Preimage Attacks on Round-reduced KECCAK-224/256 via an Allocating Approach*

Ting Li and Yao Sun(✉)

State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.
School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China.
sunyao@iie.ac.cn

Abstract. We present new preimage attacks on standard KECCAK-224 and KECCAK-256 that are reduced to 3 and 4 rounds. An allocating approach is used in the attacks, and the whole complexity is allocated to two stages, such that fewer constraints are considered and the complexity is lowered in each stage. Specifically, we are trying to find a 2-block preimage, instead of a 1-block one, for a given hash value, and the first and second message blocks are found in two stages, respectively. Both the message blocks are constrained by a set of newly proposed conditions on the middle state, which are weaker than those brought by the initial values and the hash values. Thus, the complexities in the two stages are both lower than that of finding a 1-block preimage directly. Together with the basic allocating approach, an improved method is given to balance the complexities of two stages, and hence, obtains the optimal attacks. As a result, we present the best theoretical preimage attacks on KECCAK-224 and KECCAK-256 that are reduced to 3 and 4 rounds. Moreover, we practically found a (second) preimage for 3-round KECCAK-224 with a complexity of $2^{39.39}$.

Keywords: Cryptanalysis · KECCAK · SHA-3 · Preimage attack.

1 Introduction

The KECCAK sponge function family [3], which was designed by Bertoni et al., became a candidate for the SHA-3 competition in 2008 [20]. It won this competition in 2012, and the U.S. National Institute of Standards and Technology (NIST) standardized KECCAK as Secure Hash Algorithm-3 (SHA-3) in 2015 [26]. KECCAK has received numerous security analysis since it was publicly available in 2008.

On practical collision attacks, Dinur et al. presented the first actual collision attack on 3-round KECCAK-384 based on a generalized internal differential attack [8]. Besides, they obtained practical complexities up to 4 out of 24 rounds

* This work was supported by National Natural Science Foundation of China under Grant No. 61877058, and Strategy Cooperation Project AQ-1701. *(Corresponding author: Yao Sun)*

of KECCAK-224/256 [7] [9]. Then, Qiao et al. extended Dinur et al.'s framework and achieved the first practical collision attack against 5-round SHAKE128 [22]. By improving Qiao et al.'s method, Song et al. proposed a practical collision attack on KECCAK-224 reduced to 5 rounds in 2017 [23]. Most of these collision attacks depend on the differential trails. Daemen et al. analyzed the differential propagation of KECCAK in 2012 [6]. After that, Kölbl et al. went a step further to study the differential properties of KECCAK-f[800] and KECCAK-f[1600], and presented collision attacks with practical complexity on KECCAK when the permutation is reduced to 4 rounds [14].

For preimage attacks, Naya-Plasencia et al. [19] and Morawiecki et al. [18] presented practical attacks up to 2 rounds. Guo et al. developed the technique of *linear structures*, and presented a practical attack on 3-round SHAKE128 [12]. Besides, the analysis of theoretical preimage attacking results on 3-round and 4-round instances of KECCAK are also given in their paper. Li et al. constructed a new kind of structures, called *cross-linear structures*, and improved the theoretical preimage complexities on 3-round KECCAK-256/SHA3-256/SHAKE256 [15]. Theoretical preimage attacks up to 7/8/9 rounds on KECCAK-224/256/512 are considered in [2] [5] [17]. In addition, Aumasson and Meier presented a new type of distinguisher and applied it to reduced versions of the KECCAK-f permutation in 2009 [1].

The KECCAK permutation is also used for authenticated encryption, and many researches have been made in this field. In 2014, Dinur et al. gave the first cube attacks on round-reduced KECCAK sponge function and applied it to attack MAC and stream cipher mode [11]. Then they analyzed the problems of key recovery, MAC forgery and other types of attacks on the keyed mode of KECCAK as well as the security margin of Keyak — a KECCAK-based authenticated encryption scheme [10]. After that, Huang et al. developed the conditional cube tester to analyze KECCAK in keyed modes and improved the previous distinguishing attacks in 2017 [13]. Since then, the keyed modes of KECCAK have attracted more intensive cryptanalysis [4] [16] [24].

In this paper, we present an allocating approach to make preimage attacks on KECCAK-224 and KECCAK-256 that are reduced to 3 and 4 rounds. Generally, to find a 1-block preimage for a given hash value, a system is constructed by this hash value and hash algorithm. This system contains two kinds of constraints. The first kind comes from the initial value. For example, the last $224 \times 2 = 448$ bits in the initial state of KECCAK-224 must be 0's and these bits will not XOR messages. Constraints of the second kind are produced by the hash value. That is, the first l output bits of the hash algorithm must equal the given l-bit hash value. The unknowns of the system are usually the bit values of the messages. Constrained by these two kinds of constraints, the system is often with high nonlinearity and hard to be solved. The motivation of the allocating approach is to allocate these two kinds of constraints to *two* different sets of unknowns. Specifically, we prefer to find a 2-block preimage instead of a 1-block one. The unknowns from the first message block are constrained by the first kind of constraints, while those from the second message block should make

the second kind of constraints hold. Thus, the whole attacking complexity is allocated to two stages, and we expect the complexity of each stage to be lower than that of finding a 1-block preimage. Our motivation is shown in Figure 1.

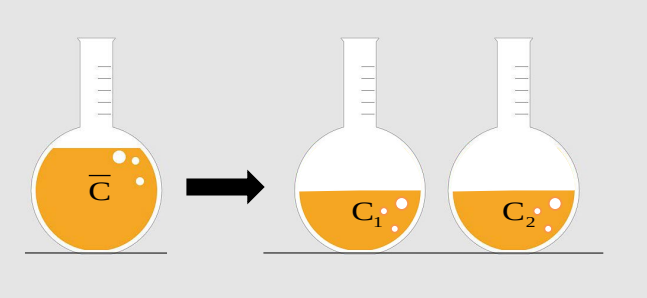

Fig. 1. The motivation of the allocating approach. The volume of water in a flask stands for the complexity of solving a system. Let $\bar{C}$ be the complexity of finding a 1-block preimage, C_1 and C_2 be the complexities of the two stages used to find a 2-block preimage by the allocating approach. We expect $\bar{C} > C_1 + C_2$.

The key step of applying this allocating approach is to find *suitable* constraints on the *middle state*, i.e. the output state of the first block as well as the initial state of the second block. Since more constraints usually cost more operations for solving the systems, to make the complexities of the two stages both lower than that of finding a 1-block preimage, the constraints on the middle state must be weaker than both kinds of constraints mentioned in the last paragraph. We improve the structure proposed by Li et al. [15], and obtain a set of *suitable* constraints on the middle state. For example, the number of constraints on the middle state of Keccak-224 is 129, which is smaller than 448 (the number of constraints from the initial value) and 224 (the number of constraints from the hash value). For Keccak-256, the number of constraints on the middle state is 193, which is also good enough to improve the preimage attacks on Keccak-256.

The contributions of this paper are summarized in three aspects.

1. We present an *allocating approach* for preimage attacks on round-reduced Keccak. This approach allocates the whole attack complexity to two stages, called *Precomputation Stage* and *Online Stage* for convenience. The complexity of each stage is lower than that of finding a 1-block preimage directly. To the best of our knowledge, this is the first two-block attack on standard Keccak, although multi-block methods have been successfully applied to MD5 [27] and SHA-1 [25].
2. We propose a new set of constraints on the middle state by improving Li et al.'s structure [15]. The improved structure could linearize the generated system at a low cost, such that we obtain more degrees of freedom to solve for the second message block.

Rounds	Capacity	Instances	Complexity	Reference
3	224	Keccak-224	2^{97}	[12]
			$\mathbf{2^{38}}$	Sec. 4.2
		SHA3-224	$\mathbf{2^{41}}$	Sec. 4.2
	256	Keccak-256	2^{192}	[12]
			2^{150}	[15]
			$\mathbf{2^{81}}$	Sec. 4.2
		SHA3-256	2^{151}	[15]
			$\mathbf{2^{84}}$	Sec. 4.2
		SHAKE256	2^{153}	[15]
			$\mathbf{2^{86}}$	Sec. 4.2
4	224	Keccak-224	2^{213}	[12]
			$\mathbf{2^{207}}$	Sec. 4.3
		SHA3-224	$\mathbf{2^{207}}$	Sec. 4.3
	256	Keccak-256	2^{251}	[12]
			$\mathbf{2^{239}}$	Sec. 4.3
		SHA3-256	$\mathbf{2^{239}}$	Sec. 4.3
		SHAKE256	$\mathbf{2^{239}}$	Sec. 4.3

Table 1. Summary of preimage attacks on 3/4-round Keccak-224/256

3. We improve theoretical complexities of preimage attacks on 3/4-round Keccak-224 and Keccak-256, as well as SHA3-224/256 and SHAKE256. Particularly, we give the first practical preimage attack on 3-round Keccak-224 with about $2^{39.39}$ operations. The theoretical results of preimage attacks in this paper, as well as the previous best ones, are summarized in Table 1. Detailed theoretical complexities of the two stages of attacking 3/4-round Keccak-224/256 are given in Table 2.

This paper is organized as follows. Some preliminaries and notations are given in Section 2. The allocating approach is proposed in Section 3. Theoretical analyses are presented in Section 4, and a practical preimage attack on 3-round Keccak-224 comes in Section 5. At last, we conclude this paper in Section 6.

Rounds	Instances	C_1 (Pre. Stage)	C_2 (Onl. Stage)	Overall Complexity	Reference
3	KECCAK-224	2^{66}	2^{31}	2^{66}	Sec. 4.1
		$2^{35.62}$	2^{38}	2^{38}	Sec. 4.2
	KECCAK-256	2^{162}	2^{62}	2^{162}	Sec. 4.1
		$2^{80.06}$	2^{81}	2^{81}	Sec. 4.2
4	KECCAK-224	2^{129}	2^{207}	2^{207}	Sec. 4.3
	KECCAK-256	2^{193}	2^{239}	2^{239}	Sec. 4.3

Table 2. Detailed theoretical complexities of the two stages. C_1 and C_2 represent the complexities of Precomputation Stage and Online Stage. Basic and improved allocating approaches are used in Section 4.1 and Section 4.2, respectively.

2 Preliminaries

2.1 The sponge construction

The sponge construction is used in KECCAK algorithm. As shown in Figure 2, it processes messages in two phases—absorbing phase and squeezing phase. With these two phases, a sponge construction receives an input stream of any length and produces an output bit stream of any desired length.

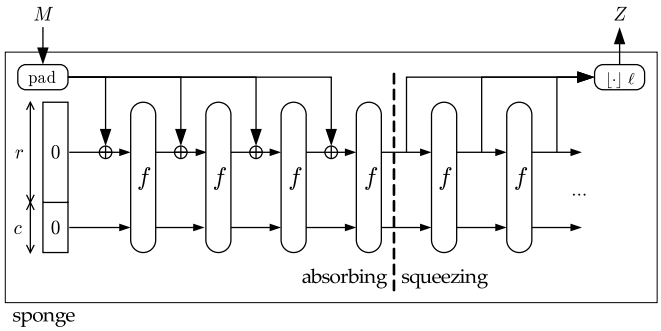

Fig. 2. The sponge construction.

At the beginning, the internal state of b-bits is initialized to be all 0's, which is the **initial value (IV)**. The message is padded and split into blocks of r-bits. In the absorbing phase, the first r bits of b-bits state are XORed with the message block, followed by the application of permutation f. This procedure is repeated until all the message blocks are processed. Then in the squeezing phase,

the first l bits are output. With an additional application of f, another l output bits are obtained. The algorithm iterates this step until the required length of a digest is reached.

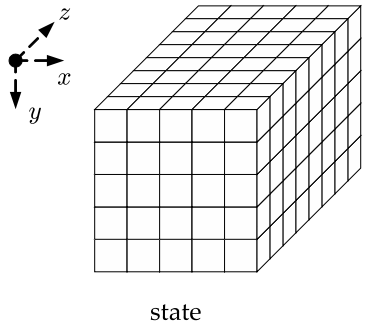

Fig. 3. The KECCAK state.

2.2 The KECCAK-f permutations

According to the KECCAK reference [3], there are 7 KECCAK-f permutations, indicated by KECCAK-$f[b]$, where $b \in \{25, 50, 100, 200, 400, 800, 1600\}$. We call b the **width** of the permutation. In this paper, we only focus on the case $b = 1600$, since KECCAK-$f[1600]$ is used widely in practice, which can be described as a 5×5 **64-bits lanes** as depicted in Figure 3. In this paper, we use L to denote the number of bits in a lane. In KECCAK-$f[1600]$, we have $L = 64$. Each bit is denoted as $A_{x,y,z}$. The integer triples (x, y, z) are the indices of bits, where x, y come from the set $\{0, 1, 2, 3, 4\}$ and $0 \leq z \leq L - 1$. The values of x and y are taken modulo 5 and we take z's values modulo L in the rest of this paper. The axis z is omitted sometimes for simplification.

The function KECCAK-$f[1600]$ consists of 24 rounds permutation R. Each round R consists of five steps:

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta,$$

where

$$\theta : A_{x,y,z} = A_{x,y,z} \oplus \bigoplus_{j=0}^{4} (A_{x-1,j,z} \oplus A_{x+1,j,z-1}),$$

$$\rho : A_{x,y,z} = A_{x,y,(z+r_{x,y})},$$

$$\pi : A_{y,2x+3y,z} = A_{x,y,z},$$

$$\chi : A_{x,y,z} = A_{x,y,z} \oplus (A_{x+1,y,z} \oplus 1) \cdot A_{x+2,y,z},$$

$$\iota : A_{0,0,z} = A_{0,0,z} \oplus RC_z.$$

In the above definitions, bit-wise XOR is denoted by "$\oplus$" and bit-wise logic AND by "$\cdot$". Besides, "$r_{x,y}$" refers to a lane-dependent rotation constant which equals the corresponding value in Table 3 taken modulo the lane length L. And "RC_z" is a round-dependent constant. The details about RC are omitted since they do not affect our attacks. For further details about KECCAK, please refer to [3].

	x=3	x=4	x=0	x=1	x=2
y=2	153	231	3	10	171
y=1	55	276	36	300	6
y=0	28	91	0	1	190
y=4	120	78	210	66	253
y=3	21	136	105	45	15

Table 3. The offsets of ρ.

2.3 Instances of KECCAK

The hash function KECCAK $[r, c, l]$ means an instance of KECCAK sponge function family with **capacity** c, **bitrate** r, and **output length** l. The official versions of KECCAK-l have $r = 1600-c$ and $c = 2{\cdot}l$, where $l \in \{128, 224, 256, 384, 512\}$. Their padding rules are identical. The message is padded by appending a bit string of "10*1", where "0*" means the shortest string of 0's such that the padded message is of multiple of r bits.

The digests of the standard SHA-3 have lengths of 224, 256, 384, and 512 bits. SHA-3 is similar to KECCAK except for the padding rule. SHA-3 pads the message with another two bits "01" before applying the KECCAK padding rule, *i.e.*, the padded string becomes "0110*1".

The SHA-3 family also includes two SHAKE instances (SHAKE128 and SHAKE256), which are called extendable-output functions (XOF's). Specifically, SHAKE128(M, l) and SHAKE256(M, l) are defined as KECCAK $[r = 1344, c = 256]$ and KECCAK $[r = 1088, c = 512]$. And the message M is padded with a suffix "1111".

In this paper, our attacks on instances of KECCAK, SHA-3 and SHAKE use the same parameters, and we focus on the instances with $l = 224/256$.

2.4 Notations

To find preimages for given hash values, we need to construct algebraic systems during the attacks. Some bits of the internal state will be set as **unknowns** and some are set as constants, where the unknowns are the bits that we need to solve. When some bits are set as unknowns in some state, bits in the consequent states can be represented as polynomials of these unknowns. For convenience, if a bit is represented as a linear polynomial of unknowns, we say **this bit is**

linear; similarly, we say **a bit is quadratic** if it is represented as a quadratic polynomial. Please note that the polynomial representation of a bit is unique in our attacks. Similarly, each column of a specific state contains 5 bits. We say **a column is linear** if all bits in this column are linear bits or constants.

We also give names to some states for sake of convenience. As the message is split into several **message blocks**, there are many hash blocks in the absorbing phase. We call the starting state of each hash block **the initial state** of this block, while the ending state of each block is called **the output state** or **outputs** of this block. Note that bits in the initial state of the first block are all 0's, and the output state of the i-th block is just the initial state of the $(i+1)$-th block. Particularly, the output state of the first block is also called **the middle state** in our attacks, since there are only two blocks in consideration. We call the state after XORing the initial state with the message **the messaged state**, and the state after the operation θ is called **the θ-diffused state**.

For notations, x, y, z refer to the axises of bits in states, and $a_{x,y,z}$, $d_{x,y,z}$, and $e_{x,y,z}$ are always used to represent bits in the messaged state, θ-diffused state, and output state of a hash block, respectively. We use i_j and o_j to denote input and output bits of the operation χ. The notation $s_{x,y}$ is used to represent the sum of a column in some state.

3 The allocating approach

With the motivation introduced in Section 1, we present the constraints on the middle state firstly by improving Li et al.'s structure [15]. Next, we show the details of the allocating approach.

3.1 Constraints on the middle state

In [15], Li et al. proposed a structure by setting some bits of the messaged state as unknowns and constants, such that the output bits after 2 forward rounds are *almost* all linear. However, there are still some nonlinear bits due to the constraints brought by the initial value. In this section, we improve this structure, and make *all* output bits linear after 2 forward rounds.

We start by studying the properties of the only nonlinear operator χ of each forward permutation. The operator χ can be regarded as a small S-box with 5 input and output bits, and the algebraic normal form of χ is

$$o_j = i_j \oplus (i_{j+1} \oplus 1) \cdot i_{j+2},\ j = 0, 1, 2, 3, 4,$$

where i_j and o_j are the j-th input and output bits. When building the algebraic systems, the input and output bits of χ are all represented as polynomials of unknowns and constants. Our goal is to find an input pattern such that: (1) the inputs contain as many linear bits as possible, i.e. the degrees of freedom need to be high; (2) the outputs contain as few non-constant bits as possible, i.e. the unknowns are not diffused much; (3) the outputs do not contain nonlinear bits.

By the requirement (1) and (3), we known that there are at most 2 linear bits in the inputs and these two linear bits must not be consecutive as well. Let x and c stand for the linear bit and constant bit, respectively. Then the input pattern is 'xcxcc', while other patterns satisfying (1) and (3) are all rotations of this one. Since each constant c could be 1 or 0, there are 8 possible cases of this input pattern. We list them and their corresponding outputs in Table 4.

inputs	x0x01	x0x00	x0x11	x1x01	x0x10	x1x00	x1x11	x1x10
outputs	x0x01	x0xx0	xxx11	x1x0x	xxxx0	x1xxx	xxx1x	xxxxx
#(linear output bits)	2	3	3	3	4	4	4	5

Table 4. Input and output bits of χ for the 'xcxcc' input pattern.

As shown in Table 4, only the input pattern 'x0x01' meets the requirement (2).

Based on the above study, we improve Li et al.'s structure in Lemma 1.

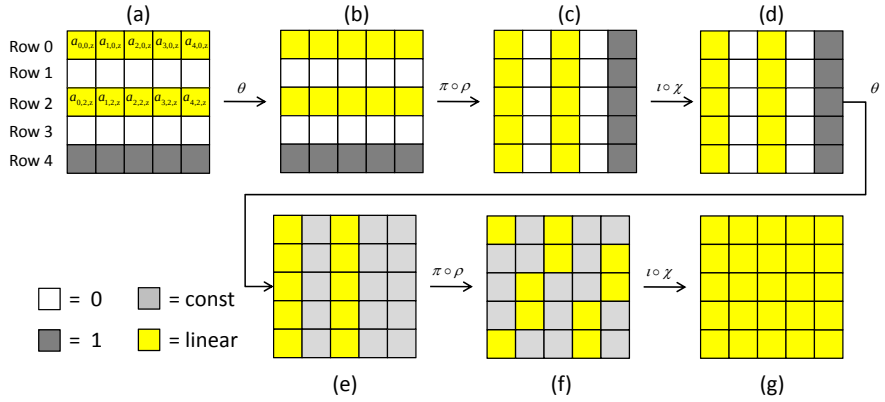

Fig. 4. The improved linear structure. Only one slice is shown, while the structures of other slices are the same.

Lemma 1. *Let the messaged state be (a) in Figure 4, i.e. bits in Row 0, 2 are unknowns, bits in Row 1, 3 are* 0*'s, and bits in Row 4 are* 1*'s. Then the* KECCAK-$f[1600]$ *permutation can be linearized up to 2 rounds with 194 degrees of freedom left.*

Proof. To avoid the propagation of unknowns after the θ operation, we assume that the bitwise sum of two columns is 0, i.e., $\bigoplus_{j=0}^{4} A_{x-1,j,z} + \bigoplus_{j=0}^{4} A_{x+1,j,z-1}$ =0 in state (a). In this way, after the operation θ, constant bits in state (a) are unchanged in state (b), but the linear bits in state (b) may be different from those

in state (a) by some constants. Initially, there are $10 \times 64 = 640$ unknowns, say $a_{x,0,z}$ and $a_{x,2,z}$. The sum assumption generates $5 \times 64 = 320$ linear constraints:

$$a_{x-1,0,z} \oplus a_{x-1,2,z} \oplus a_{x+1,0,z-1} \oplus a_{x+1,2,z-1} = 0, \text{ where } 0 \le x < 5, \text{ and } 0 \le z < 64.$$

And there is 1 constraint linear dependent on the other $320-1 = 319$ constraints. To show the linear dependence, we denote $p_{x,z} := a_{x-1,0,z} \oplus a_{x-1,2,z} \oplus a_{x+1,0,z-1} \oplus a_{x+1,2,z-1}$. Then we have

$$\bigoplus_{x,z} p_{x,z} = \bigoplus_{x,z} a_{x-1,0,z} \oplus \bigoplus_{x,z} a_{x-1,2,z} \oplus \bigoplus_{x,z} a_{x+1,0,z-1} \oplus \bigoplus_{x,z} a_{x+1,2,z-1}.$$

Since $\bigoplus_{x,z} a_{x-1,0,z} = \bigoplus_{x,z} a_{x+1,0,z-1}$ and $\bigoplus_{x,z} a_{x-1,2,z} = \bigoplus_{x,z} a_{x+1,2,z-1}$, we have $\bigoplus_{x,z} p_{x,z} = 0$, which means each $p_{x,z}$ equals the sum of the others. So after θ, there are $640 - 319 = 321$ degrees of freedom left. After ρ and π, each row of the state (c) has the pattern '$\mathsf{x0x01}$', and it is just the optimal case we studied above. So nonlinear bits are not generated and the unknowns are not diffused after the χ operation.

To keep the outputs of the second forward round linear, we also need to assume the sums of bits in Column 0 and 2 are constants in state (d), which produces $2 \times 64 = 128$ linear constraints and 1 of them is linear dependent on the other $128 - 1 = 127$ as well, while the proof is similar. Then there are $321 - 127 = 194$ degrees of freedom left. The consequent operations will not cost degrees of freedom, and all the bits in state (g) are linear.

In general, the layout of state (a) in Figure 4 is hard to meet, as it has rigid requirements on the values of constants. So we consider a more general case in Theorem 1.

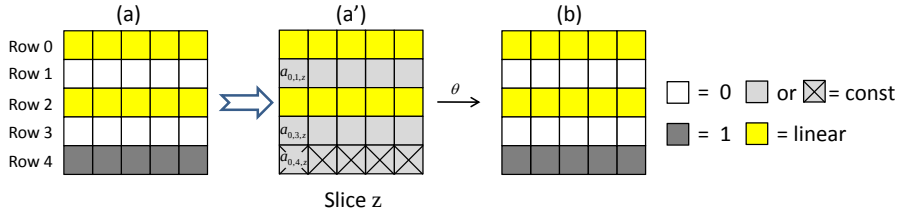

Fig. 5. Transforming state (a) to a more general case (a').

Theorem 1. *Let the messaged state be (a') in Figure 5, i.e. bits in Row 0, 2 are unknowns, and bits in Row 1, 3, 4 are constants such that*

(i) $a_{x,1,z} = a_{x,3,z} = a_{x,4,z} \oplus 1$, *and*
(ii) $\bigoplus_{x,z} a_{x,4,z} = 0$,

where $a_{x,y,z}$ *stands for the linear or constant bit at the position* (x, y, z), $0 \le x, y < 5$, *and* $0 \le z < 64$. *Then there exist constants* $s_{x,z}$*'s with* $0 \le x < 5$

and $0 \le z < 64$*, such that if assuming* $\bigoplus_{j=0}^{4} a_{x,j,z} = s_{x,z}$*, then the state (b) in Figure 5 can be obtained by operating* θ *on (a'). And hence, the* KECCAK-$f[1600]$ *permutation can be linearized up to 2 rounds with 194 degrees of freedom left.*

Proof. As introduced in Section 2.2, the operation θ is defined as:

$$\theta : d_{x,y,z} = a_{x,y,z} \oplus \bigoplus_{j=0}^{4} (a_{x-1,j,z} \oplus a_{x+1,j,z-1}) = a_{x,y,z} \oplus \bigoplus_{j=0}^{4} a_{x-1,j,z} \oplus \bigoplus_{j=0}^{4} a_{x+1,j,z-1},$$

where $d_{x,y,z}$ is a bit in the state (b) that is diffused from $a_{x,y,z}$'s. Let $s_{x,z} := \bigoplus_{j=0}^{4} a_{x,j,z}$. Then we have

$$d_{x,y,z} = a_{x,y,z} \oplus s_{x-1,z} \oplus s_{x+1,z-1}. \tag{1}$$

To ensure that the state (b) can be obtained after the operation θ, i.e. $d_{x,0,z}$ and $d_{x,2,z}$ are linear, $d_{x,1,z} = d_{x,3,z} = 0$, and $d_{x,4,z} = 1$, we only need to make the following equations hold by the condition $a_{x,1,z} = a_{x,3,z} = a_{x,4,z} \oplus 1$:

$$a_{x,4,z} \oplus s_{x-1,z} \oplus s_{x+1,z-1} = 1, \text{ where } 0 \le x < 5, 0 \le z < 64. \tag{2}$$

There are $5 \times 64 = 320$ equations in (2). All $a_{x,4,z}$ are constants. Regarding $s_{x,z}$ as symbols, the reduced Gröbner basis of the ideal $\langle a_{x,4,z} \oplus s_{x-1,z} \oplus s_{x+1,z-1} \oplus 1 \mid 0 \le x < 5, 0 \le z < 64 \rangle$ over $(GF(2)[a_{x,4,z}])[s_{x,z}]$ w.r.t. some lexicographic ordering on $\{s_{x,z}\}$ contains 320 polynomials. Among these polynomials, the only one that does not involve $\{s_{x,z}\}$ is $\bigoplus_{x,z} a_{x,4,z}$. By the properties of Gröbner bases, Equation (2) have solutions for $\{s_{x,z}\}$, if and only if $\bigoplus_{x,z} a_{x,4,z} = 0$. This is just the condition (ii), and the theorem is proved.

In fact, the above proof implies that the condition (i) and (ii) in Theorem 1 are also necessary conditions for the existence of $s_{x,z}$'s, if the messaged state is set as (a').

3.2 Preimage attacks with the allocating approach

For an instance of KECCAK-$f[1600]$, the number of bits in its internal state is 1600, which consists of two parts with r and c bits respectively. All 1600 bits are set as 0 initially. The first r bits need to XOR the message, and the last c bits remain 0's. The number c is the capacity of this instance.

By Theorem 1, if the capacity c of an instance of KECCAK-$f[1600]$ is smaller than $5 \times 64 = 320$, e.g. SHAKE128 whose capacity is 256, we can set the messaged state like (a') in Figure 5 by choosing the message carefully. Unfortunately, the capacities of most KECCAK instances are bigger than 320. This means the results in Section 3.1 cannot be used directly, because the number of 0's in the tail of the messaged state is more than 320 and the condition (i) does not hold.

Fortunately, the state (a') could serve as a good internal state in the allocating approach, where 2-block messages are considered. The outputs of the first block are not all 0's generally. We can adjust the values of the first r bits by choosing

the second message block carefully. To make the messaged state of the second block meet the conditions in Theorem 1, we also need some constraints on the last c bits in the output state of the first block.

Specifically, as shown in Figure 6, we consider the initial state (A) and messaged state (B) of the second block. The state (A) is the middle state in our attack, and it is also the output state of the first block.

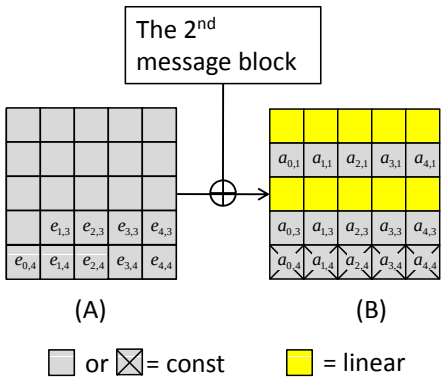

Fig. 6. The initial and messaged states of the second block of KECCAK-224/256. The axis z is omitted for simplification.

For KECCAK-224, its capacity is 448, which means the last 7 lanes of bits can not be changed after the second message block being XORed. So the last 7 lanes in the state (A) and (B) are identical. Since the values of the first 18 lanes in state (B) can be adjusted by the second message block[1], to make bits in the state (B) meet condition (i) and (ii) in Theorem 1, it suffices to ensure

$$e_{3,3,z} \oplus 1 = e_{3,4,z}, e_{4,3,z} \oplus 1 = e_{4,4,z}, \text{ and } \bigoplus_{x,z} e_{x,4,z} = 0, \tag{3}$$

which consists of $64 + 64 + 1 = 129$ equations.

The case for KECCAK-256 is similar, except that the last 8 lanes in the state (A) and (B) are identical. To make bits in the state (B) meet conditions in Theorem 1, we need the following $64 + 64 + 64 + 1 = 193$ equations hold:

$$e_{2,3,z} \oplus 1 = e_{2,4,z}, e_{3,3,z} \oplus 1 = e_{3,4,z}, e_{4,3,z} \oplus 1 = e_{4,4,z}, \text{ and } \bigoplus_{x,z} e_{x,4,z} = 0. \tag{4}$$

So in all, attacks on KECCAK-224/256 via the allocating approach consist of two stages:

1. Precomputation Stage: Find a first message block, such that Equation (3) or (4) hold for the output bits of the first block. Let C_1 be the complexity of finding this message block.

[1] The paddings will be dealt with sooner.

2. Online Stage: Construct an algebraic system using the structure in Theorem 1 for a given hash value, and solve this system for a second message block. The complexity of this stage is denoted as C_2.

We call the first stage "Precomputation Stage", because it does not need to be re-executed for different hash values, if a good first message block has been found.

Consequently, the complexity of the whole preimage attack is $C_1 + C_2$. Let $\bar{C}$ denote the complexity of finding a 1-block preimage, then we have $\max\{C_1, C_2\} < \bar{C}$. On one hand, since the numbers of equations in Equation (3) and (4) are 129 and 193, they are smaller than 224 and 256, which are the lengths of the hash values, respectively. Thus, we have $C_1 < \bar{C}$. On the other hand, if the last c bits in the initial state are set 0's, there is no way to linearize the first two rounds of KECCAK-f[1600] permutation of with 194 degrees of freedom left. So we can expect $C_2 < \bar{C}$.

The basic preimage attack via the allocating approach will be described in Section 4.1. Particularly, for the case $C_1 > C_2$, the complexity of the whole preimage attack can be made even lower. Because finding a first message block such that all equations in Equation (3) or (4) hold is usually harder than that if we allow some equations in Equation (3) or (4) *not to hold*. This means we can decrease the complexity of Precomputation Stage at the cost of increasing the complexity of Online Stage. Thus, the balanced complexity will be smaller than $\max\{C_1, C_2\}$. This balanced method will be given in Section 4.2, and it also leads to a practical preimage attack on 3-round KECCAK-224 in Section 5.

4 Theoretical results on round-reduced KECCAK-224/256

In this section, we use the allocating approach to attack KECCAK-224/256 that are reduced to 3 and 4 rounds. Theoretical complexities of these instances, as well as instances of SHA-3 and SHAKE, are given. The complexity is measured by the number of times for solving systems of linear equations, i.e., the complexity of solving a linear system is assumed to be a constant, which is the same as done in [12].

4.1 Attacks on 3-round KECCAK-224/256

In this section, we give detailed preimage attacks on 3-round KECCAK-224. Attacks on 3-round KECCAK-256 and instances of SHA-3 and SHAKE are similar.

Attacks on 3-round KECCAK-224 The attack consists of three parts. First, we find a first message block such that Equation (3) holds in Precomputation Stage. Second, we find a second message block such that the state (B) meets conditions in Theorem 1 and the outputs of the second block equal the given hash value in Online Stage. At last, we show how to deal with the paddings.

Part 1: finding a first message block

To find the first message block satisfying Equation (3), we use the structure presented by Guo et al. [12] in Figure 7, which keeps 2.5 rounds linear with 128 degrees of freedom left.

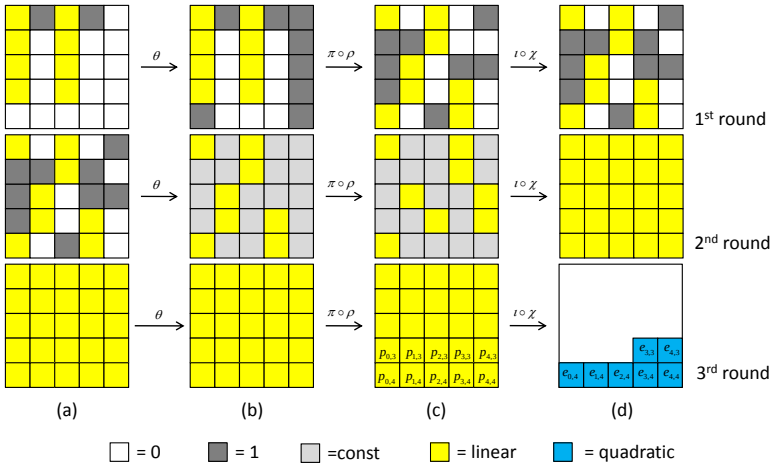

Fig. 7. The 3 forward rounds of the first block of KECCAK-224. The axis z is omitted for simplification.

In the messaged state of the first round, bits of 8 lanes are set as unknowns (shown in yellow boxes), which means there are $8 \times 64 = 512$ unknowns. White boxes and dark gray boxes in this state mean constant 0's and 1's. In the state (c) of the 3rd round, all bits are linear, and all of them become quadratic after the operation χ. During this procedure, to avoid the propagation by θ in the first and second rounds, $2 \times 64 + 4 \times 64 = 384$ linear constraints are added to the system by assuming sums of linear columns as constants. Here, a linear column refers to a column whose bits are linear or constant. By Equation (3), we obtain another $2 \times 64 + 1 = 129$ quadratic equations.

In all, we construct a system with $384 + 129 = 513$ equations in 512 unknowns. Although 384 equations are linear, this system is still not easy to solve in general. Fortunately, after noticing that there is only *one* quadratic term in each polynomial representation of $e_{i,j}$, we can enumerate 2 values of linear polynomials and obtain 4 linear equations like done in [15].

Specifically, let $p_{i,j}$ be the (linear) polynomial representation of bits in the state (c) of the 3rd round in Figure 7. By the χ operation, we have:

$$e_{3,4} = p_{3,4} \oplus (p_{4,4} \oplus 1) \cdot p_{0,4}, \;\; e_{4,4} = p_{4,4} \oplus (p_{0,4} \oplus 1) \cdot p_{1,4},$$

$$e_{3,3} = p_{3,3} \oplus (p_{4,3} \oplus 1) \cdot p_{0,3}, \;\; e_{4,3} = p_{4,3} \oplus (p_{0,3} \oplus 1) \cdot p_{1,3},$$

where the axis z is omitted for simplification. By Equation (3), we have the following equations:

$$p_{3,4} \oplus (p_{4,4} \oplus 1) \cdot p_{0,4} \oplus p_{3,3} \oplus (p_{4,3} \oplus 1) \cdot p_{0,3} = 1, \tag{5}$$

$$p_{4,4} \oplus (p_{0,4} \oplus 1) \cdot p_{1,4} \oplus p_{4,3} \oplus (p_{0,3} \oplus 1) \cdot p_{1,3} = 1. \tag{6}$$

If the values of the pair $(p_{0,3}, p_{0,4})$ are enumerated, then both Equation (5) and (6) are linearized in each slice. Together with equations from the enumeration, we obtain 4 linear equations totally.

Consequently, we enumerate the values of the pair $(p_{0,3}, p_{0,4})$ in 32 slices, and obtain 128 linear equations. The system consists of $384+(129-2\times32)+128 = 577$ equations in 512 unknowns, and $384 + 128 = 512$ of them are linear. With the same assumptions in [12], a solution to this system can be found in constant time.

Note that the original system consists of 513 equations in 512 unknowns, so the probability of the existence of a solution is regarded as 1/2. In case there is no solution to this system, we can vary the values of column sums in the state (a) of the 2nd round, and construct new systems. Besides, through the above procedure, we need to enumerate the values of $2 \times 32 = 64$ linear polynomials to solve the system. Therefore, the whole complexity of finding the first message block consists of two parts, the complexity 2^1 of ensuring the system has a solution, and the complexity 2^{64} of solving the system. The whole complexity of is $2^{1+64} = 2^{65}$.

Part 2: finding a second message block

By part 1, we obtain an initial state of the second block satisfying Equation (3). After setting bits in the second message block carefully, the messaged state, depicted in (a) of the 1st round in Figure 8, meets the conditions in Theorem 1. So there are 194 degrees of freedom left in the end of the 2nd round.

Bits in the initial state of the 3rd round are all linear, and after the linear operation θ, π, and ρ, bits in the state (c) of the 3rd round remain linear. On the other hand, KECCAK-224 generates a 224-bit hash value, which is supposed to be known for preimage attacks. Next, we construct relations between the bits before and after the operation χ. The relations are first studied in [12], and the operation ι is omitted here for simplification.

Let i_j and o_j be the input and output bit of χ. We have

$$o_j = i_j \oplus (i_{j+1} \oplus 1) \cdot i_{j+2}, \ j = 0, 1, 2, 3, 4,$$

by definition. Next, we can deduce

$$\begin{aligned} o_j &= i_j \oplus ((o_{j+1} \oplus (i_{j+2} \oplus 1) \cdot i_{j+3}) \oplus 1) \cdot i_{j+2} \\ &= i_j \oplus (o_{j+1} \oplus 1) \cdot i_{j+2} \oplus (i_{j+2} \oplus 1) \cdot i_{j+3} \cdot i_{j+2} \\ &= i_j \oplus (o_{j+1} \oplus 1) \cdot i_{j+2}. \end{aligned} \tag{7}$$

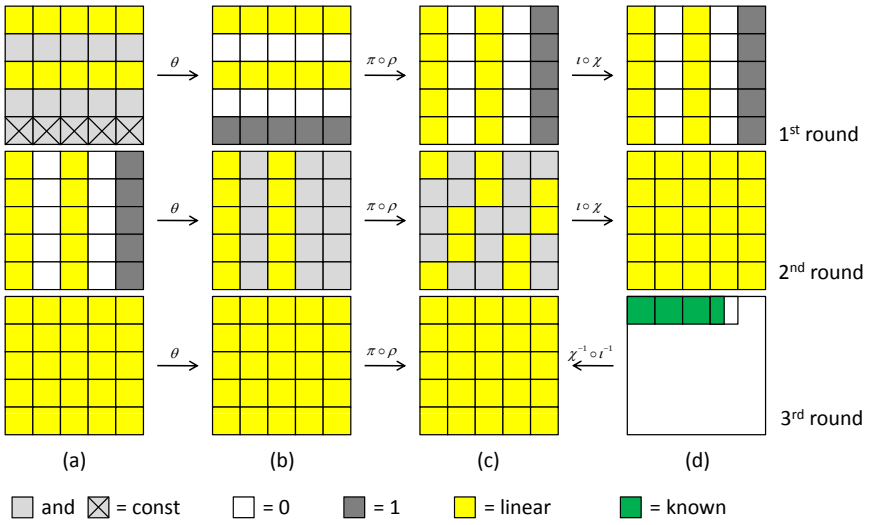

Fig. 8. The forward 3 rounds of the second block of KECCAK-224. The axis z is omitted for simplification.

Assume the i_j's are linear. If the values of 4 consecutive output bits are known, e. g. $o_0, \ldots, o_3$ are constants, then we have 3 linear equations

$$o_0 = i_0 \oplus (o_1 \oplus 1) \cdot i_2, o_1 = i_1 \oplus (o_2 \oplus 1) \cdot i_3, o_2 = i_2 \oplus (o_3 \oplus 1) \cdot i_4,$$

and 1 quadratic equation

$$o_3 = i_3 \oplus (i_4 \oplus 1) \cdot i_0. \tag{8}$$

Fortunately, Equation (8) can be simplified to linear as below.

$$i_0 = o_0 \oplus (o_1 \oplus 1) \cdot i_2 = o_0 \oplus (o_1 \oplus 1) \cdot (o_2 \oplus (o_3 \oplus 1) \cdot i_4) = A \oplus B \cdot i_4,$$

where $A = o_0 \oplus (o_1 \oplus 1) \cdot o_2$ and $B = (o_1 \oplus 1) \cdot (o_3 \oplus 1)$. Thus,

$$o_3 = i_3 \oplus (i_4 \oplus 1) \cdot i_0 = i_3 \oplus (i_4 \oplus 1) \cdot (A \oplus B \cdot i_4) = i_3 \oplus (i_4 \oplus 1) \cdot A.$$

Similarly, if the values of 3 consecutive output bits are known, say o_0, o_1, o_2, then we get 2 linear equations $o_0 = i_0 \oplus (o_1 \oplus 1) \cdot i_2$, $o_1 = i_1 \oplus (o_2 \oplus 1) \cdot i_3$, and a quadratic one

$$o_2 = i_2 \oplus (i_3 \oplus 1) \cdot i_4. \tag{9}$$

But this quadratic equation cannot be simplified.

In a digest of KECCAK-224, we have 4 consecutive bits in 32 slices, and 3 consecutive bits in the other 32 slices. Since the bits in the state (c) of the 3rd

round are linear, we can set up $(4+2)\times 32 = 192$ linear equations, and $1\times 32 = 32$ quadratic ones.

To sum up, bits of 10 lanes in the messaged state are unknowns. The number of unknowns is $10 \times 64 = 640$. To avoid the propagation by θ in the first and second rounds, $5\times 64+2\times 64 = 448$ linear constraints are added to the system by assuming that the bitewise sums of two linear columns are constants. Please note that there are 2 linear equations linear dependent on others, and the reason is shown in the proof of Lemma 1. We should also pay attention to that the values of these bitewise sums in the state (a) of the 1st round in Figure 8 must equal the values of $s_{x,z}$'s which are obtained from the proof of Theorem 1. But the sums of linear columns in the state (a) of the 2nd round could be set randomly.

Together with equations constructed by the hash value, the system has $448 - 2 + 192 + 32 = 670$ equations in 640 unknowns, and among these equations, $448 - 2 + 192 = 638$ are linear and linear independent on each other. To ensure this system has a solution, we need to enumerate $2^{670-640} = 2^{30}$ sum values of linear columns in the state (a) of the 2nd round. To solve the system, we only need to enumerate the values of a single bit i_3 in Equation (9), such that we can obtain 2 linear equations $i_3 = c$ and $o_2 = i_2 \oplus (c \oplus 1) \cdot i_4$, where c is the enumerated value of i_3. Then we get $638 + 2 = 640$ linear equations and the system can be solved within constant time. In all, the complexity that ensures the system has a solution is 2^{30}, and the complexity of solving the system is 2^1. The overall complexity is $2^{30+1} = 2^{31}$.

Part 3: dealing with paddings

For KECCAK-224, the last bit of the second message block must be 1 due to the padding rule. So to ensure that the messaged state of the second block meets the conditions in Theorem 1, we require $e_{2,3,63} \oplus 1 = e_{2,4,63} \oplus 1$, or equivalently

$$e_{2,3,63} = e_{2,4,63}. \tag{10}$$

This equation should be included in the system for finding the first message block. That is, we have 514 equations in 512 unknowns, and the probability of the existence of a solution is 1/4. The complexity for solving the first message block becomes 2^{66}.

The overall complexities of attacking 3-round KECCAK-224/SHA3-224

Summing up the analyses in the above three parts, the theoretical preimage attack on 3-round KECCAK-224 costs about 2^{66} operations. With similar analyses, the complexity of attacking 3-round SHA3-224 is 2^{69}.

Attacks on 3-round KECCAK-256 To find a first message block for 3-round KECCAK-256, the messaged state is set as (a) in Figure 9. And Equation (4) are considered. Following a similar procedure, bits in the output of the 2nd round have algebraic degree 1 at most.

To solve a first message block, we set $2 \times 64 + 3 \times 64 = 320$ linear equations by assuming the sums of linear columns in the state (a) and (d) are constants. Besides, there are $3 \times 64 + 1 = 193$ quadratic equations in Equation (4). The

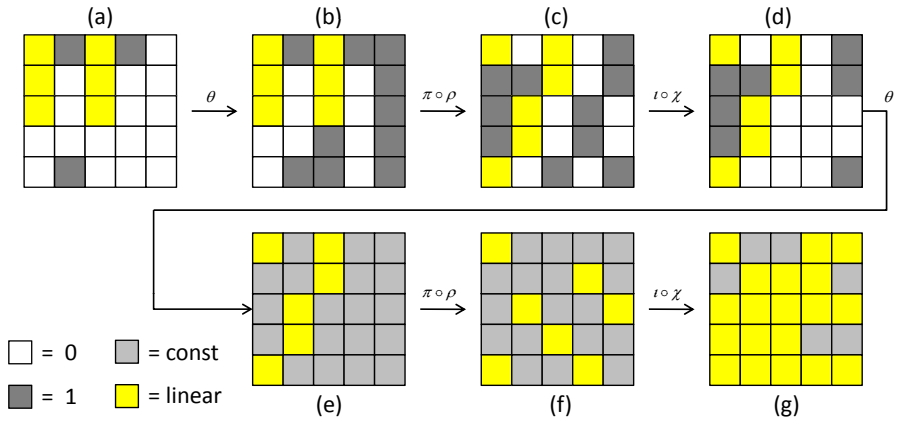

Fig. 9. The first message block of 3-round Keccak-256. The axis z is omitted for simplification.

number of unknowns is $6 \times 64 = 384$. So this system consists of $320 + 193 = 513$ equations in 384 unknowns, and the probability of the existence of a solution is $1/2^{513-384} = 1/2^{129}$. That is, we need to enumerate 2^{129} sum values of linear columns in the state (d) to ensure the system has a solution. To solve this system, similar to the case of Keccak-224, we need to enumerate the values of the pair $(p_{0,3}, p_{0,4})$ in 16 slices, and obtain 64 linear equations. Then we obtain $320 + 64 = 384$ linear equations, and the system can be solved with a constant time complexity. Thus, the complexity of finding a first message block is $2^{129+32} = 2^{161}$.

To solve a second message block, the procedure is the same as that of Keccak-224, except that we obtain $4 \times 64 = 256$ linear equations from the hash value. The system of this stage consists of $5 \times 64 + 2 \times 64 - 2 + 256 = 702$ linear equations in 640 knowns. We need to try $2^{702-640} = 2^{62}$ sum values of the linear columns in the state (a) of the 2nd round, to ensure there is a solution to this system. So the complexity of this stage is 2^{62}.

The overall complexities of attacking 3-round* Keccak*-256/SHA3-256/SHAKE256

After dealing with paddings for the instances of Keccak-256, SHA3-256, and SHAKE256, their attack complexities are 2^{162}, 2^{165}, and 2^{167}, respectively. Note that these results are not as good as those in [15].

4.2 Improved attacks on 3-round Keccak-224/256

In the attacks of Section 4.1, the complexity of Precomputation Stage is much higher than that of Online stage. One reason is that we require the state (B) in Figure 6 meets all conditions in Theorem 1. In fact, to obtain better attacks on

KECCAK-224/256, we can give up some constraints in Equation (3) or (4). In such a way, the complexity of finding the first message block will decrease at the cost of increasing the difficulty of finding the second message block. Then the complexities of the two stages will be balanced, and hence, the overall complexities of attacks will be improved.

Improved attacks on 3-round KECCAK-224 The improved attack contains two parts.

Part 1: finding a first message block

As discussed in Section 4.1, it costs 2^{66} operations to find a first message block such that all the $129 + 1$ equations in (3) and (10) hold. Consequently, the messaged state (B) in Figure 6 meets all conditions in Theorem 1, and hence, we can obtain the best complexity 2^{31} for the second block. Note that the complexity in the second stage is much lower than that in the first one. To improve the overall complexity of attacking KECCAK-224, we can balance the complexities of the two stages by allowing the messaged state (B) *to not* meet all conditions in Theorem 1. This means, not all equations in (3) and (10) must hold, which leads to a more efficient way of finding the first message block.

Since the 130 equations in (3) and (10) reflect the linear relations of the output bits of the first block, and the outputs of the first block can be regarded as random values, this means each of the 130 equations holds at the probability 1/2 in general. Thus, our strategy is that, we only include some of the equations in the system for solving, and expect the others to hold as many as possible.

Specifically, we divide the 130 equations into three sets. Set 1 contains the equations $\{e_{3,3,z} \oplus 1 = e_{3,4,z}, e_{4,3,z} \oplus 1 = e_{4,4,z}\}$ for 32 slices, and the equations that come from the other 32 slices together with Equation (10) are contained in Set 2. The equation $\{\bigoplus_{x,z} e_{x,4,z} = 0\}$ is regarded as Set 3. Then the $2 \times 32 = 64$ equations in Set 1 are included in the system for solving the first message block. We expect the $2 \times 32 + 1 = 65$ equations in Set 2 to hold as many as possible. And we do not care about whether the equation in Set 3 holds or not, because it does not affect the complexity of Online Stage.

Next, we estimate the complexity of finding a first message block. First, we have $2 \times 64 + 4 \times 64 = 384$ linear constraints by assigning the values of sums of linear columns. Second, we have $2 \times 32 = 64$ quadratic equations from Set 1. So there are $384 + 64 = 448$ equations in $8 \times 64 = 512$ unknowns. We have $512 - 448 = 64$ degrees of freedom left to solve this system. For the equations from Set 1, they are selected from 32 slices. Similar as discussed in Section 4.1, we can obtain 32 linear equations by guessing the values of 16 pairs $(p_{0,3}, p_{0,4})$, and get another 32 linear equations after the linearization of Equation (5) and (6). Then we get $384 + 64 + 64 = 512$ linear equations in 512 unknowns, which means we can obtain a solution to this system with a constant complexity. Note that when the values of $p_{0,3}$ and $p_{0,4}$ varies, solutions to the system change as well.

At last, we estimate how many operations are necessary to make as many equations in Set 2 hold as possible. In theoretical aspects, since each equation

holds with a probability 1/2, for any first message block, it makes n of the 65 equations hold with probability $\frac{C_{65}^n}{2^{65}}$. So the theoretical complexity of making n equations hold is $\frac{2^{65}}{C_{65}^n}$, and the complexities for different n's are shown in Table 5. In experimental aspects, we can obtain a lot of solutions of the systems by varying the values of $p_{0,3}$ and $p_{0,4}$. Then the number of messages that make n equations hold can be counted, and the practical probabilities are obtained as well. Complexities of practical attacks are reported in Section 5.1.

Part 2: finding a second message block

From Part 1, equations in Set 1 always hold, but some in Set 2 are not. Besides, we do not care about whether $\bigoplus_{x,z} e_{x,4,z} = 0$ holds or not. In this section, we deal with the troubles brought by these unsatisfied equations.

There are two types of unsatisfied equations for the first message block found in Part 1:

I. $e_{2,3,63} = e_{2,4,63}$, $e_{x,3,z} \oplus 1 = e_{x,4,z}$ where $x = 3$ or 4 for some z,
II. $\bigoplus_{x,z} e_{x,4,z} = 0$.

Our strategy is as follows. First, we construct an adjusted state by flipping the values of some bits in the unsatisfied equations and keeping others unchanged, such that all equations hold for the bits in the adjusted state. Second, like what we have done in the proof of Theorem 1, we can solve for the values of $s_{x,z}$'s such that the adjusted state transforms to the state (b) in Figure 5 after the operation θ. At last, we apply θ to the real outputs of the first block by assuming that the sums of columns are $s_{x,z}$'s. The obtained state will be different from the state (b) in Figure 5 only in a few bits, and hence, we only need to deal with these different bits afterwards.

Constructing the adjusted state and solving for $s_{x,z}$'s Assume there are n_I unsatisfied equations of Type I and $n_{II} \in \{0, 1\}$ unsatisfied one of Type II. We do not consider the case $n_I = 0$ since it does not happen in general cases. Our adjusting method only needs to flip n_I values of bits from the unsatisfied equations of Type I. Note that in each equation of Type I, there is one bit in Row 3 and one in Row 4. We totally flip n_{II} bit in Row 4 to ensure the sum of all bits in Row 4 of the adjusted state is 0, and flip the other $n_I - n_{II}$ bits in Row 3. Specifically, let $e'_{x,4,z} := e_{x,4,z} \oplus 1$ for n_{II} equation out of the n_I unsatisfied Type I equations, and let $e'_{x,3,z} := e_{x,3,z} \oplus 1$ from the other $n_I - n_{II}$ unsatisfied equations of Type I. Other bits are unchanged.

We illustrate the above method using a toy example with $n_I = 3$ and $n_{II} = 1$. Let $e_{x,y,z}$'s be the bits output by the first block for a given first message block, and we assume the 3 unsatisfied equations of Type I and the unsatisfied one of Type II are

$$e_{2,3,63} = e_{2,4,63} \oplus 1, e_{3,3,0} = e_{3,4,0}, e_{4,3,1} = e_{4,4,1}, \text{ and } \bigoplus_{x,z} e_{x,4,z} = 1.$$

Since $n_{II} = 1$, we let $e'_{3,4,0} := e_{3,4,0} \oplus 1$. For the other $n_I - n_{II} = 2$ bits, we set $e'_{2,3,63} := e_{2,3,63} \oplus 1$, $e'_{4,3,1} := e_{4,3,1} \oplus 1$. For the rest of bits, we have

$e'_{x,y,z} := e_{x,y,z}$. In this way, the bits $e'_{x,y,z}$'s construct the adjusted state and the equations in (3) and (10) all hold for $e'_{x,y,z}$'s. Then Equation (2) has solutions to $s_{x,z}$'s by the proof of Theorem 1. Thus, we find the desired values of $s_{x,z}$'s.

Dealing with the different bits in the θ-defused states For the original state consisting of $e_{x,y,z}$'s, let us see what happens to the state after the operation θ by assuming that the sums of columns equal the precomputed $s_{x,z}$'s. Let $d_{x,y,z}$'s be the bits in the state after the operation θ. By Equation (1), we have $d_{x,y,z} = e_{x,y,z} \oplus s_{x-1,z} \oplus s_{x+1,z-1}$. Since $s_{x,z}$'s are precomputed constants, the value of $d_{x,y,z}$ is determined by $e_{x,y,z}$. Note that there are only n_I $e_{x,y,z}$'s different from $e'_{x,y,z}$'s, so only n_I bits of the θ-defused states of the original and adjusted states are not identical, and the differences only lie in Row 3 or Row 4. Based on the row that the different bit appears, we consider two cases as shown in Figure 10.

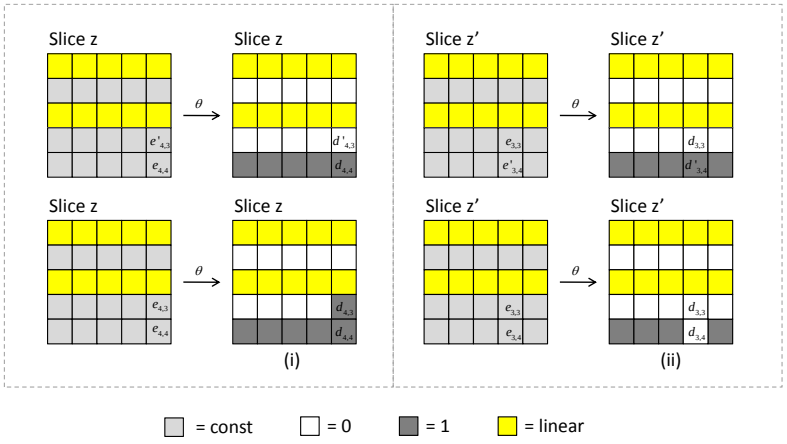

Fig. 10. The top row shows the adjusted states before and after the operation θ, and the bottom row shows the original states before and after θ. The state (i) and (ii) show two types of troubles.

Figure 11 shows how the trouble generated by the state (i) in Figure 10 is handled. The bit at (Row 3, Column 4, Slice z) of the θ-diffused state is 1 instead of 0, and it will produce two quadratic bits after two rounds. The method of eliminating this effect is that, we can enumerate the values of the bit in the orange box in the end of the first round, such that this bit becomes a constant and no quadratic bits are generated after two forward rounds. This enumeration costs 1 degree of freedom. Similarly, we can also handle the state (ii) at the cost of 1 degree of freedom in Figure 12.

To sum up, if n_I equations of Type I do not hold, the complexity of finding a second message block is 2^{31+n_I}.

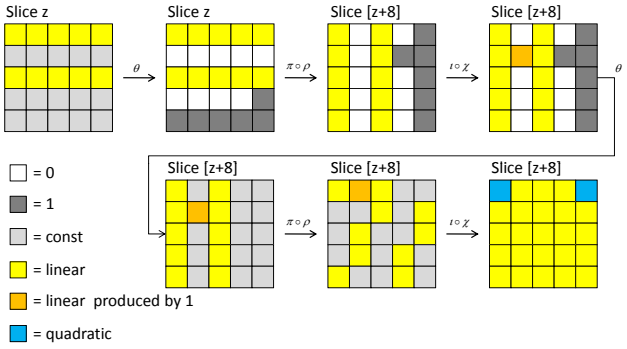

Fig. 11. The effects caused by the state (i) of Figure 10.

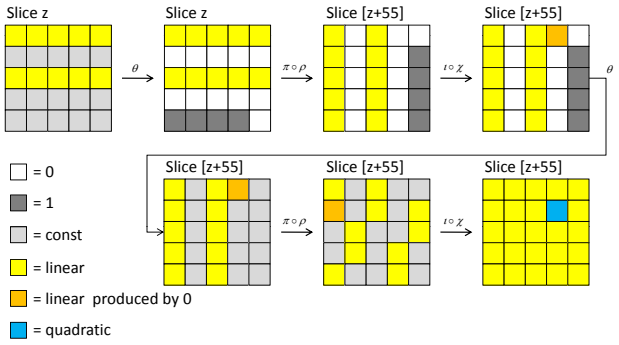

Fig. 12. The effects caused by the state (ii) of Figure 10.

The overall complexity of the improved preimage attacks on* KECCAK*-224/SHA3-224

With the improved attacks, we estimate the theoretical complexities in Table 5 for the cases that $n = 56, \ldots, 60$ equations from Set 2 hold. In this table, $n_I = 65-n$ is the number of equations that do not hold. C_1 and C_2 are the complexities of finding the first and second message blocks. The overall complexity is $C = C_1+C_2$. Please note that C_1 is estimated by probability, so its values are rounded. In Online Stage, the complexity C_2 is obtained by calculating degrees of freedom, so the values of C_2 are accurate integers.

Table 5 shows that we can obtain the best theoretical attack on 3-round KECCAK-224 with complexity 2^{38} when $n = 58$. Since this complexity is low enough, we perform a practical attack on 3-round KECCAK-224 in Section 5.

Similarly, the complexity of attacking SHA3-224 is at most 2^{41} considering padding bits.

n	n_I	C_1	C_2	$C = C_1 + C_2$
56	9	$2^{30.10}$	2^{40}	2^{40}
57	8	$2^{32.77}$	2^{39}	2^{39}
58	7	$2^{35.62}$	2^{38}	2^{38}
59	6	$2^{38.70}$	2^{37}	$2^{38.70}$
60	5	$2^{42.02}$	2^{36}	$2^{42.02}$

Table 5. Theoretical complexities of preimage attacks with $n = 56, \ldots, 60$, where n is the number of holding equations in Set 2.

Improved attacks on 3-round KECCAK-256 The improved preimage attack works on 3-round KECCAK-256 as well. There are 193 equations in Equation (4). The last 1 equation is in Set 3 and is not considered, so we hope 192 of these equations to hold, and we also need to consider 1 equation from the padding. Based on the theoretical probability, we can expect a first block message satisfying 174 out of 193 equations with complexity $2^{80.06}$. Note that among these 174 equations, 32 of them are included in the system for solving, i.e. in Set 1, and the other 142 equations hold with the probability $2^{-80.06}$. The complexity of Online Stage also increases to $2^{62+19} = 2^{81}$ in order to eliminate the impact of $193 - 174 = 19$ unsatisfied equations. Thus, the overall theoretical complexity of the preimage attack on 3-round KECCAK-256 is 2^{81}, while the previous best result is 2^{150} [15].

For SHA3-256 and SHAKE256, the differences lie in the padding rules. So extra computations are needed to find first message blocks. Using the same approach, the complexities of attacking 3-round SHA3-256/SHAKE256 are $2^{84}/2^{86}$.

4.3 Attacks on 4-round KECCAK-224/256

Attacks on 4-round KECCAK-224 The first message block for KECCAK-224 can be found by probability. Since a hash function outputs bits in a 'random' manner, the probability of finding a preimage by the random preimage attack is $1/2^l$, where l is the number of bits in digests [21]. The reason is that each output bit could be 0 or 1 with probability 1/2. The first message block are constrained by Equation (3). For the pair $(e_{3,3,z}, e_{3,4,z})$ in each slice, it has four possible values, and two of them make the equation $e_{3,3,z} \oplus e_{3,4,z} \oplus 1 = 0$ hold, which means this equation holds with a probability 1/2. The case is similar for the pair $(e_{4,3,z}, e_{4,4,z})$. As the value of $e_{x,4,z}$ is supposed to be random, the probability that $\bigoplus_{x,z} e_{x,4,z} = 0$ holds is also 1/2. Thus, the complexity of finding the desired first message block by random preimage attack is $2^{64+64+1} = 2^{129}$.

For the second block, as the messaged state meets conditions in Theorem 1, there are $(5 + 2) \times 64 - 2 = 446$ linear constraints in $10 \times 64 = 640$ unknowns after two forward rounds. The bits in the output state of the 3rd round are all quadratic and remain quadratic before the χ operation in the 4th round. Similarly to the analysis in Section 4.1, 224-bit digest leads to 192 quadratic equations and 32 quartic equations. Note that 160 out of the 192 quadratic equations are constructed by Equation (7). That is, $o_j = i_j \oplus (o_{j+1} \oplus 1) \cdot i_{j+2}$.

So if one bit o_{j+1} of the hash value is 1, then the quadratic equation becomes $o_j = i_j$ which has at most 11 quadratic terms as well. These equations are hard to solve, so the following analysis follows from the attacks on 4-round KECCAK-224 in [12]. By Guo et al.'s study, this quadratic equation can be linearized by guessing 10 values of linear polynomials. Figure 13 illustrates how to linearize this quadratic equation.

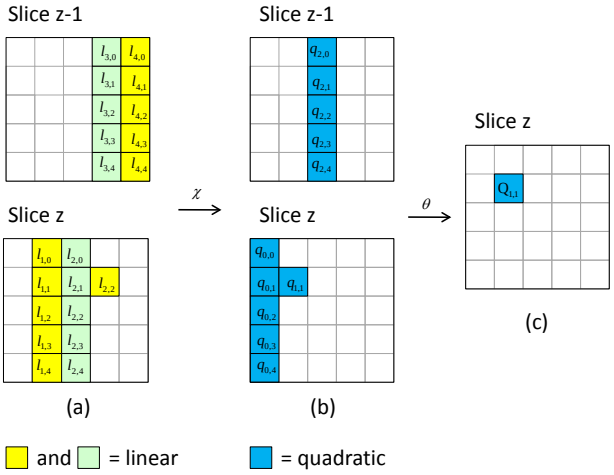

Fig. 13. Linearizing a quadratic bit by guessing the values of 10 linear polynomials.

For any quadratic bit in the state before the χ operation, it corresponds to a bit in the θ-defused state of the same round, since ρ and π are both permutations. Next, taking one quadratic bit $Q_{1,1,z}$ in the θ-defused state in the 4th round for example, we explain how to linearize $Q_{1,1,z}$ by guessing the values of 10 linear polynomials. By the definition of θ, we have $Q_{1,1,z} = q_{1,1,z} \oplus \bigoplus_{j=0}^{4}(q_{0,j,z} \oplus q_{2,j,z-1})$, where $q_{x,y,z}$ represents the output bit of the 3rd round and it is quadratic. Next, let us study how $q_{x,y,z}$ is generated by linear bits. From the states in Figure 8, the bits in the state before χ in the 3rd round are all linear, and we donate them as $l_{x,y,z}$'s. So we have $q_{x,y,z} = l_{x,y,z} \oplus (l_{x+1,y,z} \oplus 1) \cdot l_{x+2,y,z}$. Note that $q_{x,y,z}$ consists of only 1 quadratic term which is produced by two linear polynomials, $l_{x+1,y,z}$ and $l_{x+2,y,z}$. By guessing the values of either one, $q_{x,y,z}$ can be linearized. Another observation in Figure 13 is that, $q_{0,1,z}$ and $q_{1,1,z}$ share a common linear factor $l_{2,1,z}$. Thus, we can guess the values of 10 bits in the state (a), i.e. the light green bits, to linearize 11 blue quadratic bits in the state (b). Since $Q_{1,1,z}$ is represented by these blue bits, $Q_{1,1,z}$ is linearized as well. Consequently, we obtain 11 linear equations by enumerating the values of 10 linear polynomials.

As the values of bits in the digest can be regarded as random values, half of the 224 bits are supposed to be 1's. This means, we can expect 80 out of the 160 quadratic equations to have at most 11 quadratic terms. Next, we only consider

$\lfloor \frac{640-446}{11} \rfloor = 17$ quadratic equations from the above 80 ones, and leave the other $224 - 17 = 207$ equations hold by probabilities. By enumerating the values of $17 \times 10 = 170$ linear polynomials, we obtain $17 \times (10+1) = 187$ linear equations. To solve the system, we guess the values of another $640 - 446 - 187 = 7$ linear polynomials. Thus, the system consists of $446 + 187 + 7 = 640$ linear equations with 640 unknowns, so it has a solution and we can find it within constant time. Such a solution makes all the other $224-17 = 207$ equations hold at a probability 2^{-207}. So the complexity of the second block is 2^{207}.

Compared to Guo et al.'s attacks on 4-round KECCAK-224, our improvement is that, we obtain $640-446 = 194$ degrees of freedom after two forward rounds by using the allocating approach, while Guo et al. only get 127 degrees of freedom after the same rounds. So they only match $\lfloor \frac{127}{11} \rfloor = 11$ hash bits, and they need $2^{224-11} = 2^{213}$ operations to ensure all the other $224 - 11 = 213$ equations hold.

The paddings of KECCAK-224 and SHA3-224 only affect the complexity of the first block. So the complexities of attacking 4-round KECCAK-224 and SHA3-224 are both 2^{207}.

Attacks on 4-round KECCAK-256 The attacks on 4-round KECCAK-256 are quite similar. For the first message block, constraints become Equation (4). With the random preimage attack, the complexity of finding a first message block is $2^{3\times 64+1} = 2^{193}$. And we need $2^{256-\lfloor \frac{640-446}{11} \rfloor} = 2^{239}$ operations to find a second message block. So the complexities of attacking 4-round KECCAK-256, SHA3-256 and SHAKE256 are all 2^{239}.

5 Experiments

In this section, we give a practical preimage attack on 3-round KECCAK-224. Related codes, including those for verifying the found messages and for solving the systems on GPU, are available at *https://github.com/ysun0102/keccak224*.

5.1 Results of Precomputation Stage

We found about $2^{43.41}$ solutions with more than 10 NVIDIA GTX 1080 Ti cards in weeks. The numbers of solutions $\#(sol.)$ that make $n = 50, \ldots, 60$ equations hold are reported in Table 6, together with the practical and theoretical probabilities.

From the table, we can see that the theoretical and practical probabilities match very well when the number of solutions is large enough, and there is only a bit of difference between the theoretical and practical probabilities of $n = 58, 60$. This is mainly because the samples are not adequate. The results in hexadecimal format are given below.

First message block with $n = 58$:

```
|3867ED3B88A48506|FFFFFFFFFFFFFFFF|DD2D9BE5549AE517|FFFFFFFFFFFFFFFF|
|0000000000000000|97CBA3B4524267F6|0000000000000000|F607605E0D17724B|
```

n	$\#(sol.)$	Practical probability	Theoretical probability
50	65 469 825	$2^{-17.44}$	$2^{-17.44}$
51	19 262 179	$2^{-19.21}$	$2^{-19.21}$
52	5 185 994	$2^{-21.10}$	$2^{-21.10}$
53	1 271 108	$2^{-23.13}$	$2^{-23.13}$
54	281 252	$2^{-25.30}$	$2^{-25.30}$
55	56 771	$2^{-27.61}$	$2^{-27.62}$
56	9 986	$2^{-30.12}$	$2^{-30.10}$
57	1 591	$2^{-32.77}$	$2^{-32.77}$
58	227	$2^{-35.58}$	$2^{-35.63}$
59	26	$2^{-38.70}$	$2^{-38.70}$
60	2	$2^{-42.41}$	$2^{-42.02}$

Table 6. Comparisons of practical and theoretical probabilities

0000000000000000	0000000000000000	59E591785BB04788	0000000000000000
87A44FB877A61A6E	0000000000000000	0000000000000000	F649DFF78156A578
0000000000000000	AC8EB4032E2B8D32

First message block with $n = 59$:

35EF68DC35F1E5EB	FFFFFFFFFFFFFFFF	A1D249A40996BB5F	FFFFFFFFFFFFFFFF
0000000000000000	42F30B16705F6ECA	0000000000000000	26A9E432AE324F66
0000000000000000	0000000000000000	BBB37F56A6F28967	0000000000000000
A9590D7698444C80	0000000000000000	0000000000000000	CCAF1C9CE35C0246
0000000000000000	2E22A0E03FE0B8B9

First message block with $n = 60$:

CBB53657E0A66871	FFFFFFFFFFFFFFFF	7537C0597B751AA7	FFFFFFFFFFFFFFFF
0000000000000000	A27C4639BB60DFF0	0000000000000000	561C6D11D6A8DE58
0000000000000000	0000000000000000	22DF18C837CF65DB	0000000000000000
37C8309A24DD20E7	0000000000000000	0000000000000000	4B1668A66C09D25A
0000000000000000	14E39DD28900E418

5.2 Results of preimage attacks on 3-round Keccak-224

Example 1. Let the hash algorithm be 3-round Keccak-224. Find a second preimage for the message '1' with $length = 1$.

The padded message M and its digest H are given below.

$M(length = 1152)$:

0000000000000003	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	8000000000000000

$H(length = 224)$:

|F4FE7CCEA5D8B144|60F6C316572983A8|A2564CA289E5F897| CA30DB85|

Using the methods in Section 4.2, we find three second preimages M_{58}, M_{59}, M_{60} of H based on the first message blocks computed in the last subsection. When $n = 58$, finding the second message block takes a week on 6 GPU cards with approximate $2^{39.39}$ operations, while the second message block of M_{59} costs 3 days on 6 GPU cards by solving about $2^{38.20}$ linear systems. The second message block of M_{60} is obtained in 2 days using 4 GPU cards. Note that these practical complexities of Online Stage are all larger than those estimated in Table 5. We think this is because we only calculate one second message block for each of them. Interested readers can generate more preimages with the published codes, and we believe the complexities will be reasonable then.

From Table 6, the averaged complexities of Precomputation Stage for finding the first message blocks of $n = 58, 59, 60$ are $2^{35.58}$, $2^{38.70}$, and $2^{42.41}$, respectively. So the overall practical complexities of finding M_{58}, M_{59}, and M_{60}, are $2^{39.39}$, $2^{39.47}$, and $2^{42.41}$. However, since the first message blocks need to be computed only once, we suggest using the first message block of $n = 60$, if we want to produce more preimages of H or other digests. The values of M_{58}, M_{59}, M_{60} are listed below.

- $M_{58}(length = 2301)$:
3867ED3B88A48506	FFFFFFFFFFFFFFFF	DD2D9BE5549AE517	FFFFFFFFFFFFFFFF
0000000000000000	97CBA3B4524267F6	0000000000000000	F607605E0D17724B
0000000000000000	0000000000000000	59E591785BB04788	0000000000000000
87A44FB877A61A6E	0000000000000000	0000000000000000	F649DFF78156A578
0000000000000000	AC8EB4032E2B8D32
C84C8045515BF0C7	61FD4B2BBE00140E	00B252887E479E1D	4CA8454ECB4032EC
0980778FEAFC137D	1B4109C0E732BD96	820D1264F56CED03	E3A15B12575B72A2
1A068D85C2B37FE0	5DCA726A8F294970	D41129BE08A68BD4	301DD29F5E9BE657
98A7904810694A48	B3E8566CE50EA6AA	48C3E4DEB3ADD02B	853EF9C96DC6F02D
A72B40AD1F31A630	AAD47114F4750BFC
- $M_{59}(length = 2302)$:
35EF68DC35F1E5EB	FFFFFFFFFFFFFFFF	A1D249A40996BB5F	FFFFFFFFFFFFFFFF
0000000000000000	42F30B16705F6ECA	0000000000000000	26A9E432AE324F66
0000000000000000	0000000000000000	BBB37F56A6F28967	0000000000000000
A9590D7698444C80	0000000000000000	0000000000000000	CCAF1C9CE35C0246
0000000000000000	2E22A0E03FE0B8B9
CCADB05484618913	CB72585A10CF1D24	5142B6082D69F648	55FF802052E9AFA7
5002434225118309	4673F9FF53CF4651	422091CBEE6ED26C	2CED676FB523B95D
AF5FD173FA98BE32	1BB7489625D2A58A	1B58D9FB91AD563D	D2F304B902CD182E
9F519823A0C16E4D	A54F438AFE22755C	8C39E80475FCDBB0	B908F9B8CD448A94
63EF7F66EA21A245	D0A64F63C7333027
- $M_{60}(length = 2302)$:
CBB53657E0A66871	FFFFFFFFFFFFFFFF	7537C0597B751AA7	FFFFFFFFFFFFFFFF
0000000000000000	A27C4639BB60DFF0	0000000000000000	561C6D11D6A8DE58
0000000000000000	0000000000000000	22DF18C837CF65DB	0000000000000000
37C8309A24DD20E7	0000000000000000	0000000000000000	4B1668A66C09D25A
0000000000000000	14E39DD28900E418

```
|B5B27127B16157CE|1D9CDF75F80E635D|D2024BC09980F06E|43E0D61A974E2162|
|D3E8E4C133283C19|291ADC10C38952D3|0D79C02584D59EB5|5B6EDBF95FBBD637|
|FDF01822DC1C43A3|516EB953B657C03F|8C83A4CFE46AFA61|8EF91ECCAD2D5731|
|3510F4267D8A4D55|13A2BACDCE43348D|0A22C2B955093C72|8836257614188A4E|
|AFBB582F7829B0EB|6CF33EA53BEC3299|
```

6 Conclusion

In this paper, we propose preimage attacks with an allocating approach. We improved the attacks on Keccak-224/SHA3-224 and Keccak-256/SHA3-256 /SHAKE256 that are reduced to 3 and 4 rounds. The main idea is to divide the attacking procedure into two stages and to find a 2-block preimage, such that the complexity of each stage is lower than that of finding a 1-block preimage directly. The key step is that, the conditions in Theorem 1 have fewer constraints on the middle state, such that we obtain more degrees of freedom to solve for the first and second message blocks. This is why we obtain better results, compared with previous attacks.

References

1. Aumasson, J., Meier, W.: Zero-sum distinguishers for reduced keccak-f and for the core functions of luffa and hamsi. https://131002.net/data/papers/AM09.pdf (2009)
2. Bernstein, D.: Second preimages for 6(7?(8??)) rounds of keccak. In: NIST mailing list (2010)
3. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The keccak reference, version 3.0. In: https://keccak.team/keccak.html (2011)
4. Chaigneau, C., Fuhr, T., Gilbert, H., Guo, J., Jean, J., Reinhard, J.R., Song, L.: Key-recovery attacks on full kravatte. IACR Trans. Symmetric Cryptol. **2018**, 5–28 (2018). https://doi.org/10.13154/tosc.v2018.i1.5-28, https://tosc.iacr.org/index.php/ToSC/article/view/842
5. Chang, D., Kumar, A., Morawiecki, P., Sanadhya, S.: 1st and 2nd preimage attacks on 7, 8 and 9 rounds of keccak-224,256,384,512. In: SHA-3 Workshop (2014)
6. Daemen, J., Van Assche, G.: Differential propagation analysis of keccak. In: Canteaut, A. (ed.) Fast Software Encryption. pp. 422–441. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
7. Dinur, I., Dunkelman, O., Shamir, A.: New attacks on keccak-224 and keccak-256. In: Fast Software Encryption: 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. pp. 442–461 (2012)
8. Dinur, I., Dunkelman, O., Shamir, A.: Collision attacks on up to 5 rounds of sha-3 using generalized internal differentials. In: Fast Software Encryption: 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. pp. 219–240 (2013)
9. Dinur, I., Dunkelman, O., Shamir, A.: Improved practical attacks on round-reduced keccak. J. Cryptol. **27**(2), 183–209 (Apr 2014)

10. Dinur, I., Morawiecki, P., Pieprzyk, J., Srebrny, M., Straus, M.: Cube attacks and cube-attack-like cryptanalysis on the round-reduced keccak sponge function. In: Advances in Cryptology – EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. pp. 733–761. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
11. Dinur, I., Morawiecki, P.L., Pieprzyk, J., Srebrny, M., Straus, M.L.: Practical complexity cube attacks on round-reduced keccak sponge function. IACR Cryptology ePrint Archive **2014**, 259 (2014)
12. Guo, J., Liu, M., Song, L.: Linear structures: Applications to cryptanalysis of round-reduced keccak. In: Advances in Cryptology – ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. pp. 249–274. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
13. Huang, S., Wang, X., Xu, G., Wang, M., Zhao, J.: Conditional cube attack on reduced-round keccak sponge function. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 259–288. Springer International Publishing, Cham (2017)
14. Kölbl, S., Mendel, F., Nad, T., Schläffer, M.: Differential cryptanalysis of keccak variants. In: Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings. pp. 141–157 (2013). https://doi.org/10.1007/978-3-642-45239-0_9, https://doi.org/10.1007/978-3-642-45239-0_9
15. Li, T., Sun, Y., Liao, M., Wang, D.: Preimage attacks on the round-reduced keccak with cross-linear structures. IACR Trans. Symmetric Cryptol. **2017**, 39–57 (2017)
16. Li, Z., Bi, W., Dong, X., Wang, X.: Improved conditional cube attacks on keccak keyed modes with milp method. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 99–127. Springer International Publishing, Cham (2017)
17. Morawiecki, P., Pieprzyk, J., M. Srebrny, M.: Rotational cryptanalysis of round-reduced keccak. In: Fast Software Encryption: 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. pp. 241–262 (2013)
18. Morawiecki, P., Srebrny, M.: A sat-based preimage analysis of reduced keccak hash functions. Inf. Process. Lett. **113**(10-11), 392–397 (2013)
19. Naya-Plasencia, M., Rck, A., Meier, W.: Practical analysis of reduced-round keccak. In: International Conference on Cryptology in India. pp. 236–254 (2011)
20. NIST: Sha-3 competition. In: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html (2007-2012)
21. Preneel, B.: The state of cryptographic hash functions. Lecture Notes in Computer Science **1561**, 158–182 (1999)
22. Qiao, K., Song, L., Liu, M., Guo, J.: New collision attacks on round-reduced keccak. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III. pp. 216–243 (2017). https://doi.org/10.1007/978-3-319-56617-7_8, https://doi.org/10.1007/978-3-319-56617-7_8
23. Song, L., Liao, G., Guo, J.: Non-full sbox linearization: Applications to collision attacks on round-reduced keccak. In: Advances in Cryptology – CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part II. pp. 428–451. Springer International Publishing, Cham (2017)

24. Song, L., Guo, J., Shi, D.: New milp modeling: Improved conditional cube attacks to keccak-based constructions. IACR Cryptology ePrint Archive **2017**, 1030 (2017)
25. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full sha-1. In: Advances in Cryptology – CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017. pp. 570–596. Springer International Publishing, Cham (2017)
26. The U.S. National Institute of Standards and Technology Technology: Sha-3 standard: Permutation-based hash and extendable-output functions. In: Federal Information Processing Standard, FIPS 202 (2015), http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
27. Wang, X., Yu, H.: How to break md5 and other hash functions. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005. pp. 19–35. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)