented for all other members of SHA-3 hash family.

5.3 Simulation results

In Section 5.2, the method to detect fault from fault-free and fault hash/Tag were presented. If the injected fault is known, it is possible to recover χ_{23}'s effective bits using the differential equations. The average number of recovered bits on each fault is directly dependent on the size of injected faults and it increases as fault size does. To use this feature and not fall to low fault detection chance while doing it, we can start by injecting 8-bit fault, and after reaching a threshold of known state bits, start injecting 16-bit faults or 32-bit faults. In this simulation, for SHA-3-512, SHA-3-384, SHA-3-256 and SHA-3-224, the thresholds of the percentage of known state for injecting

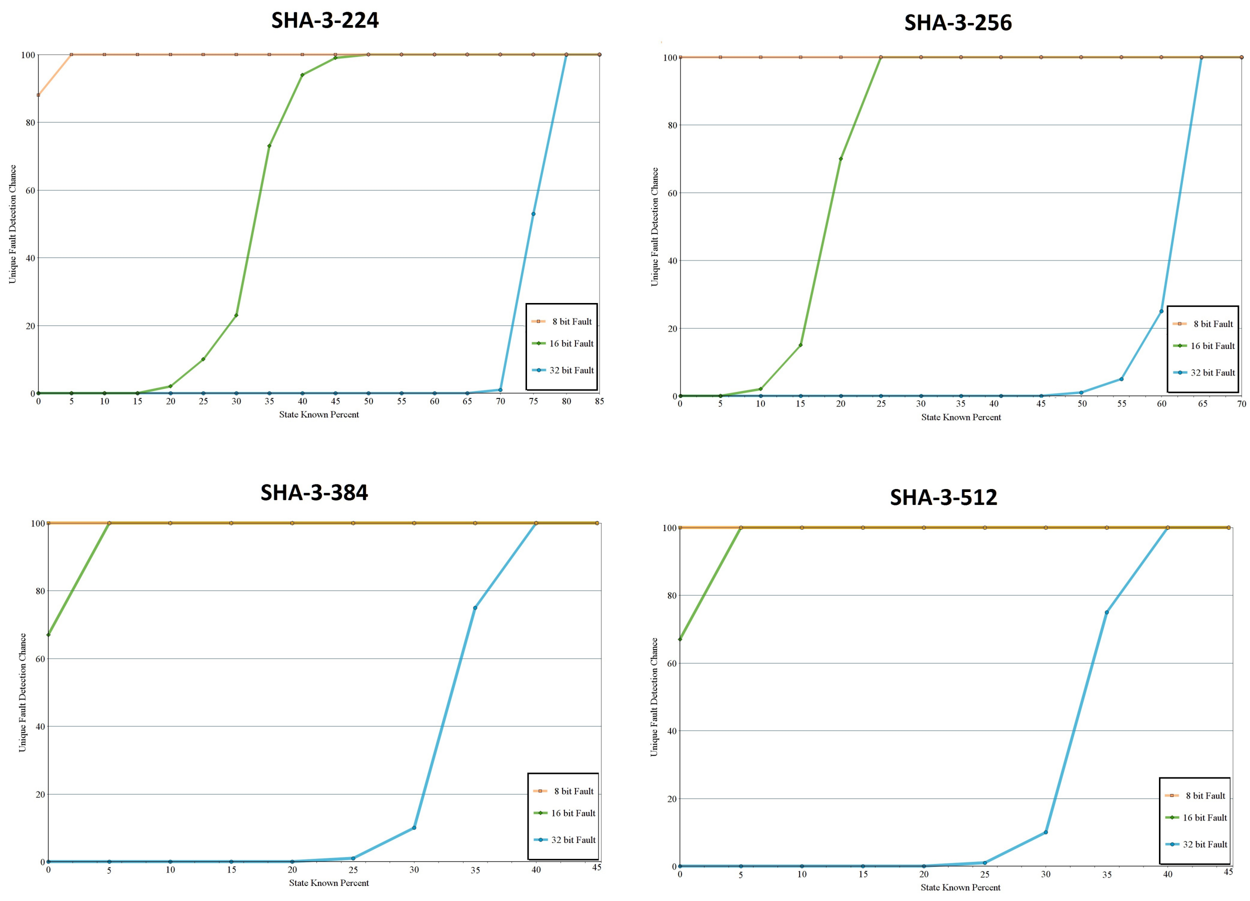

Figure 7. Fault detection probability using Algorithm 1

16-bit fault are (0, 0, 20, 30) respectively and for a 32-bit fault they are (30, 35, 65, 75) respectively. The results are illustrated in Figure 8.

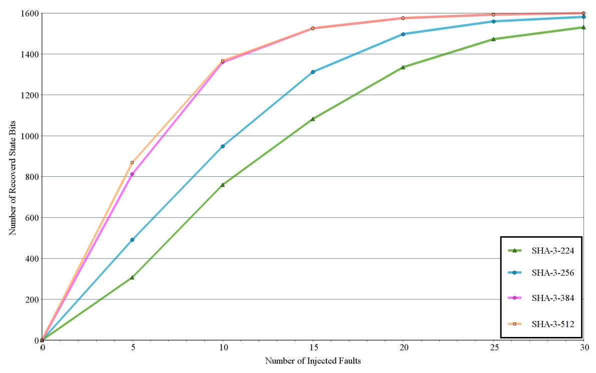

Figure 8. Simulation results of state recovery on SHA-3

In related work [20], with 60, 86 and 226 faults, the attacker could recover 1500 bits of the state respectively for SHA-3-512, SHA-3-384, SHA-3-256 and SHA-3-224, which is roughly triple of the required faults in our simulations. Although for [21] the results of the attack on SHA-3-512 and SHA-3-384 are almost the same as our attack, however, the attack is not efficient for SHA-3-256 and SHA-3-224 due to very low fault detection probability (less than 0.3%).

6 Improving the Results

In this section, we propose two improvements for the state recovery attack that was explained in the previous section, which will be presented as below:

Reverse Difference: In this improvement, by re-using the results of a detected faulty hash/Tag in differential relations, more bits of χ_{24}'s bits can be recovered which results in a more efficient half-round reverse for other faulty hashes/Tags.

Algebraic Relations: In the second improvement, the algebraic relations between two state parts which are recovered correctly during the attack will be used to increase the rate of the state's bit recovery.

Both of these improvements can be applied to the attack described in Section 5. In the following, each of these improvements is described and the simulation results of their application are expressed.

6.1 Reverse Difference

The fault analysis described in Section 5 was composed of three phases: half-round reversing, fault detection, and state recovery. In half-round reversing phase, bits of χ_{24} were retrieved from the fault-free and faulty hash/Tag. Then χ_{24}'s input differences are calculated where bit's value was known in both of them. Due to the non-linearity of function χ, this process is much less effective on SHA-3-224 and SHA-3-256, followed by weaker results for them.

To have a better view of the improvement at hand, Figure 9 is provided. In this figure, one row of fault-free and faulty hashes/Tags are being reversed with three sets of information. It should be mentioned that this figure's process goes from down to up as we are reversing the function.

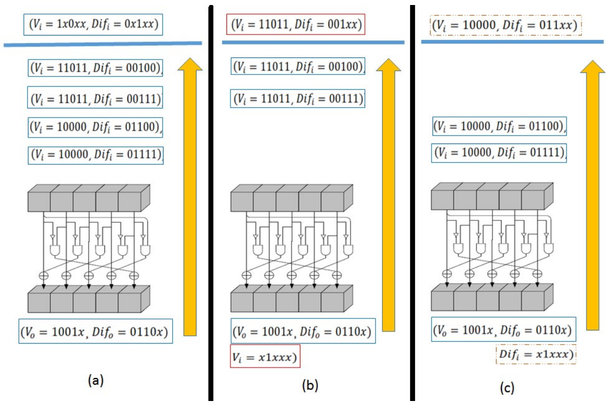
$(V_i = 1x0xx, Dif_i = 0x1xx)$
$(V_i = 11011, Dif_i = 00100)$
$(V_i = 11011, Dif_i = 00111)$
$(V_i = 10000, Dif_i = 01100)$
$(V_i = 10000, Dif_i = 01111)$
$(V_o = 1001x, Dif_o = 0110x)$
(a)
$(V_i = 11011, Dif_i = 001xx)$
$(V_i = 11011, Dif_i = 00100)$
$(V_i = 11011, Dif_i = 00111)$
$(V_o = 1001x, Dif_o = 0110x)$
$V_i = x1xxx)$
(b)
$(V_i = 10000, Dif_i = 011xx)$
$(V_i = 10000, Dif_i = 01100)$
$(V_i = 10000, Dif_i = 01111)$
$(V_o = 1001x, Dif_o = 0110x)$
$Dif_i = x1xxx)$
(c)

Figure 9. An example of Half-Round Reverse with Reverse Differences

In part (a), a row with four visible bits on output is given. By applying the same method as half-round reversing, we receive four combinations of fault-free and differential input as initial answers. Next, by checking for unchanged bits in them, two fault-free and two differential bits will be retrieved from row's input.

In part (b), row's output is similar to part (a) however the value of one fault-free bit of input is also known. By employing this new information, only two initial answers remain valid, followed by recovering five fault-free and three differential bits from row's input.

Part (c) is similar to part (a), with one known differential bit. By applying the same process, five fault-free and three differential bits from row's input can be retrieved.

As it is shown in Figure 9, finding more information about χ_{24}'s input can reduce the number of initial answers for each row which increases the efficiency of the half-round reversing process. To use this feature, it is necessary to provide a method for finding more fault-free or differential bits from χ_{24}'s input, which can be done using differential relations.

The differential relations by definition can determine the difference value of each bit in their output. If fault value-position and effective bits of χ_{23} state are given, the difference at χ_{24}'s input can be calculated. These values are known for a faulty hash/Tag with detected fault. So, by putting them to their differential relations again, all differences of χ_{24}'s input are determined, some of which were previously known. Therefore, by adding this new information to half-round reversing, a larger part of fault-free input can be obtained. Since the fault-free hash/Tag is the same for all faulty hashes, this new information can be used to help other faulty hashes/Tags in their half-round reversing process. From here on, we call this improvement reverse differences.

Figure 10 shows the number of fault-free and differential bits recovered from χ_{24}'s input using reverse differences on each fault.

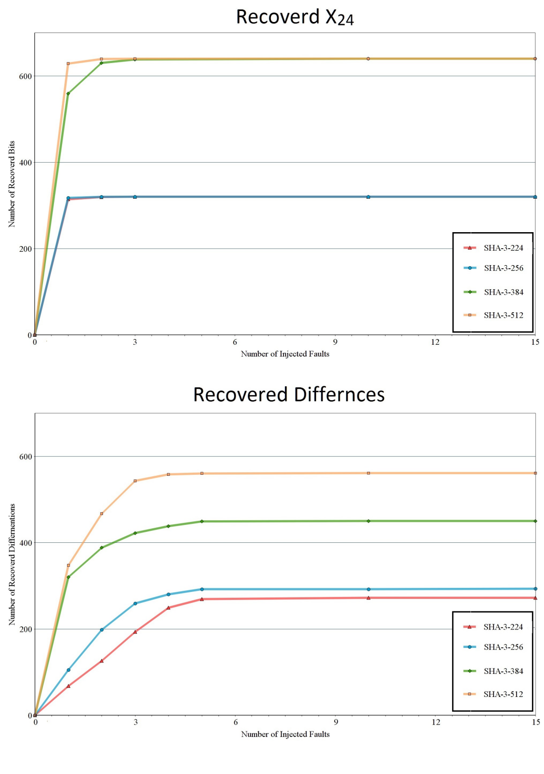

Figure 10. half-round reverse results with reverse differences in use

It can be seen the rate of increase in known bit drops after a few faults to the extent that after analyzing three detected faults, the number can almost be considered constant. Table 5 provided detailed values at this point. If we compare the numbers of Table 5 with the numbers in Table 4, the biggest relative increase is for SHA-3-224 and SHA-3-256. Since the output of half-round reverse will be used in the next two processes of attack, fault detection chances will also change with this improvement. Keeping that in mind that this improvement does not only effect half-round reverse process, but also the fault detection, we consider the fault-free bits of Plane 0 for all 4 hash functions and Plane 1 for SHA-3-384 and SHA-3-512 of χ_{24}'s as known bits and reused Algorithm 1. The new results are presented in Figure 11. In this figure, SHA-3-224 and SHA-3-256 results are almost the same as SHA-3-384's results in Figure 7, which could be a huge improvement.

Table 5. Results of half-round reverse with reverse differences in use

Function	Fault-free bits	Differential bits
SHA-3-224	319.3	318.9
SHA-3-256	320	319.1
SHA-3-384	638	633
SHA-3-512	639.8	637.2

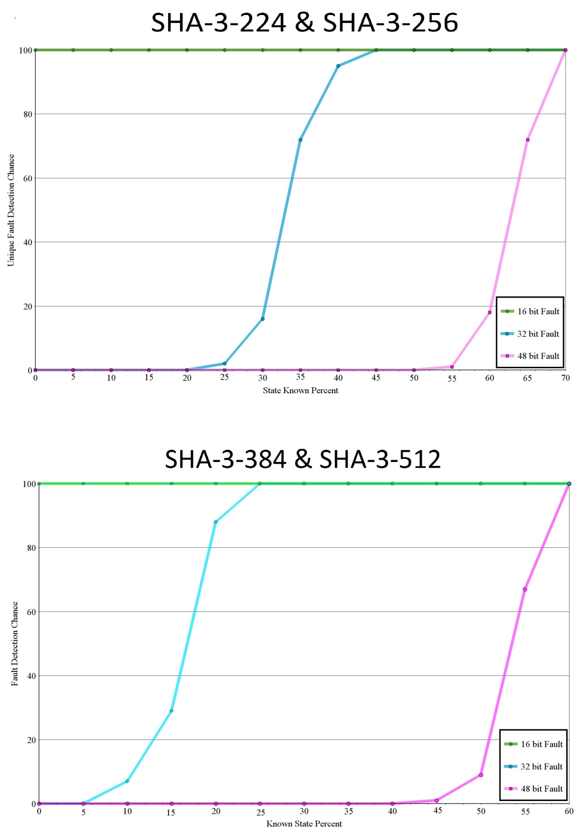

Figure 11. Fault detection when reverse differences strategy is used

6.2 Algebraic Resolution

In our fault analysis, the differences at χ_{24}'s input are used to form differential equations which then are used to detect fault and recover χ_{23}'s bits. But the differences are not the only information that the attacker have, a number of χ_{24}'s fault-free bits had also bean recovered during half-round reversing process. As mentioned before, χ_{23}'s and χ_{24}'s bits have an algebraic relation though SHA-3's round functions:

$$\chi_{24} = \pi \circ \rho \circ \theta \circ \iota_{23} \circ \chi(\chi_{23}) \qquad (16)$$

This algebraic relations between these two fault-free states can be used to form more valid equations on the target state, i.e. χ_{23}. Therefore, it is expected to increase the rate of state recovery. This improvement works very well with the initial attack. As can be seen in Figure 8, the average number of recovered bits with DFA decreases as the attack goes on; this is due to the reappearance of repetitive χ_{23}'s bits in differential relation, while algebraic relations, on the other hand, become more effective with attack's progress. Since as the attack goes on, extra information are provided for the solver, which reduces the problem solving complexity and more likely to return an unique answer.

We did 1000 simulations for DFA while using algebraic equations and without using the algebraic equations. The results show that the difference in the number of recovered bits, i.e. the recovered bits of χ_{23} when algebraic equations are used, is almost equal to the number of known fault-free bits of χ_{24}.

6.3 Simulation results

In this section, the simulation results of DFA on SHA-3 using improvements 6.1 and 6.2 will be provided. This simulation uses of 16, 32 and 48-bit faults in the same manner as the simulation described in Section 5.2 were using. The threshold of the percentage of the known state for 32-bit faults has been set to 30, 30, 10 and 10 for SHA-3-224, SHA-3-256, SHA-3-384, and SHA-3-512 respectively, and for 48-bit faults, the threshold has been set as 60, 60, 50 and 50 respectively. Also for SHA-3-224 and SHA-3-256 hash members one 8-bit fault is injected as the first fault.

Figure 12 and Table 6 show the results of this simulation. It's worth-noting that Figure 11 shows the results of this simulations as well.

To compare with the related works, the related works [19] and [20] SHA-3-385 and SHA-3-512's state can be recovered respectively with 125 and in with 80 faulty messages. Also, in [20], the chance of fault detection of SHA-3-224 and SHA-3-256 was 30.67% and 66.61% respectively and with an improvement of

Table 6. Details of our final results

Hash Function	Faults Required	16-bit Faults	32-bit Faults	16-bit Faults Detection Chance	32-bit Faults Detection Chance	48-bit Faults Detection Chance
SHA-3-224	7 - 8	3.79	2.13	99.1%	100%	88.4%
SHA-3-256	7	3.2	2.6	99.7%	100%	91.4%
SHA-3-384	5 - 6	2.41	1.76	100%	100%	96.2%
SHA-3-512	5 - 6	2	2.2	100%	100%	98.7%

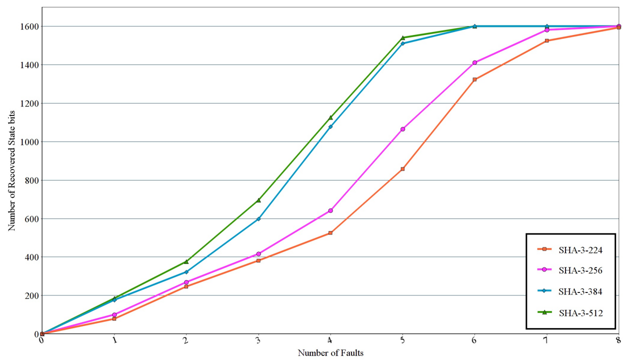

Figure 12. Final state recovery results

49.12% and 78.73%, in cost of injecting several faults to the 24th round. In [21], state recovery of SHA-3-512 and SHA-3-384 would require 60 faults and for SHA-3-256 and SHA-3-224 only 1-byte fault based model was possible and it pretty close to the results of [20] with 110 and 220 faults respectively.

We run our simulations also without 48-bit fault threshold and the number of faults required to recover state did not show a big change. For example in SHA-3-384, it went from 5.76 to 5.84. This means that one can recover the state with the given fault number by injecting 16 and 32-bit faults. We believe that this is due to the usage of algebraic resolution and its overshadowing of DFA in recovering state bit in the later faults. The next part of this section is dedicated to studying the process and results of paper [22].

Discussion

The latest paper related to fault analysis of SHA-3 is [22] and here we're going to briefly describe it and make a comparison between it and our work. In [22] a SAT solver is used with the following equations:

$$\begin{cases} H = \iota_{24} \circ \chi \circ \pi \circ \rho \circ \theta \circ \iota_{23} \circ \chi(\chi_{23}) \\ H' = \iota_{24} \circ \chi \circ \pi \circ \rho \circ \theta \circ \iota_{23} \circ \chi(\chi_{23} \oplus \Delta\chi_{23}) \\ \Delta\chi_{23} = \pi \circ \rho \circ \theta(\Delta\theta_{23}) \end{cases} \tag{17}$$

where $\Delta\theta_{23}$ have a limitation on how many faulty bytes it can have, similar to F_{size} in Algorithm 1.

Then fault-free and faulty hash/Tag $(H,\ H')$ would be given to the solver to find all possible answers for χ_{23}. For each answer, state bits are checked. If a bit have the same value as previous answers or it was not recovered before, that bit would be set as recovered; if not, that bit would be removed from recovered bits set.

That works shares some similarities with the current work which can be described as follows:

$$\begin{cases} H = \iota_{24} \circ \chi(\chi_{24} \ = \pi \circ ... \circ \chi(\chi_{23})) \\ H \oplus H' = \iota_{24} \circ \chi(\Delta\chi_{24} \ = \pi \circ ... \circ \chi(\chi_{23}, \Delta\chi_{23})) \end{cases} \tag{18}$$

after half-round reverse:

$$(18) \begin{cases} \chi_{24} = A.R.(\chi_{23}) \\ \Delta\chi_{24} = D.R.(\chi_{23}, \Delta\theta_{23}) \end{cases} \tag{19}$$

while we benefit from reverse differences, half-round reverse and bigger fault sizes as advantages of our work. In addition, the bellow equation could accelerate our attack procedure:

$$k' = a' \cdot b' = ((a \oplus \Delta a) \cdot (b \oplus \Delta b)) \tag{20}$$

$$k' = a \cdot b \oplus a \cdot \Delta b \oplus \Delta a \cdot b \oplus \Delta a \cdot \Delta b = k \oplus \Delta a \times \Delta b \tag{21}$$

In the above equation, k' is the value of a faulty bit after the application of an AND operator on it. What Equation 21 shows is that in the faulty bit's relation, e.g. k', the value of a fault-free bit is present right next to (XORed to) what we described as differential relations. If we calculate the faulty bit's relations of any function including χ, we will see this separation exists as well. This means that in [22]'s relations, it's almost impossible to find an equation that can be calculated only by fault value (This was the main feature we used in Algorithm 1 to detect fault value). So the possibility of finding fault value without χ_{23} will not be significant, except maybe for small fault sizes like 1-byte faults. We believe that this was the foundation for the change in the solver's target from their previous work [21] to [22].

In addition, we take advantage of the possibility of separating H' to $H \oplus D.R.$, which allow us to improve the attack with not just better solvers, but with known state bits and use larger faults in our attack. This separation was also the reason we were able to calculate and use reverse differences to improve our attack.

As the final results 6, 45, 12 and 27 faults are needed for state recovery respectively in SHA-3-512, SHA-3-384, SHA-3-256 and SHA-3-224 which in our work are 6, 6, 7 and 8 faults.

7 Conclusion and future works

In this paper, we used the low algebraic degree of 1.5 round differential relations of SHA-3 as a basis to apply an effective fault analysis in its hash members. Doing so resulted in state recovery, with 22-53 faults, of different members of the SHA-3 family. Next, we used differential relations again to improve attack even further and also used algebraic relations of SHA-3 to increase the rate of state's bits recovery. Combination of those approaches formed an attack that can recover state with 5-8 faults, with a high chance of fault detection.

If we compare the results of this paper with the results of the related works in literature, it shows a significant improvement in decreasing the number of required faults and increasing fault detection chance.

In our simulations, it was seen that analyzing faults larger than 32-bit, e.g. 48-bit faults, does not increase the attack efficiency. So it may be possible to use other approaches to also use those information to improve the attack further. One way to do this is to study differential relations to find a better way to solve them, given that many of our differential relations have similar faulty bits and state bits in them. Another way to change the analysis is to add a linear solver to the attack. This linear solver could be used to find θ_{24}'s faulty input from known differences on its output, $\Delta\chi_{24}$. Since for each injection point, there are a limited number of active differences on θ_{24}'s input, this would be a fast process and the result could be used to check if a fault and state exist to create it. Also differences on θ_{24}'s active bits have a bias to be 0 (because of the AND operation) which possibly useful in finding a valid θ_{24}'s input faster.

References

[1] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document. *Submission to NIST (Round 2)*, 3(30), 2009.

[2] NF Pub. Draft fips pub 202: Sha-3 standard: Permutation-based hash and extendable-output functions. *Federal Information Processing Standards Publication*, 2014.

[3] Itai Dinur, Orr Dunkelman, and Adi Shamir. Collision attacks on up to 5 rounds of sha-3 using generalized internal differentials. In *International Workshop on Fast Software Encryption*, pages 219–240. Springer, 2013.

[4] Sourav Das and Willi Meier. Differential biases in reduced-round keccak. In *International Conference on Cryptology in Africa*, pages 69–87. Springer, 2014.

[5] Paweł Morawiecki, Josef Pieprzyk, and Marian Srebrny. Rotational cryptanalysis of round-reduced keccak. In *International Workshop on Fast Software Encryption*, pages 241–262. Springer, 2013.

[6] Donghoon Chang, Arnab Kumar, Pawell Morawiecki, and Somitra Kumar Sanadhya. 1st and 2nd preimage attacks on 7, 8 and 9 rounds of keccak-224,256,384,512. In *SHA-3 Workshop*, 2014.

[7] Paweł Morawiecki and Marian Srebrny. A sat-based preimage analysis of reduced keccak hash functions. *Information Processing Letters*, 113(10-11):392–397, 2013.

[8] Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced keccak. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 249–274. Springer, 2016.

[9] Ling Song, Guohong Liao, and Jian Guo. Non-full sbox linearization: Applications to collision attacks on round-reduced keccak. In *Annual International Cryptology Conference*, pages 428–451. Springer, 2017.

[10] Silvia Mella, Joan Daemen, and Gilles Van Assche. New techniques for trail bounds and application to differential trails in keccak. *IACR Transactions on Symmetric Cryptology*, 2017(1):329–357, 2017.

[11] Kexin Qiao, Ling Song, Meicheng Liu, and Jian Guo. New collision attacks on round-reduced keccak. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 216–243. Springer, 2017.

[12] Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional cube attack on reduced-round keccak sponge function. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 259–288. Springer, 2017.

[13] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Annual international cryptology conference*, pages 513–525. Springer, 1997.

[14] Ludger Hemme. A differential fault attack

against early rounds of (triple-) des. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 254–267. Springer, 2004.

[15] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential fault analysis of the advanced encryption standard using a single fault. In *IFIP International Workshop on Information Security Theory and Practices*, pages 224–233. Springer, 2011.

[16] Navid Vafaei, Nasour Bagheri, Sayandeep Saha, and Debdeep Mukhopadhyay. Differential fault attack on SKINNY block cipher. In Anupam Chattopadhyay, Chester Rebeiro, and Yuval Yarom, editors, *Security, Privacy, and Applied Cryptography Engineering - 8th International Conference, SPACE 2018, Kanpur, India, December 15-19, 2018, Proceedings*, volume 11348 of *Lecture Notes in Computer Science*, pages 177–197. Springer, 2018.

[17] Ludger Hemme and Lars Hoffmann. Differential fault analysis on the sha1 compression function. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011 Workshop on*, pages 54–62. IEEE, 2011.

[18] Wieland Fischer and Christian A Reuter. Differential fault analysis on grøstl. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on*, pages 44–54. IEEE, 2012.

[19] Nasour Bagheri, Navid Ghaedi, and Somitra Kumar Sanadhya. Differential fault analysis of sha-3. In *International Conference in Cryptology in India*, pages 253–269. Springer, 2015.

[20] Pei Luo, Yunsi Fei, Liwei Zhang, and A Adam Ding. Differential fault analysis of sha3-224 and sha3-256. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016 Workshop on*, pages 4–15. IEEE, 2016.

[21] Pei Luo, Konstantinos Athanasiou, Yunsi Fei, and Thomas Wahl. Algebraic fault analysis of sha-3. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 151–156. IEEE, 2017.

[22] Pei Luo, Konstantinos Athanasiou, Yunsi Fei, and Thomas Wahl. Algebraic fault analysis of sha-3 under relaxed fault models. *IEEE Transactions on Information Forensics and Security*, 2018.

[23] NIST. Sha-3: a secure hash algorithm. https://competitions.cr.yp.to/sha3.html. Accessed: 1 May 2018.

NO IMAGE AVAILABLE

S.Ehsan Hosiny Nezhad received his M.S. at electrical engineering department, Shahid Rajaee Teacher Training University, Tehran, Iran. His research interest include hardware security and digital designing.

Masoumeh Safkhani obtained a Ph.D. in electrical engineering from Iran University of Science and Technology, in 2012, with the security analysis of RFID protocols as her major field. She is currently an Assistant Professor with the computer engineering department, Shahid Rajaee Teacher Training University, Tehran, Iran. Her current research interests include the security analysis of lightweight and ultra-lightweight protocols, targeting constrained environments, such as RFID, the IoT, VANET, and WSN. She is the author/co-author of over 50 technical articles in information security and cryptology in major international journals and conferences. Her official web-page is available at: https://www.sru.ac.ir/english-cv-dr-safkhani

Nasour Bagheri is an associate professor at the electrical engineering department, Shahid Rajaee Teacher Training University, Tehran, Iran. He is the author of over 100 articles in information security and cryptology. Nasour Bagheri's research interests include cryptology, more precisely, designing and analysis of symmetric schemes such as lightweight ciphers, e.g., block ciphers, hash functions and authenticated encryption schemes, cryptographic protocols for constrained environment, such as RFID tags and IoT edge devices and hardware security, e.g., the security of symmetric schemes against side channel attacks such as fault injection and power analysis. His webpage is available at: https://sites.google.com/view/nasour-bagheri.

Appendix 1: An example of Differential Relation's Calculation

$[y = 0\ ,\ x = 0,\ z = 0 : 64]$	$[y = 0\ ,\ x = 1,\ z = 0 : 64]$	$[y = 0\ ,\ x = 2,\ z = 0 : 64]$	$[y = 0\ ,\ x = 3,\ z = 0 : 64]$	$[y = 0\ ,\ x = 4,\ z = 0 : 64]$
$[y = 1\ ,\ x = 0,\ z = 0 : 64]$	$[y = 1\ ,\ x = 1,\ z = 0 : 64]$	$[y = 1\ ,\ x = 2,\ z = 0 : 64]$	$[y = 1\ ,\ x = 3,\ z = 0 : 64]$	$[y = 1\ ,\ x = 4,\ z = 0 : 64]$
$[y = 2\ ,\ x = 0,\ z = 0 : 64]$	$[y = 2\ ,\ x = 1,\ z = 0 : 64]$	$[y = 2\ ,\ x = 2,\ z = 0 : 64]$	$[y = 2\ ,\ x = 3,\ z = 0 : 64]$	$[y = 2\ ,\ x = 4,\ z = 0 : 64]$
$[y = 3\ ,\ x = 0,\ z = 0 : 64]$	$[y = 3\ ,\ x = 1,\ z = 0 : 64]$	$[y = 3\ ,\ x = 2,\ z = 0 : 64]$	$[y = 3\ ,\ x = 3,\ z = 0 : 64]$	$[y = 3\ ,\ x = 4,\ z = 0 : 64]$
$[y = 4\ ,\ x = 0,\ z = 0 : 64]$	$[y = 4\ ,\ x = 1,\ z = 0 : 64]$	$[y = 4\ ,\ x = 2,\ z = 0 : 64]$	$[y = 4\ ,\ x = 3,\ z = 0 : 64]$	$[y = 4\ ,\ x = 4,\ z = 0 : 64]$

Input State: Fault Size: 16 & injection point: [16, 0, 0]

0000 0000 0000 0000 xxxx xxxx xxxx xxxx	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000

Some of the relations			
$x[0,0,0] = 0$	$x[11,0,0] = 0$	$x[16,0,0] = I[16,0,0]$	$x[31,0,0] = I[31,0,0]$

θ_1

0000 0000 0000 0000 xxxx xxxx xxxx xxxx	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0xxx xxxx xxxx xxxx
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	x000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 xxxx xxxx xxxx xxxx	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0 xxx xxxx xxxx xxxx
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	x000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 xxxx xxxx xxxx xxxx	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0 xxx xxxx xxxx xxxx
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	x000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 xxxx xxxx xxxx xxxx	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0 xxx xxxx xxxx xxxx
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	x000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 xxxx xxxx xxxx xxxx	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0 xxx xxxx xxxx xxxx
0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	x000 0000 0000 0000 0000 0000 0000 0000

Some of the relations			
$x[0,0,0] = 0$	$x[11,0,0] = 0$	$x[16,0,0] = I[16,0,0]$	$x[31,0,0] = I[31,0,0]$

$\rho_1 \circ \pi_1$

0000 0000 0000 0000 xxxx xxxx xxxx xxxx 0000 0000 0000 0000 0000 0000 0000 0000	xxxx xxxx xxxx 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 xxxx	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 000 x xxxx xxxx xxxx x000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0 xxx xxxx xxxx xxxx x000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	xxxx xxxx xxxx x000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0 xxx	0000 0000 0000 0000 00000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0 xxx xxxx xxxx xxxx x000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0 xxx xxxx xxxx xxxx x000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 xxxx xxxx xxxx xxxx 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 00 xx xxxx xxxx xxxx xx00 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	xxxx xxxx 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 xxxx xxxx	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 00 xx xxxx xxxx xxxx xx00 0000 0000 0000 0000 0000 0000 0000

Some of the relations			
$x[0,0,0] = 0$	$x[11,0,0] = 0$	$x[16,0,0] = I[16,0,0]$	$x[31,0,0] = I[31,0,0]$

χ_1

xxxx xxxx xxxx 0000 xxxx xxxx xxxx xxxx 0000 0000 0000 0000 0000 0000 0000 xxxx	xxxx xxxx xxxx 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 xxxx	0000 0000 0000 0000 0000 0000 0000 000 x xxxx xxxx xxxx xxx0 0000 0000 0000 0000	0000 0000 0000 0000 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxx0 0000 0000 0000 0000	xxxx xxxx xxxx 0000 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxx0 0000 0000 0000 xxxx
0000 0000 0000 0000 0000 0000 0000 0000 0000 0 xxx xxxx xxxx xxxx x000 0000 0000	xxxx xxxx xxxx x000 0000 0000 0000 0000 0000 0 xxx xxxx xxxx xxxx x000 0000 0 xxx	xxxx xxxx xxxx x000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0 xxx	xxxx xxxx xxxx x000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0 xxx	0000 0000 0000 0000 0000 0000 0000 0000 0000 0 xxx xxxx xxxx xxxx x000 0000 0000
0000 0000 0000 0000 0 xxx xxxx xxxx xxxx x000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0 xxx xxxx xxxx xxxx x000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0 xxx xxxx xxxx xxxx x000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0 xxx xxxx xxxx xxxx xxxx xxxx x000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0 xxx xxxx xxxx xxxx x000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 00 xx xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx 0000	0000 0000 0000 0000 0000 0000 00 xx xxxx xxxx xxxx xx00 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 00 xx xxxx xxxx xxxx xx00 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 xxxx xxxx xxxx xxxx 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 xxxx xxxx xxxx xxxx 0000
xxxx xxxx 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 xxxx xxxx	xxxx xxxx 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 xxxx xxxx	xxxx xxxx 0000 0000 00 xx xxxx xxxx xxxx xx00 0000 0000 0000 0000 0000 xxxx xxxx	0000 0000 0000 0000 00 xx xxxx xxxx xxxx xx00 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 00 xx xxxx xxxx xxxx xx00 0000 0000 0000 0000 0000 0000 0000

Some of the relations	
$x[0,0,0] = I[20,0,0] \cdot S[0,0,2]$	$x[11,0,0] = I[31,0,0] \cdot S[11,0,2]$
$x[16,0,0] = I[16,0,0]$	$x[31,0,0] = I[31,0,0]$

θ_2

xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx x000 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx x000 xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xx00 0 xxx xxxx xxxx xxxx	xxxx xxxx xxxx x000 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx x000 xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx x000 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx x000 xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx x000 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx x000 xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx x000 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx x000 xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
Some of the relations				

$x[0,0,0] = I[20,0,0] \cdot S[0,0,2] \oplus I[20,0,0] \oplus I[20,0,0] \oplus I[19,0,0]$ $\oplus I[23,0,0] \cdot S[63,4,3] \oplus I[23,0,0] \cdot \bar{S}[63,1,2]$	$x[11,0,0] = I[31,0,0].S[11,0,2] \oplus I[31,0,0] \cdot \bar{S}[11,0,0] \oplus I[31,0,0]$ $\oplus I[29,0,0] \cdot \bar{S}[10,1,2] \oplus I[30,0,0]$
$x[16,0,0] = I[16,0,0] \oplus I[16,0,0] \cdot S[16,0,1]$	$x[31,0,0] = I[31,0,0] \oplus I[29,0,0] \oplus I[30,0,0] \cdot S[31,2,1] \oplus I[16,0,0] \oplus$ $I[31,0,0] \cdot S[31,0,1] \oplus I[20,0,0] \cdot S[30,3,3] \oplus I[21,0,0] \cdot \bar{S}[30,2,2]$

$\rho_2 \circ \pi_2$ / *Differential Relations*

xxxx xxxx xxxx xx00 xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxx0 00 xx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xx00 xxxx	xxxx xxxx xxxx xxxx xxxx xxxx x000 xxxx	xx00 00 xx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx x00 x xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxx0 00 xx xxxx xxxx xxxx xxxx
xxxx xxxx x000 xxxx xxxx xxxx xxxx xxxx	x000 xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxx0 0 xxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxx0 0 xxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx 000 x xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx x000 xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xx00 00 xx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	00 xx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx 00 xx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx x000	xxxx x000 0 xxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx 000 x xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xx00 xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx 000 x xxxx xxxx xxxx xxxx	xxxx 0000 xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx 00 xx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx xxxx 000 x xxxx xxxx	xxxx xxxx xxxx xxxx xxxx xxx0 0 xxx xxxx	xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
Some of the relations				

$x[0,0,0] = I[20,0,0] \cdot S[0,0,2] \oplus I[20,0,0] \oplus I[20,0,0] \oplus I[19,0,0]$ $\oplus I[23,0,0] \cdot S[63,4,3] \oplus I[23,0,0] \cdot \bar{S}[63,1,2]$	$x[11,0,0] = I[31,0,0].S[11,0,2] \oplus I[31,0,0] \cdot \bar{S}[11,0,0] \oplus I[31,0,0]$ $\oplus I[29,0,0] \cdot \bar{S}[10,1,2] \oplus I[30,0,0]$
$x[16,0,0] = I[16,0,0] \oplus I[16,0,0] \cdot S[16,0,1]$	$x[31,0,0] = I[31,0,0] \oplus I[29,0,0] \oplus I[30,0,0] \cdot S[31,2,1] \oplus I[16,0,0] \oplus$ $I[31,0,0] \cdot S[31,0,1] \oplus I[20,0,0] \cdot S[30,3,3] \oplus I[21,0,0] \cdot \bar{S}[30,2,2]$