IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 0, No. 0, pp. 0–0. DOI:10.46586/tosc.v0.i0.0-0

Practical Preimage Attacks on 3-Round Keccak-256 and 4-Round Keccak[r=640, c=160]

Xiaoen Lin[1], Le He[2] and Hongbo Yu[1,3,4(✉)]

[1] Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, People's Republic of China, lxe21@mails.tsinghua.edu.cn, yuhongbo@mail.tsinghua.edu.cn
[2] School of Cyber Engineering, Xidian University, Xi'an, 710126, People's Republic of China, hele@xidian.edu.cn
[3] Zhongguancun Laboratory, Beijing, People's Republic of China
[4] State Key Laboratory of Cryptography and Digital Economy Security, Tsinghua University, Beijing, 100084, People's Republic of China

Abstract. Recently, linear structures and algebraic attacks have been widely used in preimage attacks on round-reduced Keccak. Inherited by pioneers' work, we make some improvements for 3-round Keccak-256 and 4-round Keccak[r=640, c=160]. For 3-round Keccak-256, we introduce a three-stage model to deal with the unsatisfied restrictions while bringing more degrees of freedom at the same time. Besides, we show that guessing values for different variables will result in different complexity of solving time. With these techniques, the guessing times can be decreased to 2^{52}, and the solving time for each guess can be decreased to around $2^{5.2}$ 3-round Keccak calls. As a result, the complexity of finding a preimage for 3-round Keccak-256 can be decreased to around $2^{57.2}$. For 4-round Keccak[r=640, c=160], an instance of the Crunchy Contest, we use some techniques to save degrees of freedom and make better linearization. Based on these techniques, we build an MILP model and obtain an attack with better complexity of around $2^{60.9}$. The results of 3-round Keccak-256 and 4-round Keccak[r=640, c=160] are verified with real examples.

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

1 Introduction

The Keccak function, designed by Bertoni et al. [BDPA11b], is a family of cryptographic functions, which was submitted to the public competition held by NIST in 2008. In 2015, Keccak was standardized as Secure Hash Algorithm 3 (SHA-3) [Dwo15]. Up to now, plenty of security analyses have been conducted by public community.

In this paper, we mainly focus on preimage attacks. Bernstein gave theoretical preimage attacks slightly faster than brute force for up to 8-round Keccak [Ber10]. Naya-Plasencia et al. proposed practical preimage attacks on 2-round Keccak-224/256 [NRM11]. Morawiecki et al. applied rotational cryptanalysis to preimage attacks on 4-round Keccak [MPS14]. Then, Guo et al. developed a technique named *linear structure* and gave preimage attacks on different variants for up to 4 rounds [GLS16]. For round-reduced Keccak-224/256, Li et al. used the *allocating approach* and gave a practical preimage attack on 3-round Keccak-224, along with attacks improving results on 3-round Keccak-256 and 4-round Keccak-224/256 [LS19]. Lin et al. further refined the results on 3-round Keccak-224/256 by using the *5-for-3 strategy* and the *iterating strategy* [LHY21]. Pei et al. satisfied the linear structure probabilistically and made improvement on 3-round Keccak-256 while the results on 4-round Keccak-256 and 3-round Keccak-512 are also improved [PC22].

Table 1: Summary of preimage attacks on round-reduced Keccak.

	[a] 2-round	[a] 3-round	[a] 4-round	Reference
Keccak-224	–	–	[b] $2^{221} \times 2^{0} = 2^{221}$	[MPS14]
	$2^{0} \times 2^{10} = 2^{10}$	$2^{97} \times 2^{8} = 2^{105}$	$2^{213} \times 2^{8} = 2^{221}$	[GLS16]
	–	$2^{38} \times 2^{9} = 2^{47}$	$2^{207} \times 2^{9} = 2^{216}$	[LS19]
	–	–	$2^{192} \times 2^{9} = 2^{201}$	[HLY21]
	–	$2^{32} \times 2^{9} = 2^{41}$	–	[LHY21]
	–	–	[c] $2^{0} \times 2^{202} = 2^{202}$	[Din21]
	–	–	[d] $2^{184} \times 2^{4} = 2^{188}$	[WWF+21]
Keccak-256	–	–	[b] $2^{252} \times 2^{0} = 2^{252}$	[MPS14]
	$2^{0} \times 2^{10} = 2^{10}$	$2^{192} \times 2^{6} = 2^{198}$	$2^{251} \times 2^{6} = 2^{257}$	[GLS16]
	–	$2^{81} \times 2^{9} = 2^{90}$	$2^{239} \times 2^{9} = 2^{248}$	[LS19]
	–	–	$2^{218} \times 2^{9} = 2^{227}$	[HLY21]
	–	$2^{65} \times 2^{9} = 2^{74}$	–	[LHY21]
	–	–	[c] $2^{0} \times 2^{231} = 2^{231}$	[Din21]
	–	–	[d] $2^{215} \times 2^{4} = 2^{219}$	[WWF+21]
	–	$2^{64.79} \times 2^{9} = 2^{73.79}$	[d] $2^{210} \times 2^{4} = 2^{214}$	[PC22]
	–	[e] $2^{52} \times 2^{5.2} = 2^{57.2}$	–	Section 4
Keccak-384	–	–	[b] $2^{378} \times 2^{0} = 2^{378}$	[MPS14]
	$2^{129} \times 2^{10} = 2^{139}$	$2^{322} \times 2^{10} = 2^{332}$	–	[GLS16]
	[f] $2^{89} \times 2^{0} = 2^{89}$	–	–	[KMS18]
	$2^{113} \times 2^{11} = 2^{124}$	$2^{321} \times 2^{9} = 2^{330}$	$2^{371} \times 2^{9} = 2^{380}$	[Raj19]
	$2^{92} \times 2^{11} = 2^{103}$	$2^{270} \times 2^{12} = 2^{282}$	$2^{365} \times 2^{9} = 2^{374}$	[LIMY21]
	–	–	[c] $2^{128} \times 2^{231} = 2^{359}$	[Din21]
Keccak-512	–	–	[b] $2^{506} \times 2^{0} = 2^{506}$	[MPS14]
	$2^{384} \times 2^{8} = 2^{392}$	$2^{482} \times 2^{8} = 2^{490}$	–	[GLS16]
	$2^{321} \times 2^{10} = 2^{331}$	$2^{475} \times 2^{8} = 2^{483}$	–	[Raj19]
	$2^{257} \times 2^{12} = 2^{269}$	$2^{439} \times 2^{12} = 2^{451}$	–	[LIMY21]
	–	–	[c] $2^{0} \times 2^{487} = 2^{487}$	[Din21]
	–	[d] $2^{424} \times 2^{12} = 2^{436}$	–	[PC22]
	–	–	[g] $2^{504.58} \times 2^{0} = 2^{504.58}$	[QHD+23]
Keccak[b=800]	[h] solved	$2^{7} \times 2^{5} = 2^{12}$	[d] $2^{62} \times 2^{3.4} = 2^{65.4}$	[MS10, GLS16, WWF+21, BDH+a]
	–	–	[e] $2^{56.5} \times 2^{4.4} = 2^{60.9}$	Section 5

[a] Each result is shown by "guessing times × solving time = complexity". Unit: equivalent 2-round (or 3-round, 4-round) Keccak calls. For the entries without note, the "guessing times" will be the solved number of linear equation systems, and the "solving time" will be our estimated results according to the rest degrees of freedom for comparisons (similar to [LIMY21]).
[b] Achieved through rotational cryptanalysis.
[c] Based on solving multivariate equation systems with high degree. Significant memory complexity is required.
[d] In the original paper, results are derived from calculating bit operations of solving equation systems, which is idealistic. Here we re-calculate the "solving time" by the same rule of other entries or by their experimental results.
[e] These "solving time" show the actual running time according to experimental results.
[f] Obtained by the time-memory trade-offs attack which requires 2^{87} memory complexity.
[g] Resulting from the Meet-in-the-Middle attack requiring 2^{108} memory complexity.
[h] Solved by SAT-based attack (without concrete complexity).

For 4-round Keccak-224/256, He et al. [HLY21] and Wei et al. [WWF+21] gave further attacks by using different techniques including the *freedom reuse strategy* 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 [KMS18]. Rajasree allowed non-linear parts on linear structure and improved the results on round-reduced Keccak-384/512 for up to 4/3 rounds [Raj19]. Liu et al. continued to enhance the results by making full use of the linear relations through the *relinearization technique* [LIMY21]. Dinur devised a polynomial method that can be applied to 4-round Keccak where results on 4-round Keccak-384/512 are further improved [Din21]. Qin et al. used the Meet-in-the-Middle attack and gave results on 4-round Keccak-512 [QHD+23]. The results of preimage attacks on round-reduced Keccak are summarized in Table 1.

Our contribution. First, this paper gives an improved preimage attack on 3-round Keccak-256. We combine several techniques from previous papers and modify the linear structure to overcome the difficulties faced by earlier studies. The modified structure leaves more degrees of freedom and requires a new starting state which is easier to match. We propose a three-stage model that extends an additional intermediate stage generating the required new starting state. Besides, we observe that guessing different variables leads to the property: when rebuilding the equation system, only a small number of linear equations will change. To leverage this property, we propose a technique to rebuild and solve the equation system faster. With these techniques, the guessing times of finding a preimage for 3-round Keccak-256 can be decreased to 2^{52}, and the solving time for each guess can be decreased to $2^{5.2}$ 3-round Keccak calls. Finally, we demonstrate the first practical preimage attack on 3-round Keccak-256.

Second, this paper gives a practical preimage attack on 4-round Keccak[r=640, c=160]. We use some techniques to make better linearization such as selecting different bits as variables, constructing two candidates for the previous message block, taking more output bits into consideration, and making full use of enumerated variables. With these techniques, we build an MILP model optimizing the highest probability of matching the digest and result in an attack with around $2^{56.5}$ guessing times. Thus, another new solution to the Crunchy Contest [BDH+a] is obtained.

Organization. In Section 2, we give some preliminaries and notations about Keccak. The related work and literature review are discussed in Section 3. Then Section 4 presents the preimage attack on 3-round Keccak-256. Afterward, Section 5 shows the preimage attack on 4-round Keccak[r=640, c=160]. Conclusions of this paper are provided in Section 6.

2 Preliminaries

2.1 Sponge Construction

The sponge construction is a mode of operation that builds a sponge function [BDPA11a]. 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 turn the construction absorbs an r-bit message block by XORing it with the first r bits of the state. After that, the state will be operated by the Keccak-f permutation. In the squeezing phase, the construction squeezes every first r bits of the state as part of output, until the total length of the output is greater than or equal to the required length ℓ. Similar to the absorbing phase, the state will be operated by the Keccak-f permutation after each squeeze. At last, the digest is obtained by truncating the output to the required length ℓ.

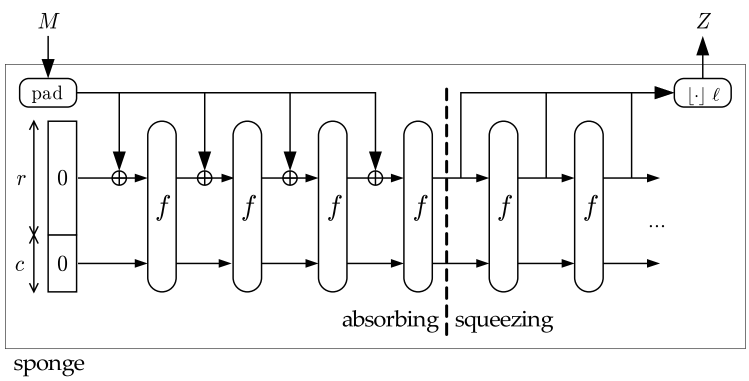

Figure 1: The sponge construction [BDPA11a].

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. As shown in Fig.2, the b-bit state can be described as 5×5 w-bit lanes. The state can be denoted as $A_{x,y,z}$, where $0 \leq x, y < 5$, $0 \leq z < w$ ($w = b/25$).

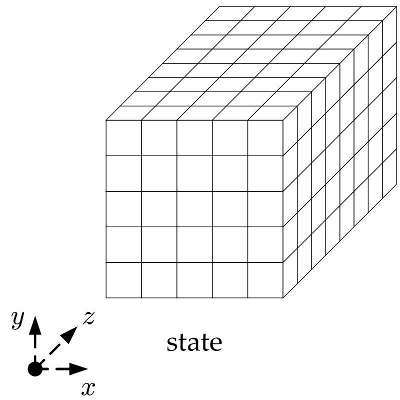

Figure 2: The Keccak-f state [BDH+b].

The permutation Keccak-f[b] consists of $12 + 2log_2(w)$ 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:

$$\begin{aligned}
\theta &: A_{x,y,z} = A_{x,y,z} \oplus \bigoplus\nolimits_{i=0\sim 4} (A_{x-1,i,z} \oplus A_{x+1,i,z-1}) \\
\rho &: A_{x,y,z} = A_{x,y,(z-r_{x,y})} \\
\pi &: A_{x,y,z} = A_{x+3y,x,z} \\
\chi &: A_{x,y,z} = A_{x,y,z} \oplus (A_{x+1,y,z} \oplus 1) \cdot A_{x+2,y,z} \\
\iota &: A_{0,0,z} = A_{0,0,z} \oplus RC_z
\end{aligned}$$

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 w. $r_{x,y}$ is a constant shown in Table 2, and RC_z is the bit in position z of a round-dependent constant RC which is shown in Table 3.

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

2.3 SHA-3 Standard

There are four SHA-3 versions standardized by NIST [Dwo15]. The parameters are $r = 1600 - 2\ell$ and $c = 2\ell$, where $\ell \in \{224, 256, 384, 512\}$. The difference between 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.

Table 3: The constant RC of round ir.

ir	RC	ir	RC	ir	RC
0	0x0000000000000001	8	0x000000000000008a	16	0x8000000000008002
1	0x0000000000008082	9	0x0000000000000088	17	0x8000000000000080
2	0x800000000000808a	10	0x0000000080008009	18	0x000000000000800a
3	0x8000000080008000	11	0x000000008000000a	19	0x800000008000000a
4	0x000000000000808b	12	0x000000008000808b	20	0x8000000080008081
5	0x0000000080000001	13	0x800000000000008b	21	0x8000000000008080
6	0x8000000080008081	14	0x8000000000008089	22	0x0000000080000001
7	0x8000000000008009	15	0x8000000000008003	23	0x8000000080008008

2.4 Properties of Matching the Output Bits

Notice that the output digest is finally truncated from the state after the last ι operation, which is just a constant-XOR and can be directly inversed. One operation backward, the state before the last χ operation can also be partially recovered from the digest.

The χ operation can be regarded as applying a 5-bit Sbox on each row. Suppose the input of the Sbox is $a_0a_1a_2a_3a_4$ and the output is $b_0b_1b_2b_3b_4$. We list some properties related to this paper that have been thoroughly discussed in previous works [GLS16, LS19, Raj19, HLY21, LIMY21].

- If $b_0b_1b_2b_3$ are known, there are two possibilities of $a_0a_1a_2a_3a_4$, and four restrictions can be obtained. Restriction on a_i (if a_i is fixed) or $a_i \oplus a_j$ (if a_i and a_j are unfixed) where $i, j \in \{0, 1, 2, 3, 4\}$ can bring a gain of 2^1.
- If $b_0b_1b_2$ are known, restriction on a_i or $a_i \oplus a_{i+2}$ (depending on b_{i+1}) where $i \in \{0, 1\}$ can bring a gain of 2^1.
- If b_i where $i \in \{1, 2\}$ is known while b_{i+1} and b_{i+2} are unconcerned, restriction on a_i can bring a gain of $0.75/0.5 \approx 2^{0.58}$. Plus, extra restriction of $a_{i+1} = 1$ or $a_{i+2} = 0$ can uplift the gain from $2^{0.58}$ to 2^1.

2.5 Notations

We use capital Greek letters Θ, P, Π, X, I with a superscript number (from 0 to 3, 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, $\Theta^0_{*,*,z}$ is a slice and $\Theta^0_{*,y,*}$ is a plane.

3 Related Work

In this section, we introduce some existing attack thoughts related to our analysis. These techniques greatly inspire our research. Hereinafter, we will introduce the *linear structure*, *allocating model* with improved linear structure, *iterating strategy* using *5-for-3 strategy*, and the *relinearization technique* linearizing quadratic structures.

3.1 The Linear Structure

The main idea of the linear structure for controlling lanes and column sums is first introduced by Dinur et al. in [DMP+15]. Then, Guo et al. formalize and develop the technique *linear structure* to linearize the permutation of round-reduced Keccak [GLS16].

Linear structures can be applied to different variants by carefully controlling lanes to be constant or linear. Taking 3-round Keccak-256 as an example, the technique is shown in Fig.3. The black lanes mean that these bits are all 1, while the white lanes indicate that

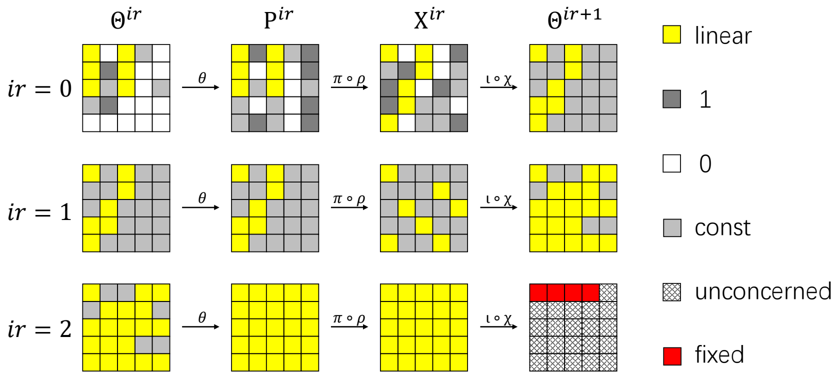

Figure 3: The linear structure used in 3-round Keccak-256 [GLS16].

these bits are all 0. The yellow lanes imply that these bits are linear (linear combination of some variable bits on Θ^0). 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 the remaining $256 - 64 = 192$ unrestricted output bits will be randomly matched to the given digest. By varying the constants on $\Theta^0_{0,3,*}$, $\Theta^0_{1,2,*}$, and $\Theta^0_{3,0,*}$ for $D_r = 2^{192}$ times (in this paper, we use D_r to denote the size of random space which provides different guesses), it is expected to obtain a preimage with guessing times of 2^{192}.

3.2 Allocating Approach with Improved Linear Structure

The all '0' capacity part of the starting state limits the design of the linear structure. To further promote the linear structure, Li et al. put forward the *allocating approach* to divide the whole attack into two easier tasks in two message blocks [LS19]. With the two-stage model, the capacity part of the starting state can be nonzero in the second stage.

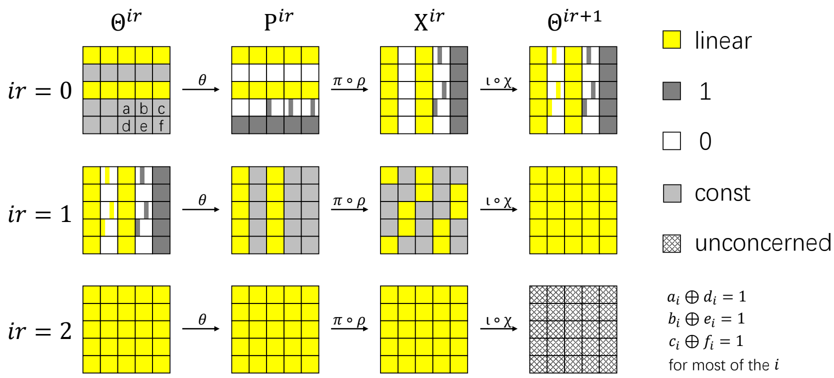

Figure 4: The linear structure for 3-round Keccak-256 in [LS19].

Li et al. [LS19] design an improved linear structure for the second stage as shown in Fig.4. The starting state [1] requires as many following restrictions [2] satisfied as possible.

$$\begin{aligned} &\Theta^0_{x,3,z} = \Theta^0_{x,4,z} \oplus 1 \ (2 \leq x \leq 4, 0 \leq z \leq 63) \\ &\Theta^0_{1,3,63} = \Theta^0_{1,4,63} \end{aligned} \tag{1}$$

As Fig.4 shows, every unsatisfied Equation (1) spends 1 degree of freedom ensuring the corresponding bit on $\Theta^1_{1,*,*}$ is constant. In summary, the number of degrees of freedom can be calculated as follows. Initially, there are $10 \times 64 = 640$ variables on Θ^0. To construct the linear structure, $5 \times 64 + 2 \times 64 = 448$ equations are added on Θ^0 and Θ^1 to control the column sums, where 2 degrees of freedom can be returned because of inherent linear dependence. Suppose there are k unsatisfied Equations (1) on the starting state. Then to eliminate the effect, k extra equations are added on Θ^1. As a result, there are $640 - 448 + 2 - k = 194 - k$ degrees of freedom remaining on X^2 which can be used for digest matching. With a trade-off, the guessing times for the first message block and the second message block will be $2^{80.06}$ and 2^{81} by leaving $k = 19$ unsatisfied Equations (1).

3.3 Iterating Strategy and 5-for-3 Strategy

In [LHY21], Lin et al. propose an *iterating strategy* that extends the first stage from the one-block model into the multi-block model, so that a state satisfying more Equations (1) can be obtained. Besides, they use degrees of freedom more efficiently with *5-for-3 strategy*. The process of *iterating strategy* can be started from any starting state (e.g. the all '0' initial value). In each turn it uses the linear structure introduced above to find a new starting state better (the number of unsatisfied Equations (1) k is smaller) than the previous one. If the linear structure has more degrees of freedom with a better starting state, it is more likely to generate another better starting state, until a good enough one is finally obtained. During each try, those $194 - k$ degrees of freedom are used with *5-for-3 strategy* consisting of the following equations, so that every 5 degrees of freedom can ensure 3 Equations (1) are satisfied.

$$\begin{cases} X^2_{0,3,z} = 1 \\ X^2_{0,4,z} = 1 \\ X^2_{2,3,z} \oplus X^2_{2,4,z} \oplus X^2_{3,3,z} = 0 \\ X^2_{3,3,z} \oplus X^2_{3,4,z} = 0 \\ X^2_{4,3,z} \oplus X^2_{4,4,z} = 1 \end{cases} \rightarrow \begin{cases} \Theta^3_{2,3,z} = \Theta^3_{2,4,z} \oplus 1 \\ \Theta^3_{3,3,z} = \Theta^3_{3,4,z} \oplus 1 \\ \Theta^3_{4,3,z} = \Theta^3_{4,4,z} \oplus 1 \end{cases} \xrightarrow[block]{next} \begin{cases} \Theta^0_{2,3,z} = \Theta^0_{2,4,z} \oplus 1 \\ \Theta^0_{3,3,z} = \Theta^0_{3,4,z} \oplus 1 \\ \Theta^0_{4,3,z} = \Theta^0_{4,4,z} \oplus 1 \end{cases}$$

Finally, the iterating process ends with a state satisfying 189 ($k = 3$) Equations (1) with guessing times of $2^{63.78}$. In the second stage (the last message block), the linear structure has $194 - 3 = 191$ degrees of freedom to match the output bits with guessing times of $2^{256-191} = 2^{65}$. For the random space, they vary the values of the column sums on Θ^1 with $D_r = 2^{128}$.

3.4 Quadratic Structure with the Relinearization Technique

The design of an entirely linear structure may be unnecessary in some cases. Take the linear structure shown in Fig.5 as an example, which is designed by Rajasree[Raj19] and used in preimage attack on 2-round Keccak-512. With the partially-linear structure, there

[1] The starting state also requires $\bigoplus_{0 \leq x \leq 4, 0 \leq z \leq 63} \Theta^0_{x,4,z} = 0$ (to ensure the setting of column sums on Θ^0 has a solution), or else there is an *adjusting method* introduced thoroughly in [LS19].

[2] $192 + 1 = 193$ equations in total, where 1 is for the padding rule when constructing the last message block.

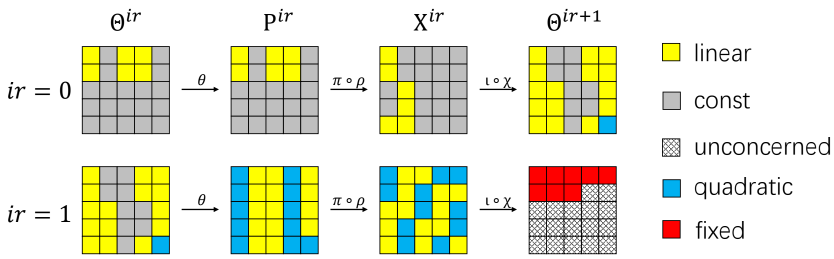

Figure 5: The linear structure used in 2-round Keccak-512 [Raj19].

remain $6 \times 64 - 1 - 3 \times 64 = 191$[3] degrees of freedom which can be used to match the output bits with a gain of 2^{191}. On the contrary, if it requires an entirely linear structure, bits on $\Theta^0_{3,0,*}$ and $\Theta^0_{3,1,*}$ should be constant which causes a loss of $64 - 1 = 63$ degrees of freedom.

The *relinearization technique* (we refer to their work as relinearization technique for ease of reference in this paper) is based on *Crossbred algorithm* [JV18] for solving special quadratic equation systems (the number of equations is much larger than the number of different non-linear terms). This algebraic method has been widely used in preimage attacks on round-reduced Keccak [WWF+21, LIMY21]. Fig.6 is an example of solving an equation system using *relinearization technique*.

$$\begin{cases} x_0 \oplus x_2 \oplus x_3 = 1 \\ x_1 \oplus x_2 = 1 \\ x_1 \oplus x_3 = 1 \\ x_0 \oplus x_1 \oplus x_3 \oplus x_0 \cdot x_1 \oplus x_1 \cdot x_2 = 0 \\ x_2 \oplus x_3 \oplus x_0 \cdot x_2 \oplus x_1 \cdot x_3 \oplus x_2 \cdot x_3 = 0 \\ x_3 \oplus x_0 \cdot x_3 = 0 \\ x_0 \cdot x_1 \oplus x_1 \cdot x_2 \oplus x_0 \cdot x_3 = 1 \end{cases} \xrightarrow{\text{relinearize}} \begin{cases} x_0 \oplus x_2 \oplus x_3 = 1 \\ x_1 \oplus x_2 = 1 \\ x_1 \oplus x_3 = 1 \\ x_0 \oplus x_1 \oplus x_3 \oplus x_4 = 0 \\ x_2 \oplus x_3 \oplus x_5 = 0 \\ x_3 \oplus x_6 = 0 \\ x_4 \oplus x_6 = 1 \end{cases} \xrightarrow{\text{solve}} \begin{cases} x_0 = 1 \\ x_1 = 0 \\ x_2 = 1 \\ x_3 = 1 \\ x_4 = 0 \\ x_5 = 0 \\ x_6 = 1 \end{cases} \xrightarrow{\text{examine}} \begin{cases} x_4 = x_0 \cdot x_1 \oplus x_1 \cdot x_2 ? \\ x_5 = x_0 \cdot x_2 \oplus x_1 \cdot x_3 \oplus x_2 \cdot x_3 ? \\ x_6 = x_0 \cdot x_3 ? \end{cases}$$

Figure 6: An example of applying the *relinearization technique*.

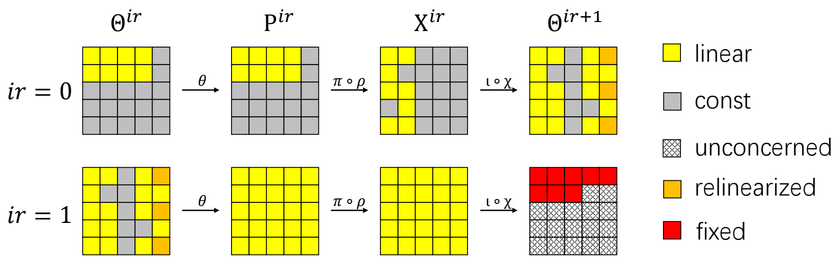

Figure 7: Preimage attack on 2-round Keccak-512 [LIMY21].

The *relinearization technique* can be applied to 2-round Keccak-512 matching more output bits [LIMY21]. 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 are $3 \times 64 = 192$ quadratic terms on Θ^1. With relinearization, 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. Finally,

[3] Here -1 corresponds to a constant bit on $\Theta^0_{3,1,*}$ (yellow lane) that matches the padding rule of Keccak, the same below. And -3×64 refers to controlling the column sums.

they construct a linear equation system consisting of these 448 linear equations on 448 variables, and it is expected to have one solution for each guess. They use the solution to recover the corresponding message and check the output bits. 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 $2^{512+192-448+1} = 2^{257}$ times on average (2^1 for matching the padding rule).

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 how to make further improvements. After that, we introduce the whole three-stage attack, which can lead to a preimage within 2^{52} guessing times. At last, we present a better way to choose the random space and enumerate it with high efficiency to speed up the solving time.

4.1 How to Get Improved

The complexity of the previous attack [LS19, LHY21] is a trade-off between two stages. For the previous stage, it is challenging to find a good starting state that satisfies all the Equations (1). The latter stage is limited by the number of degrees of freedom because the linear structure provides only at most 194 degrees of freedom while the unsatisfied Equations (1) further cost degrees of freedom. In the following, we will present an overview of how to deal with these difficulties.

We first consider the latter stage which is limited by the number of degrees of freedom. Since the output bits can recover 256 linear equations while the linear structure merely uses at most 194 of them, a basic improved idea is to apply a quadratic structure with more variables. After introducing more variables, the produced quadratic terms can be removed by *relinearization technique* (in Section 3.4). Thus, the goal is to find a good way to introduce new variables producing the fewest quadratic terms.

We then consider the previous stage which can hardly produce a state satisfying all the Equations (1). To solve this problem, some modifications can be made at the latter stage so that a few Equations (1) of $\Theta^0_{4,3,z} \oplus \Theta^0_{4,4,z} = 1$ are no longer required (see Section 4.2.3 for more details). As a result, the new required starting state is easier for the previous stage to produce. We divide the previous stage into two stages (the first stage and the second stage). In the first stage, the goal is the same as before (producing a state satisfying as many Equations (1) as possible). In the second stage, we produce a state satisfying the new requirements of the latter (third) stage.

With the above improvements, the complexity of each stage is decreased to be practical.

4.2 The Three-Stage Attack

4.2.1 The First Stage

In the first stage, we use the linear structure [LS19] with *iterating strategy* and *5-for-3 strategy* [LHY21] to get a state satisfying as many Equations (1) ($\Theta^0_{x,3,z} \oplus \Theta^0_{x,4,z} = 1$, where $2 \leq x \leq 4$) as possible, so that the linear structure of the next stage has sufficient degrees of freedom. We list the iterating process in Table 4. In the table, k and k' represent the number of unsatisfied Equations (1) before and after the current message block, respectively. Here we analyze the expected guessing times for each iteration. For the rest $194 - k$ degrees of freedom[4], the *5-for-3 strategy* is used to satisfy $\lfloor \frac{194-k}{5} \rfloor \times 3$ Equations (1). The remaining $192 - \lfloor \frac{194-k}{5} \rfloor \times 3$ Equations (1) are supposed to be satisfied randomly,

[4]In the first stage our attack is still based on the linear structure in [LS19]. Thus the initial number of degrees of freedom is still 194.

and the probability of generating a state with at most k' unsatisfied Equations (1) is $C^{k'}_{192-\lfloor\frac{194-k}{5}\rfloor\times 3}/2^{192-\lfloor\frac{194-k}{5}\rfloor\times 3}$ (here C^n_m, expressed elsewhere as $\binom{m}{n}$, is the combination number, i.e. $C^n_m = \frac{m!}{n!(m-n)!}$). Taking $\bigoplus_{0\le x\le 4, 0\le z\le 63}\Theta^0_{x,4,z} = 0$ into account[5], the calculation of guessing times is $2^1 \times (2^{192-\lfloor\frac{194-k}{5}\rfloor\times 3})/(C^{k'}_{192-\lfloor\frac{194-k}{5}\rfloor\times 3})$. Once a better state is found, we continue searching with this new state. After a 24-block iteration, we find a state satisfying $192-8=184$ Equations (1) with expected guessing times of around $2^{47.10}$. Using 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 Equations (1).

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

Here the selection of k and k' merely matches the experimental result which accepts the first better state with a smaller k'. Actually, the choice of k, k' and the number of steps hardly trouble because they are not the bottleneck.

4.2.2 The Second Stage

The second stage builds a bridge between the first stage and the third stage. The first stage gives a good state satisfying 184 Equations (1) which provides the second stage with a linear structure remaining $194-(192-184)=186$ degrees of freedom. The third stage requires a starting state (introduced later in Section 4.2.3) which should satisfy all the Equations (1) of $\Theta^0_{2,3,z}\oplus\Theta^0_{2,4,z}=1$ and $\Theta^0_{3,3,z}\oplus\Theta^0_{3,4,z}=1$ as well as $\Theta^0_{1,3,63}\oplus\Theta^0_{1,4,63}=0$ for padding rule. Additionally, the required starting state should satisfy at least 51 Equations (1) of $\Theta^0_{4,3,z}\oplus\Theta^0_{4,4,z}=1$. Therefore, the target of the second stage is generating a required starting state with the provided degrees of freedom.

We then analyze how to spend degrees of freedom adding equations and calculate the corresponding probability. Because of the independence between different slices, here we only consider the case in one slice. We focus on the property of the Sbox (operation χ) on two rows ($X^2_{*,3,z}$ and $X^2_{*,4,z}$). If we add the following three linear equations,

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

it yields that the probability of satisfying the first two Equations (1) of $x=2$ and $x=3$ is 0.625 and the probability of satisfying all three Equations (1) is 0.4375. For example, if

[5] This is not necessary if *adjusting method* is applied [LS19].

the inputs of two Sboxes are $X^2_{*,3,z} = 00001$ and $X^2_{*,4,z} = 01010$ which satisfy the three Equations (2), the outputs will be 00101 and 00011 which satisfy the first two Equations (1) of $x = 2$ and $x = 3$ ($00101 \oplus 00011 =??11?$), but do not satisfy the third Equation (1) of $x = 4$. After statistics, there are $2^5 \times 2^5 = 1024$ kinds of inputs of two 5-bit Sboxes while $1024/2^3 = 128$ of them satisfy the three Equations (2). Among these 128 kinds, 80 of them satisfy the first two Equations (1), and 56 of them satisfy all three Equations (1). If we suppose every kind of input occurs randomly, the probability of satisfying the first two or three Equations (1) will be $80/128 = 0.625$ or $56/128 = 0.4375$, respectively. We add Equations (2) on $186/3 = 62$ slices and regard the bits on the rest 2 slices as random values. The probability of satisfying all Equations (1) of $\Theta^0_{2,3,z} \oplus \Theta^0_{2,4,z} = 1$ and $\Theta^0_{3,3,z} \oplus \Theta^0_{3,4,z} = 1$ is $0.625^{62} \times 0.5^{2\times 2} \approx 2^{-46.04}$. When the first two Equations (1) are satisfied, the conditional probability of satisfying the third Equation (1) in one slice is $0.4375/0.625 = 0.7$ (51 slices are required in total). Finally, taking the padding rule ($\Theta^0_{1,3,63} \oplus \Theta^0_{1,4,63} = 0$) into account, the probability of getting a required message block is $2^{-1} \times 2^{-46.04} \times \sum_{i+j>=51}(C^i_{62} \times 0.7^i \times (1-0.7)^{62-i} \times C^j_2 \times 0.5^j \times (1-0.5)^{2-j}) \approx 2^{-51.52}$.

4.2.3 The Third Stage

In the third stage, we use the starting state provided by the second stage to match the output bits. As introduced in Section 4.1, we conclude that there are two difficulties we need to overcome. First, we need to introduce new variables increasing degrees of freedom while producing minimum quadratic terms. Second, the required starting state should be easier to produce (according to the second stage, we now have the starting state satisfying all the Equations (1) except 13 Equations (1) of $x = 4$).

To overcome these difficulties, we add 13 variables on $\Theta^1_{1,2,*}$. For the first difficulty, as shown in Fig.8, we find that adding extra variables on $\Theta^1_{1,*,*}$ leads to the fewest quadratic terms. For example, every bit on $\Theta^1_{1,2,z}$ only produces 4 quadratic terms on $X^0_{0,2,z}$, $X^0_{4,2,z}$, $I^1_{0,3,z+10}$ and $I^1_{1,3,z+10}$ which can be relinearized with minimum cost.

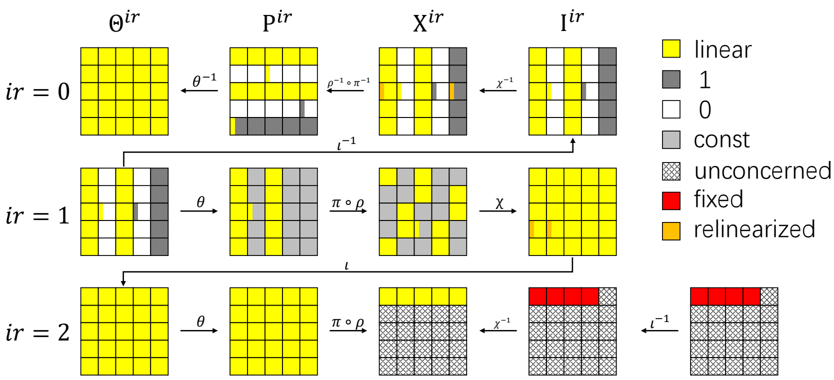

Figure 8: The modified linear structure.

By adding variables on $\Theta^1_{1,2,*}$, the second difficulty can be solved at the same time. To reduce the cost of matching the starting state, we need to ensure that the 6 lanes of $P^0_{x,y,*}(2 \le x \le 4, 3 \le y \le 4)$ are proper constants so that the relation (equal or opposite) of each bit pair $P^0_{x,3,z}$ and $P^0_{x,4,z}$ matches the relation of corresponding bit pair on the starting state. To control the required constants on P^0, we need to control some constants on X^0. As shown in Fig.9, if merely $\Theta^1_{1,*,*}$ can be extra variables, here are four types of row settings on Θ^1. Now we need to decide the type of the setting of each row on Θ^1. Among them, only the first type of setting satisfies that 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 belong to the first type. With that, 2 lanes ($a' = P^0_{2,3,*}$ and $b' = P^0_{3,3,*}$) will be constant 0, and 3 lanes ($x' = P^0_{2,4,*}$,

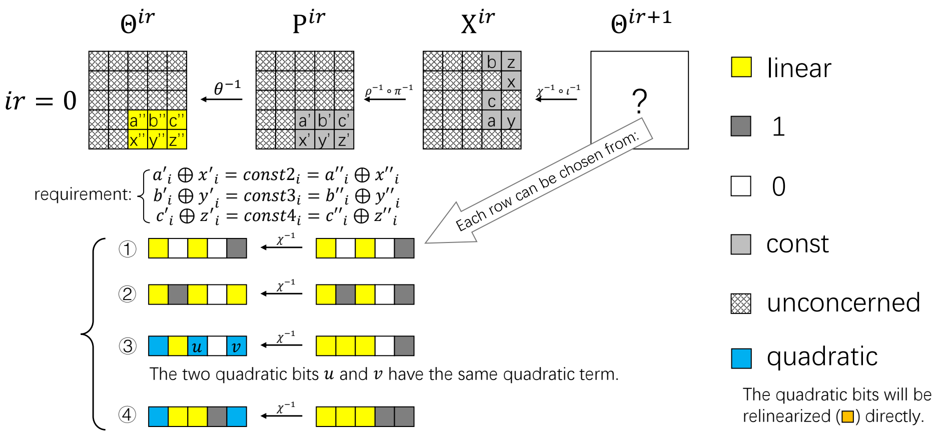

Figure 9: Controlling the constants.

$y' = P^0_{3,4,*}$ and $z' = P^0_{4,4,*}$) will be constant 1. However, the rows of $\Theta^1_{*,2,z}$ and $\Theta^1_{*,4,z}$ can be set to any type. Suppose t of them belong to the third and fourth types, which adds t degrees of freedom and generates $4t$ quadratic terms. To apply the *relinearization technique*, $192+t+4t \approx 256$ is required[6], which infers $t = 13$. Thus, at most $t = 13$ bits on lane $c' = P^0_{4,3,*}$ can be constant 1 rather than constant 0. As a result, we need a starting state satisfying all the Equations (1) ($\Theta^0_{x,3,z} \oplus \Theta^0_{x,4,z} = 1$ for $2 \le x \le 4$ and $0 \le z \le 63$) except at most $t = 13$ Equations (1) of $\Theta^0_{4,3,z} \oplus \Theta^0_{4,4,z} = 1$. That is what we obtained in the second stage exactly.

In summary, the third stage consists of six steps (see also Fig.8).

- Construct state Θ^1 by setting bits on $\Theta^1_{0,*,*}$ and $\Theta^1_{2,*,*}$ as variables (640 in total), 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 to the fourth type (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 Equations (1).
- Invert state Θ^1 one round backward (introduce 2×13 new variables to replace the quadratic bits) and add 320 equations to satisfy the starting state.
- Add 128 linear equations on Θ^1 to control the column sums and prevent the diffusion of the variables.
- Develop state Θ^1 two rounds forward (introduce another 2×13 new variables to replace the quadratic bits) and add 256 equations to match the output bits.
- Construct an equation system with $320 + 128 + 256 = 704$ linear equations and $2 \times 13 + 2 \times 13 = 52$ quadratic equations (produced by relinearization) on $640 + 13 + 2 \times 13 + 2 \times 13 = 705$ variables.

The expected number of guessing times is $2^{704-min(704,705)+52} = 2^{52}$.

Taking all three stages into account, the bottleneck of the whole attack lies in the third stage. Thus we regard the complexity of the third stage (multiplied with solving time) as the final complexity.

[6] Here the initial number of variables becomes 192 instead of 194 because our quadratic structure will break the inherent linear dependence among equations.

4.3 Speeding Up the Solving Time

We should consider not only the expected number of guessing times but also the solving time for each guess. Thus, making it easier to construct and solve the equation system is of great importance. In this section, we show that selecting different bits as random space affects the complexity of the solving time.

In general, the process of an attack can be summarized as Algorithm 1 shows. As shown in Line 2 and Line 3 of Algorithm 1, every linear equation in E'_L is obtained after all the r_i in R are determined, because every r_i is possibly involved in the linear equation (such as a term of r_0v_0).

Algorithm 1: The basic way solving an equation system.

Input: Variables composing the random space $R \leftarrow \{r_i | 0 \leq i \leq log_2(D_r) - 1\}$,
Variables $V \leftarrow \{v_j | 0 \leq j \leq n-1\}$,
Equations $E \leftarrow \{e_k(R, V) | 0 \leq k \leq m-1\}$.
Output: The solution of the equation system.

1 **while** the enumeration in the random space is not exhausted **do**
2   Assign new values for every r_i in R;
3   Generate equations E' by substituting R, while E'_L (part of E') is linear;
4     $V' \leftarrow V$, $E'_Q \leftarrow E' \setminus E'_L$, $E_{rec} \leftarrow \emptyset$;
5   **for** e' in E'_L **do**
6   Select a variable v', where v' is involved in e';
7         $E_{rec} \leftarrow E_{rec} \cup (v', e')$; `// Record equations to recover` V `later.`
8   Simplify and eliminate the variable v' in E'_L, part of E'_Q and V' by substituting e', and obtain updated E'_L, E'_Q and V'; `// Here is a trade-off that, if` E'_Q `is fully simplified, it is time-consuming but easy to verify the solution afterward.`
9   **end**
10   **if** $V' \neq \emptyset$ **then**
11   Solve the quadratic equation system E'_Q by *Crossbred algorithm* or *relinearization technique* replacing all the quadratic terms, and obtain the value of each variable in V'; `// This is not the case for attack on 3-round Keccak-256, because` E'_L `is sufficient to solve` V`.`
12   **end**
13   Recover the value of each variable in V by V' and E_{rec};
14   **if** No conflict happens before and V satisfy the rest equations E'_Q **then**
15   **return** V as a solution;
16   **end**
17 **end**
18 **return** no solution;

However, if variables set R (composing the random space) is chosen properly, only a small number of r_i are involved in each linear equation in E'_L. Thus, some linear equations will be obtained in advance when only a part of random space variables are determined. During the process of determining random space, we do the simplification of Line 6 to Line 8 of Algorithm 1 immediately once a linear equation is obtained. For some continuous guessings, the first major part of random space remains the same and the equation system can be derived from the intermediate state (the partly eliminated equation system) of the previous guessing. Consequently, only the last few simplifications should be considered, and the solving time for each guess can be greatly reduced.

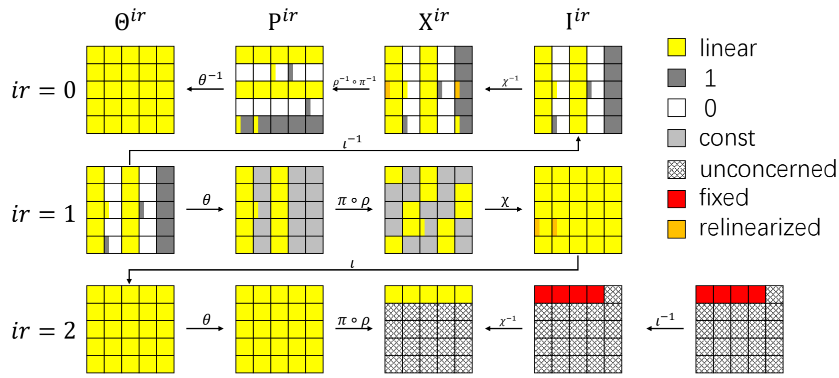

Figure 10: Vary the constants on $\Theta^1_{1,4,*}$.

In concrete, as shown in Fig.10, if we vary the values of some constant bits on $\Theta^1_{1,4,*}$, the structure keeps linear as before. Therefore, the values of constant bits on $\Theta^1_{1,4,*}$ can be selected as random space. We first regard these bits as variables and use the *relinearization technique* to deal with the newly produced quadratic terms, similarly to Section 4.1 and Section 4.2. After that, we determine the values of these bits and linearize the new quadratic equations caused by relinearization. Every time the value of a bit on $\Theta^1_{1,4,*}$ is determined, $1 + 4 = 5$ linear equations can be obtained (1 for determining bit $\Theta^1_{1,4,*}$, and 4 for linearizing 4 quadratic equations).

As a result, for a new guess, we usually 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. Last, we verify the rest quadratic equations with the solution. In most cases, we deal with a small number of linear equations, and the solving is very fast. With these techniques, we can guess around 1.01 million times per second on a personal computer.

To support the theoretical analysis of 3-round Keccak-256, we run a program to provide a preimage of all '0' digest. The experiments are run on a supercomputer. All the three-stage experimental results are finished within half a week using ten thousand core-groups (CGs). Every CG consists of a master core, and 64 slave cores (1.5 GHz). However, with basic code implementation, the ideal speed is hard to achieve even if we only run the Keccak permutation. Thus, the solving time is calculated according to the performance on a personal computer (3.7 GHz). Besides, the memory cost is around 0.2 MB. 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 A.

Table 5: The running time of preimage attack on 3-round Keccak-256.

stage	[a]running time	[b]solving speed	expected guessing times	[c]actual guessing times	[d]expected complexity	[c,d]actual complexity
the first stage	83	[e]1.79	$2^{47.1}$	$2^{48.9}$	$2^{53.9}$	$2^{55.7}$
the second stage	30	6.50	$2^{51.5}$	$2^{49.3}$	$2^{56.4}$	$2^{54.2}$
the third stage	360	[f]5.43	$2^{52.0}$	$2^{52.6}$	$2^{57.2}$	$2^{57.8}$

[a] Unit: 1000 CGs · hour.
[b] Unit: million guesses / (second · CG).
[c] The 'actual' refers to the experimental result for once.
[d] Unit: equivalent 3-round Keccak calls.
[e] The solving speed of the first stage is slower because we need to calculate the number of satisfied Equations (1), while in the other two stages, we just need to check whether Equations (1) are all satisfied.
[f] It is able to run the third stage with 1.01 million guesses per second on a personal computer, while the 3-round Keccak running speed is 37 million times per second. Thus, the solving time is around $2^{5.2}$ 3-round Keccak calls.

5 Preimage Attack on 4-Round Keccak[r=640, c=160]

The cryptanalysis of the preimage attack on 4-round Keccak[r=640, c=160] will be introduced in this section.

5.1 Related Work

Keccak[r=640, c=160] is an instance of Keccak in the Keccak Crunchy Crypto Collision and Preimage Contest [BDH+a]. In 2021, Wei et al. [WWF+21] applied *Crossbred algorithm* and gave attack on 4-round Keccak[r=640, c=160]. According to their attack, they built a linear structure linearizing two rounds with 94 degrees of freedom left. To match the digest, 10 output bits are linearized and restricted with $53+10=63$ degrees of freedom. Then, to match the first 48 output bits, a quadratic equation system with $48-10=38$ equations over $94-63=31$ variables is obtained which can be solved by *Crossbred algorithm*. Taking the last 32-bit digest into account, their attack is required to solve $2^{32+38-31=39}$ quadratic equation systems (or equivalent 2^{62} linear equation systems by relinearizing all the quadratic terms).

In addition, our improved attack incorporates the technique *zero coefficient* [HLY21]. This property describes the linear dependence among some bits on X^2. When a degree of freedom is spent to restrict a bit on X^2 a constant bit, additional bits on X^2 will simultaneously become constant. Consequently, the linearization will be more efficient. A concrete example is explained in Section 5.3.1.

5.2 Making Further Improvements

The linear structure used for 4-round Keccak[r=640, c=160] is shown in Fig.11. Compared to the linear structure used in [WWF+21] (similar to [LS19]), we take variables on $\Theta^1_{1,*,*}$ and $\Theta^1_{3,*,*}$ instead of $\Theta^1_{0,*,*}$ and $\Theta^1_{2,*,*}$. On the one hand, this modification does not affect matching the starting state because the capacity part only involves the last plane which does not require specific constant bits on $P^0_{*,3,*}$ and $P^0_{*,4,*}$. On the other hand, the result of the MILP model shows that this modification gives better linearization matching the output digest. Then, we analyze the degrees of freedom for the first two rounds. There are $10\times 32=320$ variables on Θ^1. To match the starting state, we select two candidates for the previous message block, so that $5\times 32-1=159$ equations are required to match one of them. Besides, to satisfy the padding rule, another 1 equation is added. To control the column sums on Θ^1, $2\times 32=64$ equations are added. With two linear-dependent equations, there are $320-159-1-64+2=98$ degrees of freedom left on Θ^2.

Afterward, with the 98 degrees of freedom, we should try to linearize a part of the state on the next two rounds so that the probability of matching the output digest is as high as possible. Different from [WWF+21], we consider both the technique *zero coefficient* proposed in [HLY21] and the entire 80-bit digest instead of only the first 48 bits. Besides, to make further linearization, we make full use of the variables composing the random space according to the result of the MILP model, instead of using only a part of them. At last, with these techniques, we build an MILP model [Gur24] optimizing the probability of matching the digest by linearizing the last two rounds using the 98 degrees of freedom. The detailed way to build the MILP model will be introduced in the next section (Section 5.3).

5.3 Details of the MILP Model

In Section 5.2, we use a linear structure (Fig.11) so that the state is linear for first two rounds till Θ^2 with 98 degrees of freedom left. With these 98 degrees of freedom, we will spend some degrees of freedom restricting some bits on X^2 constant bits so that some bits

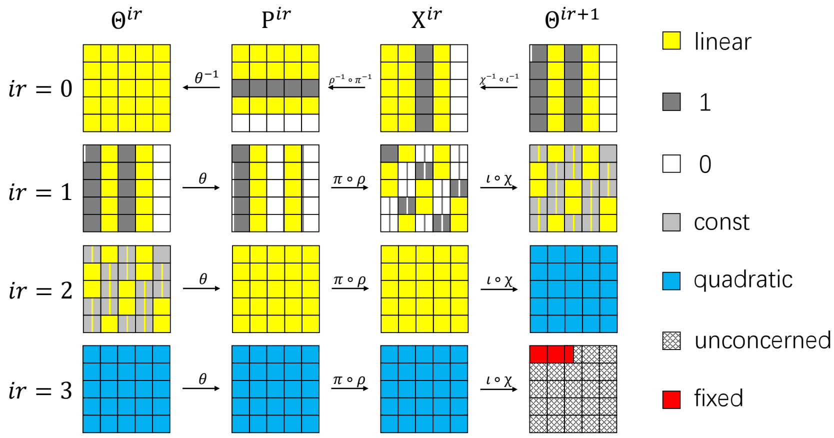

Figure 11: The linear structure used for 4-round Keccak[r=640, c=160].

on Θ^3 will be linear. When some bits on Θ^3 are linear, a few bits on X^3 will be linear accordingly. We then spend some degrees of freedom adding linear restrictions with these linear bits on X^3 to maximize the probability of matching the digest. The above tasks can be achieved by building an MILP model which will be introduced in this section.

The MILP model can be divided into two parts. The first part is linearizing last two rounds, including how to linearize bits on Θ^3 by spending some degrees of freedom restricting constant bits on X^2 (or P^2 equivalently) and how to deduce linear bits on X^3 according to linear bits on Θ^3. The second part is how to calculate the probability of matching the digest with some linear bits on X^3.

5.3.1 Linearizing Last Two Rounds

To model the first part, we use $800 + 800 + 160 = 1760$ boolean variables representing the concerned attributes of bits on three states (X^2, Θ^3, and X^3). Precisely, we are concerned with whether we should spend a degree of freedom restricting $X^2_{x,y,z}$ a constant bit (or corresponding bit on P^2 equivalently). We introduce 800 variables ($a_{X^2_{0,0,0}}$ to $a_{X^2_{4,4,63}}$) where $a_{X^2_{x,y,z}} = 1$ means a degree of freedom is spent restricting $X^2_{x,y,z}$ a constant bit. Besides, we are also concerned with whether each bit on Θ^3 or $X^3_{*,0,*}$ is linear or not. We introduce another $800 + 160 = 1600$ variables ($a_{\Theta^3_{0,0,0}}$ to $a_{\Theta^3_{4,4,63}}$, and $a_{X^3_{0,0,0}}$ to $a_{X^3_{4,0,63}}$) where $a_{\Theta^3_{x,y,z}} = 1$ (or $a_{X^3_{x,0,z}} = 1$) means the bit $\Theta^3_{x,y,z}$ (or $X^3_{x,0,z}$) is linear, or the corresponding bit is quadratic if the value of variable is 0 otherwise.

According to the property of χ operation, $\Theta^3_{x,y,z}$ will be a linear bit if $X^2_{x+1,y,z}$ or $X^2_{x+2,y,z}$ is a constant bit. Furthermore, $X^2_{x,y,z}$ will be a constant bit when a degree of freedom is spent on itself or a linear-dependent bit. For example, if $\Theta^3_{4,0,0}$ need to be linearized, the most direct way is restricting $X^2_{0,0,0}$ or $X^2_{1,0,0}$ a constant bit. Besides, restricting $X^2_{4,2,18}$ a constant bit is also an option (the technique *zero coefficient* in [HLY21]). It is because we have $X^2_{4,2,18} = P^2_{0,4,0}$ and $X^2_{0,0,0} = P^2_{0,0,0}$. Meanwhile, thanks to the selection of column sums on Θ^1, both $\Theta^2_{0,4,0}$ and $\Theta^2_{0,0,0}$ are constant bits. According to the property of θ operation, we have $\Theta^2_{0,4,0} \oplus \Theta^2_{0,0,0} = P^2_{0,4,0} \oplus P^2_{0,0,0}$. If we restrict $X^2_{4,2,18}$ ($P^2_{0,4,0}$) a constant bit, $P^2_{0,0,0}$ ($X^2_{0,0,0}$) will be a constant bit simultaneously, resulting $\Theta^3_{4,0,0}$ a linear bit as well. Similarly, there are 3 more options by restricting $X^2_{3,4,9}$, $X^2_{3,1,1}$, or $X^2_{2,3,30}$ a constant bit. With the relations above, we add an equation of

$$a_{\Theta^3_{4,0,0}} \leq a_{X^2_{0,0,0}} + a_{X^2_{1,0,0}} + a_{X^2_{4,2,18}} + a_{X^2_{3,4,9}} + a_{X^2_{3,1,1}} + a_{X^2_{2,3,30}}$$

to the MILP model which means $\Theta^3_{4,0,0}$ can be linearized when at least one of these 6 bits on X^2 is restricted spending a degree of freedom. By adding 800 equations, we successfully build the MILP model of linearizing bits on Θ^3 by spending some degrees of freedom restricting constant bits on X^2.

The MILP model of deducing linear bits on X^3 according to linear bits on Θ^3 is similar. For example, $X^3_{0,0,0}$ will be a linear bit when 11 bits on Θ^3 ($\Theta^3_{0,0,0}$, $\Theta^3_{1,0,31}$, $\Theta^3_{4,0,0}$, $\Theta^3_{1,1,31}$, $\Theta^3_{4,1,0}$, $\Theta^3_{1,2,31}$, $\Theta^3_{4,2,0}$, $\Theta^3_{1,3,31}$, $\Theta^3_{4,3,0}$, $\Theta^3_{1,4,31}$, and $\Theta^3_{4,4,0}$) are all linear bits. Thus, we add the following 11 equations to the MILP model in Table 6. After adding another $11 \times 160 = 1760$ (only 160 bits on the first plane $X_{x,0,z}$ affect digest matching) equations, the MILP model of the first part is built.

Table 6: The 11 equations modeling $X^3_{0,0,0}$.

$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{0,0,0}}$	$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{1,0,31}}$	$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{4,0,0}}$	$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{1,1,31}}$
$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{4,1,0}}$	$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{1,2,31}}$	$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{4,2,0}}$	$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{1,3,31}}$
$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{4,3,0}}$	$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{1,4,31}}$	$a_{X^3_{0,0,0}} \leq a_{\Theta^3_{4,4,0}}$	

5.3.2 Calculating the Probability

With the model of the first part, there is a variety of possibilities for the values of variables $a_{X^3_{x,0,z}}$ which means the different possibilities of linearization. In the second part, we need to parse the case of linearization and decide the number of added restrictions. After that, the probability of matching the digest can be obtained by some precomputation.

To model the second part, as shown in Fig.12, we use $32 \times 5 = 160$ variables ($a^z_{i,j}$, where $0 \leq i \leq 31$, $0 \leq j \leq 4$) for each Sbox (χ operation on a row $X^3_{*,0,z}$, and there are 32 ($z = 0 \sim 31$) Sboxes in total).

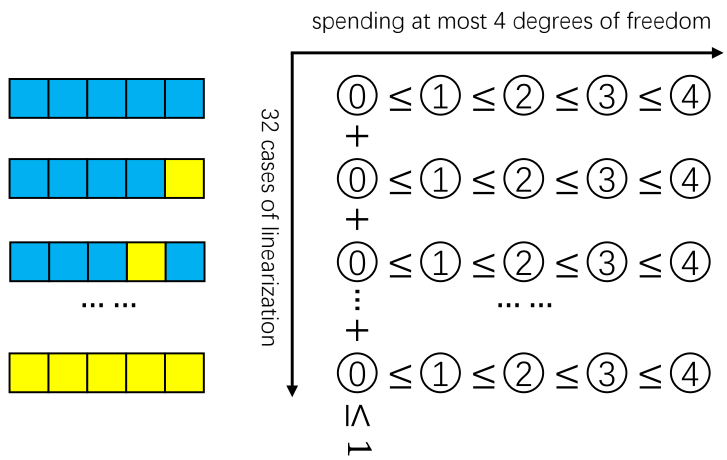

Figure 12: Modeling an Sbox.

We focus on one Sbox (for a fixed z) modeling how to add linear restrictions and how to calculate the probability. Since every bit $X^3_{x,0,z}$ on $X^3_{*,0,z}$ may be linearized ($a_{X^3_{x,0,z}} = 1$) or not ($a_{X^3_{x,0,z}} = 0$), there are $2^5 = 32$ possible cases for the Sbox showing which bits can be involved in linear restrictions. Thus, we use 32 variables ($a^z_{i,0}$, where $0 \leq i \leq 31$) distinguishing which case it is ($a^z_{i,0} = 1$ represents the i^{th} case). On the one hand, the circumstance must belong to only one case by adding an equation of

$$\Sigma_{i=0\sim 31} a^z_{i,0} \leq 1$$

to the MILP model. On the other hand, the corresponding bits should accord with the linearization of bits on $X^3_{*,0,z}$. For example, considering the case $i = 22 = (10110)_2$ which means restrictions can be added on three linear bits $X^3_{1,0,z}$, $X^3_{2,0,z}$, and $X^3_{4,0,z}$, we add three equations of

$$a^z_{22,0} \leq a_{X^3_{1,0,z}}$$
$$a^z_{22,0} \leq a_{X^3_{2,0,z}}$$
$$a^z_{22,0} \leq a_{X^3_{4,0,z}}$$

to the MILP model meaning that the Sbox may belong to the i^{th} case when the three bits on X^3 are all linearized. After determining which case it is, we consider the number of added restrictions. If one restriction is added, the $a^z_{i,1}$ will be 1. If the second restriction is added, the $a^z_{i,2}$ will be 1 as well. Surely restrictions should be added one by one on corresponding case. Thus, for each case, we add 4 equations

$$a^z_{i,1} \leq a^z_{i,0}$$
$$a^z_{i,2} \leq a^z_{i,1}$$
$$a^z_{i,3} \leq a^z_{i,2}$$
$$a^z_{i,4} \leq a^z_{i,3}$$

to the MILP model. For example, if $a^z_{i,3} = 1$ and $a^z_{i,4} = 0$, it means that we spend 3 degrees of freedom adding 3 linear restrictions on the Sbox.

So far, we have built a majority of the MILP model. In the following, we supplement a global constraint (requirement of degrees of freedom) and the objective (maximizing the probability). According to the linear structure, there are 98 degrees of freedom left on Θ^2. On X^2, we spend $\Sigma_{x=0\sim4,y=0\sim4,z=0\sim31} a_{X^2_{x,y,z}}$ degrees of freedom restricting constant bits. On X^3, we spend $\Sigma_{i=0\sim31,j=0\sim4,z=0\sim31} a^z_{i,j}$ degrees of freedom adding linear restrictions to increase the probability of matching the digest. Besides, we leave around $8 = 7 + 1$ degrees of freedom because the large quadratic equation system over 7 variables can be solved by Gauss elimination using *relinearization technique* (introduced in Section 3.4), and the rest 1 degree of freedom makes it easier to solve the equation system. Then, we add an equation

$$(\Sigma_{x=0\sim4,y=0\sim4,z=0\sim31} a_{X^2_{x,y,z}}) + (\Sigma_{i=0\sim31,j=0\sim4,z=0\sim31} a^z_{i,j}) + 8 \leq 98$$

to the MILP model.

At last, we consider the objective of the MILP model. We first introduce a precomputed probability table of $32 \times 32 \times 5 = 5120$ constants ($p_{z,i,j}$, where $0 \leq z \leq 31$, $0 \leq i \leq 31$, $0 \leq j \leq 4$). The z means the z^{th} Sbox, where some required output bits are given according to the digest. The i means an input bits mask, where the linear restrictions can only be added on some input bits masked by i (for example, $i = 22 = (10110)_2$ means $X^3_{1,0,z}$, $X^3_{2,0,z}$, and $X^3_{4,0,z}$ are linear). The j means the number of linear restrictions added in total. The $p_{z,i,j}$ means the maximum probability of the output bits matching given digest on z^{th} row with j added linear restrictions on input bits masked by i. Here we take $z = 0$ and $i = 22$ as an example. The given digest (operated by inverse ι of fourth round) on row $I^3_{*,0,z=0}$ is 101??. The linearization of row $X^3_{*,0,z=0}$ is $QLLQL$ ($X^3_{1,0,z}$, $X^3_{2,0,z}$, and $X^3_{4,0,z}$ are linear; $X^3_{0,0,z}$ and $X^3_{3,0,z}$ are quadratic). The $p_{0,22,j}$ for different j are calculated as follows.

- If no ($j = 0$) restriction is added, the probability of matching the given 3-bit digest is $p_{0,22,0} = 4/32$ (suppose every kind of input occurs randomly).
- If one ($j = 1$) restriction is added, one of the best choices is adding an equation of $X^3_{1,0,z} = 0$ so that there are 4 kinds of input (among the 16 kinds satisfying

$X^3_{1,0,z} = 0$) whose output will match the given 3-bit digest. Thus, the probability will be $p_{0,22,1} = 4/16$.

- If two ($j = 2$) restrictions are added, one of the best choices is adding two equations of $X^3_{1,0,z} = 0$ and $X^3_{2,0,z} = 1$. Then there are 3 kinds of input (among the 8 kinds satisfying the two equations) whose output will match the given 3-bit digest, leading to $p_{0,22,2} = 3/8$.
- If three ($j = 3$) restrictions are added, one of the best choices is adding three equations of $X^3_{1,0,z} = 0$, $X^3_{2,0,z} = 1$, and $X^3_{4,0,z} = 0$. After that, there are 2 kinds of input (among the 4 kinds satisfying the three equations) whose output will match the given 3-bit digest. We have $p_{0,22,3} = 2/4$.
- If we plan to spend one more ($j = 4$) degree of freedom, no more gain can be obtained because restrictions can only be added on the three input bits under the case $i = 22$. Hence, the probability remains the same with $p_{0,22,4} = p_{0,22,3} = 2/4$ (an idle case).

With the probability table p, we set an objective to the MILP model by adding

$$Maximizing : \Sigma_{z=0\sim 31, i=0\sim 31, j=1\sim 4}(log_2(p_{z,i,j}/p_{z,i,j-1}) \times a^z_{i,j})$$

which means if a degree of freedom is spent on adding the j^{th} restriction ($a^z_{i,j} = 1$), the gain (calculated by the power of 2) of the probability matching the corresponding digest bits will be $log_2(p_{z,i,j}/p_{z,i,j-1})$.

The algorithms of calculating the precomputed probability table p and building the MILP model are shown in Appendix C.

After obtaining the result of the MILP model, it is necessary to check the validation practically to avoid some corner cases. For example, if $X^2_{x,y,z}$ is a constant bit and $X^2_{x+1,y,z} = 1$, the bit $\Theta^3_{x,y,z}$ will be a constant bit instead of a linear bit. If some specific 11 bits on Θ^3 all happen to be constant bits, $X^3_{x,0,z}$ will be a constant bit instead of a linear bit. Suppose that $X^3_{x,0,z} = 1$, we can not spend a degree of freedom restricting $X^3_{x,0,z} = 0$, which causes failing of the attack.

5.4 Summary of Preimage Attack on 4-Round Keccak[r=640, c=160]

In summary, the result of the MILP model is shown in Table 7. For each guess, we determine the values of 71 bits on X^2 to linearize 19 bits on $X^3_{*,0,*}$. We add 19 linear equations bringing a gain of around $2^{16.5}$. Note that some of the 19 linear equations can be obtained when part of 71 bits on X^2 are determined. Thus, with technique introduced in Section 4.3, the equation system can be partly simplified in advance. We then solve quadratic equation systems with $48 - 14 = 34$ quadratic equations over $98 - 71 - 19 = 8$ variables. Guessing the value of only one more variable (the number of quadratic terms will be $7 \times (7-1)/2 = 21$, satisfying $21 + 7 \leq 34$), the quadratic equation system can be solved linearly by *Crossbred algorithm* (or *relinearization technique* replacing all the quadratic terms) which brings a gain of 2^7. The guessing times of the attack will be $2^{80-16.5-7} = 2^{56.5}$ while the solving time for each guess is around $2^{4.4}$ 4-round Keccak calls according to experimental results. Finally, the total complexity is $2^{60.9}$.

With the techniques introduced previously, we successfully find a solution for the 4-round preimage challenge with width $b = 800$ in the Keccak Crunchy Crypto Collision and Preimage Contest [BDH+a]. The experiment on 4-round Keccak[r=640, c=160] is run on another supercomputer. The result is obtained within 24 days using five thousand core-groups (CGs). Every CG consists of a master core, and 64 slave cores (2.25 GHz). The solving time is also calculated according to the performance on a personal computer (3.7 GHz). The memory cost is around 0.3 MB. The running time and the running speed are shown in Table 8. Appendix B presents the input message blocks.

Table 7: The result of the MILP model (in little-endian order).

71 bits on X^2 restricted to be constant [a]				
00 10 00 80	00 00 a0 19	00 00 10 00	00 00 28 00	00 00 00 00
00 00 a0 80	02 40 40 22	02 00 90 80	00 00 20 00	00 00 00 00
00 00 a0 02	00 00 00 a0	01 00 00 14	00 00 68 41	03 00 00 10
00 00 80 00	02 00 00 80	00 00 90 a8	01 00 00 51	0a 00 28 00
02 40 10 82	00 00 c0 22	24 00 00 80	01 00 40 50	00 00 00 00
19 linearized bits on $X^3_{*,0,*}$ [b]				
02 00 e0 a2	2d 2a 00 00	0a 08 00 00	00 08 00 00	08 00 00 00

[a] A bit is '1' means that this bit is constant spending 1 degree of freedom.
[b] A bit is '1' means that this bit is linear.

Table 8: The running time of preimage attack on 4-round Keccak[r=640, c=160].

stage	[a]running time	[b]solving speed	expected guessing times	[c]actual guessing times	[d]expected complexity	[c,d]actual complexity
1^{st} block	negligible	–	2^1	–	–	–
2^{nd} block	2880	[e]16.3	$2^{56.5}$	$2^{57.2}$	$2^{60.9}$	$2^{61.6}$

[a] Unit: 1000 CGs · hour.
[b] Unit: million guesses / (second · CG).
[c] The 'actual' refers to the experimental result for once.
[d] Unit: equivalent 4-round Keccak calls.
[e] On a personal computer, the solving time is around $2^{4.4}$ 4-round Keccak calls.

6 Conclusion

In this paper, we provide preimage attacks on 3-round Keccak-256 and 4-round Keccak[r=640, c=160].

For 3-round Keccak-256, we propose a three-stage model. In the third stage, we modify the linear structure by introducing some extra variables on $\Theta^1_{1,2,*}$. With that, the modified linear structure leaves more degrees of freedom for digest matching. At the same time, the difficulty of matching the starting state is solved by the new required starting state as well as the additional second stage. Besides, we speed up the solving time by selecting constants on $\Theta^1_{1,4,*}$ as random space. With that, most of the linear equations can be obtained in advance, and the change of each constant bit only causes a small number of varied linear equations. By guessing the constant bits hierarchically, we only need to deal with a small number of linear equations for each guess on average. As a result, the guessing times of finding a preimage for 3-round Keccak-256 can be decreased to 2^{52} times, and the solving time for each guess can be decreased to $2^{5.2}$ 3-round Keccak calls. Moreover, we find a preimage of all '0' digest for 3-round Keccak-256.

For 4-round Keccak[r=640, c=160], we enhance linearization through techniques such as strategic variable selection, two candidates for the previous message block, increased consideration of output bits, and comprehensive utilization of enumerated variables. With these techniques, we develop an MILP model that optimizes the highest probability for the digest matching with complexity around $2^{60.9}$ leading to another solution to the Crunchy Contest.

It is noted that our cryptanalysis is still far from threatening the security of full-round Keccak.

Acknowledgements

This work was supported by the National Key Research and Development Program of China (Grant No. 2024YFA1013003).

References

[BDH+a] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Keccak crunchy crypto collision and pre-image contest. https://keccak.team/crunchy_contest.html.

[BDH+b] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. The keccak-f state. https://keccak.team/figures.html.

[BDPA11a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011. http://sponge.noekeon.org/CSF-0.1.pdf.

[BDPA11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The keccak sha-3 submission, version 3, 2011. https://keccak.team/files/Keccak-submission-3.pdf.

[Ber10] Daniel J. Bernstein. Second preimages for 6 (7? (8??)) rounds of keccak?, 2010. https://ehash.iaik.tugraz.at/uploads/6/65/NIST-mailing-list_Bernstein-Daemen.txt.

[Din21] Itai Dinur. Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 374–403. Springer, Cham, October 2021. https://doi.org/10.1007/978-3-030-77870-5_14.

[DMP+15] Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Cube attacks and cube-attack-like cryptanalysis on the round-reduced Keccak sponge function. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 733–761. Springer, Berlin, Heidelberg, April 2015. https://doi.org/10.1007/978-3-662-46800-5_28.

[Dwo15] Morris J. Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, 2015. https://doi.org/10.6028/NIST.FIPS.202.

[GLS16] Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced Keccak. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 249–274. Springer, Berlin, Heidelberg, December 2016. https://doi.org/10.1007/978-3-662-53887-6_9.

[Gur24] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. https://www.gurobi.com.

[HLY21] Le He, Xiaoen Lin, and Hongbo Yu. Improved preimage attacks on 4-round Keccak-224/256. *IACR Trans. Symm. Cryptol.*, 2021(1):217–238, 2021. https://doi.org/10.46586/tosc.v2021.i1.217-238.

[JV18] Antoine Joux and Vanessa Vitse. A crossbred algorithm for solving boolean polynomial systems. In Jerzy Kaczorowski, Josef Pieprzyk, and Jacek Pomykała, editors, *Number-Theoretic Methods in Cryptology*, volume 10737 of *LNCS*, pages 3–21, Cham, 2018. Springer International Publishing. https://doi.org/10.1007/978-3-319-76620-1_1.

[KMS18] Rajendra Kumar, Nikhil Mittal, and Shashank Singh. Cryptanalysis of 2 round Keccak-384. In Debrup Chakraborty and Tetsu Iwata, editors, *INDOCRYPT 2018*, volume 11356 of *LNCS*, pages 120–133. Springer, Cham, December 2018. https://doi.org/10.1007/978-3-030-05378-9_7.

[LHY21] Xiaoen Lin, Le He, and Hongbo Yu. Improved preimage attacks on 3-round Keccak-224/256. *IACR Trans. Symm. Cryptol.*, 2021(3):84–101, 2021. https://doi.org/10.46586/tosc.v2021.i3.84-101.

[LIMY21] Fukang Liu, Takanori Isobe, Willi Meier, and Zhonghao Yang. Algebraic attacks on round-reduced Keccak. In Joonsang Baek and Sushmita Ruj, editors, *ACISP 21*, volume 13083 of *LNCS*, pages 91–110. Springer, Cham, December 2021. https://doi.org/10.1007/978-3-030-90567-5_5.

[LS19] Ting Li and Yao Sun. Preimage attacks on round-reduced Keccak-224/256 via an allocating approach. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 556–584. Springer, Cham, May 2019. https://doi.org/10.1007/978-3-030-17659-4_19.

[MPS14] Pawel Morawiecki, Josef Pieprzyk, and Marian Srebrny. Rotational cryptanalysis of round-reduced Keccak. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 241–262. Springer, Berlin, Heidelberg, March 2014. https://doi.org/10.1007/978-3-662-43933-3_13.

[MS10] Pawel Morawiecki and Marian Srebrny. A SAT-based preimage analysis of reduced KECCAK hash functions. Cryptology ePrint Archive, Report 2010/285, 2010. https://eprint.iacr.org/2010/285.

[NRM11] María Naya-Plasencia, Andrea Röck, and Willi Meier. Practical analysis of reduced-round Keccak. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *INDOCRYPT 2011*, volume 7107 of *LNCS*, pages 236–254. Springer, Berlin, Heidelberg, December 2011. https://doi.org/10.1007/978-3-642-25578-6_18.

[PC22] Junling Pei and Lusheng Chen. Preimage attacks on reduced-round keccak hash functions by solving algebraic systems. page 1–13. IET Inf. Secur., 2022. https://doi.org/10.1049/ise2.12103.

[QHD+23] Lingyue Qin, Jialiang Hua, Xiaoyang Dong, Hailun Yan, and Xiaoyun Wang. Meet-in-the-middle preimage attacks on sponge-based hashing. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part IV*, volume 14007 of *LNCS*, pages 158–188. Springer, Cham, April 2023. https://doi.org/10.1007/978-3-031-30634-1_6.

[Raj19] Mahesh Sreekumar Rajasree. Cryptanalysis of round-reduced KECCAK using non-linear structures. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *INDOCRYPT 2019*, volume 11898 of *LNCS*, pages 175–192. Springer, Cham, December 2019. https://doi.org/10.1007/978-3-030-35423-7_9.

[WWF+21] Congming Wei, Chenhao Wu, Ximing Fu, Xiaoyang Dong, Kai He, Jue Hong, and Xiaoyun Wang. Preimage attacks on 4-round keccak by solving multivariate quadratic systems. In Jong Hwan Park and Seung-Hyun Seo, editors, *ICISC 21*, volume 13218 of *LNCS*, pages 195–216. Springer, Cham, December 2021. https://doi.org/10.1007/978-3-031-08896-4_10.

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

The instance of preimage of 3-round Keccak-256 is shown in Table 9.

Table 9: 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 Equations (1)				
		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 Equations (1)				
		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	

B An Instance of Preimage of 4-Round Keccak[r=640, c=160]

The instance of preimage of 4-round Keccak[r=640, c=160] (start round index $ir = 0$) is shown in Table 10.

Table 10: An instance of preimage of 4-round Keccak[r=640, c=160] (in little-endian order).

the 1^{st} message block				
09 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
the 2^{nd} message block				
52 25 14 75	c1 14 5e cf	5f 91 ee 85	85 6a b3 8e	1a 6c d0 89
27 9c ae b5	93 f1 2f a1	bc 73 a4 a4	e0 02 c1 95	0e eb 6a 07
62 06 8e 83	8d 14 47 94	f7 ae 66 77	8d 5d 65 bf	5a 66 4f 54
34 91 ce 3c	82 76 34 4e	7b a6 c7 7d	81 5d 6a 38	d4 01 3c cb
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
80-bit output digest				
75 1a 16 e5	e4 95 e1 e2	ff 22		

The verification code is provided as follows, which can be added to 'KeccakCrunchyContest.cpp' directly [BDH+a].

```
counter += verifyPreimageChallenge(
// Keccak[r=640, c=160, rounds=4]: preimage challenge
640, 160, 4, (const UINT8*)"\x75\x1a\x16\xe5\xe4\x95\xe1\xe2\xff\x22",
0, // fill in this line with the start round index (0=first)
(const UINT8*)"\x09\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
    x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
    x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
    x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
    x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x52\x25\x14\x75\xc1\x14\x5e\
    xcf\x5f\x91\xee\x85\x85\x6a\xb3\x8e\x1a\x6c\xd0\x89\x27\x9c\xae\xb5\x93\
    xf1\x2f\xa1\xbc\x73\xa4\xa4\xe0\x02\xc1\x95\x0e\xeb\x6a\x07\x62\x06\x8e\
    x83\x8d\x14\x47\x94\xf7\xae\x66\x77\x8d\x5d\x65\xbf\x5a\x66\x4f\x54\x34\
    x91\xce\x3c\x82\x76\x34\x4e\x7b\xa6\xc7\x7d\x81\x5d\x6a\x38\xd4\x01\x3c\
    xcb", 1278 // fill in this line
);
```

C The Algorithms

We summarize the precomputed probability table p and the MILP model in Algorithm 2 and Algorithm 3.

Algorithm 2: Precompute the probability table p.

Input: The required digest $\Theta^4_{*,0,*}$.

Output: The probability table p, the corresponding restrictions pr satisfying the best probability in p.

$I^3_{*,0,*} \leftarrow \iota^{-1}_{ir=3}(\Theta^4_{*,0,*})$; `// Get` $I^3_{*,0,*}$ `by applying the inverse` ι `of fourth round to` $\Theta^4_{*,0,*}$.

$p \leftarrow (p_{z,i,j} = -inf)_{32\times32\times5}$; `// Prepare a table` p `to record the best probability with initial value` $-inf$.

$pr \leftarrow (pr_{z,i,j} = null)_{32\times32\times5}$; `// Prepare an empty table` pr `to record the restrictions with respect to the best probability recorded in` p.

for $z \leftarrow 0$ **to** 31 **do**
    $d \leftarrow I^3_{*,0,z}$;
    for $i \leftarrow 0$ **to** 31 **do**
        for $j \leftarrow 0$ **to** 4 **do**
            foreach valid j restrictions, denoted by r (only bits under mask i are involved in r) **do**
                Calculate the input set I recording all the 5-bit inputs satisfying restrictions r;
                $cnt \leftarrow 0$;
                foreach ip in I **do**
                    if $\chi(ip)$ matches the digest bits d **then**
                        $cnt \leftarrow cnt + 1$;
                    end
                end
                if $cnt/|I| > p_{z,i,j}$ **then**
                    $p_{z,i,j} \leftarrow cnt/|I|$, $pr_{z,i,j} \leftarrow r$;
                end
            end
        end
    end
end
return p and pr;

Algorithm 3: Build the MILP model and get the result.

Input: The required digest $\Theta^4_{*,0,*}$.
Output: The way to linearize last two rounds with the 98 degrees of freedom maximizing the probability matching the digest.

$(p, pr) \leftarrow$ Algorithm 2$(\Theta^4_{*,0,*})$; `// Get the probability table` p`, and the corresponding restrictions` pr `by Algorithm 2.`
$V \leftarrow \{a_{X^2_{x,y,z}}\} \cup \{a_{\Theta^3_{x,y,z}}\} \cup \{a_{X^3_{x,0,z}}\} \cup \{a^z_{i,j}\}$; `// The MILP model will use` $800 + 800 + 160 + 32 \times 32 \times 5 = 6880$ `variables (`$a_{X^2_{x,y,z}}$`,` $a_{\Theta^3_{x,y,z}}$`,` $a_{X^3_{x,0,z}}$`, and` $a^z_{i,j}$`).`
$O \leftarrow (\Sigma_{z=0\sim31,i=0\sim31,j=1\sim4}(log_2(p_{z,i,j}/p_{z,i,j-1}) \times a^z_{i,j}), Maximize)$; `// The objective to the MILP model which should be maximized.`
$E \leftarrow \emptyset$; `// Initialize an empty set to record the equations that should be added to the MILP model.`

for $(x, y, z) \leftarrow (0, 0, 0)$ **to** $(4, 4, 63)$ **do**
  $S \leftarrow \emptyset$;
  for $(x', y', z') \leftarrow (0, 0, 0)$ **to** $(4, 4, 63)$ **do**
    if $X^2_{x',y',z'}$ is linear-dependent to $X^2_{x+1,y,z}$ or $X^2_{x+2,y,z}$ **then**
      $S \leftarrow S \cup \{X^2_{x',y',z'}\}$;
    end
  end
  $E \leftarrow E \cup \{a_{\Theta^3_{x,y,z}} \leq \Sigma_{s\in S} a_s\}$;
end

for $(x, y, z) \leftarrow (0, 0, 0)$ **to** $(4, 4, 63)$ **do**
  $T \leftarrow \{\Theta^3_{x',y',z'}\}$; `// According to the` θ`,` ρ`, and` π `operations, determine the set` T `(of size 11) where` $X^3_{x,y,z} = \bigoplus_{t\in T} t$`.`
  $E \leftarrow E \cup \{a_{X^3_{x,y,z}} \leq a_t | t \in T\}$;
end

for $z \leftarrow 0$ **to** 31 **do**
  $E \leftarrow E \cup \{\Sigma_{i=0\sim31} a^z_{i,0} \leq 1\}$;
  for $i \leftarrow 0$ **to** 31 **do**
    for $x \leftarrow 0$ **to** 4 **do**
      if $i \& 2^x > 0$ **then**
        `// If the` x`-th bit of` i `is 1.`
        $E \leftarrow E \cup \{a^z_{i,0} \leq a_{X^3_{x,0,z}}\}$;
      end
    end
    $E \leftarrow E \cup \{a^z_{i,j} \leq a^z_{i,j-1} | 1 \leq j \leq 4\}$;
  end
end

$E \leftarrow E \cup \{(\Sigma_{x=0\sim4,y=0\sim4,z=0\sim31} a_{X^2_{x,y,z}}) + (\Sigma_{i=0\sim31,j=0\sim4,z=0\sim31} a^z_{i,j}) + 8 \leq 98\}$;
$R \leftarrow \text{MILP}(V, O, E)$; `// Get the MILP result.`
$L \leftarrow \{X^2_{x,y,z} = r_{X^2_{x,y,z}} | R.a_{X^2_{x,y,z}} = 1\} \cup \{pr_{z,i,j} | R.a^z_{i,j} = 1 \text{ AND } (j = 4 \text{ OR } R.a^z_{i,j+1} = 0)\}$; `// Parse the result of the MILP model where` $r_{X^2_{x,y,z}}$ `is an arbitrary constant as random space.`
return L;