Preimage Attacks on 4-round Keccak by Solving Multivariate Quadratic Systems

Congming Wei[1], Chenhao Wu[2], Ximing Fu[2,3], Xiaoyang Dong[1], Kai He[4], Jue Hong[4], and Xiaoyun Wang[1]

[1] Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China
[2] The Chinese University of Hong Kong, Shenzhen, Shenzhen, China
`fuximing@cuhk.edu.cn`
[3] University of Science and Technology of China, Hefei, Anhui, China
[4] Baidu Inc., Beijing, China

Abstract. In this paper, we present preimage attacks on 4-round Keccak-224/256 as well as 4-round Keccak$[r = 640, c = 160, l = 80]$ in the preimage challenges. We revisit the Crossbred algorithm for solving the Boolean multivariate quadratic (MQ) system and elaborate the computational complexity for the case $D = 2$. The result shows that the Crossbred algorithm has advantages when n is small and m outperforms n with feasible memory costs. In our attacks, we construct Boolean MQ systems in order to make full use of variables. With the help of solving MQ systems, we successfully improve preimage attacks on Keccak-224/256 reduced to 4 rounds. Moreover, we implement the preimage attack on 4-round Keccak$[r = 640, c = 160, l = 80]$, an instance in the Keccak preimage challenges, and find 78-bit matched *near preimages*.

Keywords: Keccak · Preimage attack · Multivariate quadratic systems

1 Introduction

Due to the breakthrough attacks on hash functions [17, 19, 20, 18], the National Institute of Standards and Technology (NIST) started new standardization of hash functions. The Keccak sponge function [2] won the competition and became the new generation of Secure Hash Algorithm, known as SHA-3. Since its publication in 2008, both the keyed modes and the unkeyed modes of Keccak have been widely studied.

This paper is focused on the preimage attack. Morawiecki et al. [15] gave the experiment of preimage attack with SAT solver, illustrating that using SAT solver outperforms exhaustive search when Keccak is reduced to 3 rounds. Then in 2013, rotational cryptanalysis [14] was applied to the preimage attack on 4-round Keccak[r = 1024, c = 576] with complexity 2^{506}. Then a breakthrough in the preimage attack occurred in 2016. Guo et al. proposed a new linear structure of Keccak [7] and gave 3/4-round preimage attacks based on the linear structure. After that, some improved attack methods have been proposed. Li et al. [11] constructed a new structure called cross-linear structure, and improved preimage attacks on several 3-round Keccak instances. A two-block method [10] was proposed to attack 3/4-round Keccak-224/256, such that the constraints could be allocated to two blocks and the complexity was lowered. Besides, Rajasree [16] proposed a nonlinear structure, focusing on Keccak-384/512 reduced to 2, 3 and 4 rounds. Later, He et al. [8] developed the linearization method of [10] to save degrees of freedom and improved the preimage attacks on 4-round Keccak-224/256.

Apart from solving linear systems, methods for solving nonlinear systems have been applied such that more degrees of freedom could be saved. Liu et al. [12] made full use of equations derived from the hash value by constructing Boolean quadratic systems. They used relinearization techniques to solve quadratic systems and improved the attacks on Keccak-384/512. After that, Dinur [5] gave an efficient polynomial method-based algorithm [13] for solving multivariate equation systems and applied it in cryptanalysis including preimage attacks on Keccak. The method solved equations of degree 4 and successfully improved preimage attacks on Keccak-384/512 but did not outperform attacks on Keccak-224/256 in [8].

Our Contributions. In this paper, we draw our attention to preimage attacks and present several results of attacks on 4-round Keccak-224/256 as well as 4-round Keccak$[r = 640, c = 160, l = 80]$.

One key technique in our attacks is to solve multivariate quadratic (MQ) polynomial systems. We present a new observation on MQ polynomials and elaborate the complexity of the Crossbred algorithm with $D = 2$. Our elaboration shows that the Crossbred algorithm outperforms the brute force even in the worst case, improving the complexity analysis in [6]. More impressively, although our algorithm is no better than Dinur's [5] in terms of time complexity asymptotically, our algorithm uses feasible memory in a wide range of parameters and is easy to implement. Especially in our attack, the derived MQ systems have small number of variables and larger number of equations, our method needs lower computational costs.

For preimage attacks on Keccak, we exploit the output of the inverse χ^{-1} and carefully select constant values to linearize one round backward and one round forward. Compared with the structure in [10, 8], our structure has more arbitrary constants. Guessing values of arbitrary constants helps to get messages later. Based on our structure, all input bits of χ in the 4th round are quadratic, and then we construct MQ systems in order to fully utilize degrees of freedom and derived equations. Using the Crossbred algorithm, we give a preimage attack on 4-round Keccak$[r = 640, c = 160, l = 80]$ using one message block and preimage attacks on 4-round Keccak-224/256 using two message blocks.

To the best of our knowledge, we propose the first analysis of 4-round Keccak $[r = 640, c = 160, l = 80]$ in the Keccak preimage challenges and give several 78-bit matched preimages. Besides, we improve complexities of preimage attacks on 4-round Keccak-224/256. Table 1 lists the results of this paper compared with the previous ones. The complexity in list is the times of 4-round Keccak permutation.

The rest of this paper is organized as follows. Sec. 2 shows notations and preliminaries of Keccak as well as the properties of the nonlinear layer χ, followed by the complexity elaboration of solving MQ systems. Preimage attacks on 4-round Keccak-224/256 are present in Sec. 3. And the preimage attack on the challenge with implementation details is shown in Sec. 4. Finally, Sec. 5 concludes this paper.

2 Preliminaries and Main Techniques

In this section, we will give the notation and the introduction to Keccak with some properties of the nonlinear layer χ. Then we elaborate the complexity of solving a MQ system.

2.1 Notation

r	Rate of a sponge function
c	Capacity of a sponge function
b	Bit width of a permutation, $b = r + c$
R	The round function of Keccak permutation
$\theta, \rho, \pi, \chi, \iota$	The five mapping steps of R. A subscript i denotes the mapping step in the i-th round, e.g., χ_i denotes χ in the i-th round for $i = 0, 1, 2, \ldots$.
L	Composition of θ, ρ and π and its inverse denoted by L^{-1}
M	Input message
A_i	Input of the i-th round function, $A_{i+1} = \chi(B_i)$, $i = 0, 1, 2, \ldots$
A'_i	Input of ρ in the i-th round, $A'_i = \theta(A_i)$, $i = 0, 1, 2, \ldots$
B_i	Input of χ in the i-th round, $B_i = L(A_i)$, $i = 0, 1, 2, \ldots$

2.2 Keccak-f Permutation

In the Keccak hash function, the Keccak-f permutation with width b is denoted by Keccak-$f[b]$, where $b \in \{25, 50, 100, 200, 400, 800, 1600\}$. The state for Keccak-$f[b]$ can be represented as a $5 \times 5 \times w$-bit state as depicted in Fig. 1. $A[x, y]$ denotes a lane in the state, where x, y are in $\{0, 1, 2, 3, 4\}$. Each bit in $A[x, y]$ is denoted as $A[x, y, z]$ with $0 \le z < w$.

Keccak-$f[b]$ consists of $12+2log_2(b/25)$ rounds of permutation R. Each round R consists of five steps, denoted by θ, ρ, π, χ and ι. $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$.

$$
\begin{aligned}
\theta :& A[x,y,z] = A[x,y,z] \oplus \sum_{y=0}^{4} A[x-1,y,z] \oplus \sum_{y=0}^{4} A[x+1,y,z-1],\\
\rho :& A[x,y,z] = A[x,y,z] \lll T[x,y],\\
\pi :& A[y,2x+3y,z] = A[x,y,z],\\
\chi :& A[x,y,z] = A[x,y,z] \oplus (A[x+1,y,z] \oplus 1) \cdot A[x+2,y,z],\\
\iota :& A[0,0,z] = A[0,0,z] \oplus RC_i[z].
\end{aligned}
$$

Table 1. Comparison of preimage attacks on 4-round Keccak.

Digest length	Instances	Guessing times	Solving Complexity	Total Complexity	Ref
224	Keccak-224	2^{213}	2^{6}	2^{219}	[7]
		2^{207}	2^{8}	2^{215}	[10]
		–	–	[a]2^{202}	[5]
		2^{192}	2^{8}	2^{200}	[8]
		2^{164}	2^{18}	[b]2^{182}	Sec. 3.1
256	Keccak-256	2^{251}	2^{3}	2^{254}	[7]
		2^{239}	2^{8}	2^{247}	[10]
		–	–	[c]2^{231}	[5]
		2^{218}	2^{8}	2^{226}	[8]
		2^{196}	2^{18}	[d]2^{214}	Sec. 3.2
80	Keccak[$r = 1440, c = 160, l = 80$]	2^{54}	–	solved	[3]
	Keccak[$r = 640, c = 160, l = 80$]	2^{39}	2^{19}	2^{58}	Sec. 4.1

[a]The complexity is equal to 2^{217} bit operations. [b]The complexity is equal to 2^{197} bit operations.
[c]The complexity is equal to 2^{246} bit operations. [d]The complexity is equal to 2^{229} bit operations.

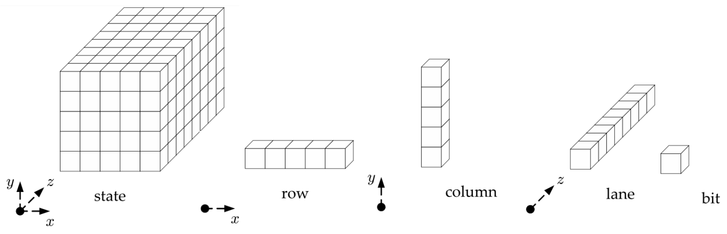

Fig. 1. State of Keccak

Here "$\oplus$" and "$\cdot$" are additions and multiplications over $\mathbb{F}_2$. $T[x, y]$ are offsets and RC_i are constants for round i. Since ι has no influence on our attacks, we ignore it in the rest of the paper.

2.3 The Keccak Hash Function

The Keccak hash function is the family of sponge functions [1] with Keccak-$f[b]$ permutation. The function is parameterized by the rate r, capacity c, and output length l which satisfies $r + c = b$, and denoted as Keccak[r,c,l]. The standardized Keccak functions restricted to Keccak[1152,448,224], Keccak[1088,512,256], Keccak[832,768,384], and Keccak[576,1024,512] are called Keccak-224, Keccak-256, Keccak-384 and Keccak-512 respectively.

As illustrated in Fig. 2, the sponge function has two phases, absorbing phase and squeezing phase. The b-bit initial state (IV) is set to be all 0's. At the beginning, padded message is divided into blocks with length of r. The first r-bits of IV is *XOR*ed with the first block and then is sent to f. Again, the first r-bits of output is *XOR*ed with the second block and computed in f. This procedure is repeated until all the blocks are absorbed. After that, if l is smaller than r, the first l-bit output of the absorbing phase is the output string. Otherwise, if l is greater than r, another function f is applied to produce r more bits. This procedure is repeated until we obtain enough output strings. Then the output strings are truncated to a l-bit digest.

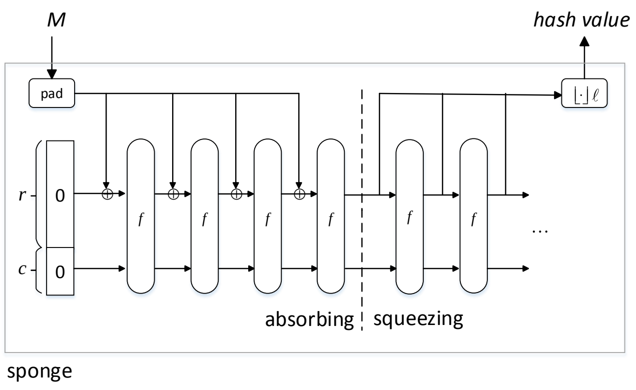

Fig. 2. Sponge Construction

The padding rule for Keccak, named multi-rate padding, extends a message M to be a message of the form $M10^{*}1$. That is, M is first padded with a single bit "1" and then with a smallest non-negative number of "0" and finally with a single bit "1" in order to produce a padded message whose bit length becomes multiple of r. The SHA-3 family adopts standardized Keccak functions except that it applies a different padding rule of the form $M0110^*1$. We refer the readers to [2] for more details.

2.4 Properties of Step χ

Before introducing attacks on Keccak functions, we first show some properties of the nonlinear step χ and its inverse χ^{-1}.

For the 5-bit input $a = a_0a_1a_2a_3a_4$ of χ, the output $b = b_0b_1b_2b_3b_4$ can be expressed as $b_i = a_i \oplus (a_{i+1} \oplus 1)a_{i+2}$.

Property 1. [7] Given two consecutive bits b_i, b_{i+1} of the output of χ, a linear equation can be set up on the input bits as $b_i = a_i \oplus (b_{i+1} \oplus 1) \cdot a_{i+2}$. Specifically, the equation turns to be $a_i = b_i$ in the case of $b_{i+1} = 1$.

The inverse operation χ^{-1} has algebraic degree 3, and its algebraic normal form can be written as

$$a_i = b_i \oplus (b_{i+1} \oplus 1) \cdot (b_{i+2} \oplus (b_{i+3} \oplus 1) \cdot b_{i+4}). \tag{1}$$

To reduce algebraic degrees of the output, the input has at most two variables and these variables should not be consecutive. Let x and c stand for the variable and constant, respectively. Each constant c could be 1 or 0. Since variables are not consecutive, the inputs are in the form of 'xcxcc'. Table 2 lists the inputs and their corresponding outputs of χ^{-1}.

Table 2. Inputs and their corresponding outputs of χ^{-1} for the 'xcxcc' input pattern

Inputs	x0x00	x0x01	x0x10	x0x11	x1x00	x1x01	x1x10	x1x11
Outputs	xx^{2}xx0	x0x01	xx^{2}xxx^2	xxx1x^2	xx^{2}xxx	x1x0x	xx^{2}xxx^2	xxx1x^2
#Linear	3	2	3	3	4	3	3	3
#Quadratic	1	0	2	1	1	0	2	1

According to Table 2, we find that the outputs of χ^{-1} for 'xcxcc' are linear only when the inputs are 'x0x01' or 'x1x01' as described in Property 2.

Property 2. [7] When $b_{i+3} = 0$, $b_{i+4} = 1$ and b_{i+1} is known, then the input a_j's can be written as linear combinations of b_j's, for all $i \in \{0, 1, 2, 3, 4\}$.

2.5 On the Concrete Complexity of Crossbred Algorithm with $D = 2$

In this section, we introduce an algorithm, the Crossbred algorithm [9] with $D = 2$ case, and elaborate the concrete complexity for solving MQ systems. The subsequent attacks on 4-round Keccak are based on this algorithm.

An MQ polynomial of n Boolean variables $x_1, x_2, \dots, x_n$ over binary field $\mathbb{F}_2$ is defined as

$$z(x_1, \dots x_n) = \sum_{1 \le i < j \le n} a_{i,j} x_i x_j + \sum_{1 \le i \le n} b_i x_i + c, \tag{2}$$

where $a_{i,j} \in \mathbb{F}_2$, $b_i \in \mathbb{F}_2$ and $c \in \mathbb{F}_2$. Then an MQ system of m equations and n variables, called an (m, n) MQ system, is given by

$$\begin{cases} z_1(x_1, \dots, x_n) = 0, \\ z_2(x_1, \dots, x_n) = 0, \\ \qquad\qquad \vdots \\ z_m(x_1, \dots, x_n) = 0. \end{cases}$$

where $z_i(x_1, \dots, x_n)$ are MQ polynomials for $i = 1, 2, \dots, m$.

The MQ polynomial z in (2) can be written in the residual form

$$z(x_1, \dots x_n) = x_1 f_1 + \cdots + x_{n-1} f_{n-1} + L + c \tag{3}$$

where f_i is a linear functions from variables $x_{i+1}, \dots, x_n$ and L is a linear combination of $x_1, \dots, x_n$. According to (3), an (m, n) MQ system can be transformed to the following form

$$\begin{cases} z_1 = x_1 f_1^1 + x_2 f_2^1 + \cdots + x_{n-1} f_{n-1}^1 + L_1 + c_1 & = 0, \\ z_2 = x_1 f_1^2 + x_2 f_2^2 + \cdots + x_{n-1} f_{n-1}^2 + L_2 + c_2 & = 0, \\ \quad \vdots \\ z_m = x_1 f_1^m + x_2 f_2^m + \cdots + x_{n-1} f_{n-1}^m + L_m + c_m & = 0 \end{cases} \tag{4}$$

Our next step is to derive polynomials such that more coefficients of x_i are constants based on the residual form (4). Taking f_{n-1}^j as instance, all possibilities of f_{n-1}^j are 0, x_n, which means that there exists a linear combination $(\alpha_1, \alpha_2, \dots, \alpha_m)$ such that $\sum_{j=1}^m \alpha_j f_{n-1}^j = 0$. We use a vector v_{n-1}^j of dimension 1 to illustrate f_{n-1}^j. If $f_{n-1}^j = x_n$, set $v_{n-1}^j = (1)$; otherwise, set $v_{n-1}^j = (0)$. The problem of finding α_j can be reduced to solving the linear system as follows

$$\left(v_{n-1}^1, v_{n-1}^2, \dots, v_{n-1}^m\right) \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{pmatrix} = 0.$$

There are at least $m - 1$ linearly independent solutions and each solution corresponds to a combination of $z_1, z_2, \dots, z_m$ which derives a polynomial

$$\sum_{j=1}^m \alpha_j z_j = x_1 \sum_{j=1}^m \alpha_j f_1^j + \cdots + x_{n-2} \sum_{j=1}^m \alpha_j f_{n-2}^j + \sum_{j=1}^m \alpha_j L_j + \sum_{j=1}^m \alpha_j c_j.$$

The same execution can be performed on other coefficients. In general, we aim to find linear combinations $\alpha = (\alpha_1, \ldots, \alpha_m)$ such that $\sum_{j=1}^{m} \alpha_j f_i^j = 0$ for each $i = n-t, n-t+1, \ldots, n-1$ with a given $1 < t < n$. Then we obtain the following *remainder equation*

$$\sum_{j=1}^{m} \alpha_j z_j = \sum_{i=1}^{n-t-1} x_i \sum_{j=1}^{m} \alpha_j f_i^j + \sum_{j=1}^{m} \alpha_j L_j + \sum_{j=1}^{m} \alpha_j c_j = 0. \tag{5}$$

Now we discuss the number of solutions of α, which determines the number of remainder equations we have. Let $f_i^j = v_i^j (x_{i+1}, \ldots, x_n)^\top$, where v_i^j is an $(n-i)$ dimensional binary row vector. Then α satisfies the condition $\alpha M = 0$, where

$$M = \begin{pmatrix} v_{n-t}^1 & v_{n-t+1}^1 & \cdots & v_{n-1}^1 \\ v_{n-t}^2 & v_{n-t+1}^2 & \cdots & v_{n-1}^2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{n-t}^m & v_{n-t+1}^1 & \cdots & v_{n-1}^m \end{pmatrix}$$

and is of dimension $m \times \frac{t(t+1)}{2}$. There are at least $m - \frac{t(t+1)}{2}$ independent solutions of α, and hence $m - \frac{t(t+1)}{2}$ remainder equations can be derived.

We use guess-and-determine techniques to solve remainder equations. It is obvious that for any fixed values of $x_1, \ldots, x_{n-t-1}$, each remainder equation (5) is reduced to a linear equation. Then a linear system of $m - \frac{t(t+1)}{2}$ equations over $t+1$ variables can be derived for each guess of variables and can be solved by Gaussian elimination. If the system is solvable, a solution can be verified by substituting it into the MQ system of equations. If the solution is verified correctly, we find the solution for the MQ equations, otherwise, the corresponding guess is wrong. In order to guarantee that there is no more than one solution on average for each guess, choose t such that $m - \frac{t(t+1)}{2} \geq t+1$.

Complexity Analysis: The computational complexity involves three parts, of which the first is for computing the remainder equations, the second is for solving the remainder equations and the third is for verifying the survived solutions.

The remainder equations can be obtained by Gaussian elimination on an $m \times \frac{t(t+1)}{2}$ binary matrix with the complexity of $m^2 \cdot \frac{t(t+1)}{2}$ bit operations. The memory cost is $m \times \frac{t(t+1)}{2} < m^2$ bits.

For solving the remainder equations, guess $n-t-1$ bits and solve a derived linear system of $m - \frac{t(t+1)}{2}$ equations over $t+1$ variables. With the help of Gray code, each equation update needs only n bit operations and totally $(m - \frac{t(t+1)}{2})n$ bit operations are needed to update a linear system. Then the linear system can be solved by Gaussian elimination with $(m - \frac{t(t+1)}{2})^2(t+1)$ bit operations. On average, there are $2^{t+1-(m-\frac{t(t+1)}{2})} = 2^{\frac{(t+1)(t+2)}{2}-m}$ solutions for each linear system. Here, we use two binary matrices of size $(m - \frac{t(t+1)}{2}) \times (t+1)$ in the memory, one for storing the iterated system and the other for solving the linear system. This memory cost can be shared by all guesses.

In order to verify the solutions, a solution is substituted into the MQ equations. Assume that each solution is verified correct for each equation with the probability 1/2, then verifying a solution needs to compute $\sum_{i=1}^{m} i\,(1/2)^i \approx 2$ equations. Computing each equation needs at most $\binom{n}{2}$ AND operations and $\binom{n}{2}+n$ XOR operations. And hence, verifying a solution needs about $2n(n+1) \approx 2n^2$ bit operations. In order to store the (m,n) MQ equations, the memory cost is at most $m\left(\binom{n}{2}+n+1\right) = m\left(\frac{n(n+1)}{2}+1\right)$ bits.

Let T and M denote the computational cost in terms of bit operations and memory cost in terms of bits for solving an (m,n) MQ system, then we have

$$\begin{aligned}T =& m^2 \cdot \frac{t(t+1)}{2} + 2^{n-t-1}\left((m-\frac{t(t+1)}{2})\cdot n + (m-\frac{t(t+1)}{2})^2(t+1)\right) + \\ & 2^{n-t-1}\cdot 2^{\frac{(t+1)(t+2)}{2}-m}\cdot 2n^2 \\ \approx & 2^{n-t-1}(t+1)\left(m-\frac{t(t+1)}{2}\right)^2 + 2^{n-m+\frac{t(t+1)}{2}+1}\cdot n^2\end{aligned}$$

and

$$\begin{aligned}M &= m\cdot\frac{t(t+1)}{2} + 2\left(m-\frac{t(t+1)}{2}\right)(t+1) + m\left(\frac{n(n+1)}{2}+1\right) \\ &< \left(\frac{(t+1)(t+5)}{2}+\frac{n(n+1)}{2}\right)m.\end{aligned}$$

For the worst case $m=n$,

$$T \approx 2^{n-t-1}\left(n-\frac{t(t+1)}{2}\right)^2(t+1) + 2^{\frac{t(t+1)}{2}+1}\cdot n^2.$$

Choose t such that $n-\frac{t(t+1)}{2} \geq t+1$ and $n-\frac{(t+1)(t+2)}{2} < t+2$, i.e., $\sqrt{2n}-3 < t < \sqrt{2n}-1$. Then we have $n-\frac{t(t+1)}{2} = n-\frac{(t+1)(t+2)}{2}+t < 2t+3$ and $\frac{t(t+1)}{2} \leq n-t-1$. Consequently, $T < 2^{n-t-1}\cdot(2t+3)^2(t+1) + 2^{n-t-1}\cdot 2n^2 < 2^{n+2-\sqrt{2n}}\cdot\left(2n^2+8n\sqrt{2n}+8n+\sqrt{2n}\right)$.

It is noted that our algorithm is suitable to any parameter (m,n). Generally, the number of equations is larger than that of equations, i.e., $m \geq n$. $m < n$ corresponds to the case with more than 1 solutions on average. $n-m$ variables can be set to constant and the case is reduced to the $m=n$ case.

Comparison In this section, we compare our method with the fast exhaustive search [4] and polynomial method [5].

In [4], the brute force can be sped up by enumeration in the standard Gray code. The fast exhaustive search (FES) needs $2d\cdot\log n\cdot 2^n$ bit operations. When $m=n$, in order to compare the complexity with that of the FES, which is $2^{n+2}\log n$ bit operations, we just need to compare $C(n) = 2^{-\sqrt{2n}}\cdot\left(2n^2+8n\sqrt{2n}+8n+\sqrt{2n}\right)$ with $\log n$. When $n \geq 64$, $C(n) < \log n$, Crossbred algorithm has lower complexity than FES in terms of bit operations.

The polynomial method [5] includes two procedures, of which the first is enumerating the isolated solutions to the polynomial system and the second is testing the solutions. In the first procedure, to achieve lower complexity, the algorithm enumerates solutions to probabilistic polynomials that are of lower degree, and hence the algorithm is probabilistic, though the probability may be high and close to 1. In this algorithm, exponential potential solutions are obtained, stored and handled in the memory with extensive use of Möbius transformations, giving rise to heavy memory cost.

By contrast, our method is a deterministic algorithm. Our algorithm avoids storing many candidate solutions and high degree polynomials. In our algorithm, the memory is used to store linear systems and the original quadratic systems, where the updated linear systems can share the same memory, such that the memory cost in our method is feasible. When used in security analysis of cryptosystems, the memory cost is within the ability of a single PC.

Here, we list the comparison of our algorithm with fast exhaustive search [4] and memory-optimized variant of polynomial method [5] in Table 3 for some practical parameters.

Table 3. Comparison of concrete complexities in terms of bit operations and memory costs in terms of bits. The memory cost of exhaustive search is small and omitted here.

variables n	Complexity (bit operations)	Memory (bits)	Algorithm
80	2^{84}	–	Exhaustive Search [4]
	2^{77}	2^{60}	Polynomial Method [5]
	2^{80}	2^{18}	Ours($m = n, t = 11$)
	2^{76}	2^{19}	Ours($m = 2n, t = 16$)
128	2^{133}	–	Exhaustive Search [4]
	2^{117}	2^{91}	Polynomial Method [5]
	2^{126}	2^{20}	Ours($m = n, t = 14$)
	2^{120}	2^{21}	Ours($m = 2n, t = 21$)
192	2^{197}	–	Exhaustive Search [4]
	2^{170}	2^{132}	Polynomial Method [5]
	2^{188}	2^{22}	Ours($m = n, t = 18$)
	2^{180}	2^{23}	Ours($m = 2n, t = 26$)
256	2^{261}	–	Exhaustive Search [4]
	2^{223}	2^{173}	Polynomial Method [5]
	2^{249}	2^{23}	Ours($m = n, t = 21$)
	2^{241}	2^{24}	Ours($m = 2n, t = 30$)

The results in Table 3 show that the Crossbred algorithm requires lower computational costs than polynomial method when n is small and m outperforms n. When n becomes larger and m is close to n, our method needs more bit operations. Hence our method is more suitable for the subsequent attacks on 4-round

Keccak which yield MQ problems with small n and large m. By this algorithm, solving an MQ problem can be reduced to calls to solving linear systems, which are easy to parallelize using single instruction multiple data (SIMD) speedup. More details about implementation are discussed in Sec. 4.2. Take an example to show the efficiency of this method. For example, an $(m = 4n, n = 80)$ MQ instance can be solved by solving 2^{45} linear systems of 45 equations over 35 variables with memory cost 2^{20} bits, which are feasible on modern microprocessors.

3 Preimage Attacks on 4-Round Keccak

In this section, we introduce preimage attacks on 4-round Keccak via solving systems of quadratic Boolean equations. Based on a 2-round linear structure, we derive an algebraic system such that the output after 3 rounds of degree at most 2, and then we solve this algebraic system using the algorithm in Sec. 2.5.

3.1 Preimage Attack on 4-round Keccak-224

In the following, we extend the linear structure in [7] and improve the preimage attack on 4-round Keccak-224, where two message blocks are used.

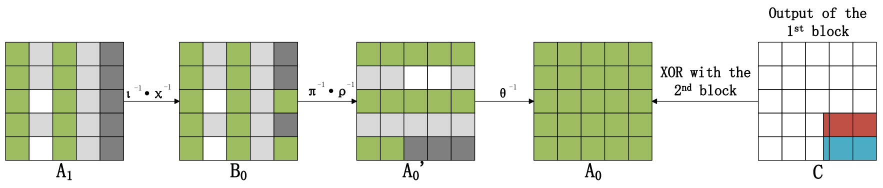

Fig. 3. The one backward round of Keccak-224

Our structure, applied on the second block, consists of one backward round $A_0 = R^{-1}(A_1)$ and two forward rounds $A_2 = R(A_1), A_3 = R(A_2)$. Here, A_0 is the XOR of the output of the first message block with the second message block. Fig. 3 shows one backward round for 4-round Keccak-224. The bits of lanes in green boxes are of degree 1. The lanes in light gray(resp. dark gray) boxes are set to constants 0's(resp. 1's). And those in white boxes represent arbitrary constants. We set 10 lanes of state $A_1[0, y]$ and $A_1[2, y], y \in \{0, \ldots, 4\}$ as variables, i.e., there are totally $10 \times 64 = 640$ variables, while the other lanes are set to constants. According to Property 2, we have

$$A_1[3, y] = 0, y \in \{0, \ldots, 4\},$$
$$A_1[4, y] = \mathtt{0xFFFF\,FFFF\,FFFF\,FFFF}, y \in \{0, \ldots, 4\},$$

then the output in B_0 are linear. Instead of directly setting all 5 lanes of $A_1[1,y]$ to zero like [10, 8], we only set

$$A_1[1,y]=0, y\in\{0,1,3\},$$

and $A_1[1,2]$ and $A_1[1,4]$ are arbitrary constants which helps to find different messages by setting all possible values. Furthermore, after ρ^{-1} and π^{-1} we have $A'_0[x,3]\oplus A'_0[x,4]=\texttt{0xFFFF FFFF FFFF FFFF}, x\in\{2,3,4\}$. When using only one message block, the last 449 bits of A_0 are set to 0 or 1 as the capacity or padding bits. Here we use two message blocks to reduce the number of constraints on A_0. Denote the output state of the first block as C, the constraints resulting from capacity and padding rules become

$$\begin{aligned}&A_0[x,3]=C[x,3], x=3,4,\\&A_0[x,4]=C[x,4], x\in\{0,1,\cdots,4\},\\&A_0[2,3,63]=C[2,3,63]\oplus 1.\end{aligned}\tag{6}$$

Due to step θ, $A_0[x,3,z]\oplus A_0[x,4,z]=A'_0[x,3,z]\oplus A'_0[x,4,z]=1, x\in\{2,3,4\}, z\in\{0,1,\cdots,63\}$. Hence, C should satisfy

$$\begin{aligned}&C[3,3]\oplus C[3,4]=\texttt{0xFFFF FFFF FFFF FFFF},\\&C[4,3]\oplus C[4,4]=\texttt{0xFFFF FFFF FFFF FFFF},\\&C[2,3,63]=C[2,4,63].\end{aligned}\tag{7}$$

Since output bits of a hash function can be considered to be uniformly distributed, the complexity of finding a preimage satisfying (7) via brute force is $2^{64+64+1}=2^{129}$. Once getting the output C, A_0 can meet the requirements of (6) with only 320 constraints as follows

$$A_0[x,4]=C[x,4], x\in\{0,1,\cdots,4\}.\tag{8}$$

Then we linearize bits in A_2 as illustrated in Fig. 4. To avoid the propagation by θ, the following $2\times 64=128$ constraints are added

$$\sum_{y=0}^{4}A_1[0,y]=\alpha, \sum_{y=0}^{4}A_1[2,y]=\beta,\tag{9}$$

where α,β are 64-bit constants. In this way, the outputs in A_2 are linear on the variables. After a round of R, the outputs in A_3 are of degree 2, which are indicated in orange in Fig. 4. Since L is a linear operation, we can set up series of quadratic polynomials of B_3.

Due to Property 1, we obtain $2\times 64+32=160$ quadratic equations from 224-bit hash value, i.e.,

$$\begin{aligned}&B_3[0,0,z]\oplus(A_4[1,0,z]\oplus 1)\cdot B_3[2,0,z]=A_4[0,0,z], z=0,1,\cdots,63,\\&B_3[1,0,z]\oplus(A_4[2,0,z]\oplus 1)\cdot B_3[3,0,z]=A_4[1,0,z], z=0,1,\cdots,63,\\&B_3[2,0,z]\oplus(A_4[3,0,z]\oplus 1)\cdot B_3[4,0,z]=A_4[2,0,z], z=0,1,\cdots,31.\end{aligned}\tag{10}$$

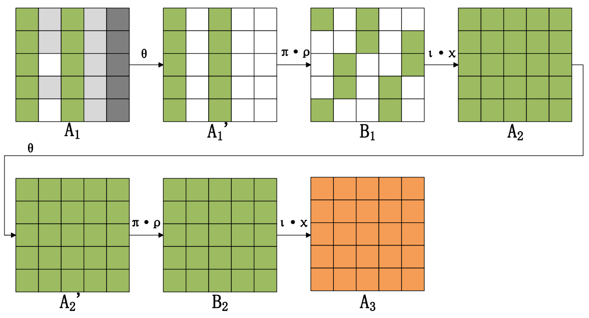

Fig. 4. The 2 forward rounds of Keccak-224

Assuming that 0s and 1s appear equally in A_4, about half of equations can be written as $B_3[x,y,z] = A_4[x,y,z]$. By Guo et al.'s study [7], quadratic bits in B_3 can be linearized by guessing values of linear polynomials. Appendix A shows how to linearize a bit in B_3. Here we give 2-bit linearization by guessing 11 values of linear polynomials. For $B_3[0,0,z]$ and $B_3[1,0,z+44]$, since both ρ and π are permutation steps, we can get the corresponding bits $A'_3[0,0,z]$ and $A'_3[1,1,z]$. According to

$$A'_3[x,y,z] = A_3[x,y,z] \oplus \sum_{y=0}^{4} A_3[x-1,y,z] \oplus \sum_{y=0}^{4} A_3[x+1,y,z-1],$$

as shown in Fig. 5, linearizing two bits requires 21 linearized bits in A_3, i.e.,

$$A_3[4,y,z], A_3[0,y,z], A_3[1,1,z], A_3[1,y,z-1], A_3[2,y,z-1], y \in \{0,\cdots,4\}. \tag{11}$$

According to the equation

$$A_3[x,y,z] = B_2[x,y,z] \oplus (B_2[x+1,y,z] \oplus 1) \cdot B_2[x+2,y,z],$$

it is obvious that the only quadratic term in $A_3[x,y,z]$ is generated by $B_2[x+1,y,z]$ and $B_2[x+2,y,z]$. Hence we can linearize $A_3[x,y,z]$ by guessing values of $B_2[x+1,y,z]$ or $B_2[x+2,y,z]$. Note that $A_3[x,y,z]$ and $A_3[x-1,y,z]$ share a common operand $B_2[x+1,y,z]$ in their quadratic terms. By fixing the value of $B_2[x+1,y,z]$, $A_3[x,y,z]$ and $A_3[x-1,y,z]$ are linearized as well. Thus by guessing 11 bits

$$B_2[3,1,z], B_2[3,y,z-1], B_2[1,y,z], y \in \{0,\cdots,4\}, \tag{12}$$

$B_3[0,0,z]$ and $B_3[1,0,z+44]$ are linearized. Similarly, two equations $B_3[1,0,z+1] = A_4[1,0,z+1]$ and $B_3[2,0,z] = A_4[2,0,z]$ can also be linearized at the same time by guessing 11 values of linear polynomials.

According to (8), (9), (10), we have 160 quadratic equations over $640 - 320 - 128 = 192$ variables. Since the hash value can be regarded as random values, about 24 pairs from 160 quadratic equations can be used during 2-bit linearization. When $m = 12$, $n = 0$[5], an MQ problem with 136 equations over 36 variables is constructed.

Complexity Analysis: The MQ system has a solution with the probability 2^{-100}. Let t be 15, according to Sec. 2.5, the computing complexity is $2^{32} + 2^{31.3} \approx 2^{33}$ bit operations, which is equivalent to 2^{18} calls to the 4-round Keccak permutation. The memory complexity for solving the MQ system is 2^{17} bits. Compared with solving the MQ system, the computational cost of performing Gaussian Elimination on linear constraints can be omitted while the memory cost is 2^{19} bits for storing the linear system. In this case, the time complexity of this attack is $2^{129} + 2^{64+100+18} = 2^{182}$ and the memory complexity is 2^{19} bits, which are shared by different MQ systems. By guessing $A[1,2], A[1,4], \alpha$, β and guessing bits in B_2, we can get $2^{64+100} = 2^{164}$ messages that satisfy the conditions and thus our attack is feasible. For SHA3-224, the time complexity of the preimage attack is 2^{182} while the padding rule changes.

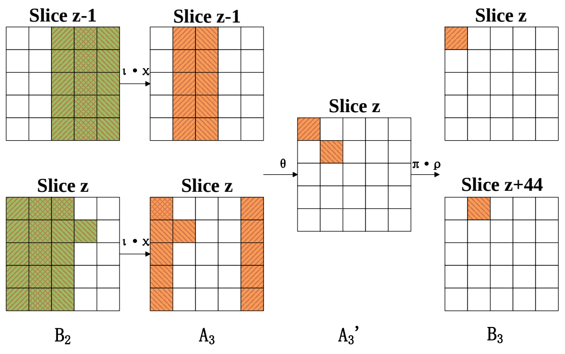

Fig. 5. Linearization of two quadratic bits in B_3. In the figure, bits related to the first bit are indicated by up diagonal slash and those related to the second bit are indicated by down diagonal slash. Besides, bits in green boxes are linear and bits in orange boxes are quadratic.

[5] When $m > 12$, computing the remainder equations and verifying solutions cost more time than solving the remainder equations during the MQ system solving process, which increases the whole computing complexity.

3.2 Preimage Attack on 4-round Keccak-256

The attack on 4-round Keccak-256 works similarly, where two message blocks are applied. Fig. 6 shows one backward round for 4-round Keccak-256. We set 10 lanes of state $A_1[0,y]$ and $A_1[2,y], y \in \{0,\ldots,4\}$ as variables. For constant bits in A_1, we have

$$\begin{aligned}
&A_1[1,y] = 0, y \in \{0,1,3,4\},\\
&A_1[3,y] = 0, y \in \{0,\ldots,4\},\\
&A_1[4,y] = \texttt{0xFFFF FFFF FFFF FFFF}, y \in \{0,\ldots,4\}.
\end{aligned}$$

The outputs of χ^{-1} are linear according to Property 2. Further, we have $A'_0[x,3] \oplus A'_0[x,4] = \texttt{0xFFFF FFFF FFFF FFFF}, x \in \{1,2,3,4\}$. According to the capacity and the padding rule, the output C of the first block and A_0 should satisfy

$$\begin{aligned}
&A_0[x,3] = C[x,3], x = 2,3,4,\\
&A_0[x,4] = C[x,4], x \in \{0,1,\cdots,4\},\\
&A_0[1,3,63] = C[1,3,63] \oplus 1.
\end{aligned}$$

Due to step θ, $A_0[x,3,z] = A_0[x,4,z] \oplus 1, x \in \{1,2,3,4\}, z \in \{0,1,\cdots,63\}$. Hence C should satisfy

$$\begin{aligned}
&C[2,3] \oplus C[2,4] = \texttt{0xFFFF FFFF FFFF FFFF},\\
&C[3,3] \oplus C[3,4] = \texttt{0xFFFF FFFF FFFF FFFF},\\
&C[4,3] \oplus C[4,4] = \texttt{0xFFFF FFFF FFFF FFFF},\\
&C[1,3,63] = C[1,4,63].
\end{aligned} \tag{13}$$

The complexity of finding a preimage whose 4-round output satisfy (13) by brute force is $2^{3\times 64+1} = 2^{193}$. Once obtaining the first message block, we set $5 \times 64 = 320$ constraints on A_0 as follows

$$A_0[x,4] = C[x,4], x \in \{0,1,\cdots,4\}. \tag{14}$$

Thus A_0 meets the requirement of the capacity and the padding rule.

The two rounds forward for Keccak-256 is similar with Keccak-224 except that $A[1,4,z] = 0, z = 0,1,\ldots,63$. To avoid the propagation by θ, we add $2 \times 64 = 128$ constraints

$$\sum_{y=0}^{4} A_1[0,y] = \alpha, \sum_{y=0}^{4} A_1[2,y] = \beta, \tag{15}$$

where α and β are 64-bit constants. Totally, there are $(5+2) \times 64 = 448$ constraints on 640 variables. Similar to Keccak-224, we have $3 \times 64 = 192$ quadratic equations from 256-bit hash value. Assuming that 0s and 1s appear equally in the states, half of equations are in the form of $B_3[x,y,z] = A_4[x,y,z]$ on average. Similar to the preimage attack on 4-round Keccak-224, quadratic bits can be linearized by guessing values of linear polynomials.

We have 192 quadratic equations and $640 - 448 = 192$ variables. Among 192 quadratic equations 32 pairs meet the requirement for the 2-bit linearization on average. When using $m = 12$ pairs of equations[6], an MQ problem with 168 equations over 36 variables is constructed and has a solution with the probability 2^{-132}. Let $t = 16$, the computing complexity is $2^{33.1} + 2^{15.3} \approx 2^{33}$ which is equivalent to 2^{18} calls to the 4-round Keccak permutation. The memory complexity for solving the MQ system and the constraint system are 2^{17} and 2^{19} bits. Totally, the time complexity of this attack is $2^{193} + 2^{64+132+18} = 2^{214}$ and the memory cost is 2^{19} bits. We can get 2^{196} two-block messages which satisfy the conditions by guessing the value of constants $A[1, 2], \alpha, \beta$ as well as bits in B_2 and thus our attack is feasible. For SHA3-256 and SHAKE256, in despite of different padding rules, the time complexities are also 2^{214}.

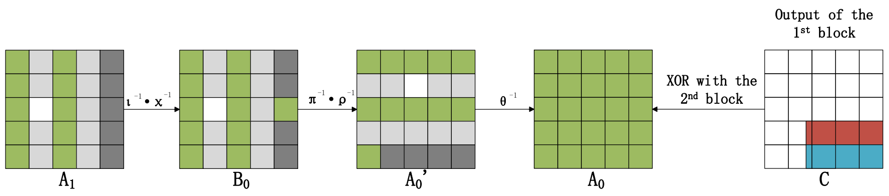

Fig. 6. The one backward round of Keccak-256

4 Application to Keccak Challenge

In this section, we implement the preimage attack on Keccak[$r = 640, c = 160, l = 80$] in the Keccak challenges.

4.1 Preimage Attack on 4 round Keccak[$r = 640, c = 160, l = 80$]

4-round Keccak[$r = 640, c = 160, l = 80$] is an instance of Keccak with the width 800 in the Keccak Crunchy Crypto Collision and Preimage Contest [3]. In this section, we apply our structure to the preimage attack on 4 round Keccak[$r = 640, c = 160, l = 80$] with only one message block.

The structure consists of one backward round $A_0 = R^{-1}(A_1)$ and two forward rounds $A_2 = R(A_1), A_3 = R(A_2)$, as illustrated in Fig. 7. We set 10 lanes of state $A_1[0, y]$ and $A_1[2, y], y \in \{0, \dots, 4\}$ as variables, i.e., there are totally $10 \times 32 = 320$ variables. The other lanes are set as 448 constants, and we have

$$A_1[3, y] = 0, y \in \{0, \dots, 4\},$$
$$A_1[4, y] = \texttt{0xFFFF FFFF}, y \in \{0, \dots, 4\}.$$

[6] When $m > 12$, computing the remainder equations and verifying solutions cost more time than solving the remainder equations during the MQ system solving process, which increases the whole computation costs.

Thus A_0 are all linear due to Property 2. According to capacity and padding rules, the last 162 bits satisfy the following constraints

$$\begin{aligned} &A_0[3,4,30]=1, A_0[3,4,31]=1,\\ &A_0[x,4,z]=0, x\in\{0,\ldots,4\}, z\in\{0,\ldots,31\}. \end{aligned} \tag{16}$$

To avoid the propagation by θ, we add extra 64 constraints

$$\sum_{y=0}^{4} A_1[0,y]=\alpha, \sum_{y=0}^{4} A_1[2,y]=\beta, \tag{17}$$

where α,β are 32-bit constants. Totally, we have $320-162-64=94$ variables. Fig. 7 presents how variables propagate in 2 forward rounds.

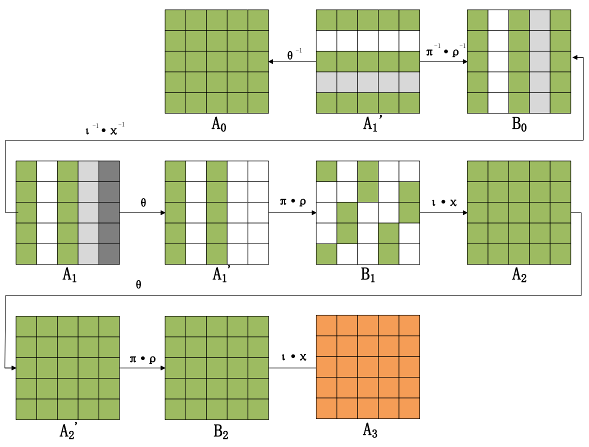

Fig. 7. The linear structure of Keccak$[r=640, c=160, n=4]$

Similar to the preimage attack on 4-round Keccak-224, two equations satisfying $B_3[0,0,z]=A_4[0,0,z]$ and $B_3[1,0,z+12]=A_4[1,0,z+12]$ can be linearized by adding 11 equations. After linearizing some equations we use the method of solving MQ systems in Sec. 2.5 to deal with the rest ones.

In the Keccak preimage challenge, the given hash value of 4-round Keccak$[r=640, c=160, l=80]$ is

```
75 1a 16 e5 e4 95 e1 e2 ff 22
```

and its bit representation is shown below.

$$\begin{aligned} H[0] &= \texttt{10\underline{1}01\underline{110}\ \underline{0}1\underline{0}1\underline{1}00\underline{0}\ \underline{0}1101\underline{000}\ 1\underline{0}100\underline{111}}\\ H[1] &= \texttt{\underline{00100111}\ 1\underline{0}101\underline{0}01\ 10000111\ 01000111}\\ H[2] &= \texttt{11111111\ 01000100} \end{aligned}$$

Due to Property 1, we obtained 48 quadratic equations from $H[i]$ and 26 equations are of the form of $B_3[x, y, z] = A_4[x, y, z]$ as underlined. We found that there are totally 5 pairs of bits satisfying the 2-bit linearization which is indicated in red. When linearizing these bits, note that there are 2 repetitive conditions so the number of extra equations is 53 rather than 55. After that, there remain 38 quadratic equations over 31 variables. The MQ system has a solution with the probability 2^{-7}. Let $t = 7$, according to Sec. 2.5, solving this MQ system needs to solve 2^{23} linear systems of 10 equations over 8 variables. The computing complexity is $2^{32.6} + 2^{31.9} \approx 2^{33}$ bit operations if the system is solvable, which is equivalent to 2^{19} calls to the 4-round Keccak permutation. And the memory complexity is 2^{14} bits. The memory cost of performing Gaussian Elimination on linear constraints is 2^{17} bits while the time cost can be omitted. Our attack obtains a preimage with the computing complexity $2^{32+7+19} = 2^{58}$ and the memory complexity 2^{17}. We give 78-bit matched preimages of 4-round Keccak$[r = 640, c = 160, l = 80]$ in Sec. 4.3.

4.2 Implementation

Our target platform for implementing attack towards 4-round Keccak$[r = 640, c = 160, l = 80]$ preimage challenge is a hybrid cluster equipped with 50 GPUs and 20 CPUs. The model of equipped GPUs is NVIDIA Tesla V100 (Volta microarchitecture) with 32 GB configuration and the model of equipped CPUs is Intel Xeon E5-2699@2.2GHz.

Our program consists of four steps:

1. Extract linear representation from linear constraints.
2. Iterate linear constraints and update MQ systems.
3. Solve MQ systems.
4. Substitute MQ solutions into original input and verify the hash result.

Since the program of solving MQ problems is easy to parallelize and suitable to GPU, we program the MQ solving routine on GPU and deploy the remaining subroutines on CPU.

As described in Section 4.1, the original Keccak system consists of 800 variables. In the preprocessing, the $480+162+64 = 706$ constraints could be imposed in advance so that the entire Keccak system can be represented by the remaining 94 variables and the computing complexity in further steps can be therefore reduced. In the main iteration, the program sets all the possible values of 53 extra equations and extracts the linear representation. These constraints, as well as 10 quadratic constraints which have been linearized by extra equations, are substituted into the 94-variables MQ system, resulting in a 31-variable MQ system of 38 equations. Subsequently, these MQ problems are copied to GPUs for the next solving process.

For each time a new constraint system is imposed, we apply Gaussian Elimination to extract the linear representations of variables. The yielded row echelon form matrix is stored in memory, thereby reproducing the complete message in the verification step.

To solve MQ systems, we employ the method in Sec. 2.5. A solution candidate can be obtained if the MQ problem is solvable. Using the solution and all the matrices stored in the previous steps, a message candidate can be reproduced. We verify a message candidate by comparing its 4-round hash with the target one. In practice, the execution time of the verification process is negligible.

Benchmarks. To inspect the practical performance of each subroutine in terms of the execution time, we present a benchmark on our implementation. All the subroutines are implemented using CUDA and C++. The computation time of each subroutine is shown in Table 4.

The subroutine to preprocess on 706 constraints will be executed only once. Need to mention that, the subroutine to iterate linear constraints, update MQ systems, and verify produced hashes are multithreaded and the program would process on a batch (2^{14}) of candidates, thus for these subroutines the execution time is measured as the elapsed time to process one batch. Also note that, the above-mentioned subroutines are executed on CPU. When a new batch of MQ systems is updated, it is copied to the off-chip memory of GPU for solving process. In practice, the GPU program to solve MQ systems can be pipelined with the subroutine to update MQ systems.

The result in Table 4 shows that setting $D = 2$ has the best practical performance for the Crossbred algorithm. According to Table 4, the entire search space is 2^{39} and the program takes 209.53 seconds to process on 2^{14} guess candidates. We estimate one preimage can be found in 223 GPU years.

MQ Solving Method	D	Runtime of preprocessing constant linear constraints (seconds)	Runtime of setting iterating constraints and updating MQ systems (seconds)	Runtime of solving MQ systems (seconds)	Runtime of verification (seconds)	Estimate runtime to obtain a preimage (GPU Year)
Crossbred	2			183.89		223
Crossbred	3	7.76	21.21	263.94	4.43	308
Fast Exhaustive Search	N/A			212.13		253

Table 4. Preimage attack on 4-round Keccak$[r = 640, c = 160, l = 80]$ with 4 CPU cores and a single Tesla V100 GPU card.

4.3 Results

We executed our program on the GPU cluster consisting of 50 NVIDIA Tesla V100 GPUs for a total of 45 days and 7 hours. Our program had traversed about $2^{33.84}$ guessing times, namely, $2^{33.84+19} = 2^{52.84}$ time complexity, which is equivalent to $2^{52.84}/2^{58} = 2.8\%$ of traversal space and obtained 2 message candidates which could produce hashes with 2 bit differentials.

The message and corresponding result hash of the first candidate are:

$$
\begin{array}{rlllll}
A = & \texttt{d7 c4 77 ec} & \texttt{e8 22 18 ca} & \texttt{80 90 8a 29} & \texttt{7d 39 78 fc} & \texttt{10 93 1c 97} \\
 & \texttt{2e 42 88 81} & \texttt{f8 21 45 4e} & \texttt{04 8f d8 cd} & \texttt{74 27 c9 67} & \texttt{00 00 00 00} \\
 & \texttt{e2 7d d6 d0} & \texttt{c4 26 8d c2} & \texttt{19 23 07 6f} & \texttt{16 03 21 61} & \texttt{99 26 41 f8} \\
 & \texttt{d1 bd 77 7e} & \texttt{07 de ba b1} & \texttt{fb 70 27 32} & \texttt{8b d8 36 98} & \texttt{01 48 1a e4} \\
 & \texttt{00 00 00 00} & \texttt{00 00 00 00} & \texttt{00 00 00 00} & \texttt{00 00 00 00} & \texttt{00 00 00 00}, \\
H = & \texttt{75 1a 16 e5} & \texttt{e4 95 c1 e2} & \texttt{f7 22}, & &
\end{array}
$$

where the differences are highlighted red.

The message and result hash of the second candidate are:

$$
\begin{array}{rlllll}
A = & \texttt{61 47 20 d5} & \texttt{57 c0 64 06} & \texttt{62 ef 6d 7c} & \texttt{f1 b3 38 2a} & \texttt{cb 8c 48 b6} \\
 & \texttt{ff 01 e4 e4} & \texttt{9f 09 9b 05} & \texttt{92 76 dd 25} & \texttt{d5 5e 82 61} & \texttt{11 c7 78 1a} \\
 & \texttt{f8 9d 2c b7} & \texttt{82 52 7b 9f} & \texttt{1e f9 59 b0} & \texttt{2d 3e a6 0b} & \texttt{60 57 6c 9f} \\
 & \texttt{00 fe 1b 1b} & \texttt{60 f6 64 fa} & \texttt{6d 89 22 da} & \texttt{2a a1 7d 9e} & \texttt{ee 38 87 e5} \\
 & \texttt{00 00 00 00} & \texttt{00 00 00 00} & \texttt{00 00 00 00} & \texttt{00 00 00 00} & \texttt{00 00 00 00}, \\
H = & \texttt{75 1a 16 e5} & \texttt{e4 95 e1 e2} & \texttt{f7 32}. & &
\end{array}
$$

Along with the obtained 2-bit differential candidates, the frequencies of hamming distance between candidates' hash and target hash are counted in Table 5.

5 Conclusion

In this paper, improved preimage attacks are proposed on 4-round Keccak-224/256. We extend the attacks to the Keccak preimage challenge, implemented on a GPU cluster. Preimages of two-bit differentials with the target hashing value are found. Specifically, our attacks are based on the complexity elaboration of solving Boolean MQ systems, which is a fundamental tool in solving cryptographic problems and hence of independent interest.

	Hamming Distance	Number	Hamming Distance	Number
	2	2	13	78146
	3	7	14	97193
	4	28	15	95992
	5	115	16	69338
	6	389	17	33109
	7	1136	18	10398
	8	2883	19	1866
	9	7223	20	175
	10	15155	21	11
	11	30203	22	1
	12	52239		
Total				425279

Table 5. Number of candidates with respect to the hamming distance from the target hash.

Acknowledgment

This work was funded by The National Key Research and Development Program of China (Grant No. 2018YFA0704701), The Major Program of Guangdong

Basic and Applied Research(Grant No. 2019B030302008) and Major Scientific and Techological Innovation Project of Shandong Province, China (Grant No. 2019JZZY010133).

References

1. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Cryptographic sponge functions, submission to NIST (Round 3), 2011. `http://sponge.noekeon.org/CSF-0.1.pdf.`
2. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The keccak reference, version 3.0, submission to NIST (Round 3), 2011. `http://keccak.noekeon.org/Keccak-reference-3.0.pdf`
3. Bertoni, G., Daemen, J., Peeters, M., Asscher, G.V.: The keccak crunchy crypto collision and preimage contest, `https://keccak.team/crunchy\_contest.html`
4. Bouillaguet, C., Chen, H., Cheng, C., Chou, T., Niederhagen, R., Shamir, A., Yang, B.: Fast exhaustive search for polynomial systems in F_2. In: Mangard, S., Standaert, F. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6225, pp. 203–218. Springer (2010)
5. Dinur, I.: Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). Cryptology ePrint Archive, Report 2021/578 (2021), `https://eprint.iacr.org/2021/578`
6. Duarte, J.D.: On the complexity of the crossbred algorithm. IACR Cryptol. ePrint Arch. **2020**, 1058 (2020)
7. Guo, J., Liu, M., Song, L.: Linear structures: Applications to cryptanalysis of round-reduced keccak. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 249–274 (2016)
8. He, L., Lin, X., Yu, H.: Improved preimage attacks on 4-round keccak-224/256. IACR Transactions on Symmetric Cryptology **2021, Issue 1**, 217–238 (2021)
9. Joux, A., Vitse, V.: A crossbred algorithm for solving boolean polynomial systems. In: Kaczorowski, J., Pieprzyk, J., Pomykala, J. (eds.) Number-Theoretic Methods in Cryptology - First International Conference, NuTMiC 2017, Warsaw, Poland, September 11-13, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10737, pp. 3–21. Springer (2017)
10. Li, T., Sun, Y.: Preimage attacks on round-reduced keccak-224/256 via an allocating approach. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11478, pp. 556–584. Springer (2019)
11. Li, T., Sun, Y., Liao, M., Wang, D.: Preimage attacks on the round-reduced keccak with cross-linear structures. IACR Trans. Symmetric Cryptol. **2017**(4), 39–57 (2017)
12. Liu, F., Isobe, T., Meier, W., Yang, Z.: Algebraic attacks on round-reduced keccak/xoodoo. IACR Cryptol. ePrint Arch. **2020**, 346 (2020), `https://eprint.iacr.org/2020/346`

13. Lokshtanov, D., Paturi, R., Tamaki, S., Williams, R.R., Yu, H.: Beating brute force for systems of polynomial equations over finite fields. In: Klein, P.N. (ed.) Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19. pp. 2190–2202. SIAM (2017)
14. Morawiecki, P., Pieprzyk, J., Srebrny, M.: Rotational cryptanalysis of round-reduced keccak. In: Moriai, S. (ed.) Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8424, pp. 241–262. Springer (2013)
15. Morawiecki, P., Srebrny, M.: A SAT-based preimage analysis of reduced keccak hash functions. Inf. Process. Lett. **113**(10-11), 392–397 (2013)
16. Rajasree, M.S.: Cryptanalysis of round-reduced KECCAK using non-linear structures. In: Hao, F., Ruj, S., Gupta, S.S. (eds.) Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11898, pp. 175–192. Springer (2019)
17. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer, R. (ed.) Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3494, pp. 1–18. Springer (2005)
18. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3621, pp. 17–36. Springer (2005)
19. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3494, pp. 19–35. Springer (2005)
20. Wang, X., Yu, H., Yin, Y.L.: Efficient collision search attacks on SHA-0. In: Shoup, V. (ed.) Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3621, pp. 1–16. Springer (2005)

A Linearization of A Bit In B_3

By Guo et al.'s study [7], a quadratic bit in B_3 can be linearized by guessing 10 values of linear polynomials, which is called the *first linearization* method. Fig. 8 illustrates how to linearize a bit in B_3. Since both ρ and π are permutation steps, we can get the corresponding bit in A_3'. According to

$$A_3'[x,y,z] = A_3[x,y,z] \oplus \sum_{y=0}^{4} A_3[x-1,y,z] \oplus \sum_{y=0}^{4} A_3[x+1,y,z-1],$$

the corresponding bit is the *XOR*ed sum of 11 bits in A_3, as shown in Fig. 8. According to the equation

$$A_3[x,y,z] = B_2[x,y,z] \oplus (B_2[x+1,y,z] \oplus 1) \cdot B_2[x+2,y,z],$$

it is obvious that the only quadratic term in $A_3[x,y,z]$ is generated by $B_2[x+1,y,z]$ and $B_2[x+2,y,z]$. Hence we can linearize $A_3[x,y,z]$ by guessing values of $B_2[x+1,y,z]$ or $B_2[x+2,y,z]$. Note that $A_3[x,y,z]$ and $A_3[x-1,y,z]$ share a common operand $B_2[x+1,y,z]$ in their quadratic terms. By fixing the value of $B_2[x+1,y,z]$, $A_3[x,y,z]$ and $A_3[x-1,y,z]$ are linearized as well. Thus the 11 bits in A_3 can be linearized by guessing only 10 bits, i.e.,

$$B_2[x+1,y,z], B_2[x+3,y,z-1], y \in \{0,\cdots,4\}. \tag{18}$$

Equivalently, the bit in A'_3 is linearized and the corresponding equation $B_3[x,y,z] = A_4[x,y,z]$ turns to be linear one.

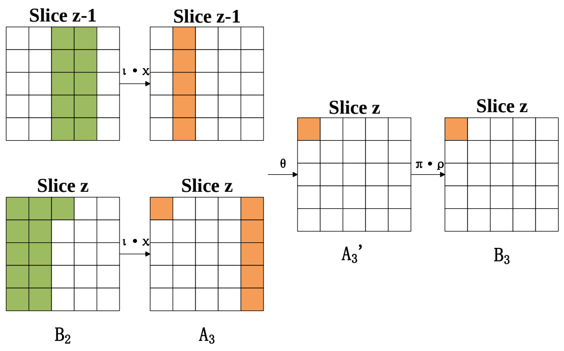

Fig. 8. The first quadratic bit linearization. We illustrate how to linearize a quadratic bit in B_3. In the figure, bits in green boxes are linear and bits in orange boxes are quadratic.