Cryptanalysis of 2 round KECCAK-384

Rajendra Kumar[1], Nikhil Mittal[1], and Shashank Singh[2]

[1] Center for Cybersecurity, Indian Institute of Technology Kanpur, India
rjndr@iitk.ac.in, mnikhil@iitk.ac.in
[2] Indian Institute of Science Education and Research Bhopal
shashank@iiserb.ac.in

Abstract. In this paper, we present a cryptanalysis of round reduced KECCAK-384 for 2 rounds. The best known preimage attack for this variant of KECCAK has the time complexity 2^{129}. In our analysis, we find a preimage in the time complexity of 2^{89} and almost same memory is required.

Keywords: KECCAK· SHA-3 · Cryptanalysis · Hash Functions · Preimage Attack.

1 Introduction

Cryptographic hash functions are the important component of modern cryptography. In 2008, U.S. National Institute of Standards and Technology (NIST) announced a competition for the Secure Hash Algorithm-3 (SHA-3). A total of 64 proposals were submitted to the competition. In the year 2012, NIST announced KECCAK as the winner of the competition. The KECCAK hash function was designed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche [3]. Since 2015, KECCAK has been standardized as SHA-3 by the NIST.

The KECCAK hash function is based on sponge construction [4] which is different from previous SHA standards. Intensive cryptanalysis of KECCAK is done since its inception [1] [15] [6] [7] [13] [8] [5] [10] [16] [17] [11]. In 2012, Dinur *et al.* gave a practical collision attack for 4 rounds of KECCAK-224 and KECCAK-256 using differential and algebraic techniques [6] and also provided attacks for 3 rounds for KECCAK-384 and KECCAK-512. They further gave collision attacks in 2013 for 5 rounds of KECCAK-256 using internal differential techniques [7]. In 2016, using linear structures, Guo *et al.* proposed preimage attacks for 2 and 3 rounds of KECCAK-224, KECCAK-256, KECCAK-384, KECCAK-512 and for 4 rounds in case of smaller hash lengths [10]. Recently, in the year 2017, Kumar *et al.* gave efficient preimage and collision attacks for 1 round of KECCAK [11]. There are hardly any attack for the full round KECCAK, but there are many attacks for reduced round KECCAK. Some of the important results are shown in the Table 1 and Table 2.

Our Contribution: We propose a preimage attack for 2 rounds of round-reduced KECCAK-384. The time complexity of attack is 2^{89} and the memory

complexity is 2^{87}. The attack is not practical, but it outperforms the previous best-known attack [10], with a good gap. The proposed attack does not affect the security of full KECCAK.

Table 1. Preimage attack results.

No. of rounds	Hash length	Time Complexity	Reference
1	KECCAK- 224/256/384/512	Practical	[11]
2	KECCAK- 224/256	2^{33}	[15]
2	KECCAK- 224/256	1	[10]
2	KECCAK- 384/512	$2^{129}/2^{384}$	[10]
3	KECCAK- 224/256/384/512	$2^{97}/2^{192}/2^{322}/2^{484}$	[10]
4	KECCAK- 224/256	$2^{213}/2^{251}$	[10]
4	KECCAK- 384/512	$2^{378}/2^{506}$	[12]

Table 2. Collision attack results.

No. of rounds	Hash length	Time Complexity	Reference
1	KECCAK- 224/256/384/512	Practical	[11]
2	KECCAK- 224/256	2^{33}	[15]
3	KECCAK- 384/512	practical	[7]
4	KECCAK- 224/256	2^{24}	[6]
4	KECCAK- 224/256	2^{12}	[16]
4	KECCAK- 384	2^{147}	[7]
5	KECCAK- 224	2^{101}	[16]
5	KECCAK- 224	Practical	[17]
5	KECCAK- 256	2^{115}	[7]

2 KECCAK Description and Notations

KECCAK is a family of sponge hash functions with arbitrary output length. A sponge construction consists of a permutation function, denoted by f, a parameter "rate", denoted by r, and a padding rule *pad*. The construction produces a sponge function which takes as input a bit string N and output length d. It is described below.

The bit string N is first padded based on the *pad* rule. The padded string is divided into blocks of length r. The function f maps a string of length b to another of same length. The capacity, denoted by c, is a positive integer such that $r + c = b$. The initial state is a b-bit string which is set to all zeros. After a string N is padded, it undergoes two phases of sponge, namely absorbing and

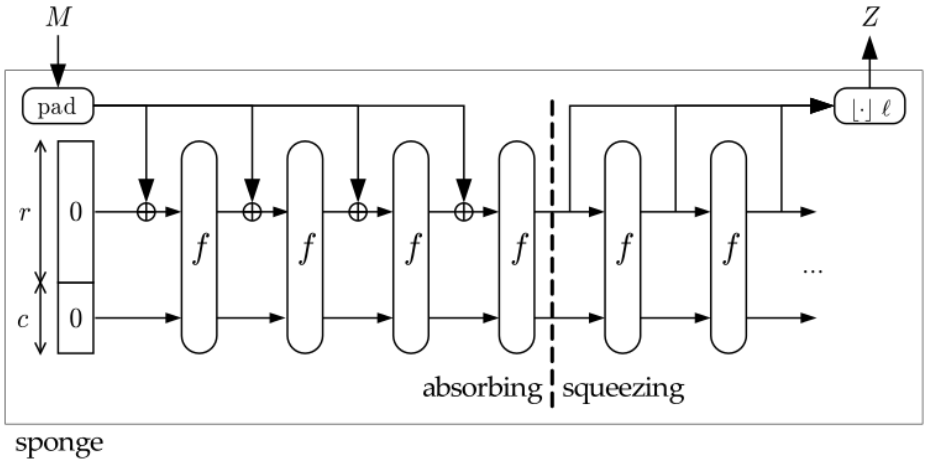

Fig. 1. The sponge construction [4]

squeezing. In the absorbing phase, the padded string N' is split into r-bit blocks, say $N_1, N_2, N_3, \ldots, N_m$. The first r bits of initial state are XOR-ed with the first block and the remaining c bits are appended to the output of XOR. Then it is given as input to the function f as shown in the diagram given in the Figure 1. The output of this f becomes the initial state for the next block and the process is repeated for all blocks of the message. After all the blocks are absorbed, let the resulting state be P.

In the squeezing phase, an string Z is initialized with the first r bits of the state P. The function f is applied on the state P and the first r bits of output, say P', is appended to Z. The P' is again passed to f and this process is repeated until $|Z| \geq d$. The output of sponge construction is given by the first d bits of Z.

The KECCAK family of hash functions is based on the sponge construction. The function f, in the sponge construction, is denoted by KECCAK-$f\,[b]$, where b is the length of input string. Internally KECCAK-$f\,[b]$ consists of a round function p which is recursively applied to a specified number of times, say n_r. More precisely KECCAK-$f\,[b]$ function is specialization of KECCAK-$p\,[b, n_r]$ family where $n_r = 12 + 2\,l$ and $l = \log_2(b/25)$ i.e.,

$$\text{KECCAK-}f\,[b] = \text{KECCAK-}p\,[b, 12 + 2l]\,.$$

The round function p in KECCAK consists of 5 steps, in each of which the state undergoes transformations specified by the step mapping. These step mappings are called θ, ρ, π, χ and ι. A state S, which is a b-bit string, in KECCAK is usually denoted by a 3-dimensional grid of size $(5 \times 5 \times w)$ as shown in the Figure 2. The value of w depends on the parameters of KECCAK. For example in the case of KECCAK-$f\,[1600]$, w is equal to 64. It is usual practice to represent a state in terms of rows, columns, lanes, and slices of the 3-dimensional

grid. Given a bit location (x, y, z) in the grid, the corresponding row is given by $(S[x + i \pmod 5), y, z] : i \in [0, 4])$. Similarly the corresponding column is given by the bits $(S[x, y + i \pmod 5), z] : i \in [0, 4])$ and the corresponding lane is given by $(S[x, y, z + i \pmod w)] : i \in [0, w - 1])$. Further the slice corresponding to a location (x, y, z), consists of $(S[x + j \pmod 5), y + i \pmod 5), z] : i, j \in [0, 4])$ bits. It is pictorially shown in the Figure 2.

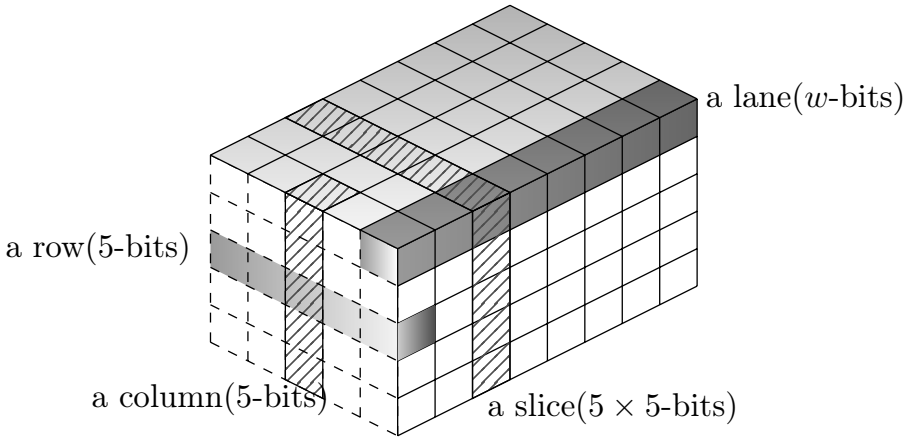

Fig. 2. A state in KECCAK

In the following, we provide a brief description of the step mappings. Let A and B respectively denote input and output states of a step mapping.

1. θ (**theta**): The theta step XORs each bit in the state with the parities of two neighboring columns. For a given bit position (x, y, z), one column is $((x - 1) \bmod 5, z)$ and the other is $((x + 1) \bmod 5, (z - 1) \bmod w)$. Thus, we have

$$B[x, y, z] = A[x, y, z] \oplus P[(x - 1) \bmod 5,\ z] \\ \oplus P[(x + 1) \bmod 5,\ (z - 1) \bmod w] \tag{1}$$

   where $P[x, z] = \oplus_{y=0}^{4} A[x, y, z]$.

2. ρ (**rho**): This step rotates each lane by a constant value towards the MSB i.e.,

$$B[x,\ y,\ z] = A[x,\ y,\ z + \rho(x, y) \bmod w], \tag{2}$$

   where $\rho(x, y)$ is the constant for lane (x, y). The constant value $\rho(x, y)$ is specified for each lane in the construction of KECCAK.

3. π (**pi**): It permutes the positions of lanes. The new position of a lane is determined by a matrix,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}, \tag{3}$$

where (x', y') is the position of lane (x, y) after π step.

4. χ (**chi**): This is a non-linear operation, where each bit in the original state is XOR-ed with a non-linear function of next two bits in the same row i.e.,

$$B[x, y, z] = A[x, y, z] \oplus \\ ((A[(x+1) \bmod 5, y, z] \oplus 1) \cdot A[(x+2) \bmod 5, y, z])) . \tag{4}$$

5. ι (**iota**): This step mapping only modifies the $(0, 0)$ lane depending on the round number i.e.,

$$B[0, 0] = A[0, 0] \oplus RC_i, \tag{5}$$

where RC_i is round constant that depends on the round number. The remaining 24 lanes remain unaffected.

Thus a round in KECCAK is given by $\texttt{Round}(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$, where A is the state and i_r is the round index. In the KECCAK-$p[b, n_r]$, n_r iterations of $\texttt{Round}(\cdot)$ is applied on the state A.

The SHA-3 hash function is KECCAK-$p[b, 12 + 2\,l]$, where $w = b/25$ and $l = \log_2(w)$. The value of b is 1600, so we have $l = 6$. Thus the f function in SHA-3 is KECCAK-$p[1600, 24]$.

The KECCAK team denotes the instances of KECCAK by KECCAK$[r, c]$, where $r = 1600 - c$ and the capacity c is chosen to be twice the size of hash output d, to avoid generic attacks with expected cost below 2^d. Thus the hash function with output length d is denoted by

$$\text{KECCAK-}d = \text{KECCAK}[r := 1600 - 2d,\ c := 2d], \tag{6}$$

truncated to d bits. The SHA-3 hash family supports minimum four different output length $d \in \{224, 256, 384, 512\}$. In the KECCAK-384, the size of $c = 2 \cdot d = 768$ and the rate $r = 1600 - c = 1600 - 768 = 832 = 13 \cdot 64$.

3 Preimage Attack for 2 Rounds of Round Reduced KECCAK-384

In this section, we present a preimage attack for a round reduced KECCAK. We will show that the preimage can be found in 2^{88} time and 2^{87} memory for 2 rounds of round-reduced KECCAK-384. Although it is not a practical attack, but it is an improvement over the existing best attack, for 2 rounds of KECCAK-384, which takes 2^{129} time [10].

3.1 Notations and Observations

In the analysis, we will represent a state by the lanes. There are in total 5×5 lanes. Each lane in a state will be represented by a variable which is a 64-bit array. A variable with a number in round bracket "(.)" represents the shift of the bits in array towards MSB. A variable with a number in square bracket "[.]" represents the bit value of the variable at that index. If there are multiple numbers in the square bracket then it represents the corresponding bit values.

We are going to use the following observations in our analysis.

1. **Observation 1:** The χ is a row-dependent operation. Guo *et al.* in [10], observed that if we know all the bits of a row then we can invert χ for that row. It is depicted in the Figure 3.

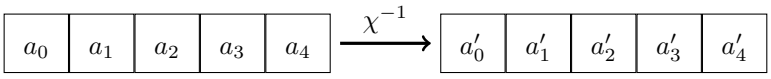

Fig. 3. Computation of χ^{-1}

$$a'_i = a_i \oplus (a_{i+1} \oplus 1) \cdot (a_{i+2} \oplus (a_{i+3} \oplus 1) \cdot a_{i+4}) \tag{7}$$

2. **Observation 2:** When only one output bit is known after χ step, then the corresponding input bits have 2^4 possibilities. Kumar *et al.* [11] gave a way to fix the first output bit to be the same as input bit and the second bit as 1. It is shown in the Figure 4.

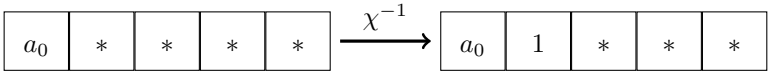

Fig. 4. Computation of χ^{-1}

3.2 Description of the Attack

The Keccak-384 outputs 384 bits hash value, which is represented by the first 6 lanes in the state obtained in the start of the squeezing phase. The diagram in the Figure 5 represents this state. The values of remaining lanes are represented by $\star$ and we do not care these values. We are interested in finding a preimage for which 6 lanes of corresponding state matches. We will call this state as *final*

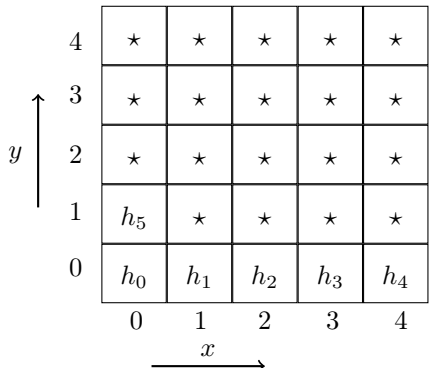

Fig. 5. Final State

state. Furthermore, we can ignore the ι step without the loss of generality, as it does not affect the procedure of the attack. However it should be taken into account while implementing the attack.

We further note that the initial state, which is fed to KECCAK-f function, is the first message block which is represented by $25 - 2 \cdot 6$ i.e., 13 lanes. The remaining 12 lanes are set to 0. Pictorially, this state is represented by the diagram in the Figure 6. We call this state *initial state*. Our aim is to find the values of a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1 and e_0, e_1 in the initial state which lead to a final state having first six lanes as h_0, h_1, h_2, h_3, h_4 and h_5.

We follow the basic idea of the attack, given in the paper [15]. We start the attack by setting variables in the initial state which ensures zero column parity. This is done by imposing the following restrictions.

$$\begin{aligned} &a_2 = a_0 \oplus a_1, \quad b_2 = b_0 \oplus b_1, \quad c_2 = c_0 \oplus c_1 \\ &d_1 = 0, \quad d_0 = 0 \quad \text{and} \quad e_1 = e_0. \end{aligned} \tag{8}$$

This type of assignment to the initial state will make the θ step mapping, an identity mapping. Even though we have put some restrictions to the initial state, we still find the input space of KECCAK-384 (with 1 message block) large enough to ensure first 6 lanes of output state, the given hash value. We explain the details of the analysis below.

Note that the output of attack is an assignment to the variables a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1 and e_0, e_1, which gives the target hash value. Recall that we are mounting an attack on the 2-Round KECCAK-384 (see the diagram in Figure 7). The overall attack is summarized in the diagram given in the Figure 8. The State 2, in the Figure 8, represents the state after $\pi \circ \rho \circ \theta$ is applied to the State 1. The θ-mapping becomes identity due to the condition (Equation 8) imposed on the initial state. The ρ and π mappings are, nevertheless, linear.

We are given with a hash value which is represented by first 6 lanes in the State 4 [Figure 8]. It represents the final state (`Round 2`) of KECCAK-384. The

0	0	0	0	0
0	0	0	0	0
a_1	b_1	c_2	0	0
a_2	b_2	c_1	d_1	e_1
a_0	b_0	c_0	d_0	e_0

Fig. 6. Setting of Initial State in the Attack

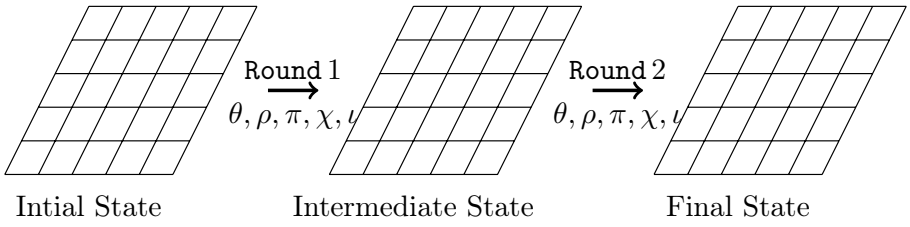

Fig. 7. Two round of Keccak-384

state can be inverted by applying $\chi^{-1} \circ \iota^{-1}$ mapping. The ι^{-1} is trivial and χ^{-1} can be computed using the Observations 1 and 2. The first 7 lanes of the output is $\{h'_0, h'_1, h'_2, h'_3, h'_4, h'_5, h'_6, 1\}$. We do not care the remaining lanes. Then the mappings π^{-1} and ρ^{-1} are applied, which are very easy to compute, to get the State 3 [Figure 8]. Note that, at this point, the blank lanes in the State 3, of the Figure 8, could take any random value and this does not affect the target hash value. The number shown in round brackets along with the variable, in the State 2 and State 3 [Figure 8], is due to ρ step mapping. On applying $\theta \circ \iota \circ \chi$, operation on the State 2, the output should match with the State 3 [Figure 8]. In the State 3, there are 7 lanes whose values are fixed. This will impose a total of 7×64 conditions on the variables we have set in the initial state. As mentioned earlier, we have also set 6 conditions (see the Equation 8) on the initial state variable and this will further add 6×64 conditions. So there are in total 13×64 conditions. Since the number of variables and the number of conditions is equal, we can expect to find one solution and it is indeed the case. In the rest of this section, we provide an algorithm to get the unique solution. Our method is based on the technique proposed by Naya-Plasencia *et al.* in the paper [15].

We aim to find the assignment of bits to the initial state which leads to a target hash value. We proceed as follows. We start with all possible assignments in the groups successive 3 slices. Using the constraints (transformation from

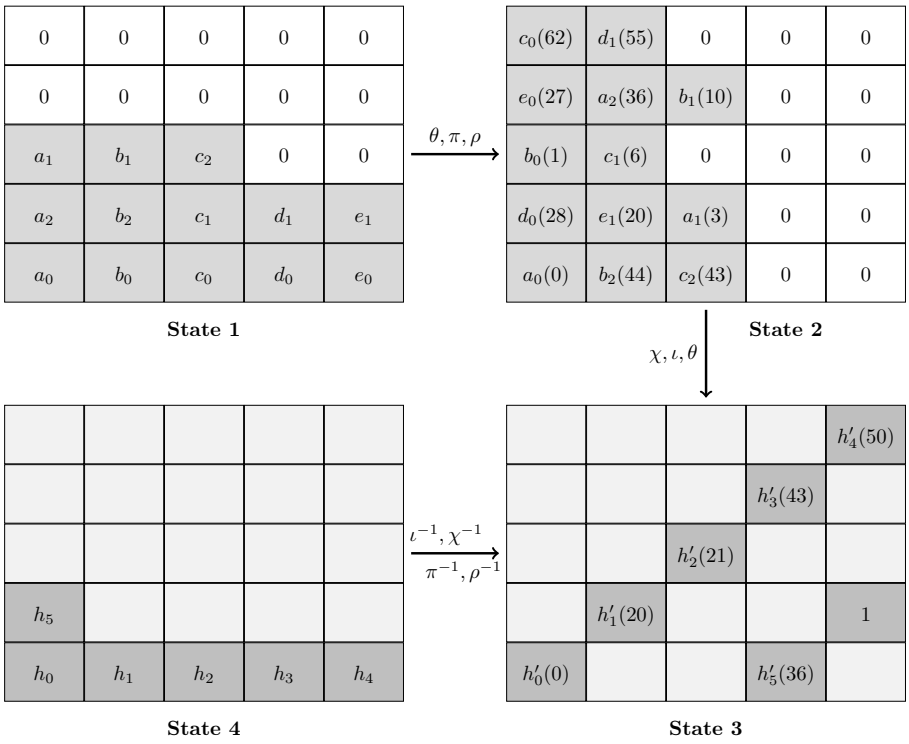

Fig. 8. Diagram for 2-round preimage attack on Keccak-384

State 2 to State 3 [Figure 8]), we discard some of the assignments, and store the remaining ones, out of which one would be a part of the solution. This is done for every 3-slice. Next step is to merge the two successive 3-slices. Again we do discard certain choices of assignments and keep the remaining ones. This process is continued to fix a set of good assignments to the 6-slices, 12-slices, 16-slices, 24-slices and 48-slice. In the last, after combining all the assignments we are left with a unique assignment, which is the required preimage. We explain the details in the Section 3.3 below.

3.3 Finding Partial Solutions

We focus on the two intermediate states of the attack i.e., the State 2 and the State 3 (see the Figure 9 below). Note that, since d_0 and d_1 are set to 0 in the beginning, we are now left with 11 lane variables a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, e_0 and e_1 only. We can ignore the ι mapping in the transformation form State 2 to State 3, without the loss of generality. The χ-mapping depends only on the row, so it will not get affected by the bit values of the other slices. It is θ-mapping

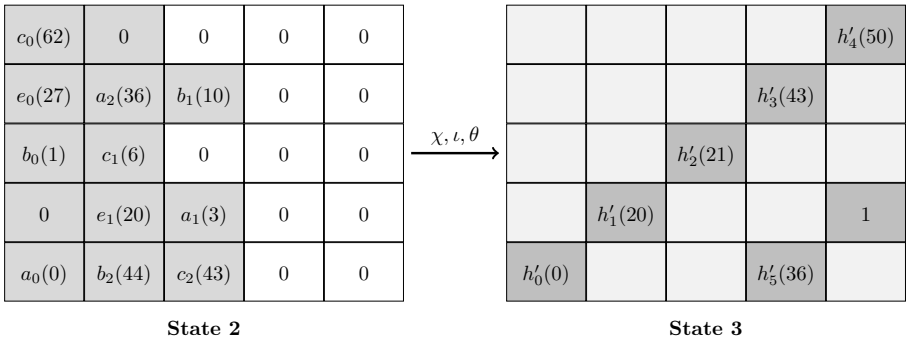

Fig. 9. Intermediate States in 2-round preimage attack on KECCAK-384

that depends on the values in the two slices; the slice on its original bit position and a slice just before it.

Possible solutions for 3-slices In a 3-slice there are $3 \cdot 11 = 33$ bit variables for which we have to find the possible assignments.

Note that the bit variables, for example take $a_0[i]$, $a_1[i]$ and $a_2[i]$, are related ($a_2 = a_0 \oplus a_1$), but due to rotation, they do not appear together when the successive 3 slices are considered. Similarly, the other variables are also independent when restricted to a 3-slice. This can be explained using the following example. If we take the first three slices then we get the following 33 independent variables, given in the Equation 9.

$$\begin{aligned} &a_0[0,1,2], \;\; a_1[3,4,5], \;\; a_2[36,37,38], \\ &b_0[1,2,3], \;\; b_1[10,11,12], \;\; b_2[44,45,46], \\ &c_0[62,63,0], \;\; c_1[6,7,8], \;\; c_2[43,44,45], \\ &e_0[27,28,29], \;\; e_1[20,21,22]. \end{aligned} \tag{9}$$

None of these variables have any dependency despite the initial restriction, given by Equation 8. So we have an input space of 33 independent variables in a given 3-slice.

Given a 3-slice in the State 2, we need to apply $\theta \circ \iota \circ \chi$ mapping to get an output in the State 3. Since the θ mapping depends on the values of two slices; the current slice and one preceding it, we will only able to get the correct output for two slices. In the State 3, we have the values of 7 lanes available with us. So for the two slices, we have $7 \cdot 2$ fixed bit values. For each of 2^{33} assignments in a 3-slice of the State 2, we compute the output of $\theta \circ \iota \circ \chi$ mapping and match it to the 14 bit locations, the values of which are available in the State 3. Thus for each 3-slice, we get $2^{33-14} = 2^{19}$ solutions. This is repeated for 16 consecutive 3-slices, other than last 16 slices. We use the fact that the time complexity of

building the list is given by the size of the list as stated in Section 6.4 of [15]. Thus the required time and memory complexity is of the order $16 \cdot 2^{19} = 2^{23}$.

Possible solutions for 6-slices The possible solutions for a 6-slice are obtained by merging the possible solutions of its constituents two 3-slices. The variables restricted to the 6-slice is again independent. This can be explained in the following manner. Consider the rotated lanes $a_0(0)$, $a_1(3)$ and $a_2(36)$. Since the lane variable a_2 is rotated by 36 and a_1 is rotated by 3, the corresponding bits of original lanes are still 33 places apart. Similarly e_0 is rotated by 27 and e_1 is rotated by 20, the corresponding bits are again 7 places apart, so there is no repetitions of bits (remember initial condition $e_0 = e_1$). Since the difference between the rotation of related variables is more than 6, the bit variables in a 6-slice are also independent. So we have $2^{19 \cdot 2} = 2^{38}$ possibilities for the bit variables in a 6-slice.

We have already noted that the θ-mapping cannot be computed for the first slice of a given 3-slice. But, when we are merging two consecutive 3-slices, θ-mapping for the first slice of second 3-slice can be computed and this will pose an additional restriction (of 7 bits) for the input space of the 6-slice. As an example consider a group of slices (0, 1, 2) and another group of slices (3, 4, 5). Note that the θ-mapping, on the slice 3, depends on the slice 3 and 2. So when we are merging these two 3-slices, we will have to satisfy the bits corresponding to slice 3, in the State 3.

So we get a total $2^{19 \cdot 2 - 7} = 2^{31}$ solutions. There are 8 number of 6-slices. The cost of this step is $8 \cdot 2^{31}$ in both time and memory. Note that the merging of two lists is done using the instant matching algorithm described in [14] by the method described in the Section 6.4 of the paper [15]. This method will be used in the following steps also, where the time complexity will be bounded by the number of solutions obtained. Thus this step has time and memory complexity of $8 \cdot 2^{31} = 2^{34}$.

Possible solutions for 12-slices For computing the possible solutions for a 12-slice, we merge two of its constituents 6-slices, in a manner similar to what we did for a 6-slice. In this case, the number of repeated bits in merge is 5, because the corresponding bits in e_0 and e_1 are set 7 places apart by the rotation in the State 2. Thus total number of possible solutions for a 12-slice is $2^{31 \cdot 2 - 5 - 7} = 2^{50}$. There are 4 groups of 12 slices, so it has time and memory complexity of $4 \cdot 2^{50} = 2^{52}$.

Possible solutions for 24-slices Similar to the previous cases, we merge each of its two consecutive 12-slices. In this case, the number of repeated bits is $24 - 7 = 17$, out of which $5 \cdot 2 = 10$ has already been considered, during the construction of possible solutions of 12-slices. So the number of new repeated bit variables are 7. Hence, the total number of possible solutions for this case is $2^{50 \cdot 2 - 7 - 7} = 2^{86}$. Note that the removal of addition seven bits is due to merging. There are 2 groups of 24 slices, so it has time and memory complexity of

$2 \cdot 2^{86} = 2^{87}$.

Possible solutions for 48-slice Finally, we merge the two groups of 24 slices. We have 2 sets of 24 slices as

1^{st} group :

$$\left.\begin{aligned} a_0 &\to 0,\ 1,\ 2, \ldots,\ 23 \\ a_1 &\to 3,\ 4,\ 5, \ldots,\ 26 \\ a_2 &\to 36,\ 37,\ 38, \ldots,\ 59 \end{aligned}\right\} \tag{10}$$

2^{nd} group :

$$\left.\begin{aligned} a_0 &\to 24,\ 25,\ 26, \ldots,\ 47 \\ a_1 &\to 27,\ 28,\ 29, \ldots,\ 50 \\ a_2 &\to 60,\ 61,\ 62, \ldots,\ 19 \end{aligned}\right\}. \tag{11}$$

After Merging these two groups [Equation (10) and Equation (11)] of 24 slices, we get

$$\left.\begin{aligned} a_0 &\to 0,\ 1,\ 2, \ldots,\ 47 \\ a_1 &\to 3,\ 4,\ 5, \ldots,\ 50 \\ a_2 &\to 36,\ 37, \ldots,\ 63,\ 0,\ 1, \ldots,\ 19 \end{aligned}\right\}. \tag{12}$$

Here the common variables for $\langle a_0, a_1, a_2 \rangle$ are the bits with positions $36, 37, \ldots, 47$ and $3, 4, \ldots, 19$. They are total 29 in number. It will impose 29 conditions on the input space for the 48-slice. Similarly for the lanes $\langle b_0, b_1, b_2 \rangle$, we get 23 conditions and for $\langle c_0, c_1, c_2 \rangle$, we get 24 such conditions. On the other hand, there are 7 new repeated bits in the lanes e_0 and e_1. Thus the total number of solutions turns out to be $2^{86 \cdot 2 - (29+23+24+7) - 7} = 2^{82}$. Since, there is only one 48-slices, so it has time and memory complexity of 2^{82}.

Possible solutions for remaining 16 slices For finding solutions for the remaining 16 slices, we first find solutions for the 12 rightmost slices, the same way as before, and obtaining 2^{50} possible solutions. Next, we obtain the possible solutions for the remaining 4 slices, we have 44 variables and none of them are repeated. Since we can get the output of θ-mapping for the last 3 slices out of the 4. We have $2^{44-7 \cdot 3} = 2^{23}$ possible solutions for this 4-slice. Now, we can merge 12-slice and 4-slice to obtain possible solutions for the last 16 slices. Between 12-slice and 4-slice, there are 4 repetitions (due to e_0 and e_1) and there are additional 7 bits of restrictions due to merging. This gives us $2^{50+23-4-7} = 2^{62}$ possible solutions.

Final Solution(s) and attack complexity Now, we have to merge the solutions for the group of first 48 slices and the group of last 16 slices. They have in common 35 bits from a_0, a_1 and a_2, 41 bits from b_0, b_1 and b_2, 40 bits from c_0, c_1 and c_2 and 14 bits from e_0 and e_1. Additionally, in merging, we can compute

the θ mapping of the remaining two slices, in turn get the additional restriction of $2 \cdot 7$ bits. Thus the total number of possible solutions, we are left with, is $2^{82+62-(35+41+40+14)-2\cdot 7} = 2^0 = 1$. This step has time complexity 2^{82}.

Total time complexity of the attack is given by $2^{33}+2^{34}+2^{52}+2^{87}+2^{63}+2^{82}$, which is of the order 2^{88}. The total memory required is 2^{87}. This confirms that there exists a set of values for the variables such that the preimage can be obtained from the hash value for the KECCAK-384.

Remark: *In our attack, we have fixed d_0, d_1 lanes to be equal to 0 as shown in Equation (8) because otherwise, these variables would have increased the number of solutions, due to shifting by ρ. And this would have increased the complexity of the attack. We chose to eliminate their effects by setting them to 0. For further implementation details, we refer to the Section 6.4 of the paper [15]. Also due to the padding rule on the message, the assignment to the $c_1[63]$ bit should be 1. This happens with probability $\frac{1}{2}$. On failure we can repeat the attack by setting any value to d_0,d_1 which satisfies $d_0[i] = d_1[i]$.*

In view of the above remark, the overall cost of the attack is $2 \cdot 2^{88}$ i.e., 2^{89}.

4 Conclusion and Future works

In this paper, we have presented a preimage attack on the 2 rounds of round-reduced KECCAK-384. The attack is not yet practical but it is much better than the existing best-known attack in term of the time complexity. The basic idea of the attack can be used to mount a practical preimage attack on the KECCAK[$r := 400 - 192$, $c := 192$] and KECCAK[$r := 800 - 384$, $c := 384$]. We are working on their implementations. We will make the source code public, once it is ready. Further, in future, we will try to explore a practical attack for the 2 or more rounds of round-reduced KECCAK-384.

Acknowledgement

We thank the reviewers of Indocrypt-2018 for providing comments which helped in improving the work. In particular, we thank an anonymous reviewer for suggesting us to implement the attack on the KECCAK[$r := 400-192$, $c := 192$] and also providing insights to further improve the attack. We take it as the future work.

References

1. Bernstein, D.J.: Second preimages for 6 (7?(8??)) rounds of keccak. NIST mailing list (2010)
2. Bertoni, G., Daemen, J., Peeters, M., Assche, G.: The keccak reference. online at http://keccak. noekeon. org/keccak-reference-3.0. pdf (2011)
3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak specifications. Submission to NIST (Round 2) (2009)

4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic sponges. online] http://sponge. noekeon. org (2011)
5. Chang, D., Kumar, A., Morawiecki, P., Sanadhya, S.K.: 1st and 2nd preimage attacks on 7, 8 and 9 rounds of keccak-224,256,384,512. In: SHA-3 workshop (August 2014) (2014)
6. Dinur, I., Dunkelman, O., Shamir, A.: New attacks on keccak-224 and keccak-256. In: Fast Software Encryption. pp. 442–461. Springer (2012)
7. Dinur, I., Dunkelman, O., Shamir, A.: Collision attacks on up to 5 rounds of sha-3 using generalized internal differentials. In: International Workshop on Fast Software Encryption. pp. 219–240. Springer (2013)
8. Dinur, I., Dunkelman, O., Shamir, A.: Improved practical attacks on round-reduced keccak. Journal of cryptology **27**(2), 183–209 (2014)
9. Dworkin, M.J.: Sha-3 standard: Permutation-based hash and extendable-output functions. Federal Inf. Process. Stds.(NIST FIPS)-202 (2015)
10. Guo, J., Liu, M., Song, L.: Linear structures: Applications to cryptanalysis of round-reduced keccak. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 249–274. Springer (2016)
11. Kumar, R., Rajasree, M.S., AlKhzaimi, H.: Cryptanalysis of 1-round keccak. In: International Conference on Cryptology in Africa. pp. 124–137. Springer (2018)
12. Morawiecki, P., Pieprzyk, J., Srebrny, M.: Rotational cryptanalysis of round-reduced keccak. In: International Workshop on Fast Software Encryption. pp. 241–262. Springer (2013)
13. Morawiecki, P., Srebrny, M.: A sat-based preimage analysis of reduced keccak hash functions. Information Processing Letters **113**(10-11), 392–397 (2013)
14. Naya-Plasencia, M.: How to improve rebound attacks. In: Annual Cryptology Conference. pp. 188–205. Springer (2011)
15. Naya-Plasencia, M., Röck, A., Meier, W.: Practical analysis of reduced-round keccak. In: International Conference on Cryptology in India. pp. 236–254. Springer (2011)
16. Qiao, K., Song, L., Liu, M., Guo, J.: New collision attacks on round-reduced keccak. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 216–243. Springer (2017)
17. Song, L., Liao, G., Guo, J.: Non-full sbox linearization: applications to collision attacks on round-reduced keccak. In: Annual International Cryptology Conference. pp. 428–451. Springer (2017)