Preimage Attacks on the Round-reduced Keccak with Cross-linear Structures

Ting Li[1,2], Yao Sun[1†], Maodong Liao[3,4] and Dingkang Wang[3,4]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China.
[3] Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China.
[4] School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing, China.
sunyao@iie.ac.cn

Abstract. In this paper, based on the work pioneered by Aumasson and Meier, Dinur et al., and Guo et al., we construct some new delicate structures from the round-reduced versions of KECCAKhash function family. The new constructed structures are called *cross-linear structures*, because linear polynomials appear across in different equations of these structures. And we apply cross-linear structures to do preimage attacks on some instances of the round-reduced KECCAK. There are three main contributions in this paper. First, we construct a kind of cross-linear structures by setting the statuses carefully. With these cross-linear structures, guessing the value of one linear polynomial could lead to three linear equations (including the guessed one). Second, for some special cases, *e.g.* the 3-round KECCAKchallenge instance KECCAK$[r = 240, c = 160, n_r = 3]$, a more special kind of cross-linear structures is constructed, and these structures can be used to obtain seven linear equations (including the guessed) if the values of two linear polynomials are guessed. Third, as applications of the cross-linear structures, we practically found a preimage for the 3-round KECCAKChallenge instance KECCAK$[r = 240, c = 160, n_r = 3]$. Besides, by constructing similar cross-linear structures, the complexity of the preimage attack on 3-round KECCAK-256/SHA3-256/SHAKE256 can be lowered to $2^{150}/2^{151}/2^{153}$ operations, while the previous best known result on KECCAK-256 is 2^{192}.

Keywords: Cryptanalysis · SHA-3 · Keccak · Preimage attacks

1 Introduction

The KECCAKsponge function family [BDPA11] is designed by Bertoni et al. and became a candidate of the SHA-3 competition [NIS] in 2008. It won this competition in 2012, and the U.S. National Institute of Standards and Technology (NIST) standardized KECCAKas Secure Hash Algorithm-3 [oST15] (SHA-3) in 2015. KECCAKhas received numerous security analysis since it was public available in 2008.

On practical collision attacks, Dinur et al. obtained practical complexities up to 4 out of 24 rounds of KECCAK-224/256 [DDS12, DDS14]. Recently, Song et al. proposed a practical collision attack on KECCAK-224 reduced to 5 rounds [SLG17]. Dinur et al. also presented theoretical complexities up to 5 rounds of KECCAK-256 [DDS13]. On practical preimage attacks, Naya-Plasencia et al. [NPRM11] and Morawiecki et al. [MS13] had presented attacks up to 2 rounds. Guo et al. [GLS16] developed the technique of linear structures,

†Corresponding author.

IACR Transactions on Symmetric Cryptology ISSN 2519-173X, Vol. 2017, No. 4, pp. 39–57
DOI:10.13154/tosc.v2017.i4.39-57

and presented a practical attack to SHAKE128. Besides, theoretical preimage attacks on KECCAKreduced to 4 rounds were also given by them. Theoretical preimage attacks up to 7/8/9 rounds on KECCAK-224/256/512 are considered in [Ber10, CKMS14, MPS13].

To encourage the practical analysis of the KECCAKsponge functions, the KECCAKteam organized the "KECCAKCrunchy Crypto Collision and Preimage Contest" (KECCAKChallenge for short) and presented challenges for reduced-round KECCAKinstances, namely KECCAK$[c = 160, r = b - c]$ with $b \geq 200$ [BDPA], where c is the capacity and b is the width. The capacities of the challenges are fixed to 160 bits, which implies a security level of 2^{80} against generic collision search. The width b of KECCAK-$f[b]$ is in $\{200, 400, 800, 1600\}$. The width values support the chosen capacity. Up to now, the best preimage solution was on 4 rounds and was submitted by Liu and Guo in December 2016, and the best collision was on 6 rounds and was submitted by Song et al. in February 2017 [SLG17]. The KECCAKteam summaries the challenge status as "Remarkably, the smaller versions are harder to break. Although they have a smaller state, they offer much less degrees of freedom, especially relative to the capacity that is the same for all versions."

The traditional method of finding a preimage for a given hashing value is usually based on solving a polynomial system generated by the hashing algorithm and hashing value. However, this system is often difficult to solve, due to the large number of unknowns and the high nonlinearity of polynomials. The Gröbner basis method is an important tool for solving non-linear polynomial systems, and has been extensively studied [Fau99, Fau02, GIW16, SW11, YSL16]. But the Gröbner basis method is not capable to solve non-linear systems in such large size, so SAT solvers and linear techniques are more popular for attacking these systems. At present, lower rounds, for instance 1 or 2 rounds, of the KECCAKChallenge instances are solved directly by SAT solvers [MS13]. Higher rounds of instances cannot be solved for 5 years, until Guo et al. found the *linear structures* of the systems. Guo et al. linearized the polynomial system at most 2 forward rounds and 1 backward round by decreasing the degrees of freedom. So they successfully found the preimages for the challenge instances KECCAK$[r = 640, c = 160, n_r = 3]$ and KECCAK$[r = 1440, c = 160, n_r = 3]$, which have 640 and 1440 degrees of freedom respectively, where n_r is the round of this instance. They also extended their method to find a preimage for the instance KECCAK$[r = 1440, c = 160, n_r = 4]$, which is the best result up to now.

Just as summarized by the KECCAKteam, the instance KECCAK$[r = 240, c = 160, n_r = 3]$ is more difficult since it has much fewer degrees (240) of freedom, so it cannot be linearized by Guo et al.'s method. We focus on attacking this instance in this paper, because it has more similar initial status with KECCAK-256/SHAKE 256/SHA3-256 than the instance KECCAK$[r = 1440, c = 160, n_r = 3]$. Figure 1 shows the initial statuses of the three instances (a) KECCAK$[r = 240, c = 160]$, (b) KECCAK$[r = 1440, c = 160]$, and (c) KECCAK$[r = 1088, c = 512]$, respectively. From this figure, we can see that, although

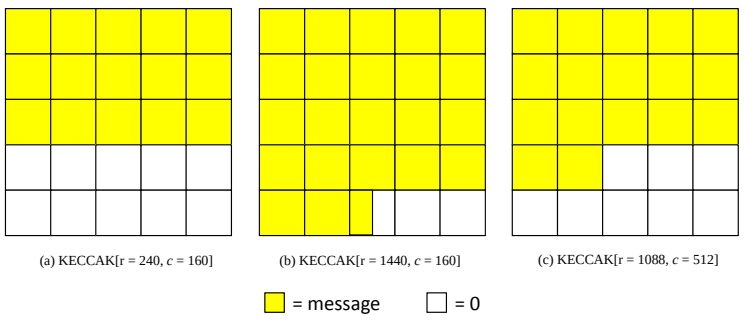

Figure 1: Initial statuses of three instances, represented by slices.

KECCAK$[r = 1440, c = 160]$ and KECCAK$[r = 1088, c = 512]$ has the same width, say 1600, the capacity of KECCAK$[r = 1088, c = 512]$ is 512. This means 8 lanes of the initial status are 0, which is quite similar to the instance KECCAK$[r = 240, c = 160]$ where 10 lanes are 0. Our experiments verified the above observation, since the methods used to attack KECCAK$[r = 240, c = 160, n_r = 3]$ can be directly used to obtain a better attack complexity of 3-round KECCAK$[r = 1088, c = 512]$/SHAKE 256/SHA3-256.

The **contributions** of this paper are summarized in three aspects:

1. We construct a new kind of structures for the KECCAKinstances. In these structures, linear polynomials exist across different polynomials, so we call these structures as *cross-linear structures*. Using these cross-linear structures, more linear equations can be obtained by enumerating some values of polynomials. Specifically, by setting the statuses of the instances carefully, we can formulate the preimage problem as quadratic polynomial equations that have two important properties. First, the quadratic part of each constructed polynomial equation consists of only one product of two linear polynomials. For sake of convenience, we call these linear polynomials as *cross-linear factors* of the quadratic parts. Second, the same cross-linear factor always appears across (at least) two different polynomials. For this system, if the values of k cross-linear factors are guessed, then $3 \cdot k$ linear equations (including the guessed one) are obtained. For simplicity, we call this kind of cross-linear structures as "$1 \to 3$" cross-linear structures. Using "$1 \to 3$" cross-linear structures, a preimage of the instance KECCAK$[r = 240, c = 160, n_r = 3]$ can be found in 2^{48} operations.

2. For some special instances, *e.g.* the KECCAKChallenge instance KECCAK$[r = 240, c = 160, n_r = 3]$, a more special kind of cross-linear structures is constructed. Because the operation ρ of the KECCAK-permutation fortunately generates some particular differences of the subscripts of unknowns, more relations can be built. Thus, by guessing the values of two correlative cross-linear factors, we can obtain 7 linear equations (including the guessed). The "$2 \to 7$" cross-linear structures enable us to find a preimage of the instance KECCAK$[r = 240, c = 160, n_r = 3]$ in 2^{45} operations.

3. We did a practical preimage attack on the 3-round instance KECCAK$[r = 240, c = 160, n_r = 3]$ in the KECCAKChallenge. We formulated 80 quadratic polynomials in 80 unknowns from the preimage problem by guessing the values of 31 linear polynomials. We guessed the values of another 14 linear polynomials to get 49 linear equations totally by using the "$2 \to 7$" cross-linear structures. Then we obtained 80 linear equations in 80 unknowns. Even if there exists a solution to this linear system, it may not be the *true* solution of the formulated system. At last, we verified the solution by the original 80 polynomials. Totally, the attack costs 5 days with 8 GPU cards, and we finally found a true solution as well as a preimage to this KECCAKinstance.

   A solution for the 3-round preimage challenge with width 400 is listed as below, where the length of the message is 238 bits and the preimage is expressed in hexadecimal.

   Image of the challenge: 5c 9d 5e 4b 38 5e 9c 4f 8e 2e.

   Preimage: 53 73 e0 75 3d ec af 5b 2e c1 00 00 00 00 00 00 00 00 00 00 53 73 e0 75 3d ec af 5b 2e c1.

   By using the similar attack done on the KECCAKChallenge instance and dealing with the paddings carefully, we obtained new theoretical preimage complexities, say $2^{150}/2^{151}/2^{153}$ operations, on 3-round KECCAK-256/SHA3-256/SHAKE256, while the previous known best result on KECCAK-256 is 2^{192} [GLS16].

This paper is organized as follows. Some preliminaries and notations are given in Sec. 2. Our main results are presented in Sec. 3. We conclude this paper in Sec. 4.

2 Preliminaries

2.1 The sponge construction

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

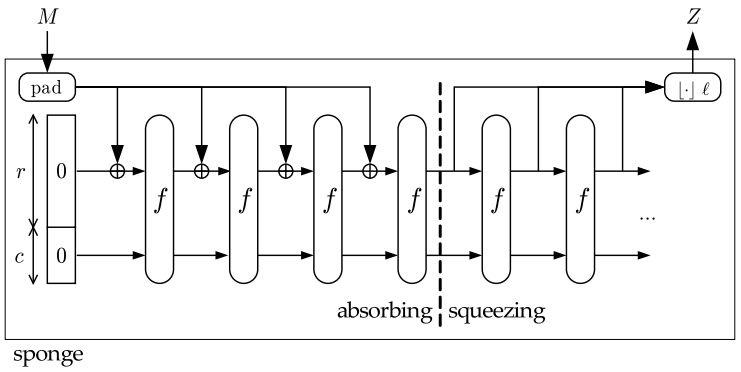

Figure 2: The sponge construction.

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 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 squeezing phase, the first r bits are outputted. With an additional application of f, another r output bits are obtained. The algorithm iterates this step until the required number of digest bits are all produced.

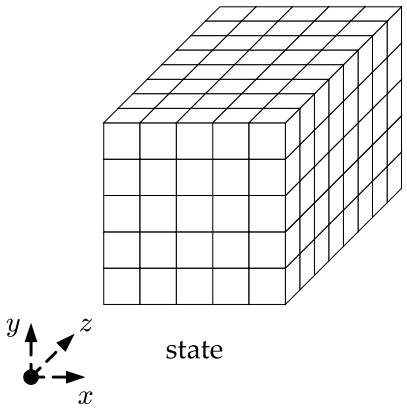

Figure 3: The KECCAKstate.

2.2 The Keccak-f permutations

According to the KECCAKreference [BDPA11], 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. Usually, KECCAK-$f[1600]$ are 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 bit length of lanes. 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 and y are from the set $\{0, 1, 2, 3, 4\}$ while $0 \leq z \leq L - 1$. Expressions in x and y are taken modulo 5 while z modulo L without other specifications.

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
$\theta : A[x, y, z] = A[x, y, z] \oplus \bigoplus_{j=0}^{4}(A[x - 1, j, z] \oplus A[x + 1, j, z - 1])$,
$\rho : A[x, y, z] = A[x, y, (z + r[x, y])]$,
$\pi : A[y, 2x + 3y, z] = A[x, y, z]$,
$\chi : A[x, y, z] = A[x, y, z] \oplus ((\sim A[x + 1, y, z])\&A[x + 2, y, z])$,
$\iota : A[0, 0, z] = A[0, 0, z] \oplus RC[z]$. In the above definitions, bit-wise XOR is denoted by "$\oplus$", bit negation by "$\sim$", and bit-wise logic AND by "&". Besides, "$r[x, y]$" denotes for lane dependent rotation constants which are the values presented in Table 1 modulo the lane length L, and "RC" are round-dependent round constants. The details of RC are omitted since it does not affect our attacks. The readers could find more details about KECCAKin [BDPA11].

Table 1: The offsets of ρ.

	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

2.3 Instances of Keccak

The hash function KECCAK[r, c, l] means the instance of KECCAKsponge function family with parameters capacity c, bitrate r, and output length l. The official versions KECCAK-l have $r = 1600 - c$ and $c = 2 \cdot l$, where $l \in \{224, 256, 384, 512\}$. 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 SHA-3 standard only has digest sizes 224, 256, 384, and 512. It is similar to KECCAKexcept for the padding rule. SHA-3 pads the message with another two bits "01" before applying the KECCAKpadding 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). More accurately, SHAKE128(M, l) and SHAKE256(M, l) are defined as KECCAK$[r = 1344, c = 256]$ and KECCAK$[r = 1088, c = 512]$. And the messages M are padded with a suffix "1111".

Without the specification, our attacks in this paper on KECCAKapplies to SHA-3 and SHAKE under the same parameters. We will only consider preimage attacks on the instance with output length $l = 256$.

2.4 The Keccak Crunchy Crypto Collision and Pre-image Contest

KECCAKChallenge, which is short for "KECCAKCrunchy Crypto Collision and Preimage Contest" [BDPA], organized by the KECCAKteam in order to boost security analysis. The KECCAKteam presented challenges for reduced-round instances with $c = 160$ and

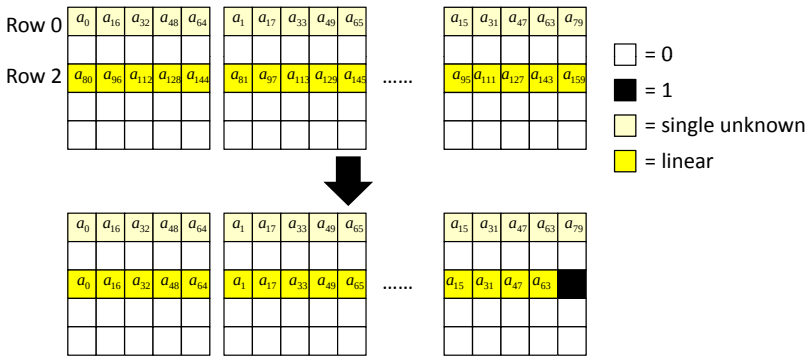

Figure 4: The initial status setting of the preimage attack on KECCAK$[r = 240, c = 160, n_r = 3]$. The constant parts in the bits of Row 2 are omitted.

$b \in \{200, 400, 800, 1600\}$. The output is truncated to 160 bits for collision challenges and 80 bits for preimages, thus both challenges have the theoretical complexity of 2^{80}.

Most of the challenges of 1 or 2 rounds were computed by SAT method around the year of 2011. There are no substantial progresses on the preimage attacks for a long time, after a preimage of KECCAK$[r = 40, c = 160, n_r = 1]$ was found in 2013. In 2016, Guo et al. developed the *linear structures*, and found 3 preimages of KECCAKChallenge instances of 3 and 4 rounds [GLS16]. They studied the nonlinear operator χ, and found a method to keep the system linear after the operation χ. Specifically, they presented the techniques of keeping 2 and 3 rounds linear at the cost of some degrees of freedom. Their techniques are capable of attacking instance with $r = 640, 1440$ on 3 rounds, but cannot work on the challenge instance KECCAK$[r = 240, c = 160, n_r = 3]$ and other smaller versions, because these instances offer much less degrees of freedom. The details of preimage attacks on the 4 round challenge instance are not publicly available yet.

After the discovery of linear structures, attacks on the collision challenges are also improved. Up to now, the best collision attacks are made by Song et al. on 6 rounds KECCAKChallenge instances [SLG17].

3 Main results

We construct two kinds of structures such that linear polynomials appear across different equations, and guessing some of the values of these linear polynomials could lead to much more linear equations. We call these new kinds of structures as *cross-linear structures*, and the mentioned linear polynomials are called *cross-linear factors*. With the first kind of cross-linear structures, we can get $3 \cdot k$ linear equations if the value of k linear polynomial are guessed, while by using the second cross-linear structures, $7 \cdot k$ linear equations can be obtained by guessing values of $2 \cdot k$ correlative polynomials. For simplicity, we call these two structures as "$1 \to 3$" and "$2 \to 7$" cross-linear structures respectively. To illustrate these two structures more clearly, we show them by detailing the procedure of finding a preimage of the challenge instance KECCAK$[r = 240, c = 160, n_r = 3]$ in SubSec. 3.1 and 3.2. In SubSec. 3.3, we give an improved complexity analysis on 3-round KECCAK-256 as well as SHA3-256 and SHAKE256. Some discussions are made in the last subsection.

3.1 The "$1 \to 3$" cross-linear structures

In this subsection, we illustrate the "$1 \to 3$" cross-linear structures by detailing the preimage attack on KECCAK$[r = 240, c = 160, n_r = 3]$. First, we formulate the polynomial

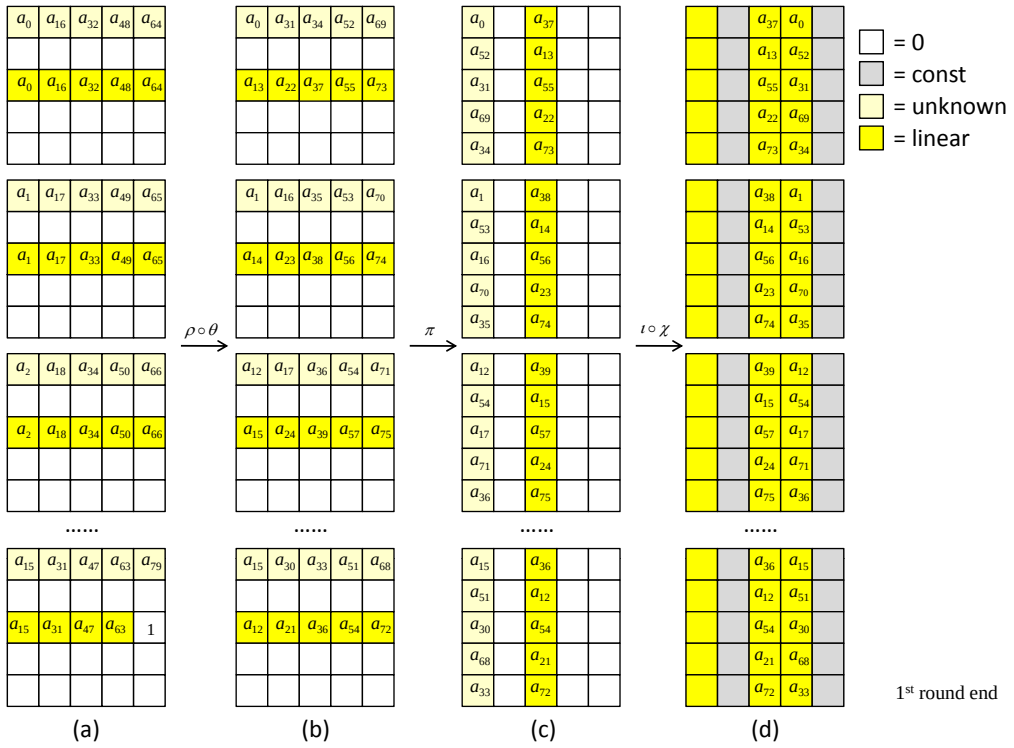

Figure 5: The first round of the preimage attack on KECCAK$[r = 240, c = 160, n_r = 3]$. The bits in white boxes are 0. The bits in (light) yellow boxes are called *linear bits*, because they can be represented by linear polynomials with the unknowns in the initial status. Particularly, bit in light yellow boxes is represented by a single unknown. Gray boxes mean the bits in them are constants. The constants in the linear/yellow boxes are omitted.

system deduced by this challenge; second, we show the "$1 \to 3$" cross-linear structures of this KECCAKinstance; at last, we generalize this "$1 \to 3$" cross-linear structures for more generic cases.

Formulation of the system: Figure 4, 5, 6, and 7 show the initial status and the statuses after the 1st, 2nd, and 3rd rounds KECCAK-f permutation. In these figures, white, black, gray, light yellow, yellow, blue, and green boxes means the bits in these boxes are 0's, 1's, constants, linear bits represented by 1 single unknown and no constant parts, linear bits, quadratic bits, and known bits respectively. Bits are called linear or quadratic if they can be represented as linear or quadratic polynomials in the unknowns of the initial status. Please remark that the constants in the linear and quadratic bits are always omitted in the figures.

Figure 4 is the initial status. To construct the cross-linear structures, we set the bits in Row 1 of all slices to 0. The bits in Row 3 and Row 4 are set to 0 by default. All unknowns a_i's appear in Row 0 and Row 2 where $i = 0, 1, \cdots, 159$. By the padding rule of KECCAK, we have $a_{159} = 1$. To avoid the mixture of bits brought by the operation θ, we assume the sums of every column are all the same, *i.e.* $a_i + a_{i+80} = c$ where $i = 0, 1, \cdots, 79$, such that the status before and after θ are identical, which is similar to that done in [GLS16]. The sum c of each column can either be 0 or 1. We hope to consider all the possibilities, so we need to enumerate the values of c, or equivalently, to enumerate the values of a_{79} because we always have $a_{79} + a_{159} = a_{79} + 1 = c$. By enumerating a_{79}, we then have $a_{i+80} = a_i + c = a_i + a_{79} + 1$ for $i = 0, 1, \cdots, 79$. This means the bits in Row 2 can be

represented as the bits in Row 0 plus a constant. Therefore, the initial status becomes the bottom one of Figure 4. For simplicity, the constants in Row 2 are omitted. This setting of the initial status is quite important for constructing the cross-linear structures, because the origin of the "cross" just lies in the fact that Row 0 and Row 2 of the initial status *share* the same unknowns.

Figure 5 (a) is the new initial status after fixing the value of a_{79}. After the operation ρ, the subscripts of a_i's are changed, as shown in Figure 5 (b). Please note that, the differences of subscripts on fixed positions are fixed modulo the length of lane, where the length in this challenge instance is 16. For example, the difference of the subscripts between position (Row 0, Column 0) and position (Row 2, Column 2) is always 37, *e.g.* (a_0, a_{37}), (a_1, a_{38}), $\cdots$. This fact is very important to the cross-linear structures.

The operation π permutes the bits in each slice, and we obtain the status (c) of Figure 5. In this status, the unknowns appear only in Column 0 and Column 2, while the bits in other columns are all 0's.

Since no adjacent bits in the rows are both linear in status (c), the nonlinear operation χ will not generate nonlinear bits. Column 0 of each slice is the sum of unknowns, and the gray bits in Figure 5 (d) are all constants.

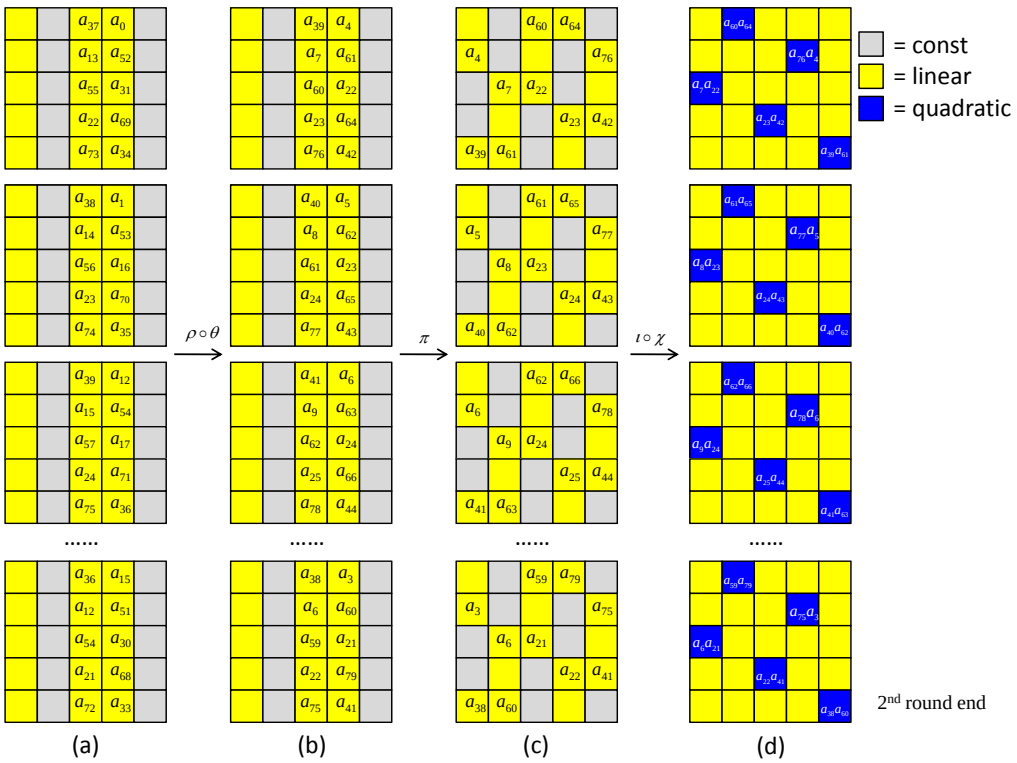

Figure 6: The second round of the preimage attack on KECCAK$[r = 240, c = 160, n_r = 3]$. The bits in gray boxes are constants. Linear bits and quadratic bits are the bits can be represented as linear and quadratic polynomials of unknowns, and they are shown in yellow and blue boxes respectively. The constant parts in the linear and quadratic boxes, as well as the linear parts in quadratic boxes, are omitted,

Figure 6 shows the statuses of the second forward round. To decrease the number of nonlinear bits after the operation χ in status (d), fewer linear bits in (c) should appear. For this aim, we should prevent the propagation of bits done by θ. However, no degrees of freedom are left now. We have to enumerate the values of the sums of Column 2 and

Column 3 in (a). Specifically, there are 16 slices, and we want to guess the values from the parities of these two columns, so we need to enumerate values for $2 \times 16 = 32$ variables in total. Due to the linear dependence, 31 instead of 32 values need to be guessed, which will be illustrated more clearly after the expressions of these 32 values are given sooner. Please also note that the bits in Column 0 are the sums of bits in Column 2 and Column 3, so the sums of bits in Column 0 can be inferred directly.

The status (b) in Figure 6 is obtained after the operations θ and ρ. The constant parts in linear boxes of Column 2 and Column 3 are omitted, *i.e.* the real representation of the bit in this linear box is the unknown marked inside the box plus a constant.

Doing the operations π, χ, and ι consequently, we get the status (d), in which there is exactly only one quadratic bit in each row and column of each slice. In Figure 6, these quadratic bits are marked in blue, and their constant and linear parts are omitted as well.

Remark that the quadratic parts of the quadratic bits in blue are all products of two linear polynomials, *e.g.* a_{60}, a_{64}, a_{76}, a_4, $\cdots$, in status (c) in Figure 6. We call these linear polynomials **cross-linear factors**, since these factors are linear and appear in different positions of slices. For example, the unknown a_6 appears both at (Row 1, Column 0, Slice 2) and (Row 2, Column 1, Slice 15) in Figure 6 (c). The property will make these cross-linear factors appear across different equations constructed by the challenge problem.

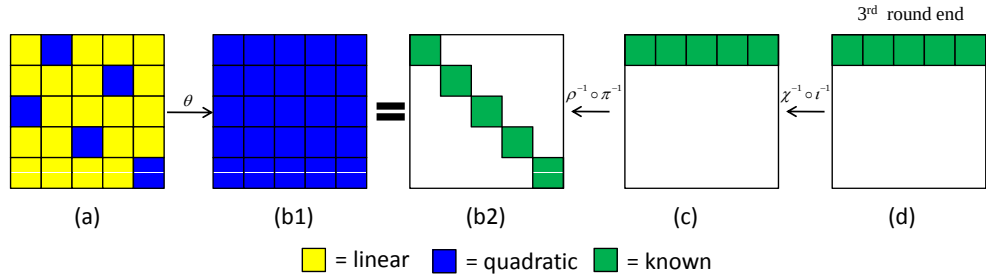

Figure 7: The third round of the preimage attack on KECCAK$[r = 240, c = 160, n_r = 3]$. Linear and quadratic bits are represented as yellow and blue boxes. Green boxes mean the bits in them are known.

Figure 7 shows the statuses of the third forward round. The status of Figure 6 (d) is the input of the third round, shown in Figure 7 (a). On one hand, all bits become quadratic bits after the forward θ operation, and the status is shown in (b1). On the other hand, we can obtain the status (b2) backward from the output of the 3rd round. The values of green bits are known by the KECCAKChallenge, and the bits in Row 0 of (c) are also known by Guo et al. [GLS16]. After the inverse operations of π and ρ. The bits in the diagonals of the status (b2) are all known. From (b1) and (b2), we formulate $5 \times 16 = 80$ quadratic polynomials in 80 variables, where 16 is the length of the lane. Remember that we have guessed the values of 31 linear polynomials at status (a) in Figure 6, and we also enumerate the value of a_{79} initially. So far, we have got all the equations we need to solve.

The quadratic equations are:

$$a_{32+\gamma(7+i)} \cdot a_{48+\gamma(13+i)} + a_{48+\gamma(11+i)} \cdot a_{64+\gamma(15+i)} + lin_1^{(i)} = c_1^{(i)},$$

$$a_{\gamma(7+i)} \cdot a_{16+\gamma(6+i)} + a_{16+\gamma(6+i)} \cdot a_{32+\gamma(9+i)} + lin_2^{(i)} = c_2^{(i)},$$

$$a_{48+\gamma(12+i)} \cdot a_{64+\gamma(i)} + a_{\gamma(3+i)} \cdot a_{64+\gamma(11+i)} + lin_3^{(i)} = c_3^{(i)},$$

$$a_{16+\gamma(7+i)} \cdot a_{32+\gamma(10+i)} + a_{32+\gamma(6+i)} \cdot a_{48+\gamma(12+i)} + lin_4^{(i)} = c_4^{(i)},$$

$$a_{\gamma(4+i)} \cdot a_{64+\gamma(12+i)} + a_{\gamma(6+i)} \cdot a_{16+\gamma(5+i)} + a_{32+\gamma(7+i)} \cdot a_{48+\gamma(13+i)} + lin_5^{(i)} = c_5^{(i)},$$

where $\gamma(k) = k \bmod 16$, $lin_j^{(i)}$ and $c_j^{(i)}$ are linear polynomials and constant values for $j = 1, 2, 3, 4, 5$ and $i = 0, 1, \cdots, 15$. And the 32 guessed sums are:

$$a_{\gamma(13+i)} + a_{16+\gamma(6+i)} + a_{32+\gamma(5+i)} + a_{48+\gamma(7+i)} + a_{64+\gamma(9+i)} = c_6^{(i)}, \tag{1}$$

$$a_{\gamma(i)} + a_{16+\gamma(15+i)} + a_{32+\gamma(2+i)} + a_{48+\gamma(4+i)} + a_{64+\gamma(5+i)} = c_7^{(i)}, \tag{2}$$

where $\gamma(k) = k \bmod 16$, and $c_6^{(i)}$, $c_7^{(i)}$ are constant values for $i = 0, 1, \cdots, 15$.

As we claimed early, 1 of the above 32 values is dependent on the others. To show the linear dependence, by summing up $c_6^{(i)}$ and $c_7^{(i)}$ over i, we have $\sum_{i=0}^{15} c_6^{(i)} = \sum_{j=0}^{79} a_j = \sum_{i=0}^{15} c_7^{(i)}$. Thus, taking $c_6^{(0)}$ for example, we have $c_6^{(0)} = \sum_{i=1}^{15} c_6^{(i)} + \sum_{i=0}^{15} c_7^{(i)}$. This means the value of $c_6^{(0)}$ is determined by the other 31 values, so it suffices to guess the other 31 ones instead of all 32.

Remark 1. If we do not assume the values of the 32 sums of Column 2 and Column 3 at status (a) in Figure 6, we can also formulate 80 quadratic polynomial equations. But the structures of the system becomes too complicated to illustrate the cross-linear structures.

The "1 → 3" cross-linear structure in Keccak$[r = 240, c = 160, n_r = 3]$**:** The quadratic equations constructed above can be simplified by linear algebraic operations, and we get:

$$\begin{gathered}
a_{\gamma(14+i)} \cdot a_{16+\gamma(13+i)} + a_{48+\gamma(i)} \cdot a_{64+\gamma(4+i)} + lin = c, \\
a_{16+\gamma(13+i)} \cdot a_{32+\gamma(i)} + a_{48+\gamma(i)} \cdot a_{64+\gamma(4+i)} + lin = c, \\
a_{32+\gamma(12+i)} \cdot a_{48+\gamma(2+i)} + a_{48+\gamma(i)} \cdot a_{64+\gamma(4+i)} + lin = c, \\
a_{64+\gamma(15+i)} \cdot a_{\gamma(7+i)} + a_{48+\gamma(i)} \cdot a_{64+\gamma(4+i)} + lin = c, \\
a_{48+\gamma(5+i)} \cdot a_{64+\gamma(9+i)} + a_{48+\gamma(3+i)} \cdot a_{64+\gamma(7+i)} + a_{48+\gamma(i)} \cdot a_{64+\gamma(4+i)} + lin = c,
\end{gathered} \tag{3}$$

where $\gamma(k) = k \bmod 16$, and $i = 0, 1, \cdots, 15$. In the above equations, lin's and c's mean linear polynomials and constant values, and they are different in different equations, while we omit the subscripts for simplicity. In fact, the equation (3) with $0 \le i \le 15$ can be inter-reduced with each other further, and we get:

$$a_{48+\gamma(i)} \cdot a_{64+\gamma(4+i)} + lin = c. \tag{4}$$

Consequently, by substituting $a_{48+\gamma(i)} \cdot a_{64+\gamma(4+i)}$ to the previous equations, we have

$$a_{\gamma(14+i)} \cdot a_{16+\gamma(13+i)} + lin = c, \tag{5}$$

$$a_{16+\gamma(13+i)} \cdot a_{32+\gamma(i)} + lin = c, \tag{6}$$

$$a_{32+\gamma(12+i)} \cdot a_{48+\gamma(2+i)} + lin = c. \tag{7}$$

$$a_{64+\gamma(15+i)} \cdot a_{\gamma(7+i)} + lin = c. \tag{8}$$

For the system of equations (4) ∼ (8), if guessing the value of *any 1* cross-linear factor, we will obtain 3 linear equations. This is because that any 1 cross-linear factor appears in 2 different equations, and guessing the value of this cross-linear factor obtains 1 linear equation directly by the guess, and also linearizes the 2 equations it appears. Because of this, we call the special structures represented by equations (4) ∼ (8) as **"1→ 3" cross-linear structures**.

So far, we have got 32 linear equations (31 linear independent guessed sums and 1 from a_{79}), and we will get 2 more by the guess of a_{79} and the "1→ 3" cross-linear structures. Next, we can guess the values of any 16 cross-linear factors, and obtain 3×16 more linear equations. Then, we have got $31 + 1 + 2 + 3 \times 16 = 82$ linear equations in 80 unknowns. There is a solution to this linear system with probability 1/4. At last, we verify the

obtained solution by the original 80 quadratic polynomials or simply by the equations that are not linearized. Because solving the linear system and verifying on the original polynomials takes constant time. The total practical preimage attack costs $2^{32+16} = 2^{48}$ operations.

Generalizing the "$1 \to 3$" cross-linear structures: For other KECCAKinstances, if the initial status is set the same as those in Figure 5 (a), then we can obtain a similar status to Figure 7 (a). The key point is that for this status, in each column of each slice, there is only one quadratic bit and it is just the product of two cross-linear factors. This ensures most of the quadratic bits after the operation θ in the 3rd round have similar structure as discussed above. That is, there are two products of cross-linear factors in most of the quadratic bits. By transformations or guessing values of cross-linear factors, the "$1 \to 3$" cross-linear structures can be constructed as well. Another example will be shown in SubSec. 3.3.

3.2 The "$2 \to 7$" cross-linear structures

Due to the special operation ρ of the KECCAKChallenge instance KECCAK$[r = 240, c = 160, n_r = 3]$, the practical preimage attack complexity can be even lowered, because we can obtain even more linear equations by using the characteristic of ρ.

We rearrange the subscripts in the equation (2), (5) and (6). We obtain the following new ones:

$$a_{\gamma(14+i)} + a_{16+\gamma(13+i)} + a_{32+\gamma(i)} + a_{48+\gamma(2+i)} + a_{64+\gamma(3+i)} = c, \tag{9}$$

$$a_{\gamma(14+i)} \cdot a_{16+\gamma(13+i)} + lin = c, \tag{10}$$

$$a_{16+\gamma(13+i)} \cdot a_{32+\gamma(i)} + lin = c, \tag{11}$$

where $\gamma(k) = k \bmod 16$, $i = 0, 1, \cdots, 15$. Again, lin's are linear polynomials and c's are constant values, and they are different in different equations. Substituting $a_{\gamma(14+i)}$ from (9) into (10), we get:

$$(a_{16+\gamma(13+i)} + a_{32+\gamma(i)} + a_{48+\gamma(2+i)} + a_{64+\gamma(3+i)}) \cdot a_{16+\gamma(13+i)} + lin = c,$$

and equivalently,

$$a_{16+\gamma(13+i)} \cdot a_{32+\gamma(i)} + a_{16+\gamma(13+i)} \cdot (a_{48+\gamma(2+i)} + a_{64+\gamma(3+i)}) + lin = c.$$

Adding the equation (11) to the above equation, we finally obtain:

$$a_{16+\gamma(13+i)} \cdot (a_{48+\gamma(2+i)} + a_{64+\gamma(3+i)}) + lin = c. \tag{12}$$

This relation means that if both the values of $a_{48+\gamma(2+i)}$ and $a_{64+\gamma(3+i)}$ are known, we will get **one more** linear equation.

By the results in the last subsection, guessing the value of any 1 cross-linear factor leads to 3 linear equations. Combining equation (12), if we guess the value of $a_{48+\gamma(2+i)}$ and $a_{64+\gamma(3+i)}$, then we will find 7 linear equations. Thus, we call the structures in equations (4) $\sim$ (8) and (12) as the **"2$\to$ 7" cross-linear structures**.

Here we list these 16 cross-linear factor pairs satisfying the "2$\to$ 7" cross-linear structures:

$$\{(a_{48}, a_{65}), (a_{49}, a_{66}), (a_{50}, a_{67}), (a_{51}, a_{68}), \cdots, (a_{62}, a_{79}), (a_{63}, a_{64})\}.$$

As mentioned above, guessing any pair in this set, 7 linear equations are obtained. But it does not mean if we pick up k pairs randomly, we can always get $7 \cdot k$ linear equations. We have to avoid guessing the variables as cross-linear factors in the same quadratic term. For example, the unknowns guessed should not contain both $a_{48+\gamma(i)}$ and $a_{64+\gamma(4+i)}$ with

the same i, because in that case, both of these two unknowns are the cross-linear factors of the quadratic term in equation (4). More specifically, specifying values of (a_{49}, a_{66}) and (a_{62}, a_{79}), we can find only 13 linear equations instead of 14, because $a_{62} \cdot a_{66}$ appears in equation (4) when $i = 14$.

For an attack of Keccak$[r = 240, c = 160, n_r = 3]$, by enumerating the values of $\{(a_{48}, a_{65}),\ (a_{52}, a_{69}),\ (a_{53}, a_{70}),\ (a_{54}, a_{71}),\ (a_{58}, a_{75}),\ (a_{62}, a_{79}),\ (a_{63}, a_{64})\}$, we obtain $14/2 \times 7 = 49$ linear equations. Note that a_{79} has already been scheduled for enumeration. The other steps are similarly as those in the SubSec. 3.1. We first solve the linear system, and then verify the solutions on the original system. The total practical preimage attack takes $2^{31+14} = 2^{45}$ operations.

Practical attack on the challenge Keccak$[r = 240, c = 160, n_r = 3]$: During the practical attack on this challenge, we implemented the codes in CUDA C++. The estimated time for enumerating all values is 8 days in 8 GPU cards (NVIDIA GTX 1080Ti). We were lucky to find the solution on the fifth day. A solution for the 3-round preimage challenge with width 400 is listed as below, where the length of the message is 238 and the preimage is expressed in hexadecimal.

Image of the challenge: 5c 9d 5e 4b 38 5e 9c 4f 8e 2e.

Preimage: 53 73 e0 75 3d ec af 5b 2e c1 00 00 00 00 00 00 00 00 00 00 53 73 e0 75 3d ec af 5b 2e c1.

By using almost the same method, we also obtain a preimage to the 3-round challenge Keccak$[r = 640, c = 160, n_r = 3]$ in a few seconds. This preimage is different from the one given by the previous challengers. The length of the message is 636 bits and we list the message below.

Image of the challenge: 00 7b b5 c5 99 80 66 0e 02 93.

Preimage: 2d dd 42 83 72 ca 84 d5 4e 20 36 69 02 55 3a 13 ad 6e 53 99 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 2d dd 42 83 72 ca 84 d5 4e 20 36 69 02 55 3a 13 ad 6e 53 99.

3.3 A preimage attack on 3-round Keccak-256

The idea presented in Sec.3.1 also applies to Keccak-256. Most of the attacking procedures are the same except some minor differences. In this subsection, we focus on dealing with these minor differences.

There are two main differences between Keccak-256 and Keccak$[r = 240, c = 160, n_r = 3]$ —— the padding rule and the length of output bits which also affects the capacity of the initial status. For Keccak-256, the last 8 lanes of Figure 8 (a) are 0's, and the messages are contained in the other 17 lanes. By the padding rule, we can set the first and second lanes in Row 3 to 0 except for the last bit in the position (Row 3, Column 1) of the last slice (Slice 63). Similarly to the top status of Figure 4, the bits in Row 0 and Row 2 are set as unknowns. To avoid the propagation of the θ step, we also assume the sums of each column in the initial status to a constant whose value is enumerated in this attack. And we use 1 linear constrain to set whether bits in Row 0 and Row 2 are identical or opposite. By doing so, Row 0 and Row 2 share the same unknowns like the bottom status of Figure 4. Please note that with this linear constrain, each bit in Row 0 is still always represented as a single unknown, while the linear bits in Row 2 may contain constant parts which are omitted in the figures as well.

So far, the main difference of Keccak-256 and Keccak$[r = 240, c = 160, n_r = 3]$ lies in that, there is 1 bit padding at the position (Row 3, Column 1) of Slice 63, shown in Figure 8 (a). Next, we show how to deal with this padding. Figure 8 illustrates the first forward round of Keccak-256, and focuses on the status containing this padding.

Figure 8 (a) is the initial status of the last slice with the padding bit 1 in Row 3. The sum of Column 1 is still assumed as a constant similarly to that done in Sec. 3.1. After the operations θ, ρ and π, we can find the statuses of slices that are affected by this bit

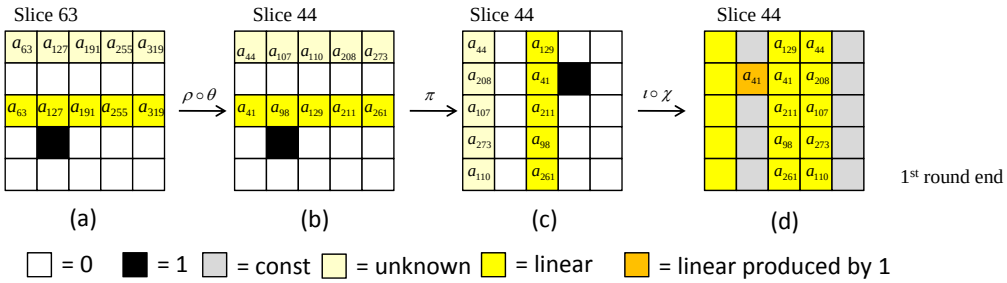

Figure 8: The First round of the preimage attack on Keccak-256. Constant parts in the Row 2 of (a) are omitted.

1, and these slices are shown in (b) and (c). After the operation χ, we obtain the status (d). Both orange and yellow bits are linear bits in Figure 8 (d). We highlight the orange bit because it is produced by bit 1 during the operation χ. Please note that a_{41} appears twice in status (d), and that is to say, we will obtain two constant bits if we enumerate the value of a_{41}. More importantly, by this enumeration, the status (d) becomes the same as that in Sec. 3.1. And hence, the second round is the same as Figure 6.

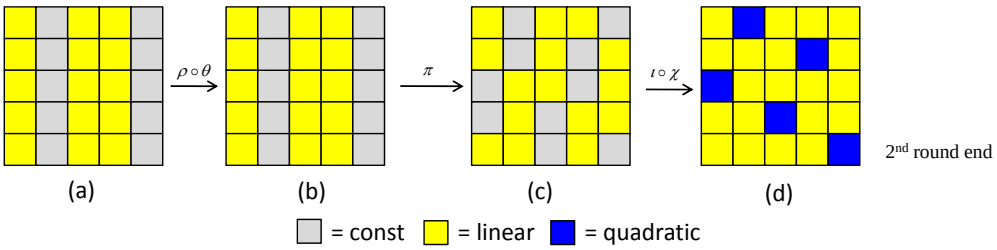

Figure 9: The second round of the preimage attack on Keccak-256.

Figure 9 shows the statuses in the second forward round. Please note that, we will also enumerate the sums of Column 2 and Column 3 as done in Sec. 3.1. But there is a little difference. We have to enumerate the values of all $2 \times 16 - 1$ sums in Sec. 3.1, because no degrees of freedom are left in that instance. But here, we still have 64 degrees of freedom, so we can directly set the value of 64 sums, and enumerate the values of the other $64 - 1$ sums. Note that there is still a sum linear dependent on the others.

Figure 10 shows the procedure of setting up the algebraic system. There are 4 lanes of the output bits known by this 3-round reduced Keccak-256. With the inverse of operation ι, χ, and π, the values of 4 green bits in each slice are known as shown in status (b2). Thus, we obtain 4×64 quadratic equations from status (b1) and (b2).

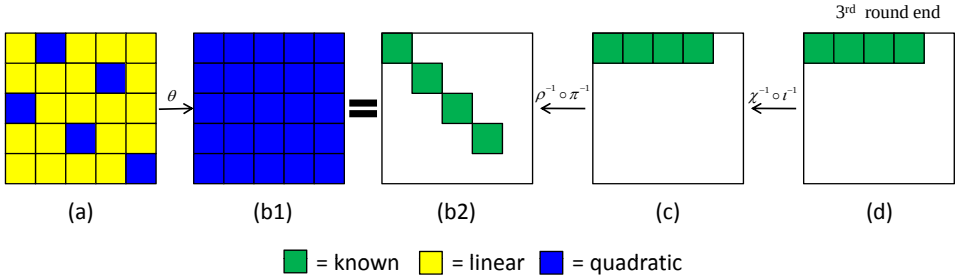

Figure 10: The third round of the preimage attack on Keccak-256.

Similar to the attack on Keccak$[r = 240, c = 160, n_r = 3]$, we obtain the following quadratic polynomials:

$$a_{128+\gamma'(26+i)} \cdot a_{192+\gamma'(48+i)} + a_{192+\gamma'(62+i)} \cdot a_{256+\gamma'(18+i)} + lin_0^{(i)} = c_0^{(i)}, \tag{13}$$

$$a_{\gamma'(58+i)} \cdot a_{64+\gamma'(41+i)} + a_{64+\gamma'(41+i)} \cdot a_{128+\gamma'(12+i)} + lin_1^{(i)} = c_1^{(i)}, \tag{14}$$

$$a_{192+\gamma'(63+i)} \cdot a_{256+\gamma'(19+i)} + a_{256+\gamma'(30+i)} \cdot a_{\gamma'(38+i)} + lin_2^{(i)} = c_2^{(i)}, \tag{15}$$

$$a_{64+\gamma'(42+i)} \cdot a_{128+\gamma'(13+i)} + a_{128+\gamma'(25+i)} \cdot a_{192+\gamma'(47+i)} + lin_3^{(i)} = c_3^{(i)}, \tag{16}$$

where $\gamma'(k) = k \bmod 64$, $lin_j^{(i)}$ are linear polynomials and $c_j^{(i)}$ are the constant values of bits at the positions (Row j, Column j), where $j = 0, 1, 2, 3$ and $i = 0, 1, \cdots, 63$. Since we only have 4 known values in the first row of Figure 10 (d), we only have 4 groups of equations.

Due to the lack of one group of equations, the above 4 groups of equations cannot be simplified to the form of the equations (4) $\sim$ (8) directly. To construct the cross-linear structures, we reorder the subscripts of equations (13) $\sim$ (16) and rearrange them to the following form:

$$a_{\gamma'(58+i)} \cdot a_{64+\gamma'(41+i)} + a_{64+\gamma'(41+i)} \cdot a_{128+\gamma'(12+i)} + lin_1^{(i)} = c_1^{(i)}, \tag{17}$$

$$a_{64+\gamma'(41+i)} \cdot a_{128+\gamma'(12+i)} + a_{128+\gamma'(24+i)} \cdot a_{192+\gamma'(46+i)} + lin_3^{(\gamma'(i-1))} = c_3^{(\gamma'(i-1))}, \tag{18}$$

$$a_{128+\gamma'(24+i)} \cdot a_{192+\gamma'(46+i)} + a_{192+\gamma'(60+i)} \cdot a_{256+\gamma'(16+i)} + lin_0^{(\gamma'(i-2))} = c_0^{(\gamma'(i-2))}, \tag{19}$$

$$a_{192+\gamma'(60+i)} \cdot a_{256+\gamma'(16+i)} + a_{256+\gamma'(27+i)} \cdot a_{\gamma'(35+i)} + lin_2^{(\gamma'(i-3))} = c_2^{(\gamma'(i-3))}, \tag{20}$$

where $i = 0, 1, \cdots, 63$. In each of these equations, there are two products of cross-linear factors, which makes the equations nonlinear. A product of cross-linear factors often appears in two different equations, *e.g.*, $a_{192+\gamma'(60+i)} \cdot a_{256+\gamma'(16+i)}$ appears in equations (19) and (20). Besides, a cross-linear factor usually appears in three different equations.

Next, we decrease the number of products of cross-linear factors in each equation to 1 by enumerating the values of another 31 cross-linear factors. Since the value of a_{41} has to be enumerated as discussed before, we enumerate the values of the following 31 cross-linear factors:

$$\{a_1, a_3, a_5, \cdots, a_{39}\} \cup \{a_{43}, a_{45}, \cdots, a_{63}\}.$$

Then if $i = 0, 2, 4, \cdots, 62$, then equations (17) $\sim$ (20) can be transformed by linear algebra to the following form:

$$a_{\gamma'(58+i)} \cdot a_{64+\gamma'(41+i)} + (a_{256+\gamma'(27+i)} \cdot a_{\gamma'(35+i)} + lin'_0) = c'_0, \tag{21}$$

$$a_{64+\gamma'(41+i)} \cdot a_{128+\gamma'(12+i)} + (a_{256+\gamma'(27+i)} \cdot a_{\gamma'(35+i)} + lin'_1) = c'_1, \tag{22}$$

$$a_{128+\gamma'(24+i)} \cdot a_{192+\gamma'(46+i)} + (a_{256+\gamma'(27+i)} \cdot a_{\gamma'(35+i)} + lin'_2) = c'_2, \tag{23}$$

$$a_{192+\gamma'(60+i)} \cdot a_{256+\gamma'(16+i)} + (a_{256+\gamma'(27+i)} \cdot a_{\gamma'(35+i)} + lin'_3) = c'_3, \tag{24}$$

where lin'_j are linear polynomials and c'_j are constants for $j = 0, 1, 2, 3$. Note that $a_{\gamma'(35+i)}$'s have been guessed as constants, so there is only one product of cross-linear factors in each equation. Similarly, if $i = 1, 3, 5, \cdots, 63$, then equations (17) $\sim$ (20) can be transformed by linear algebra to the following form:

$$a_{64+\gamma'(41+i)} \cdot a_{128+\gamma'(12+i)} + (a_{\gamma'(58+i)} \cdot a_{64+\gamma'(41+i)} + lin''_0) = c''_0, \tag{25}$$

$$a_{128+\gamma'(24+i)} \cdot a_{192+\gamma'(46+i)} + (a_{\gamma'(58+i)} \cdot a_{64+\gamma'(41+i)} + lin''_1) = c''_1, \tag{26}$$

$$a_{192+\gamma'(60+i)} \cdot a_{256+\gamma'(16+i)} + (a_{\gamma'(58+i)} \cdot a_{64+\gamma'(41+i)} + lin_2'') = c_2'', \tag{27}$$

$$a_{256+\gamma'(27+i)} \cdot a_{\gamma'(35+i)} + (a_{\gamma'(58+i)} \cdot a_{64+\gamma'(41+i)} + lin_3'') = c_3'', \tag{28}$$

where lin_j'' are linear polynomials and c_j'' are constants for $j = 0, 1, 2, 3$. Because $a_{\gamma'(58+i)}$'s are constants, there is only one product of cross-linear factors in each equation as well. This means we have constructed the "$1 \to 3$" cross-linear structures. And hence, from the equations (21) $\sim$ (28), we can get 3 linear equations by guessing the value of 1 cross-linear factor. For example, if we guess the value of a_{128+25}, then the equation (25) with $i = 13$ and equation (26) with $i = 1$ can be linearized, and we also have the linear equation by guessing a_{128+25}.

For the attack on this 3-round Keccak-256, we have obtained $2 \times 64 - 1$ linear equations from assuming the sums of columns in Figure 9 (a), and 32 linear equations by guessing a_{2k+1} for $k = 0, 1, \cdots, 31$. Besides, we use 1 linear equation to set the constants in Row 0 and Row 2 in initial status. Since there are 5×64 unknowns, we still need $3 \times 64 - 32 = 160$ linear equations. These equations can be obtained by guessing 54 cross-linear factors. For example, we can guess the values of the values of $a_{128}, a_{128+1}, \cdots, a_{128+53}$, and then we will obtain 2×54 linear equations from equations (22) and (23) or equations (25) and (26). Now we have $2 \times 64 + 32 + 3 \times 54 = 322$ linear equations in $5 \times 64 = 320$ unknowns. A solution to this system can be got in constant time, and then we verify this solution by the original equations (13) $\sim$ (16) in constant time.

The complexity of the above attack is estimated as follows. Since there are $5 \times 64 - 256 = 64$ degrees of freedom, we can directly set the value of 64 sums of columns in Figure 9 (a). So we totally need to enumerate the values of $1 + (64 - 1) + 32 + 54 = 150$ unknowns. Since the time for solving a linear system and verifying on original equations is constant, the complexity on finding a preimage for the 3-round Keccak-256 is 2^{150} operations.

3.4 Some discussions

If there is no solution to Keccak$[r = 240, c = 160, n_r = 3]$ by the setting in Figure 4: By the settings of discussed in SubSec. 3.1, the freedom degree of the constructed system is 0. Although there is a solution to this system with probability 1, we also considered the failure cases.

If the settings in Figure 4 do not work, we can set 1 bit in Row 1 of some slice to 1 instead of 0. This setting affects the forms of some statuses, but will not affect the whole method.

Figure 11 shows the statuses in the first round with a bit 1 at the position (Row 1, Column 2, Slice 2). Here we work on with this setting. The statuses (a), (b) and (c) are almost the same with Figure 5 and we trace the slices affected by the bit 1. In the status (d), one more linear bit a_{21} (orange one) is generated compared to the status (d) in Figure 5. This additional linear bit will contribute to two more quadratic bits (light blue ones in Figure 12 (d)) in the following round, which makes it difficult to construct the cross-linear structures. Thus, similarly as done in SubSec. 3.3, we enumerate a_{21} to avoid the additional quadratic bits. Moreover, a_{21} also leads to 3 linear equations so it does not double the practical complexity. After guessing the value of a_{21}, the statuses in the third round and the formulation of the system are identical to Figure 7, so we omit it here.

Our attack works on the case with 1 bit in other positions as well. Please remember to enumerate the corresponding a_i to make sure that only 1 quadratic bit appears in each column in the last status of the 2nd round.

Preimage attacks on 3-round SHAKE256 and SHA3-256: For 3-round SHAKE256 with 256 output bits, the attack in SubSec. 3.3 works as well although the padding rule is different. From SubSec. 2.3, we know that SHAKE256 is defined as Keccak$[r = 1088, c = 512]$ and the message M after padding is in the form of M111110*1. Assume that all the padding bits are in Row 3 and other message bits in Row 3 are 0. For example, we

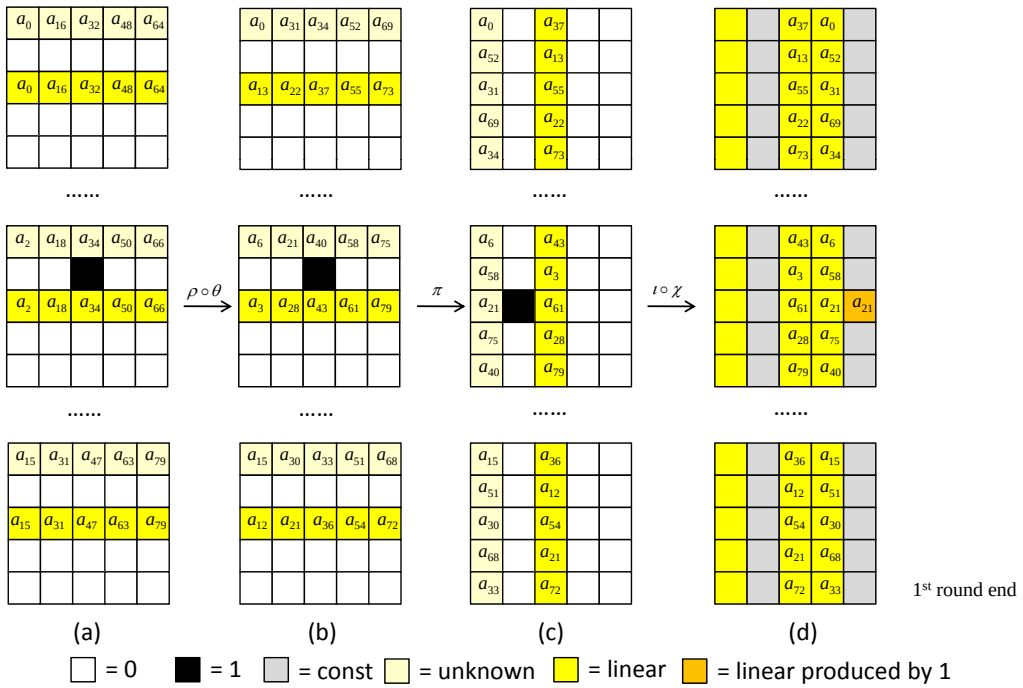

Figure 11: The first round of the preimage attack on KECCAK$[r = 240, c = 160, n_r = 3]$ with a bit 1 in Row 1 of the initial status.

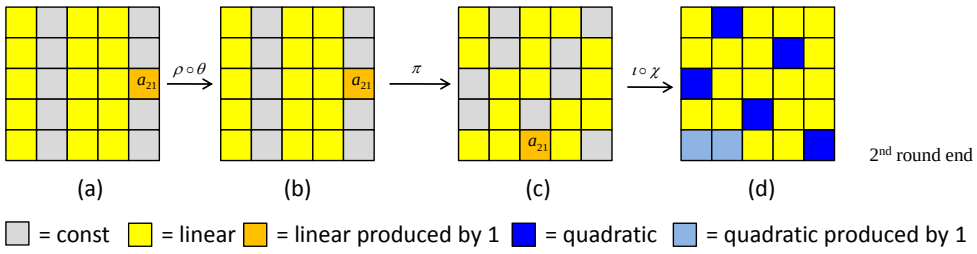

Figure 12: The second round of the preimage attack on KECCAK$[r = 240, c = 160, n_r = 3]$ with a bit 1 in Row 1 of the initial status.

set the last 6 bits at position (Row 3, Column 1, Slice 58 - 63) to 1 and the other bits to 0 in Row 3. That is to say, we have 6 slices with additional bit 1 in the initial status and they bring in 6 additional linear bits after the 1st round. The unknowns in these linear bits are $a_{36}, a_{37}, a_{38}, a_{39}, a_{40}, a_{41}$. Similar to the method we used in SubSec. 3.3, the collection of enumeration unknowns should have included a_{37}, a_{39}, a_{41}. To guarantee the cross-linear structures, we add a_{36}, a_{38}, a_{40} into the enumeration unknowns. Thus, we deduce the linear system and keep the complexity to $2^{150+3} = 2^{153}$ as well. In practical attacks, other padding bits is allowed as long as they follow the padding rules.

Thus, the above idea also works on the 3-round SHA-3 standard with digest sizes 256. Instead of padding message with 111110*1, we have the padded message M0110*1. So we could set the 3 bits in first two lanes in Row 3 to 1 (one should be in the last slice and the other two in adjacent slices according to the padding rule) while others are set to 0 and guess the value of 3 relevant variables. The other steps are totally the same and this attack finds a preimage in $2^{150+1} = 2^{151}$ operations.

The key steps of constructing the cross-linear structures: From the above attacks, we can summarize that there are three key steps of constructing cross-linear structures.

1. First, the columns of the initial status should have at most two unknowns. In this case, the same unknown could appear at two different positions of the slices. Consequently, the unknowns appear across different equations.

2. Second, there is at most 1 quadratic bit in each column of the last status of the 2nd round. This ensures all bits are quadratic bits after the operation θ of the 3rd round, and most of these quadratic bits contain only two products of cross-linear factors.

3. Third, we need to decrease the nums of products of cross-linear factors in the constructed equations by transformations or enumerations.

The "$2 \rightarrow 7$" cross-linear structures: The "$2 \rightarrow 7$" cross-linear structures are more relevant to operation ρ. Up to now, we find that this structure only exists in the KeccakinstancesKeccak with width 400.

4 Conclusions

In this paper, we construct two kinds of new structures, *cross-linear structures*, in the reduced round instances of Keccak. We use these structures to successfully find a preimage of the KeccakChallenge instance Keccak$[r = 240, c = 160, n_r = 3]$, and we also obtain better complexities for the preimage attack on 3-round Keccak-256/SHAKE256/SHA3-256. The key steps of constructing the cross-linear structures are that (1) making the same unknowns appear at different positions of slices, (2) limiting the number of nonlinear products in the quadratic part of constructed equations, and (3) decreasing the number of products of cross-linear factors by transformations or enumerations.

We believe the cross-linear structures will play important roles when attacking Keccakinstances of higher rounds. In the future, we will try to improve the cross-linear structure techniques, and break more KeccakChallenge instances.

Acknowledgement

We would like to thank Meicheng Liu for helpful discussions. Our research was supported by National Natural Science Foundation of China (No. 11301523 and No. 11371356).

References

[BDPA] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak crunchy crypto collision and pre-image contest. In *http://keccak.noekeon.org/crunchy contest.html*.

[BDPA11] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The keccak reference, version 3.0. In *http://keccak.noekeon.org*, 2011.

[Ber10] D. J. Bernstein. Second preimages for 6(7?(8??)) rounds of keccak. In *NIST mailing list*, 2010.

[CKMS14] D. Chang, A. Kumar, P. Morawiecki, and S.K. Sanadhya. 1st and 2nd preimage attacks on 7, 8 and 9 rounds of keccak-224,256,384,512. In *SHA-3 Workshop*, 2014.

[DDS12] I. Dinur, O. Dunkelman, and A. Shamir. 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*, pages 442–461, 2012.

[DDS13] I. Dinur, O. Dunkelman, and A. Shamir. 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*, pages 219–240, 2013.

[DDS14] I. Dinur, O. Dunkelman, and A. Shamir. Improved practical attacks on round-reduced keccak. *J. Cryptol.*, 27(2):183–209, Apr 2014.

[Fau99] J.-C. Faugère. A new efficient algorithm for computing gröbner bases (f4). *Journal of Pure & Applied Algebra*, 139(1-3):61–88, 1999.

[Fau02] J.-C. Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f5). In *International Symposium on Symbolic and Algebraic Computation*, pages 75–82, 2002.

[GIW16] S. Gao, F. Volny IV, and M. Wang. A new framework for computing gröbner bases. *Math. Comput.*, 85(297):449–465, 2016.

[GLS16] J. Guo, M. Liu, and L. Song. 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*, pages 249–274, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[MPS13] P. Morawiecki, J. Pieprzyk, and M. Srebrny. Rotational cryptanalysis of round-reduced keccak. In *Fast Software Encryption: 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, pages 241–262, 2013.

[MS13] P. Morawiecki and M. Srebrny. A sat-based preimage analysis of reduced keccak hash functions. *Inf. Process. Lett.*, 113(10-11):392–397, 2013.

[NIS] NIST. Sha-3 competition (2007-2012). In *http://csrc.nist.gov/groups/ST/hash/sha-3/index.html*.

[NPRM11] M. Naya-Plasencia, A. Röck, and W. Meier. Practical analysis of reduced-round keccak. In *International Conference on Cryptology in India*, pages 236–254, 2011.

[oST15] 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.

[SLG17] L. Song, G. Liao, and J. Guo. 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*, pages 428–451, Cham, 2017. Springer International Publishing.

[SW11] Y. Sun and D.K. Wang. A generalized criterion for signature related gröbner basis algorithms. In *International Symposium on Symbolic and Algebraic Computation*, page 337–344, 2011.

[YSL16] D.K. Wang Y. Sun, Z.Y. Huang and D.D. Lin. An improvement over the gvw algorithm for inhomogeneous polynomial systems. *Finite Fields and Their Applications*, 41:174–192, 2016.