Practical Preimage Attack on 3-Round Keccak-256

Xiaoen Lin[1], Le He[2], and Hongbo Yu[3(✉)]

[1] Department of Computer Science and Technology, Tsinghua University, Beijing, China, lxe21@mails.tsinghua.edu.cn
[2] Department of Computer Science and Technology, Tsinghua University, Beijing, China, he-l17@mails.tsinghua.edu.cn
[3] Department of Computer Science and Technology, Tsinghua University, Beijing, China, yuhongbo@mail.tsinghua.edu.cn

Abstract. This paper combines techniques from several previous papers with some modifications to improve the previous cryptanalysis of 3-round Keccak-256. Furthermore, we propose a fast rebuilding method to improve the efficiency of solving equation systems. As a result, the guessing times of finding a preimage for 3-round Keccak-256 are decreased from 2^{65} to 2^{52}, and the solving time of each guess is decreased from 2^9 3-round Keccak calls to $2^{5.3}$ 3-round Keccak calls. We identify a preimage of all '0' digest for 3-round Keccak-256 to support the effectiveness of our methodology.

Keywords: Keccak · SHA-3 · Preimage attack · Linear structure.

1 Introduction

The Keccak function, designed by Bertoni et al. [1], is a family of cryptographic functions, which was submitted to the public competition held by NIST (National Institute of Standards and Technology) in 2008. In 2015, it was standardized as Secure Hash Algorithm 3 (SHA-3) [2]. Up to now, plenty of research has been conducted by public community.

On collision attacks, Naya-Plasencia et al. obtained practical collision on 2-round Keccak-224/256 using low hamming weight differential paths [3]. Dinur et al. proposed a target difference algorithm by connecting a 2.5-round differential trail with a 1.5-round connector [4]. They found practical collisions on 4-round Keccak-224/256. Later, they give attacks on 5-round Keccak-256 and other variants using generalized internal differentials [5]. Qiao et al. extended the connector by one more round and gave attack on 5-round Keccak-224 [6]. Song et al. saved more degrees of freedom and found practical collision on 5-round Keccak-224 [7]. Guo et al. further improved the connector and the differential trail so that practical collision on 5-round Keccak-256 was detected [8].

On distinguishing attacks, Naya-Plasencia et al. put forward a practical differential distinguisher on 4-round Keccak-256/224 [3]. Das et al. found distinguishers on 6-round Keccak-224 [9]. Dinur et al. first introduced the cube attacks

on Keccak, and they gave practical distinguishing attacks for 6-round Keccak on different variants [10]. Using cube attacks, Huang et al. developed a new type of distinguisher named conditional cube tester in 2017 [11]. This technique improved the results significantly and gave practical distinguishing attacks for 7-round Keccak on different variants.

On preimage attacks, Naya-Plasencia et al. gave practical preimage attacks on 2-round Keccak-224/256 [3]. Then, Guo et al. developed a technique named linear structure and gave preimage attacks on different variants for up to 4 rounds [12]. For round-reduced Keccak-224/256, Li et al. used the allocating approach and gave practical preimage attack on 3-round Keccak-224 [13]. Their attacks also improved the results on 3-round Keccak-256 and 4-round Keccak-224/256. Lin et al. further refined the results on 3-round Keccak-224/256 by using the 5-for-3 strategy and the iterating strategy [14]. Pei et al. let the linear structure satisfied probabilistically, and make improvement on the result on 3-round Keccak-256 [15]. For 4-round Keccak-224/256, He et al. [16], Dinur [17], and Wei et al. [18] gave further attacks by using different techniques including the freedom reuse strategy, the polynomial method, and the Crossbred algorithm. For round-reduced Keccak-384/512, Kumar et al. demonstrated better results on 2-round Keccak-384 with high required memory [19]. Rajasree allowed non-linear parts on linear structure and improved the results on round-reduced Keccak-384/512 for up to 3/4 rounds [20]. Liu et al. continued to enhance the results by making full use of the linear relations [21]. The results of preimage attacks on Keccak-224/256 are summarized in Table 1.

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

Instance	Guessing Times	[a,b] Solving Time	[a] Total Complexity	Reference
Keccak-256	2^{192}	2^{6}	2^{198}	[12]
Keccak-256	2^{81}	2^{9}	2^{90}	[13]
Keccak-256	2^{65}	[c] 2^{9}	2^{74}	[14]
Keccak-256	$2^{64.79}$	2^{9}	$2^{73.79}$	[15]
Keccak-256	2^{52}	2^{9}	2^{61}	Section 4
Keccak-256	2^{52}	$2^{5.3}$	$2^{57.3}$	Section 5
Keccak-224	2^{97}	2^{6}	2^{103}	[12]
Keccak-224	2^{38}	[c] 2^{9}	2^{47}	[13]
Keccak-224	2^{32}	[c] 2^{9}	2^{41}	[14]
Keccak-224	2^{31}	$2^{5.3}$	$2^{36.3}$	Appendix A

[a] Unit: equivalent 3-round Keccak calls.

[b] The solving time in Section 5 is the actual running time. Other solving times are our estimated results according to the rest degrees of freedom for comparisons (similar to [21]).

[c] According to their experimental results, the actual running time is around $2^{12} - 2^{14}$ 3-round Keccak calls.

Our contribution. In this paper, we combine techniques from several previous papers and bring up a modified linear structure. There are two advantages. The first one is that the modified linear structure leaves more degrees of freedom. The second one is that we change the values of some constant bits so that the difficulty of matching the starting state can be solved. Although this structure will generate quadratic bits and result in quadratic equations, only a small number of quadratic bits will appear. We then solve the linear equations and leave the quadratic equations to be satisfied randomly. In addition, we propose a technique to rebuild and solve the equation system faster. In each guess, we only change some constants instead of randomizing all of them. When some values of constant bits vary, only a small number of linear equations will be changed. Then we rebuild the equation system faster and solve it hierarchically. With these techniques, the guessing times of finding a preimage for 3-round Keccak-256 are decreased from 2^{65} to 2^{52}, and the solving time of each guess is decreased from 2^9 3-round Keccak calls to $2^{5.3}$ 3-round Keccak calls. Moreover, we demonstrate the first practical preimage attack on 3-round Keccak-256.

Organization. In Section 2, we give some preliminaries and notations about Keccak. The related work and literature review are discussed in Section 3. Section 4 explains the improved attack with the modified linear structure. Section 5 presents the fast rebuilding method to improve the efficiency of solving equation systems. The experimental results and the conclusion of this paper are provided in Section 6 and Section 7, respectively.

2 Preliminaries

2.1 Sponge Construction

The sponge construction is a mode of operation which builds a sponge function [22]. The sponge function is a generalization of hash functions. As shown in Fig. 1, the sponge construction operates on a state of $b = r + c$ bits where the state is initially set to all '0' initial value. In the absorbing phase, the message M is padded until its length is a multiple of r. Then the padded input message is divided into several r-bit message blocks. Each time the state absorbs a message block, the first r bits of the state are XORed by the r-bit message block. After that, the state will be operated by the Keccak-f permutation. In the squeezing phase, the state squeezes every r bits by outputs the first r bits of the state until the length of the output is greater or equal to the required length ℓ. Similar to the absorbing phase, each time the state squeezes r bits output, the state is operated by the Keccak-f permutation. At last, the digest is obtained by truncating the output to the required length ℓ.

2.2 Keccak-f Permutation

The state size b can be chosen from $\{25, 50, 100, 200, 400, 800, 1600\}$, while NIST selects the value 1600 for b as SHA-3 standard. In this paper, we focus on the

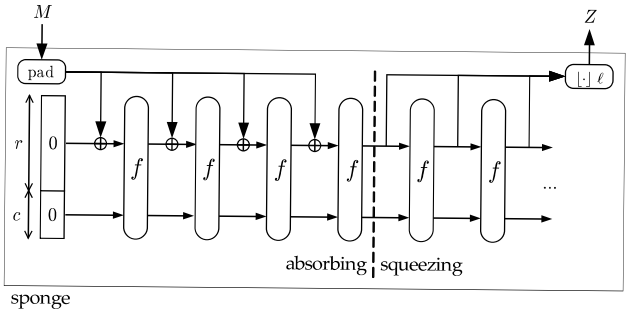

Fig. 1. The sponge construction [23].

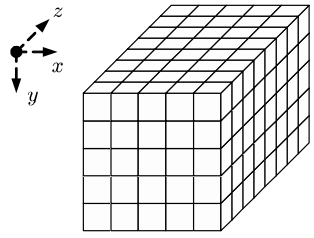

Fig. 2. The Keccak-f state.

case $b = 1600$. As shown in Fig. 2, the 1600-bit state can be described as 5×5 64-bit lanes. The state can be denoted as $A_{x,y,z}$, where $0 \le x, y \le 4$, $0 \le z \le 63$.

The permutation Keccak-$f[1600]$ consists of 24 round functions which only differ in the round-dependent constant. The round function R has 5 steps $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$, where:

$$\begin{aligned}
\theta &: A_{x,y,z} = A_{x,y,z} \oplus \bigoplus\nolimits_{i=0\sim4}(A_{x-1,i,z} \oplus A_{x+1,i,z-1}) \\
\rho &: A_{x,y,z} = A_{x,y,(z-r_{x,y})} \\
\pi &: A_{x,y,z} = A_{x+3y,x,z} \\
\chi &: A_{x,y,z} = A_{x,y,z} \oplus (A_{x+1,y,z} \oplus 1) \cdot A_{x+2,y,z} \\
\iota &: A_{0,0,z} = A_{0,0,z} \oplus RC_z
\end{aligned}$$

In the formulas above, " $\oplus$ " denotes the bit-wise XOR, and " $\cdot$ " denotes the bit-wise AND. x and y are taken modulo 5, and z is taken modulo 64. $r_{x,y}$ is a constant shown in Table 2, and RC_z is a round-dependent constant. We omit the constants RC because their values do not affect our attack.

2.3 SHA-3 Standard

There are four SHA-3 versions standardized by NIST [2]. The parameters are $r = 1600 - 2\ell$ and $c = 2\ell$, where $\ell \in \{224, 256, 384, 512\}$. The difference between

Table 2. The offsets of ρ.

	x = 0	x = 1	x = 2	x = 3	x = 4
y = 0	0	1	62	28	27
y = 1	36	44	6	55	20
y = 2	3	10	43	25	39
y = 3	41	45	15	21	8
y = 4	18	2	61	56	14

Keccak and SHA-3 is the padding rule. The message M is padded with "10*1" and "0110*1" in Keccak and SHA-3, respectively. This paper gives cryptanalysis results for Keccak. When we apply the same cryptanalysis to SHA-3, the complexity will be 4 times higher.

2.4 Notations

We use capital Greek letters Θ, P, Π, X, I with a superscript number (from 0 to 2, and 0 represents the first round) to represent the state **before** the corresponding step is executed. Besides, we use three indices in subscript to express the bit (or bits) in the inner state. We use " $*$ " to indicate the union of all values, and we use x, y, and z to indicate a specific value. For example, $\Theta^0_{*,y,z}$ is a row, $\Theta^0_{x,*,z}$ is a column, $\Theta^0_{x,y,*}$ is a lane and $\Theta^0_{*,*,z}$ is a slice.

3 Related Work

In this section, we review the previous work, including the techniques using the linear structures [12], the allocating approach [13], the iterating strategy and the 5-for-3 strategy [14] and the technique of linearizing quadratic equations [21].

3.1 The Linear Structures

Guo et al. develop the linear structures to linearize the permutation of round-reduced Keccak [12]. When used in 3-round Keccak-256, the technique is shown in Fig. 3. The black lanes mean that these bits are all 1, while the white lanes indicate that these bits are all 0. The yellow lanes imply that these bits are linear. The grey lanes suggest that some of these bits are 0, and the others are 1. To prevent the diffusion of the variables in the θ operation, they add 128 and 192 linear equations on Θ^0 and Θ^1 so that the sum of each column will be constant. Then, the state stays linear for up to 2.5 rounds.

There are $6 \times 64 = 384$ variables and $128 + 192 = 320$ linear equations, so there are $384 - 320 = 64$ degrees of freedom left which can be used to restrict the output bits. For the property of χ operation, four given output bits can be restricted by four linear equations. Thus, the 64 degrees of freedom can be used to restrict 64 output bits, and there are $256 - 64 = 192$ unrestricted output bits left. By varying the constants on $\Theta^0_{0,3,*}$, $\Theta^0_{1,2,*}$, and $\Theta^0_{3,0,*}$ for 2^{192} times, it is expected to obtain a preimage with guessing times of 2^{192}.

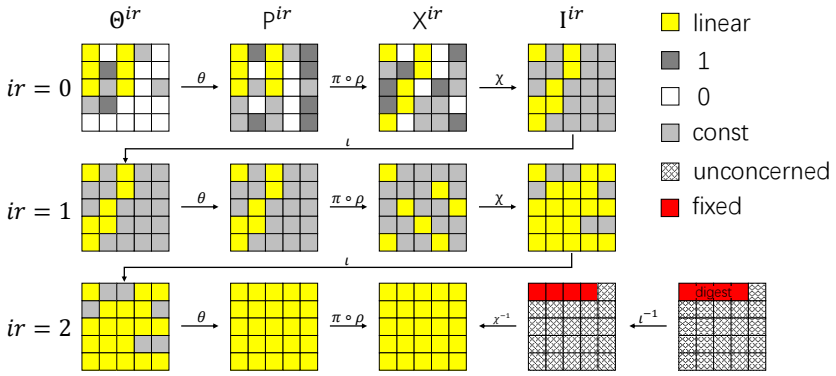

Fig. 3. The linear structure used in 3-round Keccak-256 [12].

3.2 The Allocating Approach

Li et al. put forward the allocating approach to divide the whole attack into two easier tasks [13]. They find a new linear structure that provides more degrees of freedom. However, the linear structure requires the following assumptions [13]:

$$\text{Restriction\_I: } \Theta^0_{x,3,z} = \Theta^0_{x,4,z} \oplus 1 \ (2 \le x \le 4, 0 \le z \le 63)$$

$$\text{Restriction\_II: } \bigoplus_{0 \le x \le 4, 0 \le z \le 63} \Theta^0_{x,4,z} = 0$$

Besides, to satisfy the padding rule, there is an extra restriction:

$$\text{Restriction\_III: } \Theta^0_{1,3,63} = \Theta^0_{1,4,63}$$

Hence, they add another message block using linear structure in [12] to satisfy the assumptions. Satisfying the assumptions is the first stage, and the second stage is to meet the output bits with the new linear structure. The complexity of both stages is lower than finding a preimage in one message block directly.

The first stage is shown in Fig. 4. Because the ι operation in the third round only affects the lane $I^2_{0,0,*}$, the restrictions on the Θ^0 of the second message block are equivalent to these restrictions on the I^2 of the first message block (replacing Θ^0 with I^2):

$$\text{Restriction\_I: } I^2_{x,3,z} = I^2_{x,4,z} \oplus 1 \ (2 \le x \le 4, 0 \le z \le 63)$$

$$\text{Restriction\_II: } \bigoplus_{0 \le x \le 4, 0 \le z \le 63} I^2_{x,4,z} = 0$$

$$\text{Restriction\_III: } I^2_{1,3,63} = I^2_{1,4,63}$$

Using linear structure in [12], they conclude that there are 64 degrees of freedom left. With the χ operation: $I_{x,y,z} = X_{x,y,z} \oplus (X_{x+1,y,z} \oplus 1) \cdot X_{x+2,y,z}$, they add

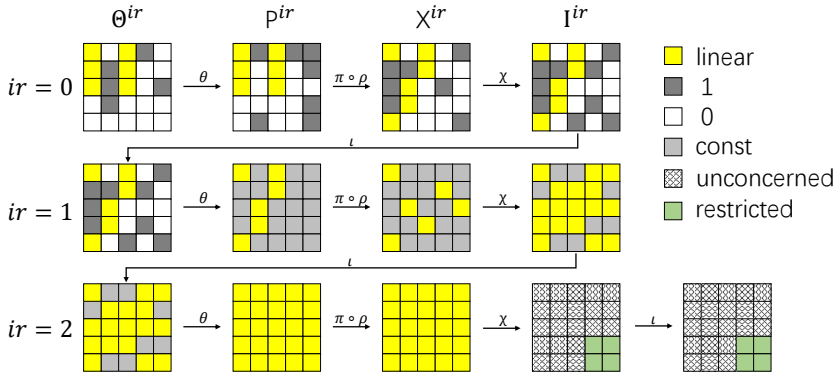

Fig. 4. The first stage of the allocating approach [13].

every 4 linear equations on X^2 to satisfy 2 Restriction_I (so-called the 4-for-2 strategy in [14]):

$$\begin{cases} X^2_{0,3,z} = c_0 \\ X^2_{0,4,z} = c_1 \\ X^2_{3,3,z} \oplus c_0 \oplus c_0 X^2_{4,3,z} \oplus X^2_{3,4,z} \oplus c_1 \oplus c_1 X^2_{4,4,z} = 1 \\ X^2_{4,3,z} \oplus X^2_{1,3,z} \oplus c_0 X^2_{1,3,z} \oplus X^2_{4,4,z} \oplus X^2_{1,4,z} \oplus c_1 X^2_{1,4,z} = 1 \end{cases}$$

where c_0 and c_1 are arbitrary constants to linearize the four bits $I_{3,3,z}$, $I_{4,3,z}$, $I_{3,4,z}$ and $I_{4,4,z}$. Therefore, there are $64 \div 4 \times 2 = 32$ Restriction_I satisfied in total. To satisfy another $192 - 32 = 160$ Restriction_I, together with Restriction_II and Restriction_III, they need to vary the column sums on Θ^1 and constants c_0 and c_1 $2^{160+1+1} = 2^{162}$ times to obtain the first message block.

The second stage is shown in Fig. 5. There are $10 \times 64 = 640$ variables and they add $5 \times 64 + 2 \times 64 - 2 = 446$ linear equations to control the sum of each column on Θ^0 and Θ^1 (2 equations are linear dependent). There are $640 - 446 = 194$ degrees of freedom left. Note that the column sums on Θ^0 should be fixed so that every bit on $P^0_{*,4,*}$ can be equal to "1", while the column sums on Θ^1 can be arbitrary constants. They vary the column sums on Θ^1 for $2^{256-194} = 2^{62}$ times to get the second message block.

Additionally, they find a way to balance the complexity of the two stages. For an ideal state P^0, it satisfies that $P^0_{x,3,z} = 0$ and $P^0_{x,4,z} = 1$. As shown in Fig. 6, every unsatisfied Restriction_I or Restriction_III will cause $P^0_{x,3,z} = P^0_{x,4,z} = 0$ or $P^0_{x,3,z} = P^0_{x,4,z} = 1$ which results in extra linear bit on I^0 and extra quadratic bit (or bits) on I^1. However, the effect of every unsatisfied Restriction_I or Restriction_III can be eliminated by using 1 degree of freedom to restrict the affected bit to constant (in Fig. 6, the affected bit is $I^0_{1,1,z}$, and other types of effects are similar). If there exists some unsatisfied Restriction_I or Restriction_III, Restriction_II can always be adjusted to satisfy by eliminating appropriate type of effect ($P^0_{x,3,z} = P^0_{x,4,z} = 0$ or $P^0_{x,3,z} = P^0_{x,4,z} = 1$). As a result, by allowing $n_I = 19$ Restriction_I or Restriction_III not to be satisfied, the

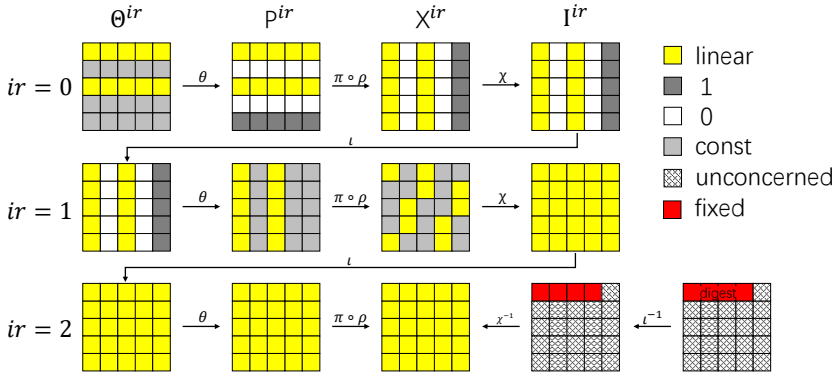

Fig. 5. The second stage of the allocating approach [13].

guessing times of the first stage are $\frac{2^{160+1}}{C_{160+1}^{n_I}} \approx 2^{80.06}$, and the guessing times of the second stage are $2^{62+n_I} = 2^{81}$.

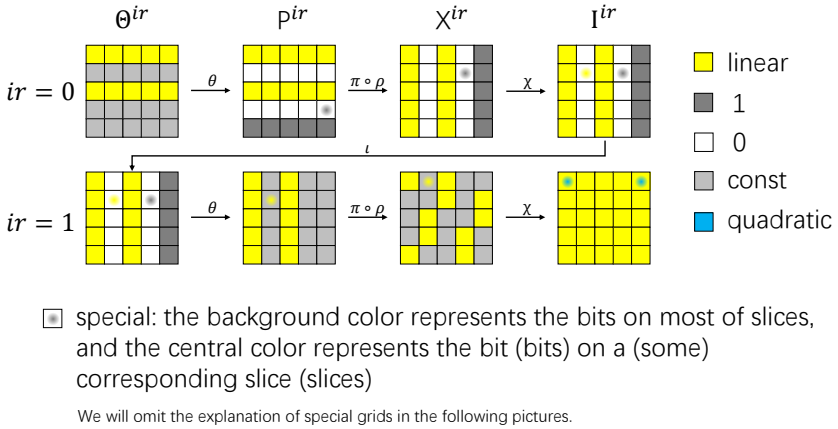

Fig. 6. The effect of unsatisfied restriction [13].

3.3 The 5-for-3 Strategy/The Iterating Strategy

Lin et al. propose the 5-for-3 strategy and the iterating strategy to improve the preimage cryptanalysis on 3-round Keccak-224/256 [14].

The 5-for-3 strategy is used for adding linear equations on X^2 to satisfy Restriction_I more efficiently. As introduced in Section 3.2, the original strategy uses every 4 degrees of freedom to satisfy 2 Restriction_I, namely the 4-for-2 strategy. By choosing appropriate constants and adding another linear equation,

the 5-for-3 strategy can use every 5 degrees of freedom to satisfy 3 Restriction_I. Within the same degrees of freedom, the 5-for-3 strategy can satisfy more Restriction_I, and provide a better state for the second stage. The linear equations of the 5-for-3 strategy are listed as follows [14]:

$$\begin{cases} X^2_{0,3,z} = 1 \\ X^2_{0,4,z} = 1 \\ X^2_{2,3,z} \oplus X^2_{2,4,z} \oplus X^2_{3,3,z} = 0 \\ X^2_{3,3,z} \oplus X^2_{3,4,z} = 0 \\ X^2_{4,3,z} \oplus X^2_{4,4,z} = 1 \end{cases}$$

The iterating strategy is also used in the first stage to provide a better state for the second stage. The first stage can be improved by using multi message blocks instead of only one message block. The goal of each message block is providing a better state (the better means satisfying more Restriction_I). When a better state is generated, the next message block will have more degrees of freedom than the previous message block, because the better starting state requires fewer degrees of freedom to eliminate the effects of unsatisfied restrictions. Thus, the next message block is more likely to generate a state better than before. Iteratively, a good-enough state can be generated eventually.

In summary, the attack of the first stage is an iterating process. The number of unsatisfied Restriction_I decreases when a new message block is found (the 5-for-3 strategy makes finding each message block efficient). When using the best starting state still can not generate a better state within acceptable guessing times, the iterating process ends with a good-enough state. An iterating process can be expressed by a table where k and k' represent the number of unsatisfied Restriction_I before and after the current message block, respectively. Their iterating process of preimage attack on 3-round Keccak-256 is shown in Table 3. For example, the starting state of the first message block is all '0' initial value which does not satisfy any of 192 Restriction_I, so the k of the first message block is 192. The first stage finally provides a state satisfying 189 Restriction_I, so the k' of the last message block is $192-189=3$. The guessing times are calculated as follows. There are 194 degrees of freedom, they use k degrees of freedom to eliminate the effects of unsatisfied restrictions. For the rest $194-k$ degrees of freedom, they use the 5-for-3 strategy to satisfy $\lfloor \frac{194-k}{5} \rfloor \times 3$ Restriction_I. The remaining $192-\lfloor \frac{194-k}{5} \rfloor \times 3$ Restriction_I are supposed to be satisfied randomly, and the probability of generating a state with at most k' unsatisfied Restriction_I is $C^{k'}_{192-\lfloor \frac{194-k}{5} \rfloor \times 3} \div 2^{192-\lfloor \frac{194-k}{5} \rfloor \times 3}$. Taking Restriction_II into account, the overall expected guessing times are $2^1 \times (2^{192-\lfloor \frac{194-k}{5} \rfloor \times 3}) \div (C^{k'}_{192-\lfloor \frac{194-k}{5} \rfloor \times 3})$.

Afterward, with $k'=3$, they construct the last message block with guessing times of $2^{256-(194-k')} = 2^{65}$ at the second stage. Hence, the overall guessing times are around $max\{2^{62.78+1}, 2^{65}\} = 2^{65}$ (the extra 1 is due to the padding rules).

Table 3. The iterating process of 3-round Keccak-256 [14].

message block id	k	k'	guessing times
# 1	192	91	$2^{5.49}$
# 2	91	48	$2^{11.97}$
# 3	48	41	$2^{8.31}$
# 4	41	37	$2^{10.23}$
# 5	37	35	$2^{10.80}$
# 6	35	33	$2^{12.65}$
# 7	33	32	$2^{12.38}$
# 8	32	31	$2^{13.40}$
# 9	31	30	$2^{14.49}$
#10	30	27	$2^{18.18}$
#11	27	25	$2^{19.32}$
#12	25	21	$2^{25.67}$
#13	21	10	$2^{48.62}$
#14	10	5	$2^{60.12}$
#15	5	4	$2^{61.33}$
#16	4	3	$2^{62.78}$

3.4 Linearizing Quadratic Equations

Liu et al. present a way to make cryptanalysis on round-reduced Keccak-384/512 by linearizing quadratic equations [21]. In this section, we only introduce their attack on 2-round Keccak-512. Their idea of solving quadratic equation systems (adding new variables to replace the quadratic terms) can be used in our attacks.

As shown in Fig. 7, the 8 yellow lanes on Θ^0 are set as variables. And they add $4 \times 64 = 256$ linear equations on Θ^0 to control the sum of each column. By simplifying the variables with the 256 equations, there remain $8 \times 64 - 256 = 256$ variables. After executing the χ operation in the first round, there remain $3 \times 64 = 192$ quadratic terms on I^0. They use another 192 variables to replace these quadratic terms. Hence, the state X^1 is linear with $256 + 192 = 448$ variables. To match the output bits, it requires 448 linear equations and 64 quadratic equations. They construct a linear equation system with 448 linear equations on 448 variables, and it is expected to have one solution. They use this solution to get the corresponding message and check the output bits. When considering the padding rule, they can get a preimage by varying the column sums on Θ^0 and constants on $\Theta^0_{4,0,*}$ for $2^{192+64+2} = 2^{258}$ times on average.

In summary, when the number of linear equations is larger than the sum of the number of variables and quadratic terms, it is possible to linearize the quadratic equations by adding variables replacing the quadratic bits.

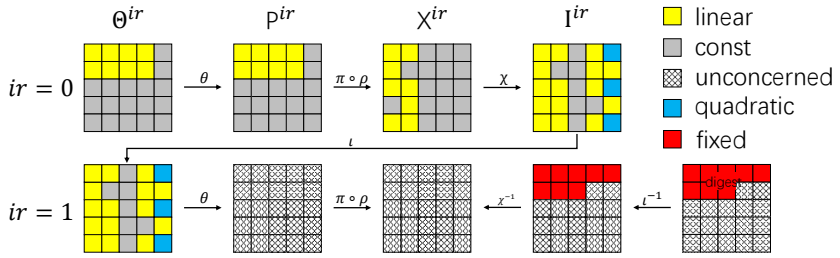

Fig. 7. Preimage attack on 2-round Keccak-512 [21].

4 Improved Attack on 3-Round Keccak-256

The preimage cryptanalysis of 3-round Keccak-256 is presented in this section. We first give an overview of our techniques. Then, we show the details of each stage respectively. It is expected that the guessing times of finding a preimage for 3-round Keccak-256 are 2^{52}.

4.1 Overview of Our Attack

Before introducing our attack, we review the three types of restriction defined in Section 3.2 which will be discussed frequently in this section.

$$\text{Restriction\_I: } I^2_{x,3,z} = I^2_{x,4,z} \oplus 1 \ (2 \le x \le 4, 0 \le z \le 63)$$

$$\text{Restriction\_II: } \bigoplus_{0 \le x \le 4, 0 \le z \le 63} I^2_{x,4,z} = 0$$

$$\text{Restriction\_III: } I^2_{1,3,63} = I^2_{1,4,63}$$

Restriction_I and Restriction_II are the prerequisites of the linear structure (proposed in [13]). Restriction_III is the prerequisite for the last message block so that the padding rule can be satisfied.

To improve the attack, we are facing two difficulties:

- The unsatisfied Restriction_I cost some degrees of freedom to eliminate the effects. However, if we want to reduce the number of unsatisfied Restriction_I, the complexity of the previous stage grows explosively. For example, we can calculate that (as introduced in Section 3.3) if we only require that the number of unsatisfied Restriction_I is no more than 10, the guessing times of the previous stage is around 2^{44}. If we require that the number of unsatisfied Restriction_I is no more than 3, the guessing times of the previous stage is around 2^{63} which is marginally unpractical. Furthermore, if we require that the starting state satisfy all the Restriction_I, the guessing times of the previous stage is around 2^{79}.
- Even if we get a starting state satisfying all the restrictions, the number of degrees of freedom is only 194 (as introduced in Section 3.2). To match the

output bits, we need to repeat trying different values of constants $2^{256-194} = 2^{62}$ times which is still unpractical.

To solve these difficulties, we modify the linear structure proposed in [13]. We discover that there exists a delicate modification which solves these difficulties at the same time. After modification, the prerequisites become a subset of previous prerequisites, and now the starting state satisfying all the prerequisites can be obtained within acceptable guessing times. Besides, the modified linear structure leaves more degrees of freedom which makes matching the output bits practical. The only negative effect of the modified linear structure is that it produces a small number of quadratic terms. However, the problem of these quadratic terms can be solved using the technique introduced in Section 3.4 [21] without extra costs.

Our attack includes three stages. The first two stages produce a particular starting state satisfying the prerequisites. At the third stage, we use the modified linear structure to match the output bits.

More specifically, at the first stage, we use the technique proposed in [14] as described in Section 3.3. We will get a good state which satisfies Restriction_II and 184 Restriction_I with guessing times of around $2^{47.10}$.

At the second stage, with a good starting state, there are $194 - (192 - 184) = 186$ degrees of freedom left. We use 3 degrees of freedom on each slice and set restrictions on 62 slices. With these restrictions, we can satisfy some restrictions on each slice with a certain probability. After many guesses, we will get a particular state satisfying Restriction_III and all those Restriction_I except at most 13 Restriction_I of type $x = 4$. The guessing times of the second stage are around $2^{51.52}$. More details of the second stage will be discussed in Section 4.3.

At the third stage, with the particular starting state, we can use the modified linear structure that has more degrees of freedom. Using this modified linear structure, we can match the output bits and obtain the last message block with guessing times of 2^{52}. More details of the third stage will be discussed in Section 4.4.

The first and the second stages introduced in this section match the attack we present in the experiment. However, the guessing times of the first and the second stages can be further decreased to $2^{43.67}$ and $2^{48.48}$, respectively. More details will be introduced in Appendix A. The bottleneck of the whole attack is still the third stage.

4.2 The First Stage

The first stage is almost the same as [14]. The only difference is that the state we require is more achievable (the number of required satisfied restrictions is fewer).

The target of the first stage is generating a state which satisfies Restriction_II and at least 184 Restriction_I.

As introduced in Section 3.3, we construct message blocks iteratively to obtain such a state. We list the iterating process in Table 4. After the 24-block iteration, we find an available state achieving the target. With this state, we have enough degrees of freedom for the next stage.

Table 4. The iterating process to get a state with only 8 unsatisfied Restriction_I.

message block id	k	k'	guessing times
# 1	192	85	$2^{6.93}$
# 2	85	67	$2^{4.97}$
# 3	67	57	$2^{4.82}$
# 4	57	48	$2^{6.18}$
# 5	48	44	$2^{6.66}$
# 6	44	42	$2^{6.95}$
# 7	42	41	$2^{7.48}$
# 8	41	37	$2^{10.23}$
# 9	37	36	$2^{9.97}$
#10	36	30	$2^{15.91}$
#11	30	29	$2^{15.65}$
#12	29	28	$2^{15.39}$
#13	28	26	$2^{17.94}$
#14	26	25	$2^{19.33}$
#15	25	22	$2^{23.96}$
#16	22	21	$2^{23.80}$
#17	21	20	$2^{25.53}$
#18	20	19	$2^{27.36}$
#19	19	18	$2^{27.26}$
#20	18	16	$2^{31.28}$
#21	16	14	$2^{35.74}$
#22	14	12	$2^{38.32}$
#23	12	9	$2^{46.58}$
#24	9	8	$2^{47.10}$

4.3 The Second Stage

The second stage builds a bridge between the first stage and the third stage. The first stage gives a good state which provides many degrees of freedom. The third stage requires a particular starting state (the motivation of this particular starting state will be introduced in Section 4.4). It should satisfy Restriction_III and all the Restriction_I of type $x = 2$ and $x = 3$. Furthermore, it should satisfy

at least 51 Restriction_I of type $x = 4$. Therefore, the target of the second stage is generating a required particular state with the provided degrees of freedom.

Because of the independence between different slices, we only consider the case in one slice. Note that the non-linear operation χ can be regarded as applying a 5-bit Sbox on each row. We focus on the property of the Sbox on two rows ($X^2_{*,3,z}$ and $X^2_{*,4,z}$). If we add the following three linear equations,

$$\begin{cases} X^2_{3,3,z} \oplus X^2_{0,4,z} \oplus X^2_{3,4,z} = 1 \\ X^2_{4,3,z} \oplus X^2_{4,4,z} = 1 \\ X^2_{2,3,z} \oplus X^2_{3,3,z} \oplus X^2_{2,4,z} = 0 \end{cases}$$

it yields that the probability of satisfying Restriction_I of type $x = 2$ and $x = 3$ is 0.625 and the probability of satisfying Restriction_I of type $x = 2$, $x = 3$ and $x = 4$ is 0.4375.

For example, if the inputs of the two 5-bit Sboxes are 00001 and 01010, the inputs satisfy the three linear equations. The outputs of the two 5-bit Sboxes are 00101 and 00011. The outputs satisfy Restriction_I of type $x = 2$ and $x = 3$, but they do not satisfy Restriction_I of type $x = 4$. After statistics, there are $2^5 \times 2^5 = 1024$ kinds of inputs of two 5-bit Sboxes while $1024 \div 2^3 = 128$ of them satisfy the three linear equations. Among these 128 kinds, 80 of them satisfy Restriction_I of type $x = 2$ and $x = 3$, and 56 of them satisfy Restriction_I of type $x = 2$, $x = 3$ and $x = 4$. If we suppose every kind of input occurs randomly, the probability of satisfying corresponding restrictions will be $80 \div 128 = 0.625$ and $56 \div 128 = 0.4375$, respectively.

We add linear equations on $186 \div 3 = 62$ slices and regard the bits on the rest 2 slices as random values. To get the result, we need to ensure that the Restriction_I of type $x = 2$ and $x = 3$ are all satisfied. In addition, we need to ensure that Restriction_III and at least 51 Restriction_I of type $x = 4$ are satisfied. The probability of satisfying all Restriction_I of type $x = 2$ and $x = 3$ is $0.625^{62} \times 0.5^{2\times 2} \approx 2^{-46.04}$. When the Restriction_I of type $x = 2$ and $x = 3$ are satisfied, the conditional probability of satisfying 3 types of Restriction_I in one slice is $0.4375 \div 0.625 = 0.7$. Finally, taking the padding rules (Restriction_III) into account, the probability of getting an available message block is $2^{-1} \times 2^{-46.04} \times \sum_{i+j>=51}(C^i_{62}0.7^i(1-0.7)^{62-i} \times C^j_2 0.5^j(1-0.5)^{2-j}) \approx 2^{-51.52}$.

4.4 The Third Stage

At the third stage, we use a modified linear structure to construct the last message block with the particular starting state (for convenience, we suppose there are exactly 51 Restriction_I of type $x = 4$ satisfied). The modified linear structure is shown in Fig. 8.

Here we introduce the motivation of the modified linear structure.

Above all, we need to increase the degrees of freedom. There are several possibilities. We can cut down some of the 128 linear equations (controlling column sums) on Θ^1 to save some degrees of freedom. We can also add extra

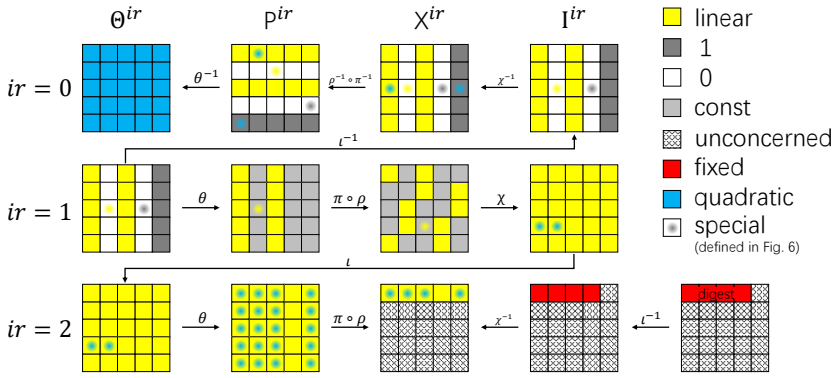

Fig. 8. The modified linear structure.

variables on different places to increase some degrees of freedom. As introduced in Section 3.4 [21], the number of produced quadratic terms must be less than or equal to the number of redundant linear equations. Thus, we need to choose the way producing minimal number of quadratic terms. Cutting down every linear equation (controlling column sums) on Θ^1 is not a good choice because some columns on Θ^1 turn from constant to variables which leads to tens of quadratic terms. Adding an extra variable on the first round is also not a good choice because a great number of quadratic terms are produced after two rounds. Adding an extra variable on $\Theta^1_{3,*,*}$ or $\Theta^1_{4,*,*}$ is still not a good choice because a column on Θ^1 turns from constant to variables which causes many quadratic terms similarly. Among all choices, adding extra variables on $\Theta^1_{1,*,*}$ is the best choice because every extra variable only produces 4 quadratic terms.

On the other hand, we want to reduce the cost of matching the starting state. We need to ensure that the 6 lanes $P^0_{x,y,*}(2 \leq x \leq 4, 3 \leq y \leq 4)$ are constants. Among them, the relation (equal or opposite) of each bit pair $P^0_{x,3,z}$ and $P^0_{x,4,z}$ needs to be the same with the relation of corresponding bit pair on the starting state. As shown in Fig. 9, to control the required constants on P^0, we need to control some constants on X^0. Thus, we need to decide the type of the setting of each row on Θ^1. For each row, 2 types of settings satisfy that all bits are linear with 2 degrees of freedom, and 2 types of settings satisfy that there are at most 4 kinds of quadratic terms (2 quadratic terms appear on X^0 and the others appear on I^1) with 3 degrees of freedom. Among them, only the first type of setting satisfies that the bit $X^0_{4,y,z}$ is constant, and its value is 1. Therefore, the rows of $\Theta^1_{*,0,z}$, $\Theta^1_{*,1,z}$ and $\Theta^1_{*,3,z}$ must be the first type. After that, 2 lanes ($a' = P^0_{2,3,*}$ and $b' = P^0_{3,3,*}$) will be constant 0, and 3 lanes ($x' = P^0_{2,4,*}$, $y' = P^0_{3,4,*}$ and $z' = P^0_{4,4,*}$) will be constant 1. However, the rows of $\Theta^1_{*,2,z}$ can be set to any type (while the number of quadratic terms produced by the third type and the fourth type must be under the limit of linearizing the quadratic equation systems). Thus, some bits on lane $c' = P^0_{4,3,*}$ can be constant 1 and others will be constant 0. As a result, we require that the particular starting state obtained

by the second stage satisfies Restriction_I except at most 13 Restriction_I of type $x = 4$. With a particular starting state, we are able to match the starting state without extra cost by carefully selecting the types of settings on $\Theta^1_{*,2,z}$.

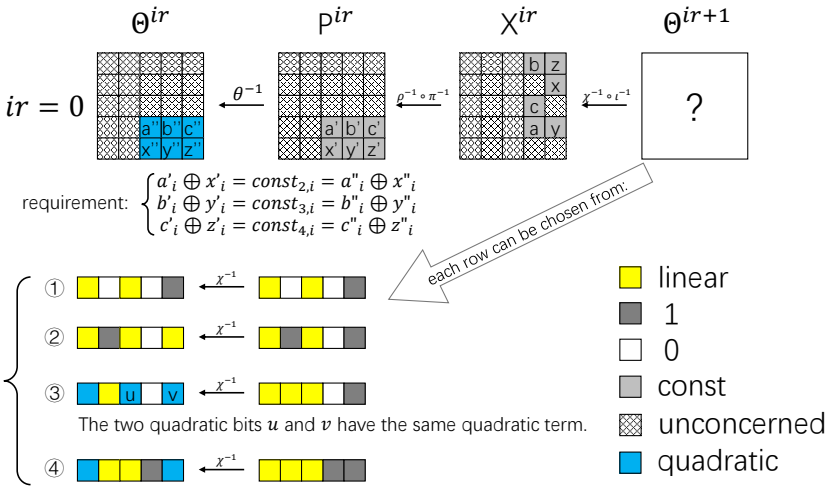

Fig. 9. Controlling the constants.

Then we give a detailed description of the modified linear structure. The variables are set on Θ^1. Except for the original 640 variables ($\Theta^1_{0,*,*}$ and $\Theta^1_{2,*,*}$), we add 13 more variables on $\Theta^1_{1,2,*}$. The 13 bits are selected according to the 13 unsatisfied Restriction_I in the particular starting state and the operation ρ. Besides, on the slice (13 in total) where corresponding bit $\Theta^1_{1,2,z}$ is chosen, the bit $\Theta^1_{3,2,z}$ is set as constant 1. Other bits on $\Theta^1_{1,*,*}$ and $\Theta^1_{3,*,*}$ are set as constant 0, and the bits on $\Theta^1_{4,*,*}$ are set as constant 1.

On the one hand, we invert the state Θ^1 one round backward. The inverse of the χ operation can be written as:

$$\begin{cases} X^0_{0,2,z} = I^0_{0,2,z} \oplus (I^0_{1,2,z} \oplus 1) \cdot (I^0_{2,2,z} \oplus (I^0_{3,2,z} \oplus 1) \cdot I^0_{4,2,z}) \\ X^0_{1,2,z} = I^0_{1,2,z} \oplus (I^0_{2,2,z} \oplus 1) \cdot (I^0_{3,2,z} \oplus (I^0_{4,2,z} \oplus 1) \cdot I^0_{0,2,z}) \\ X^0_{2,2,z} = I^0_{2,2,z} \oplus (I^0_{3,2,z} \oplus 1) \cdot (I^0_{4,2,z} \oplus (I^0_{0,2,z} \oplus 1) \cdot I^0_{1,2,z}) \\ X^0_{3,2,z} = I^0_{3,2,z} \oplus (I^0_{4,2,z} \oplus 1) \cdot (I^0_{0,2,z} \oplus (I^0_{1,2,z} \oplus 1) \cdot I^0_{2,2,z}) \\ X^0_{4,2,z} = I^0_{4,2,z} \oplus (I^0_{0,2,z} \oplus 1) \cdot (I^0_{1,2,z} \oplus (I^0_{2,2,z} \oplus 1) \cdot I^0_{3,2,z}) \end{cases}$$

For the 13 selected rows, substituting the value 1 for $I^0_{3,2,z}$ and $I^0_{4,2,z}$, we have:

$$\begin{cases} X^0_{0,2,z} = I^0_{0,2,z} \oplus (I^0_{1,2,z} \oplus 1) \cdot I^0_{2,2,z} \\ X^0_{1,2,z} = I^0_{1,2,z} \oplus I^0_{2,2,z} \oplus 1 \\ X^0_{2,2,z} = I^0_{2,2,z} \\ X^0_{3,2,z} = 1 \\ X^0_{4,2,z} = 1 \oplus (I^0_{0,2,z} \oplus 1) \cdot (I^0_{1,2,z} \oplus I^0_{2,2,z} \oplus 1) \end{cases}$$

So for each selected row, there are two quadratic bits $X^0_{0,2,z}$ and $X^0_{4,2,z}$, two linear bits $X^0_{1,2,z}$ and $X^0_{2,2,z}$ and a constant bit $X^0_{3,2,z}$. Similar to the technique in [21], we introduce 2×13 new variables to replace these quadratic bits. After that, the state develops as Fig. 8 shows till P^0. To match the starting state, we add 320 equations to restrict the state so that bits on $\Theta^0_{*,4,*}$ are satisfied. Due to the property of θ operation and the satisfaction of Restriction_I and Restriction_III (and the 13 changed constants on $P^0_{4,3,*}$), the state will match the starting state successfully.

On the other hand, we develop the state Θ^1 two rounds forward. Similar to the original linear structure, we set 128 restrictions on Θ^1 to control the column sums and prevent the diffusion of the variables. Then the linear structure produces 2×13 quadratic bits on $I^1_{0,3,*}$ and $I^1_{1,3,*}$. Similarly, we introduce another 2×13 new variables to replace these quadratic bits. And the state develops as Fig. 8 shows till X^2. To restrict the 256 output bits, we have to add 256 equations.

In summary, the third stage consists of six steps.

- Construct the state Θ^1 by setting bits on $\Theta^1_{0,*,*}$ and $\Theta^1_{2,*,*}$ as variables, bits on $\Theta^1_{1,*,*}$ and $\Theta^1_{3,*,*}$ as 1, and bits on $\Theta^1_{4,*,*}$ as 0.
- Determine which 13 rows on Θ^1 should be changed (change $\Theta^1_{1,2,z}$ from 0 to variable and change $\Theta^1_{3,2,z}$ from 0 to 1) according to the 13 unsatisfied Restriction_I.
- Invert the state Θ^1 one round backward (introduce 2×13 new variables to replace the quadratic bits) and add 320 equations to satisfy the particular starting state.
- Add 128 linear equations on Θ^1 to control the column sums and prevent the diffusion of the variables.
- Develop the state Θ^1 two rounds forward (introduce another 2×13 new variables to replace the quadratic bits) and add 256 equations to meet the output bits.
- Construct an equation system with $320 + 128 + 256 = 704$ linear equations on $640 + 13 + 2 \times 13 + 2 \times 13 = 705$ variables.

However, the $2 \times 13 + 2 \times 13 = 52$ new variables are not independent of the original $640 + 13 = 653$ variables because each new variable is equal to the expression of the replaced quadratic bit. Thus, the equation system also contains 52 quadratic equations, which must be satisfied randomly. So we need to vary the column sums on Θ^1 $2^{13\times2+13\times2} = 2^{52}$ times and solve the equation systems repeatedly, hoping to get an assignment of the variables that satisfies all quadratic equations at the same time.

5 Fast Rebuilding Method

In Section 4, we have introduced the preimage attack on 3-round Keccak-256, which requires building and solving the equation system and verifying the solution 2^{52} times. However, dealing with each guess is troublesome and time-consuming. In this section, we propose a technique named the fast rebuilding method to make it easier to rebuild and solve the equation system.

5.1 The Bottleneck of Previous Method

As introduced in Section 4, in each guess, we vary the column sums and construct an equation system with $320+128+256 = 704$ linear equations and 52 quadratic equations on 705 variables. The first 320 linear equations are added to satisfy the starting state, the 128 linear equations are added to control the column sums, and the last 256 linear equations are added to meet the output bits. Among them, the first 320 linear equations are easy to deal with because the varied column sums are not involved with these equations. We can simplify the equation system with these equations before guessing the column sums. The following 128 linear equations are not too difficult to deal with because the varied column sums are not involved with the coefficients of variables. But it is hard to deal with the last 256 linear equations because the varied column sums are deeply involved with these equations. Therefore, we have to determine all the column sums at once to deduce the coefficients of the 256 equations and solve the equation system after that.

Rebuilding and solving an equation system with at least 256 variables and verifying the 1600-bit solution are time-consuming. Thus, we hope to find a method which determines the coefficients of a small number of equations and solves the equation system with a small number of variables for each guess. In Section 5.2, we will introduce our fast rebuilding method which varies constants on $\Theta^1_{1,4,*}$ instead of column sums. As a result, it is possible to rebuild and solve 10 equations on 61 variables and verify a 61-bit solution for each guess on average.

5.2 Vary Constants on $\boldsymbol{\Theta^1_{1,4,*}}$

Before introducing the fast rebuilding method, we present the motivation for this technique.

The first difference is that, by varying constants on $\Theta^1_{1,4,*}$ instead of the column sums, we are able to do some useful preprocessing. We first regard the bits on $\Theta^1_{1,4,*}$ as variables. Similar to Section 4, these bits produce some quadratic bits (introducing some new variables to replace these quadratic bits) and result in some quadratic equations. After introducing some new variables, we will get some new linear equations and some quadratic equations about the new variables and the replaced quadratic bits. We simplify the equation system with the new linear equations, and we determine the constant value of the bits on $\Theta^1_{1,4,*}$ later. The second difference is that, after preprocessing, we can solve the equation system hierarchically. We determine the constant value of the bits on $\Theta^1_{1,4,*}$ which we regard as variables just now. Then, the quadratic equations will return to linear. Furthermore, now every equation only depends on the constant value of a bit on $\Theta^1_{1,4,*}$ instead of all the bits on $\Theta^1_{1,4,*}$. In other words, when we vary the value of every constant bit, only a small number of linear equations will be changed.

The comparison of the two methods are shown as follows.

Algorithm 1 Previous method.

build an equation system.
For each guess:
    Determine all the varied constants.
    Generate the linear equations.
    Solve the equation system and verify the quadratic equations.

Algorithm 2 Fast rebuilding method.

Build an equation system.
Do some preprocessing.
For each guess:
    Undo the simplification of a small number of linear equations.
    Determine a small number of varied constants (others keep unchanged).
    Generate a small number of linear equations.
    Simplify and solve the equation system and verify the quadratic equations.

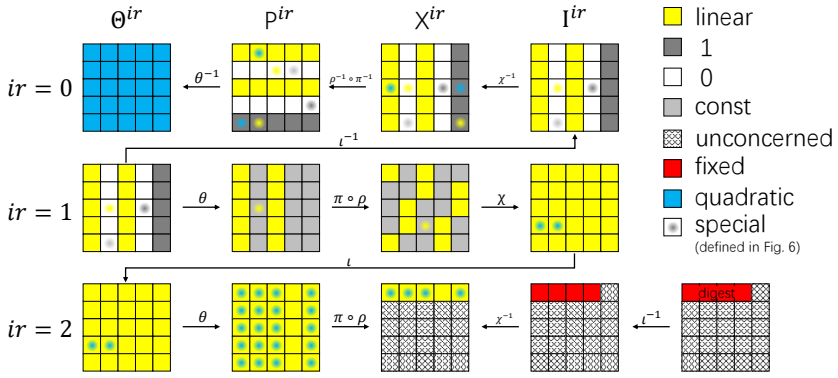

Fig. 10. Vary the constants on $\Theta^1_{1,4,*}$.

Then we give a detailed description of the fast rebuilding method. As shown in Fig. 10, we vary the constants on $\Theta^1_{1,4,*}$ instead of the column sums on Θ^1 to improve the attack. Note that when we set some bits on $\Theta^1_{1,4,*}$ as constant 1, the previous round is also linear, and the bits on $P^0_{1,4,*}$ do not affect the restrictions on the starting state. Moreover, we determine the value of $\Theta^1_{1,4,*}$ later, and we regard these bits as variables first, as shown in Fig. 11.

For the analysis in Section 4, every bit on $\Theta^1_{1,4,*}$ results in $3+2=5$ quadratic bits on X^0 and I^1. However, 2 of the 3 quadratic bits on X^0 ($X^0_{2,4,z}$ and $X^0_{4,4,z}$) have the same quadratic term. So we introduce $64\times(5-1)=256$ new variables to replace these quadratic bits. Then we build an equation system with 704 linear equations on $705+64+64\times(5-1)=1025$ variables. Besides, we have $52+256=308$ quadratic equations that must be satisfied.

Because we will vary constants on $\Theta^1_{1,4,*}$ later instead of column sums, the 704 linear equations are fixed during different guesses. Therefore, we can simplify

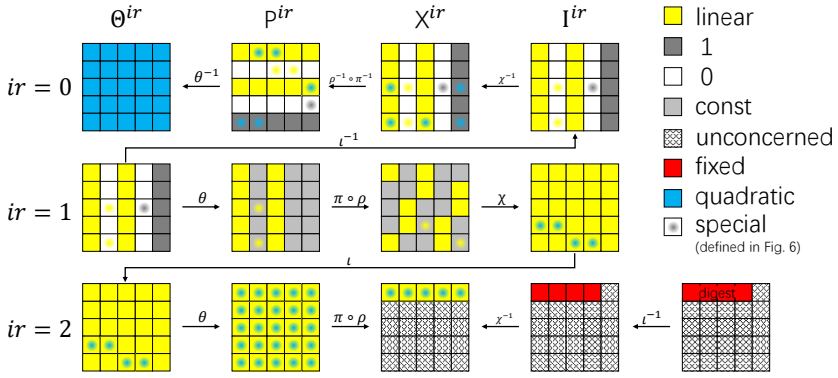

Fig. 11. Regard bits on $\Theta^1_{1,4,*}$ as variables.

the equation system with these 704 linear equations before varying the constants on $\Theta^1_{1,4,*}$. After that, the equation system consists of $1025 - 704 = 321$ free variables with no linear equation and 308 quadratic equations. We hope to find an assignment of these 321 variables satisfying all 308 quadratic equations at the same time. It seems that the current situation is worse than before. However, these changes make it possible to solve the equation system hierarchically.

Next, it is the time to assign values for $\Theta^1_{1,4,*}$. Every time we assign a value for a bit $\Theta^1_{1,4,z}$, 4 quadratic equations will become linear because the quadratic bits caused by this bit are originally linear. Including assigning value for the bit $\Theta^1_{1,4,z}$, there are 5 linear equations that can be added in total. If we vary the values of 64 bits in order (0...0000, 0...0001, 0...0010, 0...0011, 0...0100, ...), then for some continuous guesses, the first few bits keep unchanged, and the corresponding linear equations keep unchanged. With this idea, we do not assign values for all 64 bits at once. Instead, every time we value some bits, we simplify the equation system with the corresponding linear equations.

For example, consider that the values of the first 52 bits will not change for some continuous guesses. We assign value for the first 52 bits and use the $(1+4) \times 52 = 260$ linear equations to simplify the equation system. The equation system consists of $308 - 4 \times 52 = 100$ quadratic equations on $321 - 260 = 61$ variables. Then, we assign value for the rest 12 bits one by one, but this time we only simplify the added linear equations (rather than the whole equation system, including the quadratic equations) while adding every 5×1 linear equations. After all bits are valued, we will get a simplified (solved) equation system with 61-bit solution and $100 - 4 \times 12 = 52$ quadratic equations, which need to be verified.

Note that we explain the following four points.

- We value the first 52 bits at once because the rest 61 variables can be expressed by a 64-bit word easily, and the simplification is not the bottleneck. The process of valuing 52 bits can be divided into more than one step if necessary.

- We can use some linear expressions (concatenated by AND or XOR) to express a quadratic equation so that the quadratic equation is easier to be simplified or verified.
- We do not simplify the quadratic equations when there remains only a small number of variables because the simplification costs more time than the gain of reducing the number of variables.
- There are $5 \times 12 = 60$ linear equations on 61 variables, which means there is 1 degree of freedom left. There are many ways to make use of this degree of freedom. We can get two solutions of linear equations on average (but we still need to verify each of them), or we can add another linear equation to slightly improve the probability of a quadratic equation, or we can just add an independent linear equation (such as let a variable be 0) for convenience.

As a result, usually for a new guess, we need to undo the last $2 \times 5 = 10$ added linear equations on average (because the last 2 valued bits changed on average) and add another $2 \times 5 = 10$ linear equations. Then we simplify the linear equations and get the solution of linear equations. Last we verify the quadratic equations with the solution. In most cases, we deal with a small number of linear equations on 61 variables, and it is very fast. Although the preprocessing (value the 52 bits and simplify the equation system) is time-consuming, it happens every 2^{12} guesses. With these techniques, we can guess around 1.01 million times per second on a personal computer.

6 Experiments

In this section, we introduce our experimental results. The experiments are running on Sunway TaihuLight supercomputer which provides more than ten thousand nodes. All these experimental results are finished within half a week. The running time and the running speed of each stage are shown in Table 5. The whole input message blocks (26 in total) and the state after finishing each stage are shown in Appendix B.

Table 5. The runing time of each stage.

stage	[a]running time	[b]solving speed	expected guessing times	actual guessing times	[c]expected complexity	[c]actual complexity
the first stage	83	[d]1.79	$2^{46.9}$	$2^{48.9}$	$2^{53.77}$	$2^{55.80}$
the second stage	30	6.50	$2^{51.5}$	$2^{49.3}$	$2^{56.51}$	$2^{54.33}$
the third stage	360	[e]5.43	$2^{52.0}$	$2^{52.6}$	$2^{57.27}$	$2^{57.92}$

[a] Unit: 1000 nodes · hour.
[b] Unit: million guesses / (second · node).
[c] Unit: equivalent 3-round Keccak calls.
[d] The solving speed of the first stage is slower because we need to calculate the number of satisfied restrictions, while at the other two stages, we just need to check whether restrictions are all satisfied.
[e] It is able to run the third stage with 1.01 million guesses per second on a personal computer. So we think the speed of a node is around $5.43 \div 1.01 \approx 5.38$ times faster than a personal computer.

7 Conclusion

In this paper, we propose two techniques to improve the preimage attack of 3-round Keccak-256.

The first technique is a modified linear structure. Based on the linear structure proposed in [13], we select some extra bits on $\Theta^1_{1,2,*}$ as variables, so that more degrees of freedom will be left. However, these variables generate some quadratic bits which result in quadratic equations. We use the technique linearizing quadratic equations in [21] to solve the equation system. By selecting the 13 extra bits carefully, we can deal with the unsatisfied restrictions while minimizing the number of generated quadratic equations. Using this technique, we are able to decrease the guessing times of preimage attack of 3-round Keccak-256.

The second technique is a fast rebuilding method to speed up the construction of equation systems. If we regard the varied constants as variables, the equation system can be further simplified. The change of each constant bit only causes a small number of linear equations to vary. By guessing the constant bits hierarchically, we only need to deal with a small number of linear equations for each guess on average. With this technique, the solving time for each guess will decrease.

As a result, the guessing times of finding a preimage for 3-round Keccak-256 are decreased from 2^{65} times to 2^{52} times, and the solving time of each guess decreases from 2^9 3-round Keccak calls to $2^{5.3}$ 3-round Keccak calls. Moreover, we find a preimage of all '0' digest for 3-round Keccak-256. It is noted that our cryptanalysis is still far from threatening the security of full-round Keccak.

References

1. Bertoni, G., Daemen, J., Peeters, M., Assche, G.: The Keccak reference, 2011.
2. Dworkin, M.: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015.
3. Naya-Plasencia, M., Röck, A., Meier, W.: Practical Analysis of Reduced-Round Keccak. In: Bernstein, D.J., Chatterjee, S. (eds) Progress in Cryptology – INDOCRYPT 2011. LNCS vol. 7107, pp. 236–254. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25578-6_18
4. Dinur, I., Dunkelman, O., Shamir, A.: New Attacks on Keccak-224 and Keccak-256. In: Canteaut, A. (eds) Fast Software Encryption. FSE 2012, LNCS vol. 7549, pp. 442–461. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34047-5_25
5. Dinur, I., Dunkelman, O., Shamir, A.: Collision Attacks on Up to 5 Rounds of SHA-3 Using Generalized Internal Differentials. In: Moriai, S. (eds) Fast Software Encryption. FSE 2013, LNCS vol. 8424, pp. 219–240. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-662-43933-3_12
6. Qiao, K., Song, L., Liu, M., Guo, J.: New Collision Attacks on Round-Reduced Keccak. In: Coron, JS., Nielsen, J. (eds) Advances in Cryptology – EUROCRYPT 2017, LNCS vol. 10212, pp. 216–243. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56617-7_8
7. Song, L., Liao, G., Guo, J.: Non-full Sbox Linearization: Applications to Collision Attacks on Round-Reduced Keccak. In: Katz, J., Shacham, H. (eds) Advances in Cryptology – CRYPTO 2017, LNCS vol. 10402, pp. 428–451. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_15
8. Guo, J., Liao, G., Liu, G., Liu, M., Qiao, K., Song, L.: Practical Collision Attacks against Round-Reduced SHA-3. In: J Cryptol 33, pp. 228–270 (2020). https://doi.org/10.1007/s00145-019-09313-3
9. Das, S., Meier, W.: Differential Biases in Reduced-Round Keccak. In: Pointcheval, D., Vergnaud, D. (eds) Progress in Cryptology – AFRICACRYPT 2014, LNCS vol. 8469, pp. 69–87. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06734-6_5
10. Dinur, I., Morawiecki, P., Pieprzyk, J., Srebrny, M., Straus, M.: Practical Complexity Cube Attacks on Round-Reduced Keccak Sponge Function. In: IACR Cryptol. ePrint Arch, pp. 259 (2014). https://ia.cr/2014/259
11. Huang, S., Wang, X., Xu, G., Wang, M., Zhao, J.: Conditional Cube Attack on Reduced-Round Keccak Sponge Function. In: Coron, JS., Nielsen, J. (eds) Advances in Cryptology – EUROCRYPT 2017. LNCS vol. 10211, pp.259–288. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_9
12. Guo, J., Liu, M., Song, L.: Linear Structures: Applications to Cryptanalysis of Round-Reduced Keccak. In: Cheon, J., Takagi, T. (eds) Advances in Cryptology – ASIACRYPT 2016. LNCS vol. 10031, pp. 249–274. Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_9
13. Li, T., Sun, Y.: Preimage Attacks on Round-Reduced Keccak-224/256 via an Allocating Approach. In: Ishai, Y., Rijmen, V. (eds) Advances in Cryptology – EUROCRYPT 2019. LNCS vol. 11478, pp. 556–584. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_19
14. Lin, X., He, L., Yu, H.: Improved Preimage Attacks on 3-Round Keccak-224/256. In: IACR Transactions on Symmetric Cryptology 2021, Issue 3, pp. 84–101 (2021). https://doi.org/10.46586/tosc.v2021.i3.84-101

15. Pei, J., Chen, L.: Preimage attacks on reduced-round Keccak hash functions by solving algebraic systems. IET Inf. Secur. 1–13 (2022). https://doi.org/10.1049/ise2.12103
16. He, L., Lin, X., Yu, H.: Improved Preimage Attacks on 4-Round Keccak-224/256. In: IACR Transactions on Symmetric Cryptology 2021, Issue 1, pp. 217–238 (2021). https://doi.org/10.46586/tosc.v2021.i1.217-238
17. Dinur, I.: Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over GF(2). In: Canteaut, A., Standaert, FX. (eds) Advances in Cryptology – EUROCRYPT 2021. LNCS vol. 12696, pp. 374–403. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_14
18. Wei, C., Wu, C., Fu, X., Dong, X. He, K., Hong, J., Wang, X.: Preimage Attacks on 4-round Keccak by Solving Multivariate Quadratic Systems. In: Cryptology ePrint Archive (2021). https://ia.cr/2021/732
19. Kumar, R., Mittal, N., Singh, S.: Cryptanalysis of 2 Round KECCAK-384. In: Chakraborty, D., Iwata, T. (eds) Progress in Cryptology – INDOCRYPT 2018. LNCS vol 11356, pp.120–133. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05378-9_7
20. Rajasree, M.S.: Cryptanalysis of Round-Reduced Keccak Using Non-linear Structures. In: Hao, F., Ruj, S., Sen Gupta, S. (eds) Progress in Cryptology – INDOCRYPT 2019. LNCS vol. 11898, pp. 175–192. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35423-7_9
21. Liu, F., Isobe, T., Meier, W., Yang, Z.: Algebraic Attacks on Round-Reduced Keccak. In: Baek, J., Ruj, S. (eds) Information Security and Privacy. ACISP 2021. LNCS vol. 13083, pp. 91–110. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90567-5_5
22. Bertoni, G., Daemen, J., Peeters, M., Assche, G.: Cryptographic sponge functions, 2011.
23. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic sponge functions. (January 2011) http://sponge.noekeon.org/CSF-0.1.pdf

A The Further Improved First Two Stages

As introduced in Section 4.3, the target of the second stage is generating a state satisfying Restriction_I and Restriction_III except at most 13 Restriction_I of type $x = 4$. We use 186 degrees of freedom on some linear equations to satisfy these restrictions with a certain probability. However, based on the 4-for-2 strategy, if we set the two constants (c_0 and c_1) as opposite values instead of random values, the results will be better. Here we compare four strategies of setting restrictions on X^2. Similar to Section 4.3, we only focus on two 5-bit Sboxes ($X^2_{*,3,z}$ and $X^2_{*,4,z}$). The equations of these strategies are listed as follows.

Strategy A:

$$\begin{cases} X^2_{0,3,z} = 1 \\ X^2_{0,4,z} = 1 \\ X^2_{2,3,z} \oplus X^2_{2,4,z} \oplus X^2_{3,3,z} = 0 \\ X^2_{3,3,z} \oplus X^2_{3,4,z} = 0 \\ X^2_{4,3,z} \oplus X^2_{4,4,z} = 1 \end{cases}$$

Strategy B:

$$\begin{cases} X^2_{4,3,z} = 0 \\ X^2_{4,4,z} = 1 \\ X^2_{2,3,z} \oplus X^2_{2,4,z} \oplus X^2_{3,4,z} = 0 \\ X^2_{0,3,z} \oplus X^2_{3,3,z} \oplus X^2_{3,4,z} = 1 \end{cases}$$

Strategy C:

$$\begin{cases} X^2_{3,3,z} \oplus X^2_{4,3,z} \oplus X^2_{3,4,z} = 0 \\ X^2_{4,3,z} \oplus X^2_{4,4,z} = 0 \\ X^2_{2,3,z} \oplus X^2_{4,3,z} \oplus X^2_{2,4,z} = 1 \end{cases}$$

Strategy D:

$$\begin{cases} X^2_{3,3,z} \oplus X^2_{0,4,z} \oplus X^2_{3,4,z} = 1 \\ X^2_{4,3,z} \oplus X^2_{4,4,z} = 1 \\ X^2_{2,3,z} \oplus X^2_{3,3,z} \oplus X^2_{2,4,z} = 0 \end{cases}$$

We use some symbols to express some points to compare. For one strategy, it uses every d degrees of freedom to add d linear equations on X^2. Then the probability of satisfying Restriction_I of type $x = 2$ and $x = 3$ is p_a, and the probability of satisfying all 3 types of Restriction_I is p_b. We restrict r slices, and there are $u = 64 - r$ slices unrestricted. So we use $D = r \times d$ (within 186) degrees of freedom in total. After that, the probability of getting a message block satisfies Restriction_I of type $x = 2$ and $x = 3$ is $P_a = p_a^r \times 2^{-2\times u}$. When the Restriction_I of type $x = 2$ and $x = 3$ are satisfied, the conditional probability of satisfying Restriction_I of type $x = 4$ is $P_b = \sum_{i+j>=51} C_r^i (p_b/p_a)^i (1 - p_b/p_a)^{r-i} \times C_u^j (1/2)^j (1 - 1/2)^{u-j}$. At last, considering the Restriction_III, the overall probability of getting an available message block is $P = 2^{-1} \times P_a \times P_b$. The comparison of these strategies is listed in Table 6.

Among these strategies, Strategy D is introduced in Section 4.3. Besides, Strategy B shows the best result, although we do not make use of it while doing the experiments.

By using Strategy B, we only require 184 degrees of freedom. In other words, the first stage needs to provide a state satisfying Restriction_II and only 182 Restriction_I instead of 184 Restriciton_I. The bottleneck of the first stage is constructing a message block with $k = 11$ and $k' = 10$. And the guessing times of constructing this message block are $2^{43.67}$. As a result, we can decrease the guessing times of the first two stages to around $2^{43.67}$ and $2^{48.48}$, respectively.

Note that Strategy C has the best P_a, which can be used in preimage attack on 3-round Keccak-224. In [14], by iterating the first stage for 10 message blocks, they get a state satisfying 126 Restriction_I (type $x = 3$ and $x = 4$) with guessing times of $2^{25.87}$. Based on this state, we construct another message block using Strategy C$'$ (Strategy C moving a step to the right).

Strategy C$'$:

$$\begin{cases} X^2_{4,3,z} \oplus X^2_{0,3,z} \oplus X^2_{4,4,z} = 0 \\ X^2_{0,3,z} \oplus X^2_{0,4,z} = 0 \\ X^2_{3,3,z} \oplus X^2_{0,3,z} \oplus X^2_{3,4,z} = 1 \end{cases}$$

Table 6. Comparison of different strategies.

	Strategy A	Strategy B	Strategy C	Strategy D
d	5	4	3	3
r	37	46	62	62
u	27	18	2	2
D	185	184	186	186
p_a	32/32	64/64	96/128	80/128
p_b	32/32	40/64	40/128	56/128
P_a	$2^{-54.00}$	$2^{-36.00}$	$2^{-29.73}$	$2^{-46.04}$
P_b	$2^{-1.00}$	$2^{-11.48}$	$2^{-30.31}$	$2^{-4.48}$
P	$2^{-56.00}$	$2^{-48.48}$	$2^{-61.04}$	$2^{-51.52}$

Using $194 - (128 - 126) = 192$ degrees of freedom, we add restrictions on all 64 slices. The probability of satisfying all Restriction_I (type $x = 3$ and $x = 4$) is $p_a^{64} \approx 2^{-26.56}$. Considering Restriction_II and Restriction_III, the guessing times of constructing this message block are around $2^{28.56}$. After that, the second stage is the same with [14]. The results of preimage attack on 3-round Keccak-224 are shown in Table 7. Our fast rebuilding method can also be used for preimage attack on 3-round Keccak-224.

Table 7. The detailed results of preimage attack on 3-round Keccak-224.

First Stage	Second Stage	Overall Guessing Times	Reference
–	–	2^{97}	[12]
2^{66}	2^{31}	2^{66}	[13]
$2^{35.62}$	2^{38}	2^{38}	
2^{33}	2^{31}	2^{33}	[14]
2^{28}	2^{32}	2^{32}	
$2^{28.56}$	2^{31}	2^{31}	This paper

B An Instance of Preimage of 3-Round Keccak-256

The instance of preimage of 3-round Keccak-256 is shown in table 8.

Table 8: An instance of preimage of 3-round Keccak-256 (in big-endian order).

the 1^{st} message block				
b37313233b373133	5555555555555555	aaaaaaaaaaaaaaaa	aaaaaaaaaaaaaaaa	cc4c8cecc4c8cecd
ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	ffffffffffffffff
19d9b989919d9b99	ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	9919d9b9919d9b98
ffffffffffffffff	ffffffffffffffff	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 2^{nd} message block				
673fd6621904c5d4	c3cabb7867a65d30	8ff3b33ccae1b20d	a351c99bc0bd1a7b	0d22cf2e21c47bfe
48b8605866ddd794	b7b016f753eafc76	e2a72433a1de16eb	c5b77a83b99a4631	5ad7b7c347b83b0a
d2e3796fd0061aea	40a3ec9b7c8f1edb	a8044a16da4e35e4	24e2753d38030867	00989952ab6b66e7
e63843f8ce001643	107a40611e7f7b98	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 3^{rd} message block				
52e2cac588dee9fe	d5276803a3b8acef	e78ad424128b6cbb	f27c0bfd6bb3ea82	e116a542a5335bff
cdcc9bcd253a6fc9	8fa64585abb8dbef	7201c2c7e974f73d	cb0d7080c315c4f1	a424bba861d56df4
493126dc26070589	8293b4dbe162b665	ff1106edc2035d0b	90c6d779b7cc43a1	5d237e29860042d8
d15c9df5c0a777bf	926c87b5dcb1685e	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 4^{th} message block				
157c8c05aaaa492b	bc1a97672988816c	3de7e1c9452c9248	d97e56828795edbb	7c6f5bc91f53272c
ee4edd50c5b9662e	8e3864fb2c7dd15d	02c01a547b30f5ed	b735d6bbbca3167c	d4bafe63f322f89a
5aea022c2111eeb8	01d2a0445bf11961	72c22a10f7250601	2501e88923728778	2b8aa27721c9545b
5712af5c13567857	13f5ad228c093f73	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 5^{th} message block				
cbf72d75c71e5b43	c8d9cf65a00b7f1c	1437205506f22845	248af05bd3ed53f2	9945cd9c5af8aa6f
c68d43517a3a147b	39792961700804bc	0eed56f50ad29f67	90f50893c88c1347	615167dc6e81956e
4c88f8cdfdb0fe34	ca810a19281e15f4	8eb245291c783975	4418f699495b5320	82104b0a0acd1ba4
cd0962f74574bfa6	fcd0cabf4aab7de6	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 6^{th} message block				
0bbf8d72bdef5c0c	336d8e97bd32c874	13d271488d3378ee	5406ea75de2457b3	12517a561e92b75c
4c2a88ed00888fc9	f30baa06130bb284	5b17117860b7f544	4d4213363d858801	937944066cb9f5e9
086c178c9bc40d39	9162327ca8758466	6a3b2947134c2cfa	d92f33aece39b658	8ef518fc5b5c56f7
a906ebbec99f6945	26f579be884fe099	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 7^{th} message block				
ef87b752d3da4f5e	14476a96f4fdfb3f	1011c947493e62b1	6c6098539711bf18	69a7dcfe84a2604a
5dce33448829d83d	fe63fcd82f8a2bfd	2695088161e57899	50ab4559d5fa5aba	acdef158d0873b14
bdbe87c3beb786a1	8a6708c21bf3826a	7ca9deabc01b4ac0	f3a5d6c14dfd4c92	87974f43c468d186
942dffa0a3ee75a5	42de16335da1d72d	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

the 8^{th} message block
10b20df59b980828 f83f31894599cc75 0d21d228382322b2 02be27186c2bfb9c 82d9c11eb4ec2f3b
2e2640f216db8ee4 9f4313f2f74a24ee 1ed0f4ea04f34a02 9b164ad04fd76b74 678effbcf2ed0ea0
dc070aae098d8fef cf7186039aba338d dd2ba62247b6de33 2488f4e83d639a3f 7060d8a0f74a50cc
2e597fd3ec4fd07e 7c1c55f96e8c9da8 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
the 9^{th} message block
af42ed15a74f5230 21c36f088e0c99f5 d772187d68f55f41 67120ad709a9c72a 64f1735265a2e261
1cea3adbfd461622 ec46ba5013f35b01 4cbb2b5c847f6da2 d1fe597844a076a9 e99914c4b423a1a1
1e2e4d5d31812963 a602c428bedaf9a4 17c1dcdfb1e433e0 31cfc8e6ae88bca7 3ed473c8cdc5682f
9b44a41e3dd6d46e ad60e342064c98be 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
the 10^{th} message block
b3ac5b4e443fc7b1 27678230d925bfde 559415437ff8ef0b 226460c2517587af 78a65f11879d4349
6661b06a19f0e57f 512cebce2bc08e9b 3493ae909047e8c9 176807105e558612 303720b31558c933
c1a0184a7a2b1162 ed6ceb30acd1cce0 f177cd65554610df fcf6ae8ca520e1a6 aea4e94843f71e42
b54f7f090e1dfc72 c3971c7c2609a38b 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
the 11^{th} message block
f4f7b1904308b12d 86c5e19b42015969 10d0e3a53817567e 3a238e14d4197799 4e3d746c0601b274
746c501e430512d8 b6174a5d33f32292 7395112085c75ed0 9989f62d02acfd24 e4888fc2b536c9cd
49d3ea243de4bcd6 2e60ec0942ca343b 4f1e30f103bdbabf c5289c52486654b4 5172b85107091490
b8a27f7f60fdd837 6e6a457edbd51b25 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
the 12^{th} message block
bff58bb4668a1769 ed767a519c636835 9162f0cff40338ab 22cb7d6247c01890 5489b7f129ace874
053eaaeab7328636 6a133c2f7a90d569 9673f98fb2594d32 8605e5cb4a97e173 ddda14f4daf7faa3
2770269f0beb47c5 247ba9c42c701aeb 1f66825de19c7209 8fea7fb94cf366e2 985739388d19a616
e25ddb2559b5ab9f b34a532dee346cc3 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
the 13^{th} message block
f17454aaf03b4b64 4193c95ad351809a c2412fbc53f69c0d 88cb87d86bdd44fe 645b0eaf7c59a06b
9a392c1ee040d397 2d209b3fadf188c3 c551b4f208f670c8 e508216c92418d53 1ca714044770a1f3
72398f7b14d059cc 0895d6ae4c555437 cb69f9abf697023d d74f502fc91fb37e f6bd04bfda371855
2a33d37a42e5fdf7 5b84e2ebb6bbb83a 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
the 14^{th} message block
3284e7ce5d8633f7 532cd6dc071ff777 7ccafd3d5b565e12 84a6c00b3045328d 5b8e1d2f57f217ab
b5437acc9d8b2e4b d7bab63f968afad3 21ba3782d3413c54 fcf5d3429dc9b263 2f1d54ad3cfadad0
28d1acbc1f72c7ca 7f4023b0c468a000 0007def828359bf6 d0de41cfc7416ab6 4a4a43b1de3eb074
4274ac0db96edde8 a7b515a4b08543dc 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
the 15^{th} message block
1aef05eab8208394 ba3864046462d17f c6beb1b1763733c4 872120212dfa094f b05b20a21c70ba41

b019241583274fe6	98cc13d6ff996bb8	c2bea1d48afa4c4e	417ea34eb2754bf9	ef31d0730d2b79c2
a1b3f7639b13eeda	146f6670bcda6e18	012312bdef3ef43b	89b395294b8f1aae	f948a05519405544
77a874dd44ef2119	f45e17d8dbea0655	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 16^{th} message block				
74c66ef88b2b4168	be5372c2b6b7ee4d	6664096c043abcac	617a90ee574b0ca2	a3cb5cb0007cff6f
c9abfeec68ce240f	a720bcea8050bba3	320eaa769487f4cd	a01561e3b9f0c7d6	c0588d89eab44ec0
cf9ab13c8529ad9c	fe059b52b372e45d	9cb74d3b5e9e54a9	66238a7191961d1e	9f886318ef485f83
bc7efc3c0c66b25e	f11cf52ba9fbaeaa	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 17^{th} message block				
702523782bf27f28	0d2d11db773286aa	14258b2c80bd4265	176bc38ddc9ffd4d	e52d5cdb4c42cad4
319ca089d3bc82c6	b1da456c04898151	48fcd328946f20af	71fe1738cf330fa3	0d839e27de510434
8c296a263644966b	dec4b376c9f51e2c	e7a798e2a368d632	2bf44c727667a616	c3a78947b82ee2df
432c5faa2467e91c	8e28558373f2ccab	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 18^{th} message block				
bdbd2ec0173d12e2	8127e99e1e68ac01	6aed746a37ec37e1	b6acabe2e5205f78	08dd2692cd23e449
c35bab2509daad65	66c07eb0f26d4ad0	66c6c2f858690f88	1db47b83b690ca3a	e844051c319613b2
2c8fb88df528784d	588c19ccf589437c	95569822ca90dbc0	e30f8fb10c3c4e3e	5c3fe40fff6e4031
01cebf41f465991b	272d3d4934ab211a	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 19^{th} message block				
857821d22905540b	c91119948f4d1f84	276c9be260ef9b1e	f0bae5eab0b3fe3e	f57a494425fdccbc
702dc15fddefa613	6e8812bbb041aa5d	41f7482b072fb4fb	48c9617858bcfc72	4e22da96b1403036
811b165a23a9990e	f29d56d160c5d1f4	066ab1d95aa25b22	f1128c74a8daf545	a10b6c8186bd68f9
165af8be7f09def5	9fe3cc9f68c347b8	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 20^{th} message block				
1070470238f83d8c	f9cb66ff16662c4e	417fca6cf9317681	9d61e9facc0f7020	c789407b23c8e36a
f33acfb05a173c15	d098a5bb77c1437d	577ca0e67db1e8d2	39ecd0c3a0dd5a81	fcc90cec19ac1e43
b7c7b06ce385d0a2	58369aa97e72bb65	030a784e8325be06	45a786d3b1fe323c	e31a648ee8967318
0d1eff874f84ac9f	b79ec22290b02473	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 21^{st} message block				
116b68b605184cb7	a192c9a2d9aa150b	aea88b4a7d0e5178	5ab675b1d8511278	7159f23b66a4f188
440a652226bf8f99	6e3b0e860fdc99e7	0a59992c8a68dc10	c1faaa6c40db9a6f	f3cb579b9b86beff
c89fe6f93dad9680	b7d360b638e2bdc7	fe91812de7e38586	71b9b20325c5e541	119e0a287a54ce05
3eaf427da45496fb	81866c0f1a40928a	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 22^{nd} message block				
a0f3738f6aeeccbe	9120a472e84f68f0	8c6940ddcc7c9ee1	1bb88438e71ffd55	af492b1447a50e09
de6f4948cfd162e0	4386b55d47c1dbcb	08d34730280926fd	4a0a0674a4c23142	d2ea9a30db4108ba
6b34cb5abc6a6e13	d8aa92e41f2576b5	4597a65bcdaac7f6	3de783ff4bc3feea	82141a8262689299
04c5d5c9df0bd71a	192332928d0188fa	0000000000000000	0000000000000000	0000000000000000

0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 23^{rd} message block				
eeff1728343ebd82	69baaddf82153598	5a3af1eea558c47c	8344bef775da1646	29b87f96982738c1
d3e7b22bdb5f40b5	300ed144acf6bf7f	cd6de082fb5e6cd5	7965200f0b08279a	9ed9b7e3ada25b32
5b9b5d794569bd67	714816b864e114ef	214d3ed5be2592eb	c199d4e4f557713c	12033da2d7b07c81
03cfe69aaed6dc7c	a079305b3ecf8b96	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 24^{th} message block				
9fb4ab0a18ff198c	b6b26934534e6a1f	3f8270bfe9307d70	8cc2f3d05bc097af	364e5af73ec63613
05beab7cddc18bf0	323fe5e345103d25	bccb06af4244d312	db713d6b9e6fea01	0725ec27b96a1a86
2003f682a1cf5b05	bcd0dd1a4781fee0	98aeaa1bcead1a79	bada8cf402143ec8	43090b5c2830ee64
2754e09f7dffcc75	90a6e3ee492bd82f	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the state after the first stage				
3d99e3864dd978a7	60373513737dad0d	f979bbe6c728af9d	15e42f9fd704d3f8	ed0b9f1898371841
ed0fa702f897b231	3d386c110829eeeb	17a11ff1a7a75bc9	66dc2fae30780caa	f3418af070433128
03d15d5a9d12d422	03f2032b3ab8fc82	2cc7f5a6122a7646	f0fee590a8d5b7f9	b0d697274be5aff8
ea0fceb7f3f7cfe7	55e0f8617153ce65	b06128f02c49bc1e	cf9860c288fcad15	39bc989909a4eb23
39330abeaf39ba5f	9ff28abe0328b25f	5ffed70fd3b643e1	30779d3d770352ea	c6536e66f65b14dc
XOR values of Restriction_I				
		ef9fffffffffffff	ffeffdffffffffff	ffeff6ffffffffff
the 25^{th} message block				
b1adf08a211d6d0d	11f90ac31b2801f1	38c7d0d4afd7b5fd	0488b6216e6620c3	72b994921e5ca1e2
2bc35243a851f791	5d7511d23c7e36c3	b7a037018beee7d7	6cb07a0aae79c9bd	caed1b6979e7da0b
21994d3c6bb02703	63aa9ad7bbb8fb2b	d966739029e76db9	0323564413532ca0	332357afbfe97532
2cc33bf6a3318a47	35ad85a24504164d	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the state after the second stage				
26f6b72f64c7a017	904944b040829111	66bf3559df68f59e	08a764ce5bef39f5	663f9898bf40b461
c8bc323ea04684a7	3df4e9b993e3fea5	956263eb51febd9f	3cfa16d9554f8461	a105f32f994d27be
8051093937899dc8	78e471aa3ec14b76	7f146246fc095604	63313678fc6db963	fcabeefb6454ddc4
7cf1199b99f90719	907969a13f49db6d	d684663e6cab54fa	f27dbd3b15a57fda	7fef1fdb19e90f9a
af7d47dcf3e24ebf	c99bead4c54adef7	297b99c19354ab05	0d8242c4ea5a8025	c012a804cc129041
XOR values of Restriction_I				
		ffffffffffffffff	ffffffffffffffff	bffdb7dfd5fb9fdb
the 26^{th} message block				
4609b0573539b4c1	5fcf06c9ff60e4d3	1f474596b8cdae7c	cbc2c23e89cb4884	9b1fdf168988b62d
912e8a1dac5bb4e7	0f98d492841cc3ad	43e605d5354569f4	8fc78a891508739c	9ee8a4d4aaa04800
61773534510b59ad	6d36e2685d6213f5	21f3ab896958f7f0	8e335d8f2b2193b6	ff9787ac8a80f8c1
2563a1b895e43759	a215548a28b6e665	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the final state				
0000000000000000	0000000000000000	0000000000000000	0000000000000000	80e171e3611cc7f1
e28c9fbe1b6a1374	7860b435de30e34b	aeb784f6d747cbb3	12ea874996aaf826	c37af932b711fb86
2adce91fe7865ac2	29743ce03dea5172	0575f66fe6f4570c	22d91197c038438c	9075e1e53959830c

0c5aa5f2f4ba2607 7bd2c4c129b5f319 c3ac95e3aef8a884 755eacf9401d8879 c71817c519df211a
f28db1602f43a61d 39070676354565be 7117cc77c82348dc 4c8feae15571e374 4f8b1c9b9294d282
3-round digest
0000000000000000 0000000000000000 0000000000000000 0000000000000000