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

Congming Wei[1], Chenhao Wu[3], 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] University of Science and Technology of China, Hefei, Anhui, China
[3] The Chinese University of Hong Kong, Shenzhen, Shenzhen, 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, propose a new view for the case $D = 2$ and elaborate the computational complexity. The result shows that the Crossbred algorithm outperforms brute force theoretically and practically 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*. Due to the fundamental rule of solving MQ systems, the complexity elaboration of Crossbred algorithm is of independent interest.

Keywords: Keccak · Preimage attack · Multivariate quadratic systems

1 Introduction

Due to the breakthrough attacks on hash functions [WLF+05, WY05, WYY05b, WYY05a], the National Institute of Standards and Technology (NIST) started new standardization of hash functions. The Keccak sponge function [BDPAa] won the competition and became the new generation of Secure Hash Algorithm, known as **SHA-3**. Since its publication in 2008, Keccak has been widely studied. Both the keyed modes and the unkeyed modes of Keccak have made some progress in cryptanalysis.

For the keyed modes of Keccak, most of security analysis was based on the cube attack. In 2014, Dinur et al. [DMP+14] proposed a cube attack against stream cipher and MAC modes of Keccak. Later, in EUROCRYPT 2015, they proposed an attack combining cube attacks and structural properties of Keccak [DMP+15]. Huang et al. [HWX+17] introduced a new analysis model in 2017, called the conditional cube distinguisher and improved key recovery attacks reduced to 7 rounds. After that, some methods [LBDW17, SGSL18, LDB+19] were proposed to improve it. For collision attacks, Dinur et al. [DDS12] combined the low Hamming weight differential characteristic with algebraic techniques and gave actual collisions on 4-round Keccak-256. Besides, they put forward actual collision attacks on Keccak-384 and Keccak-512 up to 3 rounds based on internal differentials [DDS13]. Following the framework in [DDS12], Qiao et al. [QSLG17] implemented collision attacks of several 5-round Keccak instances and Song et al. [SLG17] further gave an actual collision on 5-round Keccak-224 as well as the 6-round collision challenge.

This paper is focused on the preimage attack. Morawiecki et al. [MS13] 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 [MPS13] was applied to the preimage attack on 4-round Keccak[r = 1024, c = 576] with complexity 2^{506}. After that, a breakthrough in the preimage attack occurred in 2016. Guo et al. proposed a new linear structure of Keccak [GLS16]. They introduced linearization of 3-round Keccak permutation functions and showed 3/4-round preimage attacks with the linear structure. After that, some improved attack methods have been proposed. Li et al. [LSLW17] constructed a new structure called cross-linear structure, and improved preimage attacks on several 3-round Keccak instances. A two-block method [LS19] 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 [Raj19] proposed a nonlinear structure, focusing on Keccak-384/512 reduced to 2, 3 and 4 rounds. Later, He et al. [HLY21] developed the linearization method of [LS19] 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. [LIMY20] 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 [Din21] gave the concrete complexity of polynomial method [LPT+17] for solving multivariate equation systems and applied it to preimage attacks on Keccak. The polynomial method can achieve exponential speedup as the number of variables goes to infinity. The new method successfully improved preimage attacks on Keccak-384/512 but did not outperform attacks on Keccak-224/256 in [HLY21].

Our Contributions. In this paper, we draw our attention on 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 solving an MQ system, which is equivalent to the $D = 2$ case of Crossbred algorithm. Our elaboration shows that the Crossbred algorithm outperforms the brute force even in the worst case, improving the complexity analysis in [Dua20]. More impressively, the algorithm uses feasible memory in a wide range of parameters and is easy to implement.

For the attack on Keccak, we propose a new 2-round linear structure with one round backward and one round forward. We exploit the output of the inverse χ^{-1} and carefully select constant values to linearize the one round backward with more arbitrary constants than in [LS19, HLY21]. According to our structure, all input bits of χ in the 4th round are quadratic. In our attacks, we construct MQ systems in order to fully utilize degrees of freedom and derived equations. Combined with solving MQ systems, we give 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 and Keccak-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. Section 2 shows notations and preliminaries of Keccak as well as the properties of the nonlinear layer χ, followed by the complexity elaboration of solving a Boolean Multivariate Quadratic system. Preimage attacks on 4-round Keccak-224 and Keccak-256 are present in Section 3. And the preimage attack on the challenge with implementation details is shown in Section 4. Finally, Section 5 concludes this paper.

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}	[GLS16]
		2^{207}	2^{8}	2^{215}	[LS19]
		–	–	[a]2^{202}	[Din21]
		2^{192}	2^{8}	2^{200}	[HLY21]
		2^{164}	2^{18}	[b]2^{182}	Subsection 3.1
256	Keccak-256	2^{251}	2^{3}	2^{254}	[GLS16]
		2^{239}	2^{8}	2^{247}	[LS19]
		–	–	[c]2^{231}	[Din21]
		2^{218}	2^{8}	2^{226}	[HLY21]
		2^{196}	2^{18}	[d]2^{214}	Subsection 3.2
80	Keccak$[r = 640, c = 160, l = 80]$	2^{39}	2^{19}	2^{58}	Subsection 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.

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 Boolean Multivariate Quadratic (MQ) system.

2.1 Notation

r	Rate of a sponge function
c	Capacity of a sponge function
b	Bit width of a permutation, $b = r + c$
n_r	Number of rounds.
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 The sponge function

The sponge construction is a framework used in Keccak. As illustrated in Figure 1, the sponge function has two phases, absorbing phase and squeezing phase. In the absorbing phase, an arbitrary number of input bits are first padded and then absorbed into the state of the permutation f. After that, the construction produces an arbitrary number of output bits in the squeezing phase.

The function f maps strings of b bits to strings of the same length. The b-bit initial state (IV) is set to be all 0's. In the absorbing phase, padded message is divided into blocks with length of r which is a positive integer and less than b. The capacity, denoted by c, equals to $b - r$. The first r-bit of IV is *XOR*ed with the first block and then is sent to f. Again, the first r-bit of output is *XOR*ed with the second block and computed in f.

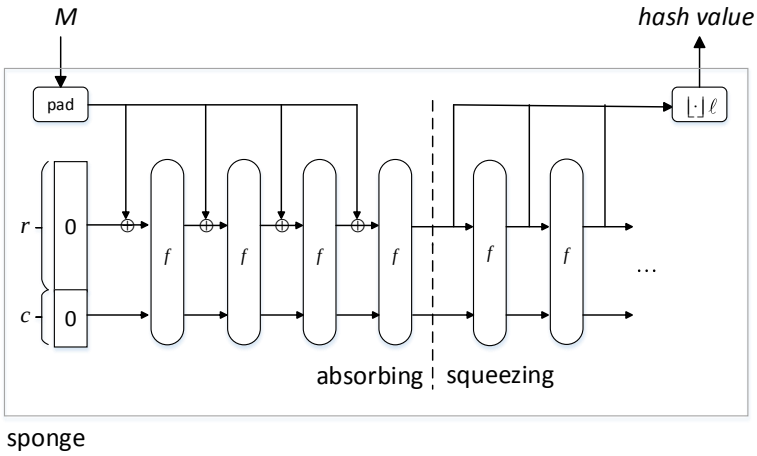

Figure 1: Sponge Construction

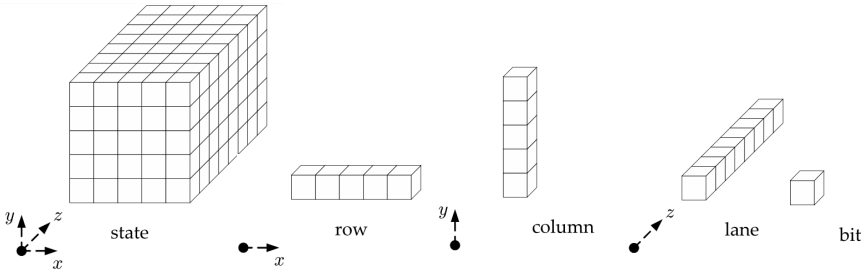

Figure 2: State of Keccak

This procedure is repeated until all the blocks are absorbed. After that, a l-bit digest is obtained in the squeezing phase. If l is less than r, the first l-bit output of the absorbing phase is the output string. Otherwise, if l is more 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 of the sponge construction.

2.3 Keccak-f PERMUTATIONS

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 5×5 w-bit lanes as depicted in Figure 2. For the Keccak-f permutation, w is chosen as $b/25$. $A[x, y]$ 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 $0 \leq z < w$.

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

Table 2: Offsets of ρ

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	153	231	3	10	171
$y=1$	55	276	36	300	6
$y=0$	28	91	0	1	190
$y=4$	120	78	210	66	253
$y=3$	21	136	105	45	15

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

Here "$\oplus$" and "$\cdot$" are additions and multiplications over F_2. $T[x,y]$ are offsets listed in Table 2 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.

2.4 The Keccak Hash Function

The Keccak hash function is the family of sponge functions 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 standard 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. The padding rule for Keccak, named multi-rate padding, extends message M to be a message of the form $M10^*1$. That is, message 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 standard Keccak functions except that it applies a different padding rule of the form $M0110^*1$, i.e., it first pads a message with two bits "01", then followed by the multi-rate padding rule in Keccak. Moreover, the two extendable-output functions (XOFs), SHAKE128 and SHAKE256, are introduced in the SHA-3 family. SHAKE128(M,l) and SHAKE256(M,l) are defined as Keccak[$r=1344, c=256$] and Keccak[$r=1088, c=512$] with a four-bit suffix '1111' to M. We refer the readers to [BDPAa] for more details.

2.5 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. [GLS16] 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 $b_{i+1}=1$.

Table 3: Inputs and their corresponding outputs of χ^{-1} for the 'xcxcc' input pattern

Inputs	x0x00	x0x01	x0x10	x0x11	x1x00	x1x01	x1x10	x1x11
Outputs	xx^2xx0	x0x01	xx^2xxx^2	$xxx1x^2$	xx^2xxx	x1x0x	xx^2xxx^2	$xxx1x^2$
#Linear	3	2	3	3	4	3	3	3
#Quadratic	1	0	2	1	1	0	2	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 must not have consecutive variables. Let x and c stand for the variable and constant, respectively. Each constant c could be 1 or 0. Table 3 lists the inputs 'xcxcc' and their corresponding outputs of χ^{-1}.

According to Table 3, 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. [GLS16] When $b_{i+3} = 0$, $b_{i+4} = 1$ and b_{i+1} is known, then all input bits a_j's can be written as linear combinations of b_j's, for all $i \in \{0, 1, 2, 3, 4\}$.

From (1), when $b_{i+3} = 0$, $b_{i+4} = 1$, we have

b_{i+1}	a_i	a_{i+1}	a_{i+2}	a_{i+3}	a_{i+4}
0	$b_i \oplus b_{i+2} \oplus 1$	0	$b_{i+2} \oplus 1$	0	1
1	b_i	1	$b_i \oplus b_{i+2}$	0	b_i

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

In this section, we propose a algorithm, equivalent to the Crossbred algorithm [JV17] with $D = 2$ case, and elaborate the concrete complexity for solving a Boolean Multivariate Quadratic (MQ) system. The subsequent attacks on 4-round Keccak are based on this algorithm.

An MQ polynomial of n Boolean variables $x_1, x_2, \ldots, x_n$ over binary field F_2 is defined as

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

where $a_{i,j} \in F_2$, $b_i \in F_2$ and $c \in F_2$. Then an MQ system of m equations and n variables, called an (m, n) MQ system, is given by

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

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

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

$$z(x_1, \ldots x_n) = x_1 f_1 + \cdots + x_n f_n + L + c \tag{3}$$

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

form

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

Next, 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*

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

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. It is obvious that for each guess 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 and can be solved by Gaussian elimination. If the system is solvable, the solution can be verified by substituting it into the MQ system of equations. If the solution is verified correct, 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 gray code guess, each equation update needs only $t+1$ bit operations and totally $(m - \frac{t(t+1)}{2})(t+1)$ 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, the 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

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

and

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

For the worst case $m = n$,

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

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, $T < 2^{n-t-1}\cdot(2t+3)^2(t+1) + 2^{n-t-1}\cdot 2n^2 < 2^{n+2-\sqrt{2n}}\cdot\left(2n^2+8n\sqrt{2n}+8n+\sqrt{2n}\right)$. In order to compare the complexity with that of the fast exhaustive search (FES), which is $2^{n+2}\log n$, we just need to compare $C(n) = 2^{-\sqrt{2n}}\cdot\left(2n^2+8n\sqrt{2n}+8n+\sqrt{2n}\right)$ with $\log n$. Intuitively, $C(n)$ decreases subexponentially to zero as n and hence is smaller than $\log n$ asymptotically. More precisely, when $n \geq 64$, $C(n) < \log n$, then Crossbred algorithm has lower complexity than FES in terms of bit operations.

Here, we list the comparison of our algorithm with fast exhaustive search [BCC+10] and polynomial method [Din21] in Table 4 for some practical parameters.

The results in Table 4 show that the Crossbred algorithm needs more bit operations than polynomial method when n becomes larger and m is close to n. When n is small and m outperforms n, our method needs lower computational costs. In addition, our method needs much less memory cost. When used in security analysis of cryptosystems, the memory cost is within the ability of a single PC. 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 Subsection 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 of 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. Given a hash value, we improve the structure in [GLS16] and derive an algebraic system such that the output bits after 3 rounds of degree at most 2. Then we solve this algebraic system using the algorithm in Subsection 2.6.

Table 4: 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 [BCC+10]
	2^{77}	2^{60}	Polynomial Method [Din21]
	2^{80}	2^{18}	Ours($m = n, t = 11$)
	2^{76}	2^{19}	Ours($m = 2n, t = 16$)
128	2^{133}	–	Exhaustive Search [BCC+10]
	2^{117}	2^{91}	Polynomial Method [Din21]
	2^{126}	2^{20}	Ours($m = n, t = 14$)
	2^{120}	2^{21}	Ours($m = 2n, t = 21$)
192	2^{197}	–	Exhaustive Search [BCC+10]
	2^{170}	2^{132}	Polynomial Method [Din21]
	2^{188}	2^{22}	Ours($m = n, t = 18$)
	2^{180}	2^{23}	Ours($m = 2n, t = 26$)
256	2^{261}	–	Exhaustive Search [BCC+10]
	2^{223}	2^{173}	Polynomial Method [Din21]
	2^{249}	2^{23}	Ours($m = n, t = 21$)
	2^{241}	2^{24}	Ours($m = 2n, t = 30$)

3.1 Preimage Attack on 4-round Keccak-224

In the following, we give an introduction to attacks on 4-round Keccak-224, namely, 4-round Keccak$[r = 1152, c = 448, l = 224]$. In [GLS16], Guo et al. proposed a linear structure for up to 3 round by setting some bits in the middle state as constants. In this section, we extend this structure to the preimage attack on 4-round Keccak-224, where two message blocks are used.

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. Figure 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. The other bits in A_1 are set to constants. For constant bits in A_1, we have

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

Then all bits in B_0 are linear over A_1 according to Property 2. Especially, bits in $B_0[3, y], y \in \{0, 1, \cdots, 4\}$ are set to 0 and bits in $B_0[4, 0]$, $B_0[4, 1]$ and $B_0[4, 3]$ are set to 1. Furthermore, 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. Instead of setting 449 constraints directly, we add another message block to reduce the number of constraints on A_0. Denote the output state of the first block as C. According to the capacity and the padding rule, we have the relations between C and A_0

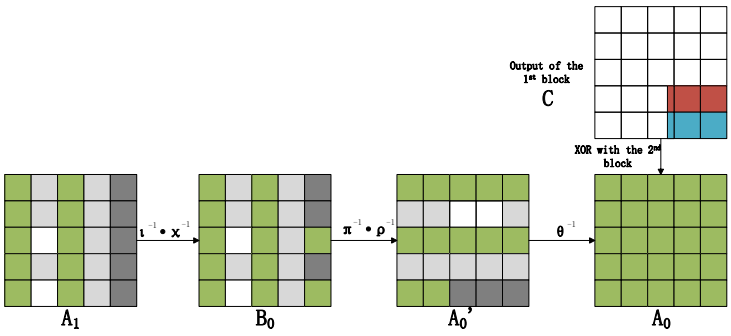

Figure 3: The one backward round of Keccak-224

as

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

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 first message block and its output C, A_0 can meet the requirement of (6) with only $5 \times 64 = 320$ 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 Figure 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 bits in A_2 keep linear. After a round of R, the bits in A_3 are of degree 2, which are indicated in orange in Figure 4. Since L is a linear operation, bits in B_3 are also quadratic.

Due to Property 1, given two consecutive bits b_i, b_{i+1} of the output of χ, we have $b_i = a_i \oplus (b_{i+1} \oplus 1) \cdot a_{i+2}$ for input bits a_i and a_{i+1}. When $b_{i+1} = 1$, it turns to be $a_i = b_i$. Then we obtain $2 \times 64 + 32 = 160$ quadratic equations from 224-bit hash value, i.e.,

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

Assuming that 0s and 1s appear equally in A_4, about half of equations are in the form of $B_3[x,y,z] = A_4[x,y,z]$. Furthermore, the equations of the form $B_3[x,y,z] = A_4[x,y,z]$ can be linearized by adding extra constraints as follows.

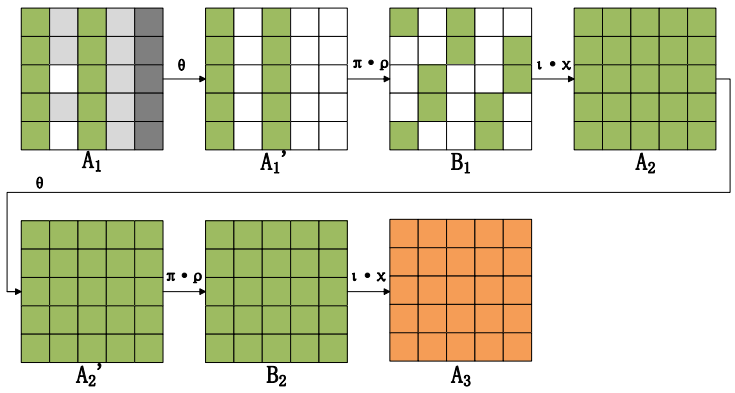

Figure 4: The 2 forward rounds of Keccak-224

By Guo et al.'s study [GLS16], a quadratic bit in B_3 can be linearized by guessing 10 values of linear polynomials, which is called the *first linearization* method. Figure 5 illustrates how to linearize a bit $B_3[x, y, z]$. 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 Figure 5. Thus linearizing the corresponding bit is equivalent to linearizing those 11 bits. It is depicted that all bits in B_2 are linearized. 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 either one. 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 guessing $B_2[x+1, y, z]$, $A_3[x, y, z]$ 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.,

$$B_2[x+1, y, z], B_2[x+3, y, z-1], y \in \{0, \cdots, 4\}. \tag{11}$$

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

Furthermore, 2 quadratic bits can be linearized by guessing 11 values of linear polynomials [GLS16]. We call it the *second linearization*. Figure 6 illustrates how to linearize two bits at the same time. For bits $B_3[0, 0, z]$ and $B_3[1, 0, z+44]$, linearizing the corresponding bits in A'_3 requires 21 linearized bits in A_3, i.e.,

$$A_3[4, y, z], A_3[0, y, z], A_3[1, 1, z], A_3[1, y, z-1], A_3[2, y, z-1], y \in \{0, \cdots, 4\}. \tag{12}$$

Again, we can achieve it by guessing values of bits in B_2. Since $A_3[x, y, z]$ and $A_3[x-1, y, z]$ share a common operand $B_2[x+1, y, z]$ in their quadratic terms, by guessing 11 bits

$$B_2[3, 1, z], B_2[3, y, z-1], B_2[1, y, z], y \in \{0, \cdots, 4\}, \tag{13}$$

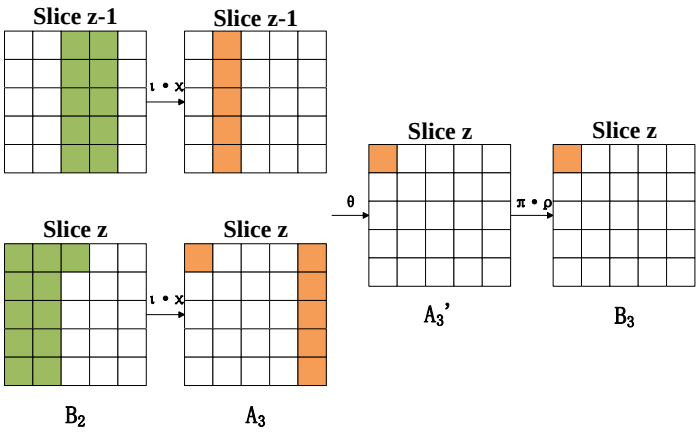

Figure 5: The first quadratic bit linearization. We illustrate how to linearize a quadratic bit in B_3. In the figure, bits in green boxes are linear and bits in orange boxes are quadratic.

the 21 bits in (12) are linear. Hence the bits in A'_3 are linearized and the corresponding ones in B_3 turn to be linear ones. 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 be linearized at the same time by guessing 11 values of linear polynomials.

According to (8), (9), (10), we have 160 quadratic equations over $640-(5+2)\times 64 = 192$ variables. After linearizing $2m$ equations by guessing $11m$ bits and n equations by guessing $10n$ bits, there remain $M = 160-2m-n$ quadratic equations over $N = 192-13m-11n$ variables. To deal with the rest equations, we use the method of solving MQ systems as shown in Subsection 2.6. The MQ system has a solution with the probability 2^{N-M}. With t chosen as $\lfloor\sqrt{2M+\frac{1}{4}}-\frac{3}{2}\rfloor$, totally, we need to solve $2^{224-160-(N-M)+(N-t-1)} = 2^{223-2m-n-t}$ linear systems. Note that we use the second linearization method first as it can linearize bits with less equations. Since the hash value can be regarded as random values, about 24 pairs from 160 quadratic equations can be used in the second linearization. When $m = 12$, $n = 0$[1], 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 Subsection 2.6, 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. Since $A[1,2], A[1,4], \alpha$, β and linearizing bits in the 3rd round are series of arbitrary constants, we can get $2^{64+100} = 2^{164}$ messages that satisfy the conditions by guessing the value of constants and thus our attack is feasible. For SHA3-224, the time complexity of the preimage attack is 2^{182} while the padding rule changes.

[1]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.

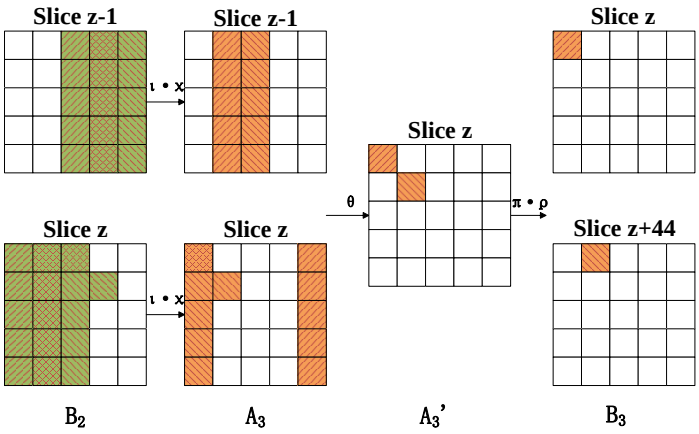

Figure 6: The second quadratic bit linearization of two quadratic bits in B_3. In the figure, up diagonal slash presents bits related to the first bit and down diagonal slash presents those related to the second bit. Besides, bits in green boxes are linear and bits in orange boxes are quadratic.

3.2 Preimage Attack on 4-round Keccak-256

The attack on 4-round Keccak-256 works similarly, where two message blocks are applied. Figure 7 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, i.e., there are totally $10 \times 64 = 640$ variables. Other bits in A_1 are set to constants. For constant bits in A_1, we have

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

Then all bits in B_0 keep 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

$$\begin{aligned} &C[2, 3] \oplus C[2, 4] = \texttt{0xFFFF FFFF FFFF FFFF}, \\ &C[3, 3] \oplus C[3, 4] = \texttt{0xFFFF FFFF FFFF FFFF}, \\ &C[4, 3] \oplus C[4, 4] = \texttt{0xFFFF FFFF FFFF FFFF}, \\ &C[1, 3, 63] = C[1, 4, 63]. \end{aligned} \tag{14}$$

The complexity of finding a preimage whose 4-round output satisfy (14) 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

$$A_0[x, 4] = C[x, 4], x \in \{0, 1, \cdots, 4\}. \tag{15}$$

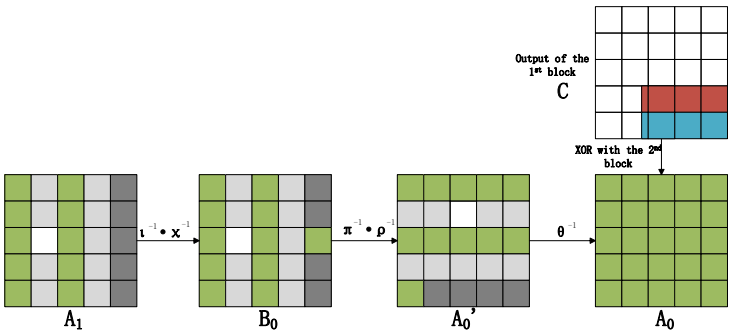

Figure 7: The one backward round of Keccak-256

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

$$\sum_{y=0}^{4} A_1[0,y] = \alpha, \sum_{y=0}^{4} A_1[2,y] = \beta, \tag{16}$$

where α and β are 64-bit constants. Totally, there are $(5+2) \times 64 = 448$ constraints on 640 variables. Similar to Keccak-224, the bits in A_3 are quadratic. Since L is a linear operation, bits in B_3 are also quadratic.

Due to Property 1, we have $3 \times 64 = 192$ quadratic equations from 256-bit hash value.

$$\begin{aligned} &B_3[0,0,z] \oplus (A_4[1,0,z] \oplus 1) \cdot B_3[2,0,z] = A_4[0,0,z], z = 0,1,\cdots,63, \\ &B_3[1,0,z] \oplus (A_4[2,0,z] \oplus 1) \cdot B_3[3,0,z] = A_4[1,0,z], z = 0,1,\cdots,63, \\ &B_3[2,0,z] \oplus (A_4[3,0,z] \oplus 1) \cdot B_3[4,0,z] = A_4[2,0,z], z = 0,1,\cdots,63. \end{aligned} \tag{17}$$

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.

According to (15), (16), (17), we have 192 quadratic equations and $640 - 448 = 192$ variables. We use the second linearization method as it can linearize bits with less equations. Among 192 quadratic equations 32 pairs meet the requirement for the second linearization on average. When using $m = 12$ pairs of equations[2], 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 linearizing bits in the 3rd round and thus our attack is feasible. For SHA3-256 and SHAKE256, in despite of different padding rules, the time complexities are also 2^{214}.

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.

[2]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.

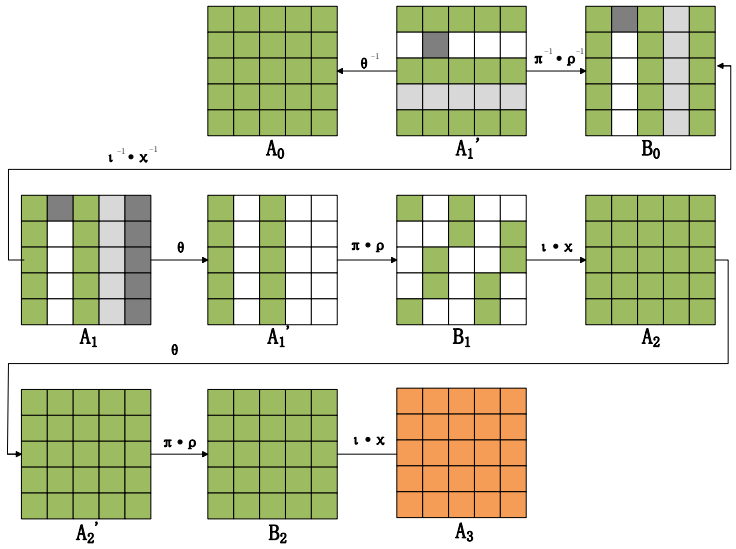

Figure 8: The linear structure of Keccak$[r = 640, c = 160, n = 4]$

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 [BDPAb]. 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 Figure 8. 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 $10 \times 32 = 320$ variables. For constant bits in A_1, we have

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

Thus $B_0 = \chi^{-1}(A_1)$ are linearized according to Property 2. Since L^{-1} consists of several linear steps, all bits in A_0 are linearized. Note that the last 162 bits are set to 0 or 1 such that A_0 satisfies the padding rule.

$$\begin{aligned} &A_0[3, 4, 30] = 1, A_0[3, 4, 31] = 1, \\ &A_0[x, 4, z] = 0, x \in \{0, \ldots, 4\}, z \in \{0, \ldots, 31\}. \end{aligned} \tag{18}$$

To avoid the propagation by θ, extra constraints are added to set the sums of columns as constants.

$$\sum_{y=0}^{4} A_1[0, y] = \alpha, \sum_{y=0}^{4} A_1[2, y] = \beta, \tag{19}$$

where α, β are 32-bit constants. Totally, there are $2 \times 32 = 64$ constraints. Figure 8 presents how variables propagate in 2 forward rounds. In the structure, the bits keep linear in A_2. And bits in A_3 are quadratic, which are indicated in orange in Figure 8.

Bits in $B_3 = L(A_3)$ keep quadratic as L is linear. According to Property 1, from 80-bit

hash value in A_4, we get $32+16=48$ quadratic equations of B_3.

$$\begin{aligned} &B_3[0,0,z] \oplus (A_4[1,0,z] \oplus 1) \cdot B_3[2,0,z] = A_4[0,0,z], z = 0,1,\cdots,31, \\ &B_3[1,0,z] \oplus (A_4[2,0,z] \oplus 1) \cdot B_3[3,0,z] = A_4[1,0,z], z = 0,1,\cdots,15. \end{aligned} \tag{20}$$

About half of them are in the form of $B_3[x,y,z] = A_4[x,y,z]$ due to that bits in the hash value can be considered as random values. We aim to linearize some equations in the form of $B_3[x,y,z] = A_4[x,y,z]$. Similar to the preimage attack on 4-round Keccak-224, a quadratic bit can be linearized by adding 10 equations and 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.

According to (18), (19), (20), we have 48 quadratic equations and $320-162-64=94$ variables. After linearizing some equations we use the method of solving MQ systems in Subsection 2.6 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.

$$\begin{aligned} H[0] &= \texttt{10101110 01011000 01101000 10100111} \\ H[1] &= \texttt{00100111 10101001 10000111 01000111} \\ H[2] &= \texttt{11111111 01000100} \end{aligned}$$

Due to Property 1, we obtained 48 quadratic equations from $H[i]$ and 26 equations are in the form of $B_3[x,y,z] = A_4[x,y,z]$ which can be used in linearization. We underlined those bits that can derive such equations and found that there are totally 5 pairs of bits satisfying the second linearization, namely, $(H[0][21], H[1][1])$, $(H[0][22], H[1][2])$, $(H[0][23], H[1][3])$, $(H[0][25], H[1][5])$, $(H[0][29], H[1][9])$. Hence in the experiment, according to (13), 2×5 equations are linearized by imposing linear conditions on bits as follows:

$$\begin{aligned} &B_2[3,1,21], B_2[3,y,20], B_2[1,y,21], y \in \{0,\cdots,4\}, \\ &B_2[3,1,22], B_2[3,y,21], B_2[1,y,22], y \in \{0,\cdots,4\}, \\ &B_2[3,1,23], B_2[3,y,22], B_2[1,y,23], y \in \{0,\cdots,4\}, \\ &B_2[3,1,25], B_2[3,y,24], B_2[1,y,25], y \in \{0,\cdots,4\}, \\ &B_2[3,1,29], B_2[3,y,28], B_2[1,y,29], y \in \{0,\cdots,4\}. \end{aligned}$$

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 Subsection 2.6, 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 Subsection 4.3.

4.2 Implementation Details

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 micro-architecture) 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 solution into original input and verify the hash result.

As described in Section 4.1, the original Keccak system consists of 800 free bits, represented by $A[5][5][32]$. With groups of constraints being imposed to the system, the number of free bits is reduced and finally yields an MQ problem with $n = 31$, $m = 38$. Among all the involved constraints, we simply categorize them into three different types,

- *Unchanging Linear Constraints*: constraints that the coefficient of each variable and the right-hand-side constants stay unchanged throughout the program execution. Specifically, we impose 706 unchanging linear constraints, which consists of 162 constraints to satisfy the padding rule, 480 constraints given by Property 2, and 64 constraints to avoid propagation by θ.
- *Iterating Linear Constraints*: constraints that the coefficient of each variable is fixed but the right-hand-side constant is being iterated during the program execution. Specifically, 53 iterating linear constraints are imposed to linearlize the quadratic equations.
- *Quadratic Constraints*: constraints that originally include quadratic terms before the attack starts. For this type of constraints, when the iterating constraints are set, all the quadratic terms can be linearlized. Specifically, we impose 10 quadratic constraints in our attack.

To obtain a possible preimage of given hash, we first impose 706 unchanging linear constraints to the 800-variable system. Next, we iterate and impose all the possible settings of 53 iterating constraints, and then eliminate quadratic terms in 10 quadratic constraints and impose these quadratic constraints to the system. Finally, we solve the yielded MQ problems. If the MQ problem is solvable, we then use the solution of the MQ problem to produce all the other bits in the input message according to all the constraints imposed in the previous steps. The program will terminate and return the produced message if it is verified as the preimage of the given hash, otherwise, the program will keep trying new settings of the iterating constraints. The overall procedure is shown as Figure 9.

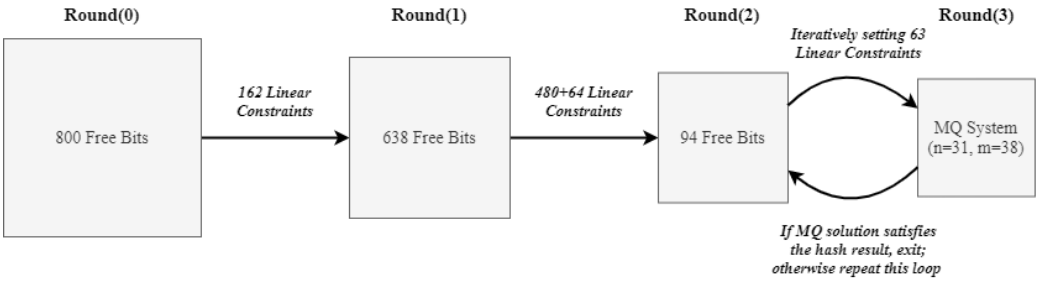

Figure 9: Four Main Routines towards Keccak Preimage Challenge

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

4.2.1 Linear Representation Extraction

An important subroutine in our program is to handle all the linear constraints. Because the linear constraints are imposed in several groups, in general we handle linear constraints by uniting one group of constraints into one constraint system and consider only one constraint system at a time.

Assume that there are k linear constraints over n variables, composing to a constraint system $[\mathcal{S}_{n,k}|\mathbf{c}]$, such that

$$\mathcal{S}_{n,k} \cdot \mathbf{x} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \dots & a_{kn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \mathbf{c}$$

If constraints in $[\mathcal{S}_{n,k}|\mathbf{c}]$ have no linear dependency with each other, imposing $[\mathcal{S}_{n,k}|\mathbf{c}]$ will have $n-k$ variables in the system be free variables and the other k variables be dependent variables. A simple method to extract the linear representations among variables is to perform Gaussian Elimination on the constraint system $[\mathcal{S}_{n,k}|\mathbf{c}]$, which will yield a row echelon form matrix $[\hat{\mathcal{S}}_{n,k}|\hat{\mathbf{c}}]$ after the back substitution step. According to the definition of Gaussian Elimination, the index of pivot columns in $[\hat{\mathcal{S}}_{n,k}|\hat{\mathbf{c}}]$ correspond to the index of dependent variables and the index of non-pivot columns correspond to the index of free variables. For each time a new constraint system $[\mathcal{S}_{n,k}|\mathbf{c}]$ is imposed, we apply Gaussian Elimination to extract the linear representations of dependent variables. As the Gaussian Elimination finishes, the yielded row echelon form matrix $[\hat{\mathcal{S}}_{n,k}|\hat{\mathbf{c}}]$ is stored and in the verification step the program will backward reproduce the complete message using the value of free variables and stored $[\hat{\mathcal{S}}_{n,k}|\hat{\mathbf{c}}]$.

According to Section 4.1, there are 706 unchanging linear constraints set in the first and the second round. Because they are unchanging constraints, we could unite these constraints as $[\mathcal{S}_{800,706}|\mathbf{c}]$ and extract $[\hat{\mathcal{S}}_{800,706}|\hat{\mathbf{c}}]$ in advance of the main iteration. By this preprocessing, the entire Keccak system can be represented by the remaining 94 free variables and the computation complexity in further steps can be therefore reduced.

4.2.2 Iterate Constraints and Update MQ Systems

In the main iteration, the program guesses on the possible settings of right-hand-side constants of the 53 iterating linear constraints. A complete iteration generates 2^{53} possible outputs of A_3, and consequently, the iteration is equivalent to have total 53 bits in the 4th round hash being guessed in a bruteforce manner. Before the main iteration starts, the program stores a 94-variable MQ problem in memory. In every single iteration, after the right-hand-side constants of the 53 iterating constraints are set, the program extracts the linear representation of 53 dependent variables. Together with 10 linearlized quadratic constraints, the program substitutes these constraints into the 94-variables MQ problem, resulting in a 31-variable MQ problem with 38 equations. Subsequently, these MQ problems are copied to GPUs for the next solving process. To accelerate the MQ problem updating, we parallelize this procedure using *POSIX Thread API* such that the workloads are distributed to multiple CPU cores.

4.2.3 The Crossbred Algorithm

In our implementation, we employ the Crossbred Algorithm to solve the MQ problems. The Crossbred Algorithm first extends the original MQ problem into its degree-D Macaulay matrix $\mathcal{M}_D$ and then extract k equations from $\mathcal{M}_D$ in which all the non-linear monomials of the leading k variables are eliminated. After the assignment of remaining $n-k$ variables, a solution candidate that satisfies the extracted k equations can be obtained by solving a

linear system with Gaussian Elimination. After a complete iteration on the assignments of $n-k$ remaining variables, all the solution candidates can be collected. To effectively iterate the assignment of $n-k$ remaining variables, we employ the gray-code to evaluate the non-linear polynomials. After a solution candidate has been collected, the remaining $m-k$ equations will be used to verify the correctness of the candidate. If the MQ problem is solvable, a complete iteration on the remaining $n-k$ variables will finally give a solution of the MQ problem.

4.2.4 Verify Message Candidates

By the last step, all 800 variables in the Keccak system can be linearly represented by 31 variables in the MQ problem. Using the solution obtained from the last step and all the constraints imposed in the previous steps, a message candidate with complete 800-bit content can be reproduced. Since 53 bits in the hash are randomly guessed, we still need to compare the 4-round hash of the reproduced message with the target hash. In practice, the execution time of this verification process is negligible compared with that of updating and solving MQ problems.

4.2.5 Benchmarks

To inspect the practical performance of each subroutine in terms of the execution time, in this section, we present a benchmark on our implementation towards the preimage attack of 4-round Keccak$[r=640, c=160, l=80]$. All the subroutines are implemented using CUDA and C++. The computation time of each subroutine is shown in Table 5.

As described in Section 4.2.1, the subroutine to preprocess on constant linear constraints is invoked at the beginning of the program and it will be executed only once. Need to mention that, the subroutine to set iterating constraints, update MQ problems, and verify produced hashes are multithreaded and the program would process on a batch (2^{14}) of guess candidates, thus for these subroutines the execution time is measured as the elapsed time to process one batch of guess candidates. Also note that, three subroutines are executed on CPU, i.e., the subroutine to preprocess constant linear constraints, the subroutine to set iterating constraints and update MQ problems, and the subroutine to verify the reproduced message. When a new batch of MQ problems is updated, it is copied to the off-chip memory of GPU for solving process. In practice, the GPU program to solve MQ problems can be pipelined with the subroutine to update the MQ problems.

Table 5: Preimage attack on 4-round Keccak$[r=640, c=160, l=80]$ with 4 CPU cores and a single Tesla V100 GPU card.

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

In the Crossbred Algorithm, the performance of MQ problem solving is impacted by the setting of Macaulay matrix degree D and the number of equations and variables. We perform several experiments to determine the best choice of degree D among $D=2,3$. The result in Table 5 shows that setting $D=2$ has the best practical performance. According

to Table 5, the entire search space is 2^{39} and the program takes 209.53 seconds to process on 2^{14} guess candidates. Given the probability to have a solvable MQ system($n = 31$, $m = 38$) is 2^{-7}, we estimate one preimage can be found in 223 GPU years.

4.3 Results

As described in Subsection 4.1, the target hash value of 4-round Keccak$[r = 640, c = 160, l = 80]$ preimage challenge is

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

We executed our program on the GPU cluster consisting of 50 NVIDIA Tesla V100 GPUs for a total of 45 days and 7 hours, and obtained 2 message candidates which could produce hashes with 2 bit differentials.

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

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

where the differences are highlighted red.

The message and result hash of the second candidate are:

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

Along with the obtained 2-bit differential candidates, the frequencies of hamming distance between candidates' hash and target hash are counted in Table 6.

5 Conclusion

In this paper, improved preimage attacks are proposed on 4-round Keccak-224 and Keccak-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 foundamental tool in solving cryptographic problems and hence of independent interest.

Table 6: Number of candidates with respect to the hamming distance from the target hash.

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

References

[BCC+10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in F_2. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2010.

[BDPAa] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. The keccak reference, version 3.0. Submission to NIST (Round 3), 2011. http://keccak.noekeon.org/Keccak-reference-3.0.pdf.

[BDPAb] G. Bertoni, J. Daemen, M. Peeters, and G. V. Asscher. The keccak crunchy crypto collision and preimage contest. https://keccak.team/crunchy_contest.html.

[DDS12] Itai Dinur, Orr Dunkelman, and Adi Shamir. New attacks on keccak-224 and keccak-256. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 442–461. Springer, 2012.

[DDS13] Itai Dinur, Orr Dunkelman, and Adi Shamir. Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 219–240. Springer, 2013.

[Din21] Itai Dinur. 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.

[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.

[DMP+15] Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Cube attacks and cube-attack-like cryptanalysis on the round-reduced

keccak sponge function. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 733–761. Springer, 2015.

[Dua20] João Diogo Duarte. On the complexity of the crossbred algorithm. *IACR Cryptol. ePrint Arch.*, 2020:1058, 2020.

[GLS16] Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced keccak. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *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*, volume 10031 of *Lecture Notes in Computer Science*, pages 249–274, 2016.

[HLY21] Le He, Xiaoen Lin, and Hongbo Yu. Improved preimage attacks on 4-round keccak-224/256. *IACR Transactions on Symmetric Cryptology*, 2021, Issue 1:217–238, 2021.

[HWX+17] Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional cube attack on reduced-round keccak sponge function. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 259–288, 2017.

[JV17] Antoine Joux and Vanessa Vitse. A crossbred algorithm for solving boolean polynomial systems. In Jerzy Kaczorowski, Josef Pieprzyk, and Jacek Pomykala, editors, *Number-Theoretic Methods in Cryptology - First International Conference, NuTMiC 2017, Warsaw, Poland, September 11-13, 2017, Revised Selected Papers*, volume 10737 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2017.

[LBDW17] Zheng Li, Wenquan Bi, Xiaoyang Dong, and Xiaoyun Wang. Improved conditional cube attacks on keccak keyed modes with MILP method. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 99–127. Springer, 2017.

[LDB+19] Zheng Li, Xiaoyang Dong, Wenquan Bi, Keting Jia, Xiaoyun Wang, and Willi Meier. New conditional cube attack on keccak keyed modes. *IACR Trans. Symmetric Cryptol.*, 2019(2):94–124, 2019.

[LIMY20] Fukang Liu, Takanori Isobe, Willi Meier, and Zhonghao Yang. Algebraic attacks on round-reduced keccak/xoodoo. *IACR Cryptol. ePrint Arch.*, 2020:346, 2020.

[LPT+17] Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2190–2202. SIAM, 2017.

[LS19] Ting Li and Yao Sun. Preimage attacks on round-reduced keccak-224/256 via an allocating approach. In Yuval Ishai and Vincent Rijmen, editors, *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*, volume 11478 of *Lecture Notes in Computer Science*, pages 556–584. Springer, 2019.

[LSLW17] Ting Li, Yao Sun, Maodong Liao, and Dingkang Wang. Preimage attacks on the round-reduced keccak with cross-linear structures. *IACR Trans. Symmetric Cryptol.*, 2017(4):39–57, 2017.

[MPS13] Pawel Morawiecki, Josef Pieprzyk, and Marian Srebrny. Rotational cryptanalysis of round-reduced keccak. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 241–262. Springer, 2013.

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

[QSLG17] Kexin Qiao, Ling Song, Meicheng Liu, and Jian Guo. New collision attacks on round-reduced keccak. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 216–243, 2017.

[Raj19] Mahesh Sreekumar Rajasree. Cryptanalysis of round-reduced KECCAK using non-linear structures. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings*, volume 11898 of *Lecture Notes in Computer Science*, pages 175–192. Springer, 2019.

[SGSL18] Ling Song, Jian Guo, Danping Shi, and San Ling. New MILP modeling: Improved conditional cube attacks on keccak-based constructions. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 65–95. Springer, 2018.

[SLG17] Ling Song, Guohong Liao, and Jian Guo. Non-full sbox linearization: Applications to collision attacks on round-reduced keccak. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 428–451. Springer, 2017.

[WLF+05] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[WYY05b] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on SHA-0. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.