IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2021, No. 3, pp. 84–101. DOI:10.46586/tosc.v2021.i3.84-101

Improved Preimage Attacks on 3-Round Keccak-224/256

Xiaoen Lin, Le He and Hongbo Yu(✉)†

Department of Computer Science and Technology, Tsinghua University, Beijing, China
linxe17@tsinghua.org.cn, he-l17@mails.tsinghua.edu.cn,
yuhongbo@mail.tsinghua.edu.cn

Abstract. In this paper, we provide an improved method on preimage attacks of standard 3-round Keccak-224/256. Our method is based on the work by Li and Sun. Their strategy is to find a 2-block preimage instead of a 1-block one by constructing the first and second message blocks in two stages. Under this strategy, they design a new linear structure for 2-round Keccak-224/256 with 194 degrees of freedom left, which is able to construct the second message block with a complexity of $2^{31}/2^{62}$. However, the bottleneck of this strategy is that the first stage needs much more expense than the second one. Therefore, we improve the first stage by using two techniques. The first technique is constructing multi-block messages rather than one-block message in the first stage, which can reach a better inner state. The second technique is setting restricting equations more efficiently, which can work in 3-round Keccak-256. As a result, the complexity of finding a preimage for 3-round Keccak-224/256 can be decreased from $2^{38}/2^{81}$ to $2^{32}/2^{65}$.

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

1 Introduction

The SHA (Secure Hash Algorithms) is a family of cryptographic hash functions which have been standardized as the FIPS (Federal Information Processing Standards) by NIST (National Institute of Standards and Technology). Up to now, three generations of SHA standard have been proposed. Among these generations, SHA-1 is not secure now because collision resistance has been cracked by Wang et al. in [WY05]. Although SHA-2 is still secure till now, its resemblance with SHA-1 has also aroused doubts in terms of security. Therefore, NIST decided to launch a public competition to find a new hash function standard in 2008, and the Keccak function won the competition finally.

Since the publication of Keccak in 2008, numerous researches have been conducted. On collision attacks, most attacks depend on the differential trials. Dinur et al. [DDS12] proposed *target difference algorithm* in 2012 which can linearize 1.5 rounds and connect to 2.5-round differential trails so that realistic collisions for 4-round Keccak-224/256 can be found. After that, Qiao, Song, Guo et al. [GLL+20, QSLG17, SLG17] improved the method by making full use of the degrees of freedom and finding better differential trails so that realistic collisions for 5-round Keccak-224/256 can be found. On distinguishing attacks, Dinur et al. gave the first cube distinguisher on the Keccak sponge function [DMP+14] in 2014. In 2017, Huang et al. [HWX+17] developed the cube distinguisher and the conditional cube tester to realize practical distinguishing attacks on 7-round Keccak sponge

† Corresponding Author

Received: 2021-06-01 Accepted: 2021-08-01 Published: 2021-09-17

function under different capacities. Besides, there are many other attacks under different security settings. We would not list them all here since they are less relevant to our work.

In this paper, we focus on preimage attacks of round-reduced Keccak. In [NRM11], Naya-Plasencia et al. presented practical preimage analysis for 2-round Keccak-224/256. In [GLS16], Guo et al. improved the technique of linear structure and presented preimage analysis for up to 4-round Keccak. In [LS19], Li and Sun proposed a 2-block model and a new linear structure with more degrees of freedom left. The bottleneck of their strategy is that constructing the first block needs much more expense than the second one (the details will be further discussed in Section 3). As a result, they found a trade-off between the two blocks and succeeded in constructing the practical attacks on 3-round Keccak-224. In addition, their method also performed well on 3-round Keccak-256 and 4-round Keccak-224/256. All the preimage cryptanalysis on round-reduced Keccak-224/256 above are summarized in **Table 1**.

Table 1: Summary of preimage cryptanalysis on round-reduced Keccak-224/256.

Round	Instance	Complexity	Reference
2	Keccak-224	2^{33}	[NRM11]
		[a]2^{0}	[GLS16]
2	Keccak-256	2^{33}	[NRM11]
		[a]2^{0}	[GLS16]
3	Keccak-224	[a]2^{97}	[GLS16]
		[a]2^{38}	[LS19]
		[a]2^{32}	Section 4.1
3	Keccak-256	[a]2^{192}	[GLS16]
		[a]2^{81}	[LS19]
		[a]2^{65}	Section 5.2
4	Keccak-224	[a]2^{213}	[GLS16]
		[a]2^{207}	[LS19]
4	Keccak-256	[a]2^{251}	[GLS16]
		[a]2^{239}	[LS19]

[a] Note: those results do not include the complexities for solving the linear equation system (with a factor $O(n^3)$ where n is the number of linear equations).

Our contributions. Based on Li and Sun's work [LS19], we propose two techniques to improve the first stage of their work which is the bottleneck of their algorithm. The first idea is to construct multi-block messages rather than one-block message which can improve the inner state better and better so that more degrees of freedom can be left in the second stage. The second idea is to improve the setting of restricting equations so that more restrictions can be satisfied within the same degrees of freedom. Using these new techniques, we reduce the complexity of preimage attacks of 3-round Keccak-224/256 from $2^{38}/2^{81}$ to $2^{32}/2^{65}$.

Organization. We first give some preliminaries and notations about Keccak in Section 2. Then we introduce the related work in Section 3. In Section 4 and Section 5, we analyze our techniques used in 3-round Keccak-224 and 3-round Keccak-256 respectively. Some experimental results are presented in Section 6. At last, conclusions are summarized in Section 7.

2 Preliminaries

2.1 Sponge Construction

The sponge construction is a new iterative hash function framework proposed by Bertoni et al. [BDPA11]. As shown in **Figure 1**, it has two phases—absorbing phase and squeezing phase. In the absorbing phase, it receives the input message M (after padding) by r bits and mixes the inner state by function f repeatedly with an all "0" initial value (IV). In the squeezing phase, it outputs r bits and mixes the inner state repeatedly until the output reaches the required length ℓ.

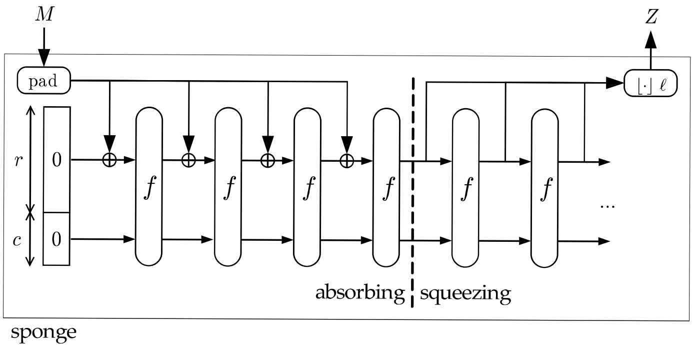

Figure 1: The sponge construction.

2.2 Keccak-f Permutation

The core of the sponge construction is permutation Keccak-$f[b]$, and the case of $b = r + c = 1600$ is chosen by NIST as SHA-3 standards. So, we only focus on the case of $b = 1600$.

As shown in **Figure 2**, the 1600 bit inner state can be organized as 5×5 64-bit lanes, denoted as $A_{x,y,z}$, where $0 \leq x, y \leq 4$, $0 \leq z \leq 63$.

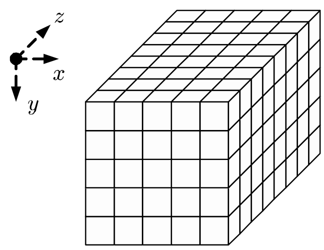

Figure 2: The Keccak-f state.

The Keccak-f consists of 24 rounds of permutation R, and each R consists of 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 bit-wise XOR and " $\cdot$ " denotes bit-wise AND. The indices of x and y are taken modulo 5, and the index of z is taken modulo 64. $r_{x,y}$ is a constant as listed in **Table 2**, and RC_z is a round-dependent constant which does not affect our attacks.

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

NIST standardized several versions of SHA-3 [Dwo15] with parameters $r = 1600 - 2\ell$ and $c = 2\ell$, where $\ell \in \{224, 256, 384, 512\}$. As for the padding rules, the message M is padded till the length is a multiple of r by concatenating a bit string of "10*1" and "0110*1" for Keccak and SHA-3 respectively.

2.4 Notations

For a certain Keccak-f permutation, we use capital Greek letters Θ, P, Π, X, I with a superscript to express the inner state after the corresponding step is executed. For example, in the last Keccak-f permutation, the first 256 bits of I^3 are the output of 3-round Keccak-256. And we may use extra three indices in subscript to express the bits in the inner state. Besides, we use " $*$ " to indicate all legal values. For example, $A_{*,y,z}$ is a row, $A_{x,*,z}$ is a column, $A_{x,y,*}$ is a lane and $A_{*,*,z}$ is a slice. Especially, we use H to denote the starting inner state of the *last* Keccak-f permutation (or the ending inner state I^3 of the penultimate Keccak-f permutation).

3 Related Work

In this section, we will briefly introduce Li and Sun's work [LS19] about preimage attack of 3-round Keccak-224/256.

3.1 Overall Idea

To obtain a 3-round Keccak-224/256 preimage, their work consists of three parts. First, they construct a first message block with a complexity of $2^{65}/2^{161}$ which can let the inner state H meet some restrictions that the second stage requests. Then, with the given inner state H, they adopt a new linear structure which can match specified 224/256 output bits of 3-round Keccak-224/256 with a complexity of $2^{31}/2^{62}$. At last, they find a trade-off between the two stages above, and reach an overall complexity of $2^{38}/2^{81}$.

3.2 The Basic Allocating Approach

To be more specific, they prove a theorem (Theorem 1) as shown below. This paper just cites the theorem and for the entire proof, please refer to [LS19].

Theorem 1 [LS19]. Let the messaged state[1] be (a) in **Figure 3**, i.e. bits in Row 0, 2 are unknowns, and bits in Row 1, 3, 4 are constants such that

[1] "the messaged state" means the inner state before θ operation in the first round

I. $a_{x,1,z} = a_{x,3,z} = a_{x,4,z} \oplus 1$, *and*
II. $\bigoplus_{x,z} a_{x,4,z} = 0$ [2]

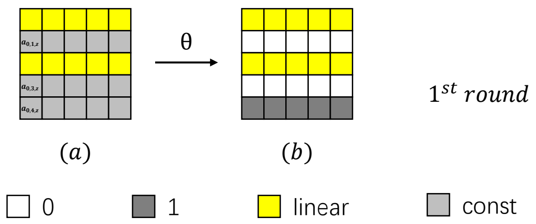

Figure 3: The inner states about θ operation in the first round.

where $a_{x,y,z}$ stands for the linear or constant bit at the position (x, y, z), $0 \leq x, y < 5$, and $0 \leq z < 64$. Then there exist constants $s_{x,z}$'s with $0 \leq x < 5$ and $0 \leq z < 64$, such that if assuming $\bigoplus_{j=0}^{4} a_{x,j,z} = s_{x,z}$, then the state (b) in **Figure 3** can be obtained by operating θ on (a). And hence, the KECCAK-f[1600] permutation can be linearized up to 2 rounds with 194 degrees of freedom left.

The application of Theorem 1 is shown in **Figure 4**. Suppose Condition_I and Condition_II are satisfied. Their structure begins with 10 undetermined lanes (640 variables). Then in θ operations of the first two rounds, they add 320 and 128 linear equations respectively (with one linear dependent equation in each round) to control the column sums and prevent the variable diffusions. After that, the inner state will be transformed as **Figure 4** shows. In the third round, since operations θ, ρ, π are linear, the inner state Π^3 will still be linear with $640 - 319 - 127 = 194$ degrees of freedom left.

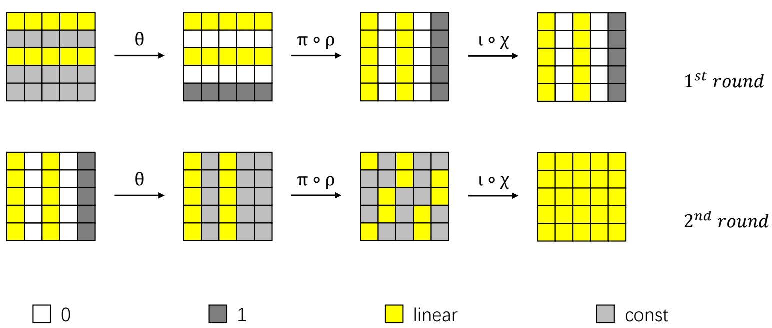

Figure 4: The 2.5-round linear structure with 194 degrees of freedom left.

Next, they use the 194 degrees of freedom to match specified 224/256 output bits. According to [LS19], each row with 4-bit output is corresponding to 4 linear equations, while each row with 3-bit output is corresponding to 2 linear equations and 1 quadratic equation. Let i_j and o_j with $j = 0, 1, 2, 3, 4$ be the input and output bits of χ, then:

$$\begin{aligned}
o_0 &= i_0 \oplus (o_1 \oplus 1) \cdot i_2 \\
o_1 &= i_1 \oplus (o_2 \oplus 1) \cdot i_3 \\
o_2 &= i_2 \oplus (o_3 \oplus 1) \cdot i_4 \\
o_3 &= i_3 \oplus (o_4 \oplus 1) \cdot i_0
\end{aligned}$$

[2]We use Condition_I and Condition_II to denote these two conditions in this paper.

Moreover, if 4 consecutive output bits are known, the expression of o_3 can be rewritten as $o_3 = i_3 \oplus (i_4 \oplus 1) \cdot (o_0 \oplus (o_1 \oplus 1) \cdot o_2)$. However, if only 3 consecutive output bits are known, the quadratic expression can not be simplified.

Notice that each linear equation can be ensured by spending 1 degree of freedom, and the rest can hold with a probability of $\frac{1}{2}$ for each. As a result, using the 194 degrees of freedom, they can construct the second message block matching specified 224/256 output bits with a complexity of $2^{31}/2^{62}$. Notice that there is an extra 2^1 complexity for 3-round Keccak-224. That's because the 224-bit digest contains 32 3-bit output rows while the 256-bit digest only contains 4-bit output rows. Thus, for Keccak-224, the 194 degrees of freedom can only satisfy all 192 linear equations and 1 quadratic equation, bringing an extra 2^1 complexity.

From Theorem 1 we can see that it is important to construct the first message block which makes the inner state H meet Condition_I and Condition_II as efficient as possible. Moreover, the first $1600 - 2n$ ($n = 224/256$) bits of any message state can be chosen arbitrarily. So all $a_{x,1,z} = a_{x,4,z} \oplus 1$ and part of $a_{x,3,z} = a_{x,4,z} \oplus 1$ in Condition_I can always be satisfied. Therefore, Condition_I can be simplified to $a_{x,3,z} = a_{x,4,z} \oplus 1$, where $3/2 \leq x \leq 4$ for Keccak-224/256.

To meet Condition_I and Condition_II in the starting inner state H of the second message block, they use Guo et al.'s work [GLS16] to construct the first message block. As shown in **Figure 5** and **Figure 6**, by eliminating the propagation of the θ operation in the first two rounds, the linear structure can fully linearize 2.5 rounds with 128/64 (for Keccak-224/256) degrees of freedom left. Using these degrees of freedom, they set 2 bits $\Pi^3_{0,3,z}$ and $\Pi^3_{0,4,z}$ of a certain slice $\Pi^3_{*,*,z}$ to be constant so that the 4 corresponding bits $X^3_{3,3,z}$, $X^3_{3,4,z}$, $X^3_{4,3,z}$ and $X^3_{4,4,z}$ in $X^3_{*,*,z}$ are linear, obtaining 2 satisfiable restrictions in Condition_I. In a word, they spend every 4 degrees of freedom satisfying 2 restrictions of Condition_I, and we call it *4-for-2 Strategy* in this paper (we improve this strategy and propose a technique named *5-for-3 Strategy* in Section 5). Under this strategy, they can satisfy 64/32 restrictions in Condition_I, while there are 129/193 restrictions of two kinds of conditions in total. So they need to enumerate $2^{65}/2^{161}$ times to meet all the rest restrictions. In summary, under the strategy in [LS19], the first message block can be fully constructed with a complexity of $2^{65}/2^{161}$.

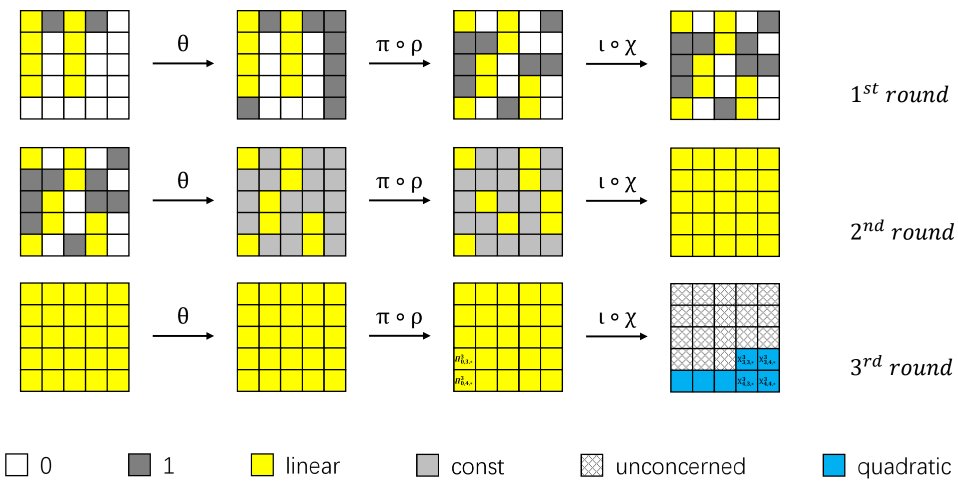

Figure 5: The 2.5-round linear structure for 3-round Keccak-224.

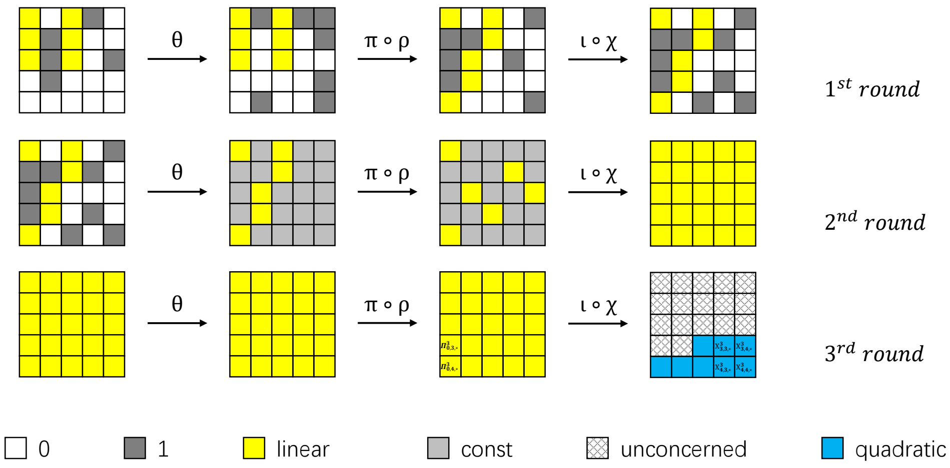

Figure 6: The 2.5-round linear structure for 3-round Keccak-256.

3.3 The Trade-Off of Allocating Approach

It is obvious that the bottleneck is constructing the first message block. So, they tolerate n_I pairs of bits ($a_{x,3,z}$ and $a_{x,4,z}$ for some x and z) not satisfying Condition_I which can reduce the complexity greatly. However, as shown in **Figure 7**, this causes quadratic bits in the inner state X^2 of the second stage. To eliminate the effects of these quadratic bits, each pair of bits that does not meet Condition_I will cost another 1 degree of freedom to set a linear bit to be constant (the orange square). So the overall complexity becomes $\frac{2^{65}}{C_{65}^{n_I}} + 2^{31+n_I}/\frac{2^{161}}{C_{161}^{n_I}} + 2^{62+n_I}$, reaching a trade-off complexity of $2^{38}/2^{81}$ ($n_I = 7/19$) for 3-round Keccak-224/256.

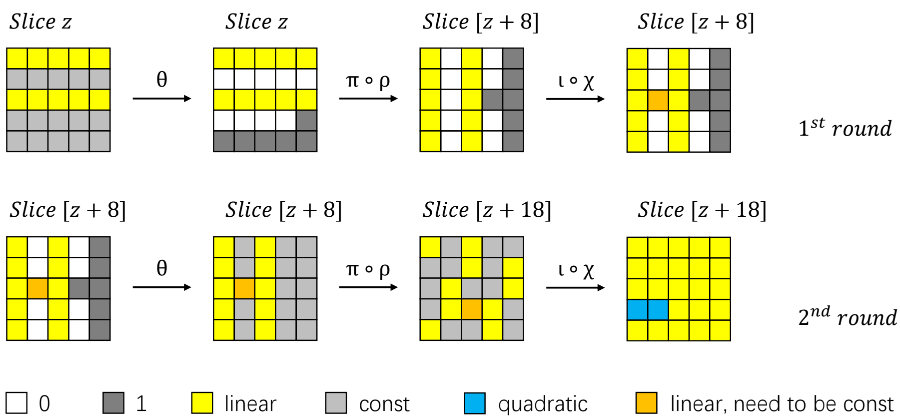

Figure 7: A case of effects caused by one unsatisfied restriction of Condition_I.

4 Improved Preimage Attack on 3-Round Keccak-224

In this section, we will analyze preimage cryptanalysis on 3-round Keccak-224. We will discuss a technique named *Iterating Strategy*, which can provide a better inner state H (satisfying more restrictions under the same complexity) for the second stage.

4.1 Iterating Strategy

Li and Sun's strategy [LS19] uses two message blocks corresponding to the two stages. And the goal of the first stage is to reach an inner state H which meets Condition_II and as many restrictions of Condition_I as possible. However, if we construct multi-block messages rather than one-block message to implement the same effect in the first stage, the complexity can be further decreased.

For Keccak-224, as shown in **Figure 8**, we do not spend the degrees of freedom in the second message block matching the output bits directly, but we spend them restricting more opposite pairs of bits (satisfy some restrictions of Condition_I) as the first message block does. Similarly, we use the third message block to restrict more opposite pairs of bits as the first two message blocks do. Iteratively, there will be more and more opposite pairs of bits in each inner state H, which means more and more restrictions will be satisfied. After a good-enough inner state H is found, we construct the last message block matching the target output bits. And we get the entire preimage eventually.

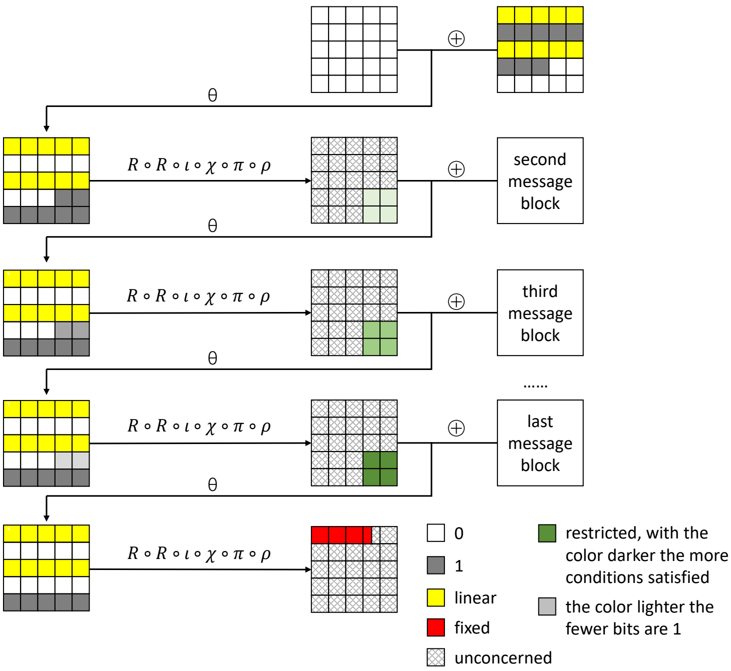

Figure 8: *Iterating Strategy* on 3-round Keccak-224. ($R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$)

The overall complexity of improved preimage attacks on 3-round Keccak-224 is analyzed as follows.

For each new message block in the first stage, we must satisfy Condition_II randomly with a complexity of 2^1. Suppose that there are k restrictions of Condition_I which are *not* fulfilled in the previous message block. Then we need to spend k degrees of freedom eliminating the effects of quadratic bits in X^2, and there remain $194-k$ degrees of freedom. Within these degrees of freedom, we spend $\lfloor \frac{194-k}{4} \rfloor \times 4$ of them on satisfying $\lfloor \frac{194-k}{4} \rfloor \times 2$ restrictions of Condition_I via *4-for-2 Strategy* (one more restriction can be satisfied if there exactly remain 3 degrees of freedom). Therefore, we can ensure at least $\lfloor \frac{194-k}{4} \rfloor \times 2$ out of 128 restricting equations. If we iterate once and want to get the new message block with k' ($k' \leq 128 - \lfloor \frac{194-k}{4} \rfloor \times 2$) restrictions of Condition_I *not* be fulfilled, then we need to enumerate $2^1 \times (2^{128-\lfloor \frac{194-k}{4} \rfloor \times 2}) \div (C^{k'}_{128-\lfloor \frac{194-k}{4} \rfloor \times 2})$ times in average.

A possible iterating process is listed in **Table 3**.

Table 3: A possible iterating process of 3-round Keccak-224 via *Iterating Strategy*.

message block id	k	k'	complexity
# 1	128	35	$2^{9.71}$
# 2	35	14	$2^{11.23}$
# 3	14	10	$2^{10.18}$
# 4	10	9	$2^{10.51}$
# 5	9	8	$2^{12.15}$
# 6	8	7	$2^{14.01}$
# 7	7	5	$2^{18.48}$
# 8	5	4	$2^{19.50}$
# 9	4	3	$2^{22.45}$
#10	3	2	$2^{25.87}$
#11	2	1	$2^{28.00}$
	2	0	$2^{33.00}$

After an 11-block iteration, we get an inner state H which satisfies Condition_II and most restrictions of Condition_I (except 1 restriction) with a complexity less than 2^{29}. Considering the padding rules, we need to ensure $H^3_{2,3,63} = H^3_{2,4,63}$ with an extra complexity of 2^1. Totally, we get the inner state H with a complexity less than $2^{29+1} = 2^{30}$. Finally, we enumerate the 12^{th} message block $2^{224+1-194+1} = 2^{32}$ times (the first "1" is for 1 quadratic equation, and the second "1" is for 1 unsatisfied restriction) to get an entire preimage of 3-round Keccak-224. The overall complexity is 2^{32}. Besides, we can get an inner state H which satisfies all restrictions of Condition_I and Condition_II with a complexity of 2^{33} (the experimental results are presented in Section 6).

5 Improved Preimage Attack on 3-Round Keccak-256

Improved preimage attacks on 3-round Keccak-256 will be analyzed in this section. In addition to the technique of *Iterating Strategy*, we will discuss another technique named *5-for-3 Strategy* which can make better use of the degrees of freedom in the first stage.

5.1 Iterating Strategy

For Keccak-256, the only differences are the number of output bits and the number of restrictions of Condition_I, so the *Iterating Strategy* can also be used in preimage attack on 3-round Keccak-256 directly as shown in **Figure 9**.

For each new message block in the first stage, we must satisfy Condition_II randomly with a complexity of 2^1 as well. We still use symbols k and k' to express the number of unsatisfied restrictions of Condition_I in the previous and current message block respectively. Then k degrees of freedom will be spent on eliminating the effects of quadratic bits in X^2 with $194 - k$ degrees of freedom left. However, there are as many as 192 restrictions of Condition_I while we still only satisfy $\lfloor \frac{194-k}{4} \rfloor \times 2$ of them. Therefore, if we iterate once for the new message block with k' restrictions of Condition_I *not* be fulfilled, we need to enumerate $2^1 \times (2^{192-\lfloor \frac{194-k}{4} \rfloor \times 2}) \div (C^{k'}_{192-\lfloor \frac{194-k}{4} \rfloor \times 2})$ times in average.

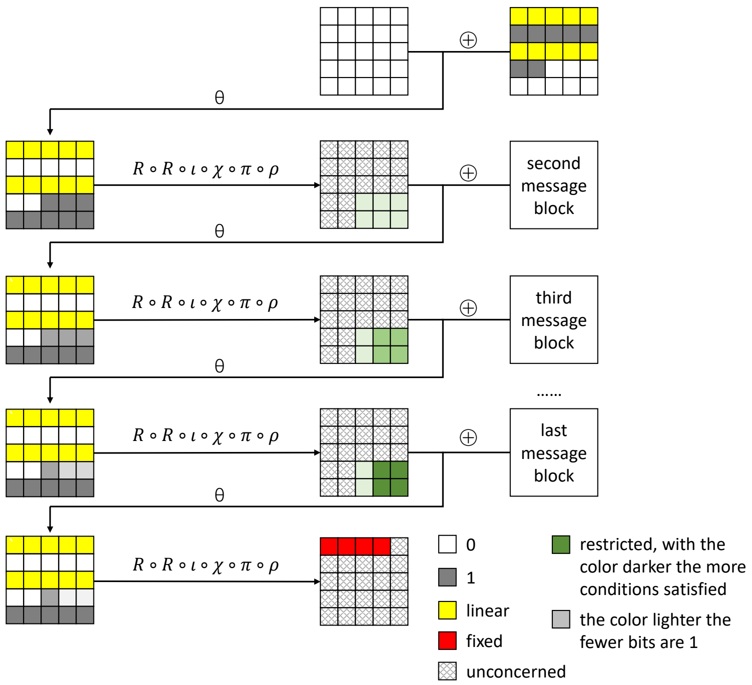

Figure 9: *Iterating Strategy* on 3-round Keccak-256. $(R = \iota \circ \chi \circ \pi \circ \rho \circ \theta)$

A possible iterating process is listed in **Table 4**.

Table 4: A possible iterating process of 3-round Keccak-256 via *Iterating Strategy*.

message block id	k	k'	complexity
#1	192	80	$2^{8.97}$
#2	80	20	$2^{58.45}$
#3	20	10	$2^{62.14}$
#4	10	8	$2^{63.56}$
#5	8	7	$2^{67.10}$

After a 5-block iteration, we get an inner state H which satisfies Condition_II and most restrictions of Condition_I (except 7 restrictions) with a complexity less than 2^{68}. And to deal with the padding rules, we need to ensure $H^3_{1,3,63} = H^3_{1,4,63}$ with an extra complexity of 2^1. Totally, we get the inner state H with a complexity less than $2^{68+1} = 2^{69}$. Finally, we enumerate the 6^{th} message block $2^{256-194+7} = 2^{69}$ times to get an entire preimage of 3-round Keccak-256. The overall complexity is 2^{69}.

5.2 5-for-3 Strategy

Comparing with Keccak-224, Keccak-256 has one more type of Condition_I ($x = 2$). Due to the limitation of linearization, we totally ignore this type of restrictions. Surprisingly, by spending one more degree of freedom for a slice, we can satisfy one more restriction of type $x = 2$, which is more efficient.

Consider the two slices $\Pi^3_{*,*,z}$ and $X^3_{*,*,z}$ (we can use $X^3_{*,*,z}$ to replace $I^3_{*,*,z}$ since the last two rows never change after ι operation). In order to meet Condition_I, we need to satisfy that:

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

From the χ operation $A_{x,y,z} = A_{x,y,z} \oplus (A_{x+1,y,z} \oplus 1) \cdot A_{x+2,y,z}$, we have:

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

To ensure the satisfaction of equations (2), we add 5 linear equations on Π^3:

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

To sum up, we spend every 5 degrees of freedom on satisfying 5 linear equations so that 3 restrictions of Condition_I will also be satisfied. We name this strategy *5-for-3 Strategy*. Then for Keccak-256, the *5-for-3 Strategy* can take the place of *4-for-2 Strategy* as shown in **Figure 10**.

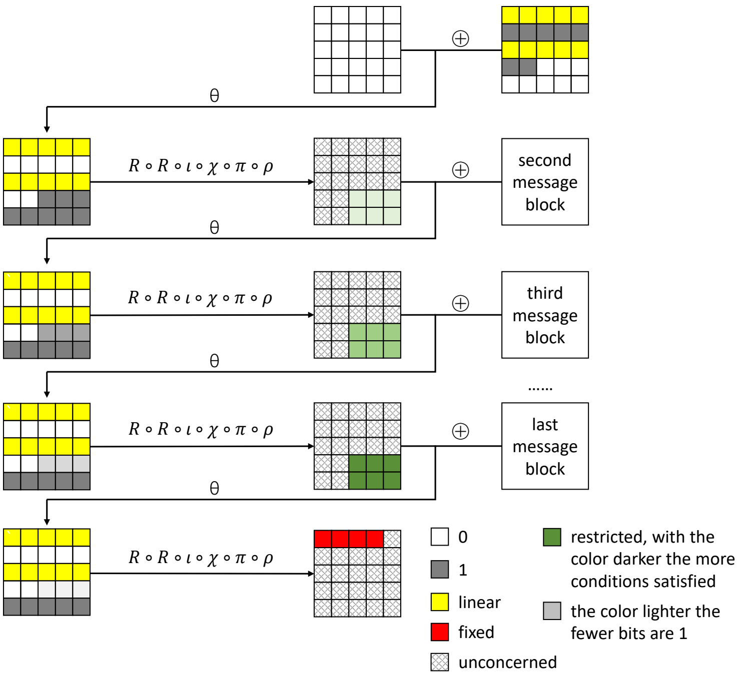

Figure 10: *Iterating Strategy* via *5-for-3 Strategy* on 3-round Keccak-256. ($R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$)

The overall complexity of improved preimage attacks on 3-round Keccak-256 via *Iterating Strategy* and *5-for-3 Strategy* is analyzed as follows.

We satisfy Condition_II with a complexity of 2^1. And we spend k degrees of freedom eliminating the effects of quadratic bits in X^2 and there remain $194 - k$ degrees of freedom. Next, we spend $\lfloor \frac{194-k}{5} \rfloor \times 5$ degrees of freedom satisfying $\lfloor \frac{194-k}{5} \rfloor \times 3$ restrictions of Condition_I. Suppose that each of the rest $192 - \lfloor \frac{194-k}{5} \rfloor \times 3$ restrictions of Condition_I will fulfil with a probability of $\frac{1}{2}$. Then the probability that exactly k' restrictions *not* be fulfilled is $(C^{k'}_{192-\lfloor\frac{194-k}{5}\rfloor\times 3}) \div (2^{192-\lfloor\frac{194-k}{5}\rfloor\times 3})$. Taking into account Condition_II, we are expected to enumerate $2^1 \times (2^{192-\lfloor\frac{194-k}{5}\rfloor\times 3}) \div (C^{k'}_{192-\lfloor\frac{194-k}{5}\rfloor\times 3})$ times to get a new inner state.

A possible iterating process is listed in **Table 5**.

Table 5: A possible iterating process of 3-round Keccak-256 via *Iterating Strategy* and *5-for-3 Strategy*.

message block id	k	k'	complexity
# 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	$2^{62.78}$

After a 16-block iteration, we get an inner state H which satisfies Condition_II and most restrictions of Condition_I (except 3 restrictions) with a complexity less than 2^{63}. And to deal with the padding rules, we need to ensure $H^3_{1,3,63} = H^3_{1,4,63}$ with an extra complexity of 2^1. Totally, we get the inner state H with a complexity less than $2^{63+1} = 2^{64}$. Finally, we enumerate the 17^{th} message block $2^{256-194+3} = 2^{65}$ times to get an entire preimage of 3-round Keccak-256. The overall complexity is 2^{65}. The experimental results of the first 12 message blocks are presented in Section 6.

6 Experiment

We will present the experimental results in this section. First, we will show the results of preimage attacks on 3-round Keccak-224 including the two stages. Next, we will show the results of the first stage of preimage attacks on 3-round Keccak-256.

Results of Keccak-224. In the first stage, we run 400 processes on 2.50 GHz CPU for about 2 hours to get the results. According to our experiment, solving a linear equation system costs about $2^{19.3}$ cycles in average. So, the expected costs of getting the results are $2^{33} \times 2^{19.3} = 2^{52.3}$ cycles. Meanwhile, the experimental result costs $400 \times 2.5 \times 2^{30} \times 7200 = 2^{52.78}$ cycles, which is in line with expectations.

The experimental results consist of 11 message blocks, and the produced inner state H can meet all 128 restrictions of Condition_I as well as Condition_II and the padding rules.

Due to the limitation of space, the 11 message blocks together with the last message block are listed in Appendix A. Here we only list part of the results in **Table 6**.

Table 6: The inner state H (in little-endian order).

the inner state after 11 message blocks				
e8607ad31bf82c29	108f3f79af33a40b	91f9fc271728393f	1312a1a67d97af82	d2d7f7468979007b
ee14a076f8c3956a	0917f9faceecfc18	b0ba65b1a2889be7	fd54b7280431cf9d	7ff153da60d37e49
03cbf192382c2826	877d2d5fdf9542a2	036d316b1bd49c02	ce3683a1e78c9dd2	3c3ffc6c8dbfc786
0c321c19a083c89f	2a4f2a6d8fa38c09	410eea37f6cf19f5	f806a2ff56a7105a	410a3228e0868a50
fd255898fbbae50c	e5e3b70a10e1acac	5edc01abb491bd9e	07f95d00a958efa5	bef5cdd71f7975af
XOR values of restrictions of Condition_I				
----------------	----------------	----------------	ffffffffffffffff	ffffffffffffffff

Using this inner state H, we get a preimage matching 224-bit all '0' digest in the second stage (we use the code published in [LS19] to get the preimage with an NVIDIA GTX 1080 Ti card for around 10 hours). The results are listed in **Table 7**.

Table 7: The last message block and the 3-round digest (in little-endian order).

the 12^{th} (last) message block				
94cbfb3a690a8d98	04a85c22dab8e6b0	8f0cfb9b0c442bd2	50e15a0c65acf5ed	04ace5f5db4c6d9d
ecce0711fc868f99	130bb10f21f2af4b	11999be5e9e6d986	055215d75296dfc7	3efb61f28055f419
b4432a530ccb79d0	8c966bcac722ad59	5549925e1d71107d	a73a1343cd3689de	a334a0e63f0cc6e4
0ee8bb7ea4c6d26c	3053629860bddf5a	e02d1463bda15b94	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
3-round digest				
0000000000000000	0000000000000000	0000000000000000	00000000	

Results of Keccak-256. We run 200 processes on 2.50 GHz CPU for 1 hour to get the results of the first stage (we get four results satisfying 171 restrictions, and we only present one of them). According to our experiment, solving a linear equation system costs about $2^{20.6}$ cycles in average. So, the expected costs to get four results are $4 \times 2^{25.67} \times 2^{20.6} = 2^{48.27}$ cycles. Meanwhile, the experimental result costs $200 \times 2.5 \times 2^{30} \times 3600 = 2^{50.78}$ cycles, which is roughly in line with expectations.

The experimental results consist of 12 message blocks, and the produced inner state H can meet 171 restrictions of Condition_I as well as Condition_II and the padding rules. The entire 12 message blocks are listed in Appendix B. Here we only list the inner state H in **Table 8**.

Table 8: The inner state H (in little-endian order).

the inner state after 12 message blocks				
2765917b3a69a027	babfba7f5b4b9e9b	8e8e9da01404fefe	f436e5591cac135b	3ac10eb1fc266a3f
1f9179d78612d85a	daa6ac87f59b5c9b	98faf46dd3916d84	31262adedde125d1	92d3a6ef9e96d541
d0ad46049bf7e2f7	d9eed66f20681de6	608e0871b2bd46c2	4201125674bca08d	64f129f30c304db7
effa5f9f4e707313	6d7a0c483758079d	69b60e60ca1b52f2	e766493834523982	17c7ef107de53f8f
af968f7afc60c504	ae02837a7c48ec83	9649f1df3cacac07	1899b6c749adc25d	e83810efe292d1d8
XOR values of restrictions of Condition_I				
----------------	----------------	ffffffbff6b7fef5	ffffffff7dfffbdf	ffffffff9f77ee57

7 Conclusion

In this paper, we provide an improved preimage cryptanalysis on 3-round Keccak-224/256 based on the work of Li and Sun. The main idea is to improve the first stage which is the bottleneck of their work. For this goal, two techniques are proposed:

- We propose *Iterating Strategy* which can provide more degrees of freedom by using more than two message blocks.
- We propose *5-for-3 Strategy* which can satisfy more restrictions within the same degrees of freedom.

Using these techniques, we decrease the complexity of finding the restricted inner state. After trading off, the total complexity is decreased as well. It is expected that the complexity of preimage attacks on 3-round Keccak-224/256 can be decreased into $2^{32}/2^{65}$.

It is noted that our techniques are still far from threatening the security of other Keccak variants or more rounds. Larger digest versions use one-block message framework [GLS16] which depend on the all '0' IV rather than an inner state with some specific conditions. As for the attack with more than 3 rounds [LS19], the bottleneck is in the second stage which we do not optimize. However, our techniques may be applied when some new attacks are proposed.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (Grant Nos. 2017YFA0303903 and 2018YFB0803405). We are grateful to Yao Sun and Ting Li for their help of calculating the second-stage results of the preimage attacks on 3-round Keccak-224.

References

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011.

[DDS12] Itai Dinur, Orr Dunkelman, and Adi Shamir. New attacks on Keccak-224 and Keccak-256. In *FSE 2012*, volume 7549 of *LNCS*, pages 442–461. Springer, Heidelberg, 2012.

[DMP+14] Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Practical complexity cube attacks on round-reduced keccak sponge function. *IACR Cryptol. ePrint Arch.*, 2014:259, 2014.

[Dwo15] M. Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. 2015.

[GLL+20] Jian Guo, Guohong Liao, Guozhen Liu, Meicheng Liu, Kexin Qiao, and Ling Song. Practical collision attacks against round-reduced SHA-3. *Journal of Cryptology*, 33(1):228–270, 2020.

[GLS16] Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced Keccak. In *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 249–274. Springer, Heidelberg, 2016.

[HWX+17] Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional cube attack on reduced-round Keccak sponge function. In *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 259–288. Springer, Heidelberg, 2017.

[LS19] Ting Li and Yao Sun. Preimage attacks on round-reduced Keccak-224/256 via an allocating approach. In *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 556–584. Springer, Heidelberg, 2019.

[NRM11] María Naya-Plasencia, Andrea Röck, and Willi Meier. Practical analysis of reduced-round Keccak. In *INDOCRYPT 2011*, volume 7107 of *LNCS*, pages 236–254. Springer, Heidelberg, 2011.

[QSLG17] Kexin Qiao, Ling Song, Meicheng Liu, and Jian Guo. New collision attacks on round-reduced Keccak. In *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 216–243. Springer, Heidelberg, 2017.

[SLG17] Ling Song, Guohong Liao, and Jian Guo. Non-full sbox linearization: Applications to collision attacks on round-reduced Keccak. In *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 428–451. Springer, Heidelberg, 2017.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 19–35. Springer, Heidelberg, 2005.

A One Instance of Preimage of 3-Round Keccak-224

Table 9: One instance of preimage of 3-round Keccak-224 (in little-endian order).

the 1^{st} message block				
e70b144e6be90cb8	19dcf87383f85c33	9304bc275c51774d	748e51e030d3a5f0	d84a88aeed0d026c
ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	ffffffffffffffff
b25e411b3ebc59ed	4c89ad26d6ad0966	c651e97209042218	de24fb4a9a790f5a	72e0220447a7a8c6
ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 2^{nd} message block				
126c0858bd87808f	d169a3a2b42a7d87	236a463ebcfa1e72	b575a7f6a25448b3	67b053a68d0e5d68
b571ba98044d13ef	d11d3181c1658d22	0b97e54a9e4561d1	23f4195bcf025054	58c064b300430101
58dd068893abacac	4192a6bf207e38f8	2fb0ef20c7d2b44e	a9ac8c4012aa0bb3	99718c55c10d45b7
2e9ea20a92420435	171e69bc10dcb8d5	4b2994a22b987322	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 3^{rd} message block				
29cdbbf36beae8b1	4ed14c7c084d7542	88a82f933c29b2fa	eb106ddd368c5079	072dccfab5cac97e
690bc81515956d06	06857b25ca1dbfa1	d8e4ebd52a84cabe	feae97e22572674b	df5e1223a7b9dae3
593163e38b1ffcfd	36b9db4640b1ae8f	516696d5d487b5a3	16109192752222d5	08a012a7e6fcbe84
e471a94872d60abd	23414a4e0ee274d0	58c49b3f3bcb10e3	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 4^{th} message block				
b8f14d68c2466398	b04bec7bf706651b	cc816d641c44d7f4	6e7791a6de6aebdb	bc3621710c563787
f1114ed799c15859	1159303eb53d1904	a43da120f9a496ce	79e6753ca0d56ef1	a4833be79aa1ebca
79fea00268a47847	86cc61404b825937	9daeb7376181dfbf	d5b10848d85c23cd	ee2745175c70a401
2cad0bf77d69c08c	8a0ddc71a2026415	e6f88b6646072178	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

the 5^{th} message block				
5b0f71376d3a70ba	cd2739c5fadda85a	0b9ddf9bbbe51bc2	aa8f5604e72cca04	327b2859ef99fac4
bfbc34703411a2fa	8fcc4d914af0b3f2	546384b82efe91e4	519bd2ef4eeaf985	14aeaa231610f348
3c3105d79f3bfd80	939766c39a837c76	bfd5be0ef5f69520	722d2793f2318895	d905b7527c7f5902
08fe2e665d7dd3a6	6748247fd85ceae3	0d32f9c5ed6965c4	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 6^{th} message block				
feb58371dfcf2216	89c766f532b8316b	e85e903bffc3cc2f	d1df32dc334cedc7	4d001955adcc0ddd
b1e9642ae989c7a6	f09b69513963a013	4007d09344a0857e	067337a7e954a153	297c3064bec54c68
224d2a3c8bfb8952	abeda9614102927d	9002f20550263063	3ffe6c48d14fa3c8	fc3a4f0cdd601b65
08b0ea7424822299	c081496ac3cd5a4b	4c0eeff16df70d04	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 7^{th} message block				
09a6de6d57941573	0d04af2a9e8396c2	0e8359e5ef304860	de97db39e8c40760	8e51fce527607ea6
9581e722d724a6dd	f883a4f03c806a42	c776a9bd3f2ac6be	b2eb552628967520	7e9a02670e4b3363
56ddbafd17ad6a6a	5f30fcf2a6da8f3c	0ccfb643ade5d88b	92bb34d46adb073a	ce5f115897805146
e8ede2750bc647b0	1e9aee7ff5049662	3a5f971686d56a0c	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 8^{th} message block				
c3717eb68e997064	f1955c85f149fb75	d012b8cbaea66317	240e292c3a0f9581	d95fa140d7c9b6c2
5aa977ee563d65f5	1381fa5dc4027683	8afb2ba571d7ed3d	af03d7e0886a7608	7dd6eca4b971a63a
7f3be7d88dfee519	06cb619412ee1786	c8d02210a57753be	7dddabe802cc0064	29117733d884d79d
5987eedb1e90e223	6621ba073436c820	08d29253cb32150a	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 9^{th} message block				
fa2f1364cf301a4e	b3df5bc05be02cc3	8af99bd748114722	350423c6ace57a31	83702fcc38ac39f0
58881c22e2b72723	ab2f837983735e25	24bd203f75521b46	6745c6b9fbe5d1a5	27134eb9b4afc7c5
99afc3f20e0f3dfd	a1637c7713a854c9	2b057402681026a0	0badba430e13f5f5	5f382f11009c76ef
b39e2c586ffd9ab8	464b33aef2b3f411	78913c656e29dfa9	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 10^{th} message block				
637229c9f6c8c626	570d82f708765ca0	7082d7954711739f	6111dbe140d3eec2	7e625020c241971e
4c95b9931496b14f	3339cd2b009a07dd	c5d8c62552215442	bd4046ef8ebb548a	3614eb4e25c4f78e
85c93218aa5281a5	c5c829c8d9eb165d	b3233f210351e37f	c674fd81f5298bf4	fd59a295e6a869bd
48249213fcee254b	57047ac5c0a3cb8c	2272e60ff46761ab	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 11^{th} message block				
5d623af067561182	e6c936759f313b6a	76f598a72df9c7a8	d6fc8957f41ed999	0a80227e8ea73404
249015615d43d6dc	1baff4359dcb70d2	20667c69c7de9173	43be9dc29ba6c8f1	274604c962a3dfb1
8f19ce954c6d8a8d	ad1f8dcf25e6aa43	870c5e19fddb1be8	2bee83fc5c8dfd97	ab85478a45b5a3df
292423d29d6d1dd2	507a9a5bc2e3bc61	1c509b99450abfa4	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 12^{th} message block				
94cbfb3a690a8d98	04a85c22dab8e6b0	8f0cfb9b0c442bd2	50e15a0c65acf5ed	04ace5f5db4c6d9d
ecce0711fc868f99	130bb10f21f2af4b	11999be5e9e6d986	055215d75296dfc7	3efb61f28055f419
b4432a530ccb79d0	8c966bbcac722ad59	5549925e1d71107d	a73a1343cd3689de	a334a0e63f0cc6e4
0ee8bb7ea4c6d26c	3053629860bddf5a	e02d1463bda15b94	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

B One 12-Block Instance of the First Stage of Preimage Attacks on 3-Round Keccak-256

Table 10: The first 12 blocks in the first stage of preimage attacks on 3-round Keccak-256 (in little-endian order).

the 1^{st} message block				
a87e4b4591f0687c	84def99bf4272cd9	c723d6e67f3e7b6b	e26ce2551b109fdb	f8c07a91b5e04142
ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	ffffffffffffffff
02d4e1ef3b5ac2d6	2e7453315e8d8673	927683b32a6b2e3e	b739b7004e45ca8e	ad952fc4e0b51417
ffffffffffffffff	ffffffffffffffff	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 2^{nd} message block				
0131127868c5eca4	481e88fef7f835b2	3f7c30ee2b18bc9b	29169e7fe1fa7a05	c8488dd9b24b1612
8c9a16cbe89414be	aa800aa0467902a4	8ace1ce757f8d35c	352b978f459471ee	116648278b89f778
04d1167b50c6b654	27a4060a2ce64c45	33da42207746a6c8	2a6d8cc7c06aba6b	1af754519656a39a
3b1a48403b167ac0	0b3e40bb0e25d475	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 3^{rd} message block				
676509fd04883731	c312babe7ad90df3	5522440426d28a78	c6b2b6dce094cda2	cd79d6bd710b8e31
64f0fb307ab9290f	56100677b01c58b6	ba2e24ac8d38f687	38ff427956f53a0c	9fbe70623351aef2
e9e8f7b6de7e7060	6dc7f1b29e81f89b	5a8e29c217d109d5	c772c00b58b826c4	3a2be13d0c3c5499
5747eb9510effc99	d6052132c2427f33	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 4^{th} message block				
a4cda31899256497	715563d6a427264a	755ee24e3fd7ce95	315509f648246f32	5b213820188c9d8e
1481618cc5e427d8	2e75a7ff053e721d	496dc64aa616d685	d897634e7f2b2df4	71be70cf3889499e
ff7a96e424a94555	a9f6c5aeb830853c	6c670055219266df	3c3a068154aa9390	96cf729838728202
db64a740f9589513	2764bb94b367a884	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 5^{th} message block				
dcbe58b183dc41d4	e34fd3c7e5aa21ef	a0f66fd7ca634450	ba4c4fb944ab81ac	dcc19648624e9e8b
e7fe18b5136741f8	0a976baec941006b	db18ddf29be6a93b	e7b5768114b31638	cb033abd5c1c30dc
08293758f3e869c3	e5ac7ea44c8ef07b	a7f0477695ef4804	1f3d6259c1b0d9f8	884a30db099f2230
cff7efffe0a6343e	ed545a88086e87a7	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 6^{th} message block				
d485e3166f82636f	aa42ebc6e29aa2be	e095e87c8ebc6db1	a27a311655072c2c	46820369e658deac
2d1574ee508e7bf2	91d253375dc273b8	172f37e02d6e2a86	1d45c223f2b41991	d6cf75f27cfb0dad
afd53e5dffc7caa7	f88350234c478dbd	624adf441e7ef330	4bba7d186e494252	b4008fbfc0631902
9555eabaec092b47	b50622a2f9efae69	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 7^{th} message block				
83044e63844ddbb8	d135b0d01ed47bfb	c83bf865df75ebbb	7aa1b9676f00456d	5546998bc4dfa501
da591b6a1f39ea43	b792d66ad8372922	c7cc994c55e7e6cd	eb66775db8e3e0ba	d5b8a76ac3a5a58b
aecc0becddf84002	0cd9696cfd226270	81e3cd39659715a7	33b7a38cd7c2c151	284f2e868b30eee7
e3eabb69eb6b40e7	37a65b3cbbab1f84	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 8^{th} message block				
ea416a86ba6bba7a	07dd944d5a5207d2	e2216052ca5eec77	3c4a4fe6085bd5dc	6acd039872ec89bf

99addf32207e4b56	500381ce5c0d8346	1ae7e42d5785e08f	2d9a4114702571d4	57c502924aa9f8ac
b9898734c4c411bf	c7ed05a43fc5e4eb	b45ab22d24647598	e39ddf99b2a560b7	e3685a210700580d
edef697bdd2923ef	6138f8f38a91a2b4	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 9^{th} message block				
74cf7ad3cb5c688b	293f72ffaac3be04	daa3e1c626d20c92	60e555cb822d2345	e10996a979a9c520
22546ae5928c17b8	24bf5d7e40539aeb	a414620eeab4b716	13336ce0280f51b4	da3652dc78ebfdb9
028f93b799f1c702	b635e84f848cc584	ec9416f01ac3530f	9afde1444f2839d7	002753dd1e0c0bcf
db7d222f5992b23a	c5ce1a296c383421	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 10^{th} message block				
799035349bd98195	b3e6579ffbb7d6d4	5318dbdd25b98223	41426631d911493e	dd85dacbc065c2cb
143fc388b42666ed	616c75eeff801ce1	3fe11c04e8e09a94	7af8ddd6a87a868b	fc22dd5cd84e636d
5c0604d54f600df7	0bc31c8706eb4fb0	66239737aa0ab292	86d3116db845895f	b764a2fd34b28ed5
c6afa57f4fdae837	80eaac6257d32938	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 11^{th} message block				
487aa7cc49c5536f	8311c849d9fc74fa	966a2052bbe40e51	b8a63ab937c7c257	d10e0b299815a791
0379109824528280	fc9aff577593b9b6	3758db6078aeb1ca	b2e887ebbdca1e22	7e317366421357f2
ee81d9847ca04df5	a9fc5005a9a42c32	2f5b69fb9e0d4001	a24bd5e2145bbc0e	8e77215236c03891
d7b8738f92eaa99e	6ba852e2c5f8647e	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
the 12^{th} message block				
dc0dbf7d91fb4441	6aa5fbafa98221f7	89050db49c6e9897	2a349d48906c64e2	dca1a319838c015f
b35f33fc4a4930cd	6ec20bf573640766	05ec836c5995b2c1	e292afadd198efaf	4d29bc75680254c1
a77563d658388351	87e8a16725f03427	7ac7d5326c0cc6fe	87fdf37cef971818	cce3d2d85bed65e5
79ffefa6cc33e89f	3d9bee727ec02c95	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000