Conditional Cube Attack on Reduced-Round Keccak Sponge Function

Senyang Huang[1], Meiqin Wang[2,3], Xiaoyun Wang[1,2,3⋆], Jingyuan Zhao[4]

[1] Institute for Advanced Study, Tsinghua University, Beijing, 100084, China
huangsenyang12@mails.tsinghua.edu.cn
[2] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan 250100, China
[3] School of Mathematics, Shandong University, Jinan 250100, China
[4] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
xiaoyunwang@mail.tsinghua.edu.cn

Abstract. Since Keccak was selected as SHA-3 hash function by NIST, it has attracted considerable attention from cryptographic researchers. Keccak sponge function [1] has also been used to design message authentication codes (MAC) and authenticated encryption (AE) scheme Keyak. Till now, the most efficient key recovery attacks on Keccak-MAC and Keyak are cube attacks and cube-attack-like cryptanalysis proposed at EUROCRYPT'15. In this paper, we provide a new type of cube distinguisher named *conditional cube tester* for Keccak sponge function, where we append some bit conditions for some cube variables to reduce the dimension of the original cube tester. We apply the conditional cube tester to recover the key for reduced-round Keccak-MAC and Keyak. Compared to the previous key recovery attacks for Keccak-MAC and Keyak, our attacks are the best attacks according to the number of rounds or the complexity. Moreover, by constructing an MILP (mixed integer linear programming) model, we provide a searching algorithm to produce the most efficient conditional cube tester, which can be utilized as a distinguisher for Keccak sponge function. As a result, we improve the previous distinguishing attacks on Keccak sponge function. Although the attacks in this paper are the best ones compared with previous results, they cannot threat the security margin of Keccak sponge function.

Keywords: Keccak-MAC, Keyak, cube tester, conditional cube variable, ordinary cube variable

1 Introduction

In 2007, the U.S. National Institute of Standards and Technology (NIST) held a public competition in order to design a new standard for a cryptographic hash function SHA-3. In 2012, after five years of research and analysis, Keccak, designed by Bertoni et al., was selected as the winner of the contest.

⋆ Corresponding author

Since then, Keccak has attracted much more attention from cryptographic researchers. For keyless modes of reduced-round Keccak, many results have been brought out on collision attack [2], preimage attack [3] and second preimage attack [4] etc. In addition, a distinguisher of full 24-round Keccak internal permutation has been proposed in [5] with time complexity of 2^{1579} Keccak calls. Using the rebound attack and efficient differential trails, Duc et al.[6] provided a distinguisher for 8-round Keccak internal permutation with the complexity 2^{491}. After that, Jérémy et al.[7] proposed an 8-round internal differential boomerang distinguisher on Keccak with practical complexity. These results on Keccak internal permutation seem not to be close to the security margin of Keccak sponge function. For the distinguishing attack on Keccak sponge function, some results have been given in [8], [9] and [10].

In keyed modes, Keccak can not only generate an infinite key stream as a stream cipher but also be used as message authentication codes (MAC) and authenticated encryption (AE) schemes. Besides the side channel attack in [11], the key recovery attack given in [10] is the only public result on keyed modes of Keccak. In [10], Dinur et al. set the cube variables in the CP kernel, which controls the propagation of θ in the first round. Cube variables are carefully selected so that they will not get multiplied with each other after the first round. In this way, the cube dimension is reduced. Besides, the cube sums over the cube variables on output polynomials are only related to a part of key bits. This property is used to construct the first key recovery attack on reduced-round Keccak-MAC and Keyak, which is called cube-attack-like cryptanalysis. Moreover, the cube attack and cube-attack-like are very efficient methods to analyse Keccak-like cryptosystems [12] and [13]. However, most of the previous attacks only control the propagation of cube variables after the first round. How to control the relation of cube variables after the second round to improve the current attacks has been an interesting problem for cryptanalysis of Keccak sponge function. In this paper, we propose the *conditional cube tester* to solve this problem.

1.1 Our Contributions

Conditional Cube Tester for Keccak Sponge Function. In this paper, we propose a new model named *conditional cube tester* which is inspired by dynamic cube attack on Grain stream cipher in [14] and conditional differential attacks in [15]. In [14], by appending bit conditions on initial value (IV), the intermediate polynomials can be simplified and the degree of output polynomial can be reduced. However, the structure of Keccak sponge function is so different from Grain stream cipher that the dynamic cube attack in [14] can not be utilized for Keccak sponge function. In this paper, we control the propagation of cube variables caused by the nonlinear operation χ by appending bit conditions. With the propagation of cube variables controlled, it will benefit to find the cube variables not multiplied with each other after the second round of Keccak sponge function. We provide the algorithms for searching the cube variables and appending the corresponding bit conditions. Then, we can construct a cube tester based on the obtained cube variables and the corresponding conditions. Thus, we name it as 'conditional cube tester'. In this way, the dimension of the conditional cube tester can be reduced compared to the previous cube tester in some cases.

Improved Key Recovery Attack on Reduced-Round Keccak-MAC. We apply the conditional cube tester to recover the key for Keccak-MAC. As a result, we can provide a key recovery attack on five rounds of Keccak-MAC-512 with the complexity of 2^{24} Keccak calls. For six rounds of Keccak-MAC-384, full key bits could be recovered with the complexity of 2^{40} Keccak calls. We could break seven rounds of Keccak-MAC-256 with 2^{72} Keccak calls. The previous key recovery attacks have been provided in [10], where the attack on 5-round Keccak-MAC-288 has complexity of 2^{35} Keccak calls, the attacks on 6-round and 7-round Keccak-MAC-128 have complexity of 2^{66} and 2^{97} Keccak calls, respectively. Note that the attack on Keccak-MAC-n_1 can work on Keccak-MAC-n_2 with the same complexity as $n_1 > n_2$. We compare our key recovery attacks on Keccak-MAC with those previous attacks in Table 1. From Table 1, our attacks are the best attack according to the number of rounds. Moreover, the attacks on 5-round Keccak-MAC-512 and 6-round Keccak-MAC-384 are practical, which have been verified with experiments.

Rounds	Capacity	Time	Data	Memory	Reference
5	576	2^{35}	2^{35}	negligible	[10]
6	256	2^{66}	2^{64}	2^{32}	[10]
7	256	2^{97}	2^{64}	2^{32}	[10]
5	576/1024	2^{24}	2^{24}	negligible	Section 4
6	256/768	2^{40}	2^{40}	negligible	Section 4
7	256/512	2^{72}	2^{72}	negligible	Section 4

Table 1. Summary of key recovery attacks on Keccak-MAC

Improved Key Recovery Attack on Reduced-Round Keyak. We also use the conditional cube tester to recover the key for reduced-round Keyak, which is an AE scheme based on Keccak sponge function [16]. We focus on the key recovery attack on Keyak with two blocks of messages and assume that the nonce could be reused. According to the specification of Keyak [16], a nonce cannot be reused during providing confidentiality but the nonce can be fixed when only authenticity and integrity are required. Thus, our attacks on Keccak are to break the properties of authenticity and integrity. We give the attacks on 7-round and 8-round Keyak with the time complexity 2^{42} and 2^{74}, respectively. The previous key recovery attack on Keyak can work on 7-round with the time complexity 2^{76}, which also assumes the nonce could be reused. Table 2 compares our results with previous attacks on Keyak. From Table 2, our attacks on Keyak are the best attacks according to the number of rounds and the complexity. Note that memory complexity in our attacks is negligible.

Rounds	Capacity	Time	Data	Memory	Reference
7	256	2^{76}	2^{75}	2^{43}	[10]
7	256	2^{42}	2^{42}	negligible	Section 5
8	256	2^{74}	2^{74}	negligible	Section 5

Table 2. Summary of key recovery attacks on Keyak

Improved Distinguishing Attack on Keccak Sponge Function. The conditional cube tester could also be used to proceed the distinguishing attack on Keccak sponge function. We choose the conditional cube variables with MILP, which helps us to find the combination of conditional cube variables automatically. As a result, the distinguishing attack has been achieved practically up to seven rounds of Keccak sponge function. For the previous distinguishing attacks on Keccak sponge function, Naya-Plasencia et al. put forward a 4-round differential distinguisher over Keccak-256/224 in [8] and Das et al. gave a 6-round distinguisher over Keccak-224 in [9]. Besides, there is another straightforward distinguisher on n-round Keccak sponge function from [10] with time complexity of $2^{2^{n-1}+1}$ Keccak calls ($n \leq 7$).

Table 3 lists the previous and our distinguishing attacks on Keccak sponge function. Note that the attack on Keccak with the capacity c_1 can work on Keccak with the capacity c_2 with the same complexity where $c_1 > c_2$.

Rounds	Capacity	Time	Data	Memory	Referance
4	448/512	2^{25}	2^{24}	negligible	[8]
6	448	2^{52}	2^{52}	negligible	[9]
6	448/512/576	2^{33}	2^{33}	negligible	[10]
7	448/512/576	2^{65}	2^{65}	negligible	[10]
5	448/512	2^{9}	2^{9}	negligible	Section 6
6	1024	2^{9}	2^{9}	negligible	Section 6
6	448/512	2^{17}	2^{17}	negligible	Section 6
7	768	2^{17}	2^{17}	negligible	Section 6
7	448	2^{33}	2^{33}	negligible	Section 6

Table 3. Summary of distinguishing attacks on Keccak sponge function

This paper is organized as follow. In Section 2, we will introduce the preliminaries of this paper, including Keccak sponge function, two keyed modes of Keccak and the idea of cube tester. In Section 3, our new model – conditional cube tester will be introduced. In Section 4 and Section 5, we apply our new model on key recovery attack for Keccak-MAC and Keyak. In Section 6, distinguishing attacks on Keccak sponge function will be constructed based on conditional cube tester. Finally, we conclude the paper in Section 7.

2 Preliminaries

Keccak is a family of sponge hash functions, which could also be used as MAC, AE scheme and stream cipher. In this section, we will show the details of Keccak sponge function. Then, we will introduce two keyed modes of Keccak, including Keccak-MAC and AE scheme Keyak. Finally, the idea of cube tester will be stated.

2.1 Notations

k_i	the i-th key bit
v_i	public variable
A	state of Keccak
$A[i][j][k]$	input bit of Keccak in the $i-$th column, the $j-$th row, the $k-$th slice
Round(A,RC)	round function of Keccak, RC is round constant.
rot$[L, n]$	$L >>> n, L$ is a lane.

2.2 Keccak Sponge Function

Description of Keccak Sponge Function. We give a brief description here for understanding our attacks better. For a complete version, please refer to Keccak specification [1].

The sponge function works on a b-bit state. As we discuss the default version, b is set up to be 1600 in this paper. The 1600 bits are divided into two parts according to the parameters r and c, which are bitrate and capacity respectively. As described in [1], we use another parameter $n = c/2$ to denote the sponge function as Keccak-n. The length of hash value is n bits. There are four versions of Keccak sponge function, which are Keccak-224, Keccak-256, Keccak-384 and Keccak-512. Initially, all the 1600 bits are filled with 0s and the message will be split into r-bit blocks. The sponge function will work on the message in two phases.

The first phase is the absorbing phase. In this phase, the next r-bit message block will be XORed with its first r-bit intermediate state and process the state with internal permutation. The number of internal permutations is 24 for default version. After all the blocks are absorbed, there comes to the second phase, which is the squeezing phase. In this phase, Keccak-n will return the first r bits as the output of the function with internal permutation iteratively until the desired length of the digest is produced.

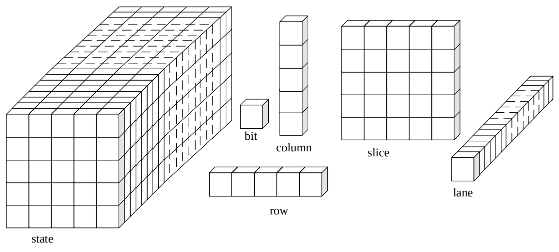


Fig. 1. Terminologies used in Keccak

Keccak is designed in the view of 3 dimensions. Fig. 1 shows the terminologies used in Keccak. The state of Keccak can be viewed as an array of 5×5 lanes. Each of the lane, denoted by $A[x, y]$, is a 64-bit string. And the internal permutation is all the same except for the round constant (RC) XORed into the state at the end of each round.

Next, we will show one round internal permutation with pseudo-code. The array B in the pseudo code denotes for an intermediate state. $C[x]$ and $D[x]$ are intermediate value for a single lane of 64 bits. $r[x, y]$ is the offset of the internal permutation for the position where the lane $A[x, y]$ should be moved to. $r[x, y]$ will be shown in Table 8. In our pseudo-code, the indices in all the operations are modulo 5 by default.

```
Round(A, RC)
{
    $\theta$ step
    for x in (0...4)
        C[x] = A[x,0] xor A[x,1] xor A[x,2] xor A[x,3] xor A[x,4]
        D[x] = C[x-1] xor rot(C[x+1],1)
    for x in (0...4)
        for y in (0...4)
           A[x,y] = A[x,y] xor D[x]

    $\rho$ step
    for x in (0...4)
        for y in (0...4)
            A[x,y] = rot[A[x,y],r[x,y]]

    $\pi$ step
    for x in (0...4)
        for y in (0...4)
            B[y,2*x+3*y] = A[x,y]
    $\chi$ step
    for x in (0...4)
        for y in (0...4)
            A[x,y] = B[x,y] xor ((not B[x+1,y]) and B[x+2,y])

    $\tau$ step
    A[0,0] = A[0,0] xor RC

    return A
};
```

The aim of θ is to diffuse the state. It could be seen that if certain variable in every column of state has even parity, the variable will not diffuse to other columns. In Keccak specification [1], this property is called **column parity kernel**, **CP kernel** for short. In this way, diffusion of some input variables caused by θ could be controlled in the first round. This property has been widely used in cryptanalysis of Keccak. It has also been applied in [10] to slow down diffusion of θ and decrease the dimension of cube used in the attack.

The operations ρ and π just change the position of bits. As τ will not impact our attacks, we will omit it in our paper. We refer to the first three linear operations, i.e. θ, ρ and π, as half a round. It is obvious to see that the only nonlinear operation in the internal permutation is χ. The algebraic degree of χ is only 2. After n-round Keccak internal permutation, the algebraic degree of output bit polynomial is 2^n at most.

2.3 Keyed Modes of Keccak

MAC based on Keccak. A message authentication code can be applied for ensuring data integrity and authentication of a message. During communicating, Alice sends a message M associated with a short tag T to Bob and Bob will verify whether M could produce T with the key they have shared. In this way, Bob confirms the integrity and source of M. Thus, a secure MAC algorithm is expected to meet two security requirements: no key recovery and resistance of MAC forgery.

Fig. 2 shows the construction of Keccak-MAC-n working on a single block. As described in 2.2, n is half of capacity length. For Keccak-MAC in this paper, message which we could control is a single block. Both sizes of key and tag are 128 bits. The two significant lanes are key bits. Block sizes are different based on the variants we analyse.

Authenticated Encryption Scheme based on Keccak During communication, AE schemes are cryptographic primitives to provide confidentiality, integrity and authenticity of data. Keccak sponge function can also be applied as authenticated encryption scheme. Here we will introduce the design of Keyak [16], an authenticated encryption scheme based on Keccak. Keyak is a second-round candidate algorithm submitted to CAESAR [17].

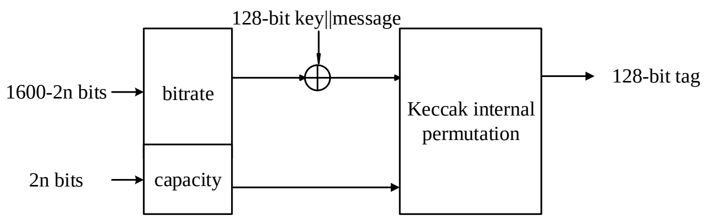


Fig. 2. Construction of Keccak-MAC-n

Fig.3 shows the construction of Keyak on two-block message. Keyak is the recommended version with 1600-bit state and no associated data. The internal permutation of Keyak submitted to CAESAR has twelve rounds. Both sizes of key and nonce are 128 bits and the capacity is set to 256 bits, i.e. $r = 1600-256 = 1344$.

According to the specification of Keyak [16], when confidentiality of data is not required, a nonce could be reused. In this paper, we focus on the two-block Keyak. We could control the value of the two-block plaintext as long as $2 \times 1344 = 2688$ bits.

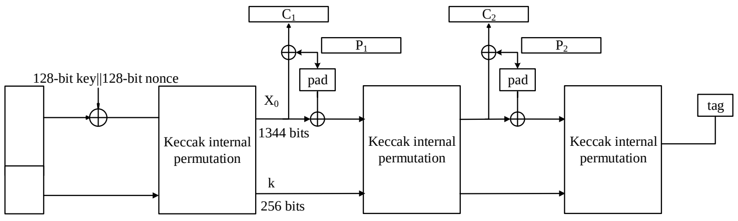

C1
P1
pad
C2
P2
pad
128-bit key||128-bit nonce
X0
1344 bits
k
256 bits
Keccak internal permutation
Keccak internal permutation
Keccak internal permutation
tag

Fig. 3. Construction of Keyak on two blocks

2.4 Cube Tester.

Cube tester introduced in [18] is a distinguisher to detect algebraic property of cryptographic primitives. Given a Boolean function with algebraic degree d, the adversary sums over its outputs over $2^k (k \leq d)$ inputs, in which a set of cube variables with size k ranges over all possible values and the other variables are fixed to some constants. The cube sum is the coefficient of the multiplication term on the cube variables in the set. The cube sum can be taken as differentiating the output polynomial over cube variables, which is higher order derivative [19] with respect to the cube variables. This non-random behaviour is based on the following theorem.

Theorem 1. *([10]) Given a polynomial $f : \{0,1\}^n \to \{0,1\}$ of degree d. Suppose that $0 < k < d$ and t is the monomial $x_0 \dots x_{k-1}$. Write f as:*

$$f(X) = t \cdot P_t(x_k, \dots, x_{n-1}) + Q_t(X),$$

where none of the monomials in $Q_t(X)$ is divisible by t. Then the sum of f over all values of the cube (cube sum) is

$$\sum_{x' \in C_t} f(x', x_k, \dots, x_{n-1}) = P_t(x_k, \dots, x_{n-1}),$$

where the cube C_t contains all binary vectors of the length k.

Proof of this theorem is given in [20]. All the polynomials in this paper are over $\mathbb{F}_2$. Cube sum here is exactly the value of $P_t(x_k, \dots, x_n)$, which can be regarded as the coefficient of t. Previous results in [18] and [20] are more concerned about the properties of P_t, such as its low algebraic degree, highly unbalanced truth table and so on.

***n*-Round Cube Tester on Keccak Sponge Functions.** Cube tester could be constructed based on algebraic property of Keccak sponge function, which could distinguish round-reduced Keccak from random permutation. As algebraic degree of output bits after n-round Keccak is bounded by 2^n, there is no terms of degree $2^n + 1$. Cube tester can detect this with a cube of dimension $2^n + 1$. This idea has been used in [10] to forge MAC instead of recovering key.

3 Conditional Cube Tester for Keccak Sponge Function

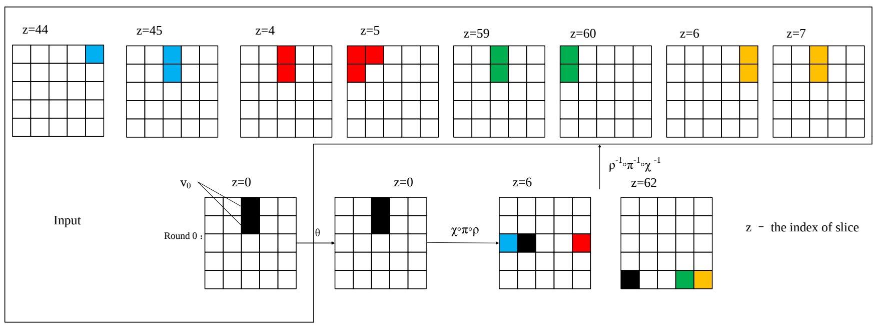


Fig. 4. Overview of bit conditions

The previous cube attacks against keyed modes of Keccak in [10] were built by extending one round before n-round cube tester to construct a $(n+1)$-round cube tester. That is to select the cube variables not multiplied with each other after the first round. For example, $A[2][0][0] = A[2][1][0] = v_0$ in Fig. 4 is a cube variable. This cube variable only impacts two bits before the operation χ in the first round. It is very easy to find a combination of $2^n + 1$ cube variables. The cube tester has been used to proceed the forgery attack on $(n + 1)$-round Keccak-MAC in [10]. More important contribution in [10] is that the cube tester has been applied to the key recovery attack for Keccak-MAC and Keyak with many dedicated and novel techniques.

In our new model, we focus to choose the cube variables which are not multiplied with each other after the second round. In this way, we need to collect $2^n + 1$ such cube variables to construct a cube tester for $(n + 2)$-round.

The scheme for the new model is shown in Fig. 4. In Fig. 4, we also choose $A[2][0][0] = A[2][1][0] = v_0$ as a cube variable. Furthermore, in order to reduce the possibilities that v_0 gets multiplied with other cube variables, we should append some additional conditions related to the colorful input bits of the second round according to the property of χ in Fig. 4. Then we denote the conditions with the input variables of the first round, where the corresponding input variable has same color as the input bit for the second round. Since we choose $A[2][0][0] = A[2][1][0] = v_0$ as a cube variable and set the colorful input bits in the first round with the proper conditions, the propagation of v_0 is slowed down and the possibility that v_0 gets multiplied with other cube variables after the second round is reduced. With this scheme, we construct a cube tester which is named as *conditional cube tester* in our paper. Next, we will give the definitions of some variables in a conditional cube tester.

Definition 1 *The variable summed in the cube tester is called **cube variable**. The variable in the CP kernel pattern is called **ordinary cube variable**. The variable assigned with a function is called **conditional variable**. The variable in the CP kernel pattern with bit conditions set to the conditional variables is called **conditional cube variable**. Other variables fixed with random values are called **free variables**.*

In this section, we will give the requirements of conditional and ordinary cube variables in a conditional cube tester. Then, some properties on Keccak sponge function will be discussed, based on which the algorithms to evaluate the relation between two cube variables after the second round are given.

3.1 Requirements on the Cube Variables in a Conditional Cube Tester

Actually, the advantage of an ordinary cube variable is that no extra conditions should be satisfied. However, the disadvantage is that it is difficult to avoid the ordinary cube variables multiplied with each other after the second round. Thus, in order to get an optimal cube tester for Keccak sponge function, we should choose a proper combination of some ordinary cube variables and conditional cube variables. The following theorem provides the basic requirements for the two types of cube variables in a conditional cube tester.

Theorem 2. *For $(n+2)$-round Keccak sponge function, there are p $(0 \leq p < 2^n + 1)$ conditional cube variables, i.e. $v_1, \ldots, v_p$, and $q = 2^{n+1} - 2p + 1$ ordinary cube variables, i.e. $v_{p+1}, \ldots, v_{p+q}$ (A special case is $q = 0$ if $p = 2^n + 1$). If the cube variables satisfying the following three requirements, the term $v_1 v_2 \ldots v_{p+q}$ will not appear in the output polynomials of $(n+2)$-round Keccak sponge function $(n > 0)$. With p conditional cube variables and q ordinary cube variables, a $(n+2)-$round cube tester can be constructed.*

a. *The conditional cube variables v_i $(1 \leq i \leq p)$ are not multiplied with each other after the second round.*
b. *The ordinary cube variables v_j $(p+1 \leq i \leq p+q)$ are not multiplied with each other after the first round.*
c. *Every conditional cube variable v_i $(1 \leq i \leq p)$ isn't multiplied with ordinary cube variable v_j $(p+1 \leq j \leq p+q)$ respectively after the second round.*

Proof. We use reduction to absurdity to prove. Assume that there is a monomial with $p+q$ cube variables after $(n+2)$-round Keccak and then we prove that the assumption is not true.

According to the above three requirements, for the intermediate polynomials of the second round, the degree of monomials M_2^p containing $v_1, v_2, \ldots, v_p$ is one and the degree of monomials M_2^q formed by $v_{p+1}, \ldots, v_{p+q}$ is two at most.

For the output polynomials after the following n-round operation, the monomial M_{n+2} with the highest degree should consist of 2^n monomials from M_2^p and M_2^q.

If all p conditional cube variables appear in M_{n+2} as we have assumed, there will be $2^n - p$ monomials from M_2^q in M_{n+2}. In this way, as the degree for M_2^q is 2, then the degree for M_{n+2} is at most $p + (2^n - p) \cdot 2 = 2^{n+1} - p$ instead of $2^{n+1} - p + 1$ Thus it is impossible that all p conditional cube variables and $q = 2^{n+1} - 2p + 1$ ordinary cube variables appear in M_{n+2}.

If $p = 2^n + 1$, no ordinary cube variable is needed. After $(n+2)$-round Keccak internal permutation, the degree of the monomial M_{n+2} is 2^n at most. Therefore, it is also impossible that all $p = 2^n + 1$ conditional cube variables appear in M_{n+2}. □

Note that if $p = 0$, there is no conditional cube variable, which is the same as forgery attacks in [10]. If $1 \leq p \leq 2^n + 1$, we can apply the conditional cube tester to recover the key for the $(n+2)$-round keyed modes of Keccak based on Theorem 2, which will be introduced in Section 4 and 5. Moreover, the distinguishing attack on Keccak sponge function can be implemented when $p = 2^n + 1$, which will be described in Section 6.

3.2 Properties of Keccak Sponge Function

In general, the differential propagation is much easier to trace than the Boolean function. So we will construct an equivalent relation between the differential characteristic and the Boolean function. Then the propagation of differential characteristic will be used to trace the Boolean function. Firstly, we will give the following property to describe the differential property for χ.

Property 1. If the input difference to χ in a row is (1,0,0,0,0), the output difference is (1,0,0,0,0) if and only if $x_1 = 0$ and $x_4 + 1 = 0$.

Proof. According to Boolean equations of χ , when the input of operation χ is assigned to $(x_0, x_1, x_2, x_3, x_4)$ and the other input is assigned to $(\Delta x_0 + x_0, x_1, x_2, x_3, x_4)$, the output of the pairs is as follow.

$$\begin{aligned}
y_0 &= x_0 + (x_1+1)x_2, & y_0' &= \Delta x_0 + x_0 + (x_1+1)x_2,\\
y_1 &= x_1 + (x_2+1)x_3, & y_1' &= x_1 + (x_2+1)x_3,\\
y_2 &= x_2 + (x_3+1)x_4, & y_2' &= x_2 + (x_3+1)x_4,\\
y_3 &= x_3 + (x_4+1)x_0, & y_3' &= x_3 + (x_4+1)(\Delta x_0 + x_0),\\
y_4 &= x_4 + (x_0+1)x_1, & y_4' &= x_4 + (\Delta x_0 + x_0 + 1)x_1.
\end{aligned}$$

It is clear that, the output difference is $(1, 0, 0, 0, 0)$ if and only if $x_1 = 0$ and $x_4 + 1 = 0$. □

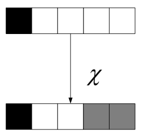


Fig. 5. Diffusion caused by operation χ

Now we describe the relation between the differential characteristic and the Boolean function based on Fig. 5. As the input difference for χ is $(1, 0, 0, 0, 0)$, the truncated output difference is $(1, 0, 0, ?, ?)$, where '?' is the unknown bit. From the view of Boolean function, the vector $(1, 0, 0, ?, ?)$ means that the first output bit y_0 denoted as '1' is linearly related to x_0, the second and third output bits y_1 and y_2 denoted as '0' are not related to x_0 and the fourth and fifth output bits y_3 and y_4 denoted as '?' are nonlinearly related to x_0. Thus, the truncated differential characteristic can represent the Boolean function equivalently. For the differential characteristic, if the conditions $x_1 = 0$ and $x_4 + 1 = 0$ are satisfied, the differential characteristic $(1, 0, 0, 0, 0) \rightarrow (1, 0, 0, 0, 0)$ will hold with probability 1. At the same time, if the conditions $x_1 = 0$ and $x_4 + 1 = 0$ are satisfied, $(1, 0, 0, 0, 0) \rightarrow (1, 0, 0, 0, 0)$ for Boolean function means that y_0 is linear related to x_0 and y_i is not related to x_0 for $1 \leq i \leq 4$.

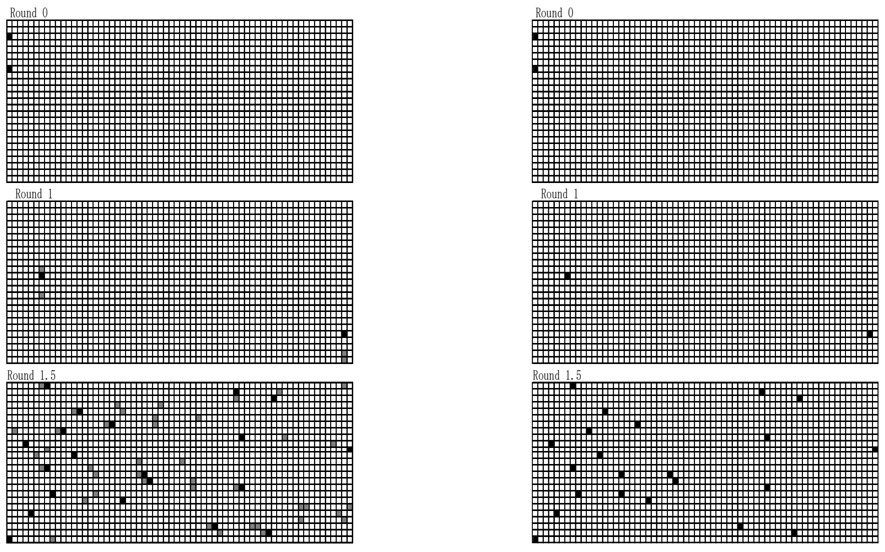
We summarize all the five differential characteristics cases that we will use in Table 4 where the input and output differences have only one non-zero bit.

Input/Output Difference	Conditions
$(1,0,0,0,0) \rightarrow (1,0,0,0,0)$	$x_1 = 0, x_4 = 1$
$(0,1,0,0,0) \rightarrow (0,1,0,0,0)$	$x_2 = 0, x_0 = 1$
$(0,0,1,0,0) \rightarrow (0,0,1,0,0)$	$x_3 = 0, x_1 = 1$
$(0,0,0,1,0) \rightarrow (0,0,0,1,0)$	$x_4 = 0, x_2 = 1$
$(0,0,0,0,1) \rightarrow (0,0,0,0,1)$	$x_0 = 0, x_3 = 1$

Table 4. Summary of conditions for differential characteristic of χ

As described in the above, in these cases, every input bit x_i of χ is only linearly related to one output bit y_i under two bit conditions, which will be used in constructing our conditional cube tester.

In order to show the advantage of conditional cube variable compared with the ordinary cube variable, we consider the propagation of an ordinary cube variable $A[2][0][0] = A[2][1][0] = v_0$ in the view of truncated differential characteristic in Fig. 6(a) and the propagation of a conditional cube variable $A[2][0][0] = A[2][1][0] = v_0$ in the view of differential characteristic in Fig. 6(b).



(a) Propagation of an ordinary cube variable

(b) Propagation of a conditional cube variable

Fig. 6. 1.5-round differential of an ordinary and a conditional cube variable

It is obvious to see that two active bits at the beginning of the second round will affect 22 bits caused by the step θ. Thus, the conditional cube variable in Fig. 6(b) is only linearly related 22 active bits after 1.5-round Keccak internal permutation and the ordinary cube variable in Fig. 6(a) is linearly related to the black bits and non-linearly related to the gray bits after 1.5-round Keccak. In total, 62 bits are related to v_0 after 1.5-round Keccak. Thus, it is more likely for a ordinary cube variable to get multiplied with other cube variables after the second round Keccak.

Note that we name the pattern of the conditional cube variable in Fig. 6(b) as 2-2-22 pattern according the number of active bits in three states (input state, the output state of the first round, the output state of the first 1.5 rounds).

As we introduce more conditional cube variables, contradictions may occur from the bit conditions derived from different conditional cube variables. In order to avoid such contradiction, we give the following property by observing the operation χ.

Property 2. For the χ operation in the first round, if in a certain row the first input bit is linear related with a conditional cube variable v_0 and the third input bit is linear related with a conditional cube variable v_1, it is impossible to cancel all the nonlinear terms containing v_0 or v_1 after the first round.

Proof. The output truncated difference over v_0 is (1,0,0,?,?). To remove the nonlinear terms including v_0 after the χ operation, conditions $x_1 = 0, x_4 = 1$ should be added. The output truncated difference over v_1 is (?,?,1,0,0), conditions $x_3 = 0, x_1 = 1$ should be satisfied to delete the nonlinear terms with v_1. In this way, there is a contradiction for the conditions for x_1. The contradictions for other input bits for χ operation can be derived similarly. □

If we can deduce the exact intermediate polynomials after the second round, whether two cube variables are multiplied with each other could be determined. But it is very complicated to derive. The truncated differential will help us to determine the relation between two cube variables efficiently, which will be introduced in Algorithm 1, 2 and 3.

Based on Keccak sponge function, if the neighboring input bits in a row of the operation χ are related to two different variables, the two variables would get multiplied in the next step. Line 6 in Algorithm 1, line 7 and 10 in Algorithm 2, and line 10 and line 13 in Algorithm 3 are based on this property. The three algorithms are corresponding to requirements [b], [a] and [c] in Theorem 2 respectively.

In the three algorithms, as we compute the output truncated difference or difference, we use '0', '1' and '2' to denote the inactive bit, the active bit and the unknown bit respectively. When deducing the difference or truncated difference over some cube variable v_0, the input bits of the first round set to v_0 are denoted to 1 and other input bits are set to 0. It should be noted that the indices in the pseudo-code are modulo 5.

Algorithm 1 Determine Relation of Two Ordinary Cube Variables

```
Input: input bit positions set $P_0$ of an ordinary cube variable $v_0$
       input bit positions set $P_1$ of an ordinary cube variable $v_1$
Output: relation of two ordinary cube variables $v_0$ and $v_1$ set on $P_0$ and $P_1$ respectively
1: compute the 0.5-round output difference from the input difference set $P_0$ ($P_1$) as 1
   and other input positions as 0, and store them in $B_0$ ($B_1$);
2: flag=0
3: for each $i \in [0, 63]$ do
4:   for each $j \in [0, 4]$ do
5:     for each $k \in [0, 4]$ do
6:       if ($B_0[k][j][i] = 1$, $B_1[k+1][j][i] = 1$)or ($B_0[k+1][j][i] = 1$, $B_1[k][j][i] = 1$)
         then
7:         flag=1;
8:       end if
9:     end for
10:  end for
11: end for
12: if (flag) then
13:   return multiplied after the first round;
14: else
15:   return not multiplied after the first round;
16: end if
```

Algorithm 2 Determine Relation of a Conditional Cube Variable and an Ordinary Cube Variable

```
Input: input bit positions set $P_0$ of a conditional cube variable $v_0$
       input bit positions set $P_1$ of an ordinary cube variable $v_1$
Output: relation of conditional cube variable $v_0$ and ordinary cube variable $v_1$ set on
   $P_0$ and $P_1$ respectively
1: flag=[0,0]
2: compute the 0.5-round output difference from the input difference set $P_0$ ($P_1$) as 1
   and other input positions as 0, and store them in $B_0$ ($B_1$);
3: compute the 1.5-round truncated output difference from the input difference set
   $P_0$ ($P_1$) as 1 and other input positions as 0, and store them in $C_0$ ($C_1$);
4: for each $i \in [0, 63]$ do
5:   for each $j \in [0, 4]$ do
6:     for each $k \in [0, 4]$ do
7:       if ($B_0[k][j][i] = 1$, $B_1[k+1][j][i] = 1$)or ($B_0[k+1][j][i] = 1$, $B_1[k][j][i] = 1$)
         then
8:         flag[0]=1;
9:       end if
10:      if ($C_0[k][j][i] = 1$, $C_1[k+1][j][i] \neq 0$)or ($C_0[k+1][j][i] = 1$, $C_1[k][j][i] \neq 0$)
         then
11:        flag[1]=1;
12:      end if
13:    end for
14:  end for
15: end for
16: if (flag[0]) then
17:   return multiplied after the first found;
18: else if (flag[1]) then
19:   return multiplied after the second round;
20: end if
21: return not multiplied after the second round;
```

Algorithm 3 Determine Relation of Two Conditional Cube Variables

```
Input: input bit positions set $P_0$ of conditional cube variable $v_0$
       input bit positions set $P_1$ of conditional cube variable $v_1$
Output: relation of two conditional cube variables $v_0$ and $v_1$ set on $P_0$ and $P_1$ respectively
 1: flag=[0,0,0]
 2: compute the 0.5-round output difference from the input difference set $P_0$ ($P_1$) as 1
    and other input positions as 0, and store them in $B_0$ ($B_1$);
 3: compute the 1.5-round output difference from the input difference set $P_0$ ($P_1$) as 1
    and other input positions as 0, and store them in $C_0$ ($C_1$);
 4: for each $i \in [0, 63]$ do
 5:   for each $j \in [0, 4]$ do
 6:     for each $k \in [0, 4]$ do
 7:       if ($B_0[k][j][i] = 1$, $B_1[k+2][j][i] = 1$)or ($B_0[k+2][j][i] = 1$, $B_1[k][j][i] = 1$)
          then
 8:         flag[0]=1;        ▷ Property 2.
 9:       end if
10:       if ($B_0[k][j][i] = 1$, $B_1[k+1][j][i] = 1$)or ($B_0[k+1][j][i] = 1$, $B_1[k][j][i] = 1$)
          then
11:         flag[1]=1;
12:       end if
13:       if ($C_0[k][j][i] = 1$, $C_1[k+1][j][i] = 1$)or ($C_0[k+1][j][i] = 1$, $C_1[k][j][i] = 1$)
          then
14:         flag[2]=1;
15:       end if
16:     end for
17:   end for
18: end for
19: if (flag[0]) then
20:   return contradiction;
21: else if (flag[1]) then
22:   return multiplied after the first round;
23: else if (flag[2]) then
24:   return multiplied after the second round;
25: end if
26: return not multiplied by the second round;
```

4 Key Recovery Attack on Reduced-Round Keccak-MAC

In this section, we will use conditional cube tester to achieve key recovery attacks against Keccak-MAC. Firstly, we discuss the general process for key recovery attack, including the attack process, complexity analysis and searching the combination of conditional and ordinary cube variables. Then, conditional cube attacks will be applied on different variants of Keccak-MAC, including Keccak-MAC-512, Keccak-MAC-384 and Keccak-MAC-224.

4.1 General Process for Key Recovery Attack on Keccak-MAC

Given a cube tester with p conditional cube variables and $q = 2^{n+1} - 2p + 1$ ordinary cube variables ($1 \leq p \leq 2^n + 1$), we could proceed key recovery attack on $(n + 2)$-round Keccak-MAC. It is assumed that k_0 equivalent key bits are related to the bit conditions derived from conditional cube variables. The attack process is described in the following.

Step 1. Fix the free variables with random values.
Step 2. Guess the value of k_0 equivalent key bits.
Step 3. Calculate the values of conditional variables under the guessed values of key bits.
Step 4. For all the possible values of cube variables, compute the corresponding tag and sum all the 128-bit tags over $(2^{n+1} - p + 1)$-dimension cube.
Step 5. If 128-bit cube sums over output tags are zero, the guessed key values are probably the right candidate; otherwise, it's the wrong guess and return to Step 2.

After executing the above process one time, the values of k_0 key bits could be recovered. The process takes $2^{2^{n+1}-p+1} \cdot 2^{k_0}$ Keccak calls. The rotation of cube variables will change the key bits in the bit conditions instead of the relations between the cube variables. To recover the remaining $128 - k_0$ key bits, we just shift the positions of all the cube variables in the attack and repeat the process for $128/k_0 - 1$ times. Therefore, both time and data complexity of the key recovery attack are $\frac{1}{k_0} \cdot 2^{2^{n+1}-p+k_0+8}$. Thus, for a $(n + 2)$-round conditional cube attack, the complexity is determined by $\frac{1}{k_0} \cdot 2^{k_0-p}$. However, more conditional cube variables involved in the attack will cause more key bits involved in the bit conditions to be guessed. The number of conditional cube variables can not be too large to make the attacks better. Thus, we use one conditional cube variable and $2^{n+1} - 1$ ordinary cube variables to construct an optimized key recovery attack on Keccak-MAC.

If we choose $A[2][0][0] = A[2][1][0] = v_0$ as the conditional cube variable, there are only two equivalent key bits involved in the bit conditions (bit condition is the relation between conditional variables, equivalent key bits and free variables). But if we choose other positions to set the conditional cube variable, the number of key bits involved in the bit conditions may be greater than two. Therefore, we choose $A[2][0][0] = A[2][1][0] = v_0$ as the conditional cube variable. Algorithm 4 will show how to find the corresponding ordinary cube variables with the conditional cube variable $A[2][0][0] = A[2][1][0] = v_0$.

Now, a cube tester has been constructed with v_0 and $2^{n+1} - 1$ ordinary cube variables. With the cube tester, we can proceed the key recovery attack on Keccak-MAC.

4.2 Key Recovery on 5/6/7-Round Keccak-MAC

Full key bits of 5-round Keccak-MAC-512 could be recovered with a conditional cube variable and 15 ordinary cube variables. The block size of this version is

Algorithm 4 Search the Ordinary Cube Variables along with the Conditional Cube Variable $A[2][0][0] = A[2][1][0] = v_0$ for Keccak-MAC

```
Output: a combination of $2^{n+1} - 1$ ordinary cube variables;
 1: $m$=#{ordinary cube variable candidates in bitrate part}
 2: $S = \varnothing$
 3: for each $i \in [0, m-1]$ do
 4:     proceed Algorithm 2 with $v_0$ and the $i$-th ordinary cube variable candidate $u_i$
        as the input;
 5:     if Algorithm 2 returns 'not multiplied by the second round' then
 6:         $S \leftarrow S \cup \{u_i\}$
 7:     end if
 8: end for
 9: Choose $2^{n+1} - 1$ cube variables from $S$ which will not be multiplied with each other
    after the first round and put these variables into $T$
10: return $T$
```

$1600 - 2 \cdot 512 = 576$ bits. As described in Section 4.1, A[0][2][0]=A[1][2][0]=v_0 is set to be the conditional cube variable and the corresponding ordinary cube variables are shown in Table 5. The time and data complexity of this attack is 2^{24}, which could be done on a desktop in a few minutes.

As the instance shown next for attacking Keccak-MAC-512, the key is generated randomly. For the convenience of statement, all the static variables are fixed to be zero, which could be random in the attack. It can be seen that the right key can be easily distinguished.

128-bit key:
1110000100010100000010110100100010111111100000011001011100111 0101
11000111110001011110100011111111010000101011000000011000100100010
right value: $k_5 + k_{69} = 1, k_{60} = 0$
guessed value:00, cube sum: 0xe93169ae5c86d086, 0xf6ec898c859bea1a
guessed value:01, cube sum: 0xc7d0bc36dc141c5e, 0x523a33c8753eb171
guessed value:10, cube sum: 0x0,0x0
guessed value:11, cube sum: 0x2ee1d5988092ccd8, 0xa4d6ba44f0a55b6b

Ordinary Cube Variables	A[2][0][8]=A[2][1][8]=v_1, A[2][0][12]=A[2][1][12]=v_2, A[2][0][20]=A[2][1][20]=v_3, A[2][0][28]=A[2][1][28]=v_4, A[2][0][41]=A[2][1][41]=v_5, A[2][0][43]=A[2][1][43]=v_6, A[2][0][45]=A[2][1][45]=v_7, A[2][0][53]=A[2][1][53]=v_8, A[2][0][62]=A[2][1][62]=v_9, A[3][0][3]=A[3][1][3]=v_{10}, A[3][0][4]=A[3][1][4]=v_{11}, A[3][0][9]=A[3][1][9]=v_{12}, A[3][0][13]=A[3][1][13]=v_{13}, A[3][0][23]=A[3][1][23]=v_{14}, A[3][0][30]=A[3][1][30]=v_{15}
Conditional Cube Variables	A[2][0][0]=A[2][1][0]=v_0
Bit Conditions	A[4][0][44]=0, A[2][0][4]= k_5 + k_{69} + A[0][1][5] + A[2][1][4] + 1, A[2][0][59]= k_{60} + A[0][1][60] + A[2][1][59] + 1, A[2][0][7]= A[4][0][6] + A[2][1][7] + A[3][1][7]
Guessed Key Bits	$k_{60}, k_5 + k_{69}$

Table 5. Parameters set for attack on 5-round Keccak-MAC-512

For 6-round Keccak-MAC-384, one conditional cube variable and 31 ordinary cube variables are used in conditional cube attack, which could recover full 128-bit key with 2^{40} Keccak calls. We search the corresponding ordinary cube variables with Algorithm 4. The parameters of this attack are shown in Table 9. The process could be finished on a desktop with four $i5$ processors in a few days. We will show an instance for attacking 6-round Keccak-MAC-384. Similarly, the key is generated randomly and all the static variables are fixed to be zero.

128-bit key:
1111011111001001000111010010100111100011110001110111100100000010
0111000010010100010101110110111110100010101010001110111001100011
right value: $k_5 + k_{69} = 1, k_{60} = 0$
guessed value:00, cube sum: 0x3f9d5fa4e143f779, 0x26607b3ce1c56f2b
guessed value:01, cube sum: 0x99bbf2ae6b93a7fb, 0xdbbb864fcc563747
guessed value:10, cube sum: 0x0,0x0
guessed value:11, cube sum: 0x398b37a846e81e42, 0x691cf4345e2164ee

For 7-round Keccak-MAC-256, conditional cube attack could recover full 128-bit key with 2^{72} Keccak calls with a cube of dimension 64. The parameters of this attack are shown in Table 10.

5 Key Recovery Attacks on Reduced-Round Keyak

As it has been done with Keyak in [10], we focus on the key recovery attack on Keyak with two blocks of messages in Fig. 3. In this way, we can use the first block as the input of the second permutation and the second block to get the output of the second permutation. Besides, we also assume that the nonce could be reused as [10]. Actually, the attack here is a state recovery attack. In Fig. 3, we can get the value of X_0, but the 256 bits in the capacity part are unknown which are denoted as $k \in \mathbb{F}_2^{256}$. We denote the i-th bit of k as $k_i (0 \leq i \leq 255)$. If we can recover k, then we can get the master key by proceeding the inverse of the first Keccak internal permutation.

During the attack, cube variables are set in P_1 and P_2 is fixed to zero and then C_2 is equal to the upper part of the output of Keccak internal permutation. The attack process is almost the same as the general process introduced in Section 4.1 except for the bit conditions. Fortunately, conditional cube attack could be extended by one more round forward without increasing the dimension of cube. The size of Keyak output is 1344 bits. In this way, the operation χ of the last round on the most significant 1280 bits could be reversed. The linear operations of the final round would never increase the degree of output polynomials, which guarantees the previous $(n+2)$-round cube tester could be hold for $(n+3)$-round with the same cube dimension.

For 7-round Keyak, conditional cube attack could be built with the same cube in Table 9 except for the bit conditions. The four bit conditions are shown in Table 6. By shifting the positions of cube variables and repeating the attack for $192/4 = 48$ times, three lanes of secret values, i.e. $k_0, \cdots, k_{191}$ could be recovered with $2^{36} \cdot 48 = 2^{41.58}$ Keyak calls. The other lane of key bits could be recovered by changing the conditional cube variable to $A[3][0][i] = A[3][1][i] = v_0$. Only one key bit is involved in the bit conditions after recovering three lanes of secret values. Recovering the remaining lane of secret values requires $2^{33} \cdot 2^6 = 2^{39}$ Keyak calls. In total, the time complexity to recover the full 128-bit master key is about 2^{42} Keyak calls.

For 8-round Keyak, the cube variables in Table 10 will be used and the bit conditions in the Table 6 will be applied. Similar as the attack on 7-round Keyak, the data and time complexity for 8-round attack is 2^{74}. Note that the memory complexity for both attacks could be neglected.

Bit conditions for 8(7)-round Keyak	A[4][0][44]=k_{169} (+A[4][1][44]) + A[2][2][45] + A[3][2][45] + A[4][2][44] + A[2][3][45] + A[4][3][44], A[0][0][5]= k_{128} + A[1][0][5] + A[2][0][4] + A[0][1][5] + A[2][1][4] + A[0][2][5] + A[2][2][4] + A[0][3][5] + A[2][3][4] + A[0][4][5] + 1, A[0][0][60]= k_{56} + k_{183} + A[2][0][59] + A[0][1][60] + A[2][1][59] + A[0][2][60] + A[2][2][59] + A[0][3][60] + A[2][3][59]+ A[0][4][60] + 1, A[2][0][7]= k_{131} + A[4][0][6] + A[2][1][7] + A[3][1][7] + A[4][1][6] + A[2][2][7]+ A[4][2][6] + A[2][3][7] + A[4][3][6]
Guessed Key Bits	k_{169}, k_{128}, k_{56} + k_{183}, k_{131}

Table 6. Parameters for attacking 7-round and 8-round Keyak

6 Distinguishing Attacks on Keccak Sponge Function

In this section, conditional cube tester will be applied to proceed the distinguishing attacks on Keccak sponge function with practical complexity. From Theorem 2, if we use $2^n + 1$ conditional cube variables under the three requirements, the monomial including such $2^n + 1$ conditional cube variables will not appear in the output polynomials of $(n + 2)$-round Keccak sponge function. That's to say, the dimension of the cube to distinguish $(n + 2)$-round Keccak is reduced to $2^n + 1$.

Constructing the cube tester includes two parts:

– Find a combination of enough conditional cube variables, which do not get multiplied with each other after the second round;
– Derive the corresponding bit conditions for the chosen conditional cube variables.

6.1 Constructing Conditional Cube Tester with MILP

In this section, a new model will be introduced to find a combination of enough conditional cube variables, which can be regarded as a mixed integer linear programming (MILP) problem. The MILP problem has been also applied to find the best differential characteristic in [21].

In this new model, a variable $x_i \in \mathbb{F}_2, 1 \leq i \leq m$ is set for each conditional cube variable candidate. If $x_i = 1$, it means that the i-th conditional cube variable candidate is selected as a conditional cube variable; otherwise, it is not chosen. To find enough conditional cube variables, we need to find the assignment $X = \{(x_1, x_2, \ldots, x_m) | x_i \in \mathbb{F}_2, 1 \leq i \leq m\}$ of hamming weight larger than $2^n + 1$. From previous analysis, we know that some conditional cube variables could not be selected together. So we firstly deduce the constrains on X. It will be shown how to deduce the constrains in Algorithm 5.

Algorithm 5 Deduce Constrains on X

```
Input: $m$ conditional cube variable candidates;
Output: A set $F$ of constrains on $X$
 1: $F = \varnothing$
 2: for each $i \in [1, m-1]$ do
 3:     for each $j \in [i+1, m]$ do
 4:         proceed Algorithm 3 on the $i$-th and the $j$-th conditional cube variables;
 5:         if Algorithm 3 does not return 'not multiplied after the second round' then
 6:             $F \leftarrow F \cup \{x_i + x_j \leq 1\}$
 7:         end if
 8:     end for
 9: end for
10: return $F$
```

With the constrains F, the selection problem for conditional cube variables is modeled into a binary linear programming problem as follow:

$$\sum_{i=1}^{m} x_i \geq 2^n + 1$$

$$\text{s.t.} A_0 X \leq b, X = \{(x_1, x_2, \ldots, x_m) | x_i \in \mathbb{F}_2, \quad 1 \leq i \leq m\}$$

where A_0 is a binary matrix and b a binary vector based on the inequalities in set F in Algorithm 5. Although MILP is proved to be NP-hard, fortunately a programming solver, Gurobi Optimizer [22], could solve certain instances based on branch and cut algorithm. We use Gurobi to solve this model directly.

Now the combination of enough conditional cube variables has been found. Next, we will proceed the distinguishing attacks on Keccak sponge function by deriving the corresponding conditions for the chosen conditional cube variables.

6.2 Distinguishing Attack on Keccak-512 and Keccak-384

Conditional cube variables candidates in one slice for Keccak-512 are set to colorful bit positions in Fig. 7. The bit positions in the same color is for one cube variable candidate. There are 256 such candidates in 64 slices. We execute Algorithm 5 to deduce all the constrains for the 256 conditional cube variable

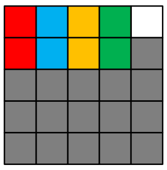


Fig. 7. Conditional cube variable candidates in a slice for Keccak-512

candidates and solve the problem with Gurobi Optimizer [22]. A combination of 9 conditional cube variables has been found, with which we can construct a 5-round conditional cube tester. As long as the conditions are satisfied, the algebraic degree of output polynomials over the conditional cube variables is 8 at most. Therefore, the cube sum of 5-round Keccak-512 output is zero. The most significant 320 output bits of Keccak-512 could be reversed, which wins the distinguishing attack one more round without increasing the complexity. The time complexity for the distinguishing attack on 6-round Keccak-512 with the conditional cube tester is 2^9 Keccak calls and the data complexity is 2^9. Similarly, we find a combination of 17 conditional cube variables for Keccak-384 which can be used to construct a 7-round conditional cube tester. We proceed the distinguishing attack on 7-round Keccak-384 with the complexity 2^{17}.

The conditions for these two conditional cube tester are shown in Table 11 and Table 12.

6.3 Distinguishing Attack on Keccak-224

The same process could be done with the conditional cube variables candidates in 2-2-22 pattern to distinguish 7-round Keccak-224. But the searching problem is too difficult to solve, where 1536 conditional cube variable candidates should be considered. Therefore, the conditional cube candidates are turned to be the ones in double kernel pattern. It means that if the conditions are satisfied, propagation of the variable is invariant to the operation θ in the second round.

The four differential characteristics in double kernel pattern are shown in Table 7 presented by hexadecimal numbers. We use '-' to denote zero difference and δ_0 is the input difference, δ_1 is the input difference of the second round and $\delta_{1.5}$ is the output difference after 1.5 rounds Keccak. The first two differential characteristics are found in [9] in 6-6-6 pattern and the other two are found with the method introduced in [8] in 8-8-8 pattern. For example, a conditional cube variable could be set as

$$A[0][0][0]=A[0][1][0]=A[2][1][30]=A[2][2][30]= A[1][0][63]=A[1][2][63]=v_0$$

This variable only impacts 6 bits after 1.5 round, which reduces the possibilities for the conditional cube variables to be multiplied with each other. We proceed Algorithm 5 on these 256 conditional cube variable candidates. The problem is

NO	δ_i	Differential				
0	δ_0	---------------1 ---------------1 ---------------- ---------------- ----------------	8--------------- ---------------- 8--------------- ---------------- ----------------	---------------- --------4------- --------4------- ---------------- ----------------	---------------- ---------------- ---------------- ---------------- ----------------	---------------- ---------------- ---------------- ---------------- ----------------
0	δ_1	---------------1 ---------------- ---------------1 ---------------- ----------------	---------------- ---------------- ------1--------- ------1--------- ----------------	-------------2-- ---------------- ---------------- -------------2-- ----------------	---------------- ---------------- ---------------- ---------------- ----------------	---------------- ---------------- ---------------- ---------------- ----------------
0	$\delta_{1.5}$	---------------1 ---------------- ---------------- ---------------- --------------8-	---------------- ---------------- ---------------- ---------------- ----------------	---------------- ---------------8 ---------------- ----4----------- ----------------	---------------- -----------2---- ---------------- ---------1------ ----------------	---------------- ---------------- ---------------- ---------------- ----------------
1	δ_0	---------------- -----------8---- ---------------- -----------8---- ----------------	---------------- ---------------- ------------8--- ------------8--- ----------------	---------------- ---------------- ---------------- ---------------- ----------------	---------------- ---------------1 ---------------1 ---------------- ----------------	---------------- ---------------- ---------------- ---------------- ----------------
1	δ_1	---------------- ---------------- ---------------- ---------------- ----------------	---------------- ---------------- ---------------- --8------------- --8-------------	---------------- ---------------- ---------2------ ---------2------ ----------------	---------------- 1--------------- ---------------- ---------------- 1---------------	---------------- ---------------- ---------------- ---------------- ----------------
1	$\delta_{1.5}$	---------------- ---------------- ---------------- ---------------- ----------------	---------------- ---------------- ---------------- ---------------- ---8------------	--------------1- ---------------- ---------------- ---------------- ----------------	---------------- ------1--------- ---------------- -----1---------- ----------------	---------------- ---------------- ---------------- --1------------- -2--------------
2	δ_0	---------------- --------------1- ---------------- --------------1- ----------------	---------------1 ---------------- ---------------- ---------------1 ----------------	---------------- -------4-------- -------4-------- ---------------- ----------------	---------------- ---------------- ---------------- ---------------- ----------------	------4--------- ---------------- ------4--------- ---------------- ----------------
2	δ_1	---------------- ---------------- ---------------2 ---------------2 ----------------	---------------- ---------------- -----1---------- -----1---------- ----------------	------------2--- ---------------- ---------------- ---------------- ------------2---	---------------- ----2----------- ---------------- ---------------- ----2-----------	---------------- ---------------- ---------------- ---------------- ----------------
2	$\delta_{1.5}$	---------------- ---------------- ---------------- ---------------- -------------8--	---------------- ---------------- ---------------- ---------------- ------1---------	---------------- --------------1- ---------------- ---4------------ ----------------	---------------- ----------2----- ---------------- ---------------- -----4----------	---------------- -------------4-- ---------------- ------2--------- ----------------
3	δ_0	---------------- --------------8- --------------8- ---------------- ----------------	---------------1 ---------------- ---------------1 ---------------- ----------------	---------------- ---------------- ---------------- ---------------- ----------------	---------------- --1------------- --1------------- ---------------- ----------------	------4--------- ---------------- ------4--------- ---------------- ----------------
3	δ_1	---------------- ---------------- ---------------2 ---------------2 ----------------	---------------- ---------------- ---------------- -----8---------- -----8----------	---------------- -------------4-- ------------2--- -------------4-- ------------2---	---------------- ---------------- ---------------- ---------------- ----------------	---------------- ---------------- ---------------- ---------------- ----------------
3	$\delta_{1.5}$	---------------- ---------------- ---------------- ---------------- ----------------	---------------- ---------------- -----------1---- ---------------- ----------------	-1-------------- --------------1- ---------------- ---------------- ----------------	---------------- ---------1------ ---------------- ---------2------ -----4----------	---------------- -------------4-- ---------------- ---------------- ----2-----------

Table 7. The four equivalent classes of differential paths

solved by Gurobi and a combination of 30 conditional cube variables has been found, which never get multiplied with each other after the second round. Three conditional cube variables in 2-2-22 pattern are appended with the combination. A 7-round cube tester on Keccak-224 is produced. These 33 conditional cube variables are independent.

Time complexity of this distinguishing attack is 2^{33}. Memory complexity is negligible. This distinguishing attack could be done on a desktop in hours. These conditional cube variables are shown in Table 13. The conditions could be derived exactly from the conditional cube variables. Due to the limited space, we list the conditions in the auxiliary supporting material.

7 Conclusion

In this paper, we propose the conditional cube tester for Keccak sponge function which has smaller dimension compared with the original cube tester in some cases. Using the conditional cube tester, we improve the previous key recovery attacks on Keccak-MAC and Keyak in terms of the number of rounds or the complexity. Moreover, the improved distinguishing attacks on Keccak sponge function have been provided in this paper based on the conditional cube tester. Note that our proposed conditional cube tester could be used for Keccak-like cryptosystems.

References

1. Bertoni Guido, Daemen Joan, Peeters Michaël, and Gilles Van Assche. Keccak Sponge Function Family Main Document. `http://Keccak.noekeon.org/Keccak-main-2.1.pdf`.
2. Itai Dinur, Orr Dunkelman, and Adi Shamir. Improved Practical Attacks on Round-Reduced Keccak. *Journal of Cryptology*, 27(2):183–209, 2014.
3. Morawiecki Pawel, Pieprzyk Josef, Srebrny Marian, and Straus Michal. Preimage Attacks on the Round-Reduced Keccak with the Aid of Differential Cryptanalysis. Cryptology ePrint Archive, Report 2013/561, 2013. `http://eprint.iacr.org/`.
4. Daniel J. Bernstein. Second Preimages for 6 (7 (8??)) Rounds of Keccak. *NIST mailing list (2010).*
5. Ming Duan and XueJia Lai. Improved Zero-sum Distinguisher for Full Round Keccak-f Permutation. *Chinese Science Bulletin*, 57(6):694–697, 2012.
6. Alexandre Duc, Jian Guo, Thomas Peyrin, and Lei Wei. Unaligned Rebound Attack: Application to Keccak. In Anne Canteaut, editor, *FSE,* volume 7549 of LNCS, pages 402–421. Springer, 2012.
7. Jérémy Jean and Ivica Nikolić. Internal Differential Boomerangs: Practical Analysis of the Round-Reduced Keccak-f Permutation. In Gregor Leander, editor, *FSE 2015,* volume 9054 of LNCS, pages 537–556. Springer, 2015.
8. María Naya-Plasencia, Andrea Röck, and Willi Meier. Practical Analysis of Reduced-Round Keccak. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology – INDOCRYPT 2011,* volume 7107 of LNCS, pages 236–254. Springer, 2011.

9. Sourav Das and Willi Meier. Differential Biases in Reduced-Round Keccak. In David Pointcheval and Damien Vergnaud, editors, *Progress in Cryptology – AFRICACRYPT 2014,* volume 8469 of LNCS, pages 69–87. Springer, 2014.
10. Itai Dinur, Paweł Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michał 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,* volume 9056 of LNCS, pages 733–761. Springer, 2015.
11. Mostafa Taha and Patrick Schaumont. Differential Power Analysis of MAC-Keccak at Any Key-Length. In Kazuo Sakiyama and Masayuki Terada, editors, *IWSEC,* volume 8231 of LNCS, pages 68–82. Springer, 2013.
12. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Cryptanalysis of Ascon. In Kaisa Nyberg, editor, *CT-RSA,* volume 9056 of LNCS, pages 371–387. Springer, 2015.
13. Morawiecki Pawel, Pieprzyk Josef, Straus Michal, and Srebrny Marian. Applications of Key Recovery Cube-attack-like. Cryptology ePrint Archive, Report 2015/1009, 2015. `http://eprint.iacr.org/`.
14. Itai Dinur and Adi Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In Antoine Joux, editor, *FSE,* volume 6733 of LNCS, pages 167–187. Springer, 2011.
15. Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of Nlfsr-Based Cryptosystems. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010,* volume 6447 of LNCS, pages 130–145. Springer, 2010.
16. Bertoni Guido, Daemen Joan, Peeters Michaël, and Gilles Van Assche. Keyak. `http://keyak.noekeon.org`.
17. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. `http://competitions.cr.yp.to/caesar.html`.
18. Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In Orr Dunkelman, editor, *FSE,* volume 5665 of LNCS, pages 1–22. Springer, 2009.
19. Xuejia Lai. Higher Order Derivatives and Differential Cryptanalysis. In Richard E. Blahut, Daniel J. Costello, Ueli Maurer, and Thomas Mittelholzer, editors, *Communications and Cryptography: Two Sides of One Tapestry*, pages 227–233. Springer, 1994.
20. Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009,* volume 5479 of LNCS, pages 278–299. Springer, 2009.
21. Siwei Sun, Lei Hu, Meiqin Wang, Qianqian Yang, Kexin Qiao, Xiaoshuang Ma, Ling Song, and Jinyong Shan. Extending the Applicability of the Mixed-Integer Programming Technique in Automatic Differential Cryptanalysis. In Javier Lopez and J. Chris Mitchell, editors, *ISC,* volume 9290 of LNCS, pages 141–157. Springer, 2015.
22. Gurobi optimization. *Gurobi:Gurobi optimizer reference manual (2015).* `http://www.gurobi.com`.

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

Table 8. offsets r[x,y] in operation ρ

Ordinary Cube Variables	A[2][0][12]=A[2][1][12]=v_1, A[2][0][20]=A[2][1][20]=v_2, A[2][0][28]=A[2][1][28]=v_3, A[2][0][41]=A[2][1][41]=v_4, A[2][0][43]=A[2][1][43]=v_5, A[2][0][45]=A[2][1][45]=v_6, A[2][0][53]=A[2][1][53]=v_7, A[2][0][61]=A[2][1][61]=v_8, A[2][0][62]=A[2][1][62]=v_9, A[3][0][3]=A[3][1][3]=v_{10}, A[3][0][9]=A[3][1][9]=v_{11}, A[3][0][13]=A[3][1][13]=v_{12}, A[3][0][15]=A[3][1][15]=v_{13}, A[3][0][23]=A[3][1][23]=v_{14}, A[3][0][30]=A[3][1][30]=v_{15}, A[3][0][40]=A[3][1][40]=v_{16}, A[3][0][46]=A[3][1][46]=v_{17}, A[3][0][56]=A[3][1][56]=v_{18}, A[3][0][57]=A[3][1][57]=v_{19}, A[4][0][5]=A[4][1][5]=v_{20}, A[4][0][10]=A[4][1][10]=v_{21}, A[4][0][12]=A[4][1][12]=v_{22}, A[4][0][14]=A[4][1][14]=v_{23}, A[4][0][47]=A[4][1][47]=v_{24}, A[4][0][58]=A[4][1][58]=v_{25}, A[4][0][62]=A[4][1][62]=v_{26}, A[4][0][63]=A[4][1][63]=v_{27}, A[0][1][28]=A[0][2][28]=v_{28}, A[0][1][34]=A[0][2][34]=v_{29}, A[0][1][37]=A[0][2][37]=v_{30}, A[0][1][46]=A[0][2][46]=v_{31}
Conditional Cube Variables	A[2][0][0]=A[2][1][0]=v_0
Bit Conditions	A[4][0][44]= A[4][1][44] + A[2][2][45], A[2][0][4]= k_5 + k_{69} + A[0][1][5] + A[2][1][4] + A[0][2][5] + A[2][2][4] + 1, A[2][0][59]= k_{60} + A[0][1][60] + A[2][1][59] + A[0][2][60] + A[2][2][59] + 1, A[2][0][7]= A[4][0][6] + A[2][1][7] + A[4][1][6] + A[2][2][7] + A[3][1][7].
Guessed Key Bits	$k_{60}, k_5 + k_{69}$

Table 9. Parameters set for attack on 6-round Keccak-MAC-384

Ordinary Cube Variables	A[2][0][8]=A[2][1][8]=v_1, A[2][0][12]=A[2][1][12]=v_2, A[2][0][20]=A[2][1][20]=v_3, A[2][0][28]=A[2][1][28]=v_4, A[2][0][41]=A[2][1][41]=v_5, A[2][0][43]=A[2][1][43]=v_6, A[2][0][45]=A[2][1][45]=v_7, A[2][0][53]=A[2][1][53]=v_8, A[2][0][62]=A[2][1][62]=v_9, A[3][0][3]=A[3][1][3]=v_{10}, A[3][0][9]=A[3][1][9]=v_{11}, A[3][0][13]=A[3][1][13]=v_{12}, A[3][0][30]=A[3][1][30]=v_{13}, A[3][0][40]=A[3][1][40]=v_{14}, A[3][0][46]=A[3][1][46]=v_{15}, A[3][0][56]=A[3][1][56]=v_{16}, A[4][0][5]=A[4][1][5]=v_{17}, A[4][0][10]=A[4][1][10]=v_{18}, A[4][0][12]=A[4][1][12]=v_{19}, A[4][0][14]=A[4][1][14]=v_{20}, A[4][0][31]=A[4][1][31]=v_{21}, A[4][0][47]=A[4][1][47]=v_{22}, A[4][0][58]=A[4][1][58]=v_{23}, A[4][0][62]=A[4][1][62]=v_{24}, A[4][0][63]=A[4][1][63]=v_{25}, A[0][1][37]=A[0][2][37]=v_{26}, A[0][1][47]=v_{27}, A[0][2][47]=v_{27}+v_{28}, A[0][3][47]=v_{28}, A[0][1][46]=A[0][2][46]=v_{29}, A[0][1][59]=A[0][2][59]=v_{30}, A[1][1][7]=A[1][2][7]=v_{31}, A[1][1][15]=A[1][2][15]=v_{32}, A[1][1][20]=A[1][2][20]=v_{33}, A[1][1][26]=A[1][2][26]=v_{34}, A[1][1][30]=A[1][2][30]=v_{35}, A[1][1][38]=A[1][2][38]=v_{36}, A[1][1][39]=A[1][2][39]=v_{37}, A[1][1][40]=A[1][2][40]=v_{38}, A[1][1][52]=A[1][2][52]=v_{39}, A[1][1][54]=A[1][2][54]=v_{40}, A[2][1][11]=A[2][2][11]=v_{41}, A[2][1][15]=A[2][2][15]=v_{42}, A[2][1][19]=A[2][2][19]=v_{43}, A[2][1][24]=A[2][2][24]=v_{44}, A[2][1][52]=A[2][2][52]=v_{45}, A[2][1][58]=A[2][2][58]=v_{46}, A[2][1][61]=A[2][2][61]=v_{47}, A[3][1][23]=A[3][2][23]=v_{48}, A[3][1][29]=A[3][2][29]=v_{49}, A[3][1][58]=A[3][2][58]=v_{50}, A[4][1][1]=A[4][2][1]=v_{51}, A[4][1][28]=A[4][2][28]=v_{52}, A[4][1][44]=A[4][2][44]=v_{53}, A[4][1][50]=A[4][2][50]=v_{54}, A[4][1][61]=A[4][2][61]=v_{55}, A[0][2][17]=A[0][3][17]=v_{56}, A[0][2][28]=A[0][3][28]=v_{57}, A[0][2][34]=A[0][3][34]=v_{58}, A[0][2][56]=A[0][3][56]=v_{59}, A[1][2][44]=A[1][3][44]=v_{60}, A[1][2][49]=A[1][3][49]=v_{61}, A[1][2][57]=A[1][3][57]=v_{62}, A[2][0][5]=A[2][2][5]=v_{63}
Conditional Cube Variables	A[2][0][0]=A[2][1][0]=v_0
Bit Conditions	A[4][0][44]= A[2][2][45] + A[3][2][45], A[2][0][4]= k_5 + k_{69} + A[0][1][5] + A[2][1][4] + A[0][2][5] + A[2][2][4] + A[0][3][5] + 1, A[2][0][59]= k_{60} + A[0][1][60] + A[2][1][59] + A[0][2][60] + A[2][2][59] + A[0][3][60] + 1, A[2][0][7]= A[3][1][7] + A[4][1][6] + A[2][1][7] + A[4][1][6] +A[2][2][7] + A[4][2][6]
Guessed Key Bits	$k_{60}, k_5 + k_{69}$

Table 10. Parameters set for attack on 7-round Keccak-MAC-256

Conditional Cube Variables	A[2][0][0]=A[2][1][0]=v_0, A[2][0][1]=A[2][1][1]=v_1,A[2][0][2]=A[2][1][2]=v_2, A[2][0][3]=A[2][1][3]=v_3,A[2][0][22]=A[2][1][22]=v_4, A[2][0][23]=A[2][1][23]=v_5, A[2][0][44]=A[2][1][44]=v_6, A[2][0][45]=A[2][1][45]=v_7,A[3][0][15]=A[3][1][15]=v_8
Bit Conditions	A[2][0][4]= A[0][0][5]+ A[1][0][5]+ A[0][1][5]+ A[2][1][4]+ 1, A[2][0][5]= A[0][0][6]+ A[1][0][6]+ A[0][1][6]+ A[2][1][5]+ 1, A[2][0][6]= A[0][0][7]+ A[1][0][7]+ A[0][1][7]+ A[2][1][6]+ 1, A[2][0][7]= A[0][0][8]+ A[1][0][8]+ A[0][1][8]+ A[2][1][7]+ 1, A[2][0][8]= A[4][0][7]+ A[2][1][8]+ A[3][1][8], A[2][0][9]= A[4][0][8]+ A[2][1][9]+ A[3][1][9], A[2][0][10]= A[4][0][9]+ A[2][1][10]+ A[3][1][10], A[2][0][17]= A[0][0][18]+ A[0][1][18]+ A[2][1][17]+ 1, A[2][0][25]= A[4][0][24]+ A[2][1][25], A[2][0][26]= A[0][0][27]+ A[1][0][27]+ A[0][1][27]+ A[2][1][26]+ 1, A[2][0][27]= A[0][0][28]+ A[1][0][28]+ A[0][1][28]+ A[2][1][27]+ 1, A[2][0][29]= A[4][0][28]+ A[2][1][29]+ A[3][1][29], A[2][0][30]= A[4][0][29]+ A[2][1][30]+ A[3][1][30], A[2][0][40]= A[0][0][41]+ A[0][1][41]+ A[2][1][40]+ 1, A[2][0][46]= A[4][0][45]+ A[2][1][46], A[2][0][47]= A[4][0][46]+ A[2][1][47], A[2][0][48]= A[4][0][47]+ A[2][1][48], A[2][0][49]= A[0][0][50]+ A[1][0][50]+ A[0][1][50]+ A[2][1][49]+ 1, A[2][0][51]= A[4][0][50]+ A[2][1][51]+ A[3][1][51], A[2][0][52]= A[4][0][51]+ A[2][1][52]+ A[3][1][52], A[2][0][59]= A[0][0][60]+ A[0][1][60]+ A[2][1][59]+ 1, A[2][0][60]= A[0][0][61]+ A[0][1][61]+ A[2][1][60]+ 1, A[2][0][61]= A[0][0][62]+ A[0][1][62]+ A[2][1][61]+ 1, A[2][0][62]= A[0][0][63]+ A[0][1][63]+ A[2][1][62]+ 1, A[3][0][23]= A[0][0][22]+ A[0][1][22]+ A[3][1][23], A[3][0][31]= A[0][0][30]+ A[0][1][30]+ A[3][1][31], A[3][0][45]= A[1][0][46]+ A[1][1][46]+ A[3][1][45]+ 1, A[4][0][3]= A[0][0][5]+ A[1][0][5]+ A[0][1][5]+ 1, A[4][0][6]= A[0][0][8]+ A[1][0][8]+ A[0][1][8]+ A[3][1][7]+ 1, A[4][0][25]= A[0][0][27]+ A[1][0][27]+ A[0][1][27]+ 1, A[0][1][49]= A[0][0][49]+ A[1][0][49]+ A[4][0][47]+ 1, A[4][0][44]=0, A[4][0][2] = 1.

Table 11. Conditions to distinguish Keccak-512

Conditional Cube Variables	A[0][0][14]=A[0][1][14]=v_0, A[2][0][23]=A[2][1][23]=v_1,A[2][0][24]=A[2][1][24]=v_2,
	A[2][0][43]=A[2][1][43]=v_3, A[2][0][44]=v_4, A[2][1][44]=v_4+v_5, A[2][2][44]=v_5,
	A[3][0][56]=A[3][1][56]=v_6, A[3][0][58]=A[3][1][58]=v_7, A[0][1][57]=A[0][2][57]=v_8,
	A[0][1][58]=A[0][2][58]=v_9, A[1][1][49]=A[1][2][49]=v_{10}, A[1][1][50]=A[1][2][50]=v_{11},
	A[2][1][41]=A[2][2][41]=v_{12}, A[0][0][20]=A[0][2][20]=v_{13}, A[1][0][13]=A[1][2][13]=v_{14},
	A[2][0][0]=A[2][2][0]=v_{15}, A[2][0][16]=A[2][2][16]=v_{16}

Bit Conditions	A[0][0][1]= A[3][0][2]+ A[0][1][1]+ A[3][1][2]+ A[4][1][2]+ A[0][2][1]
	A[0][0][2]= A[3][0][3]+ A[0][1][2]+ A[3][1][3]+ A[4][1][3]+ A[0][2][2]+ 1
	A[0][0][5]= A[3][0][6]+ A[0][1][5]+ A[3][1][6]+ A[0][2][5]+ 1
	A[0][0][7]= A[3][0][8]+ A[0][1][7]+ A[3][1][8]+ A[0][2][7]
	A[0][0][9]= A[3][0][10]+ A[0][1][9]+ A[3][1][10]+ A[0][2][9]
	A[0][0][12]= A[2][0][11]+ A[0][1][12]+ A[2][1][11]+ A[0][2][12]+ A[2][2][11]+ 1
	A[0][0][15]= A[2][0][14]+ A[0][1][15]+ A[2][1][14]+ A[0][2][15]+ A[2][2][14]
	A[0][0][16]= A[2][0][15]+ A[0][1][16]+ A[2][1][15]+ A[0][2][16]+ A[2][2][15]
	A[0][0][19]= A[2][0][18]+ A[0][1][19]+ A[2][1][18]+ A[0][2][19]+ A[2][2][18]+ 1
	A[0][0][22]= A[3][0][23]+ A[0][1][22]+ A[3][1][23]+ A[4][1][23]+ A[0][2][22]+ A[2][2][24]+ 1
	A[0][0][28]= A[1][0][29]+ A[2][0][27]+ A[2][0][28]
	+ A[4][0][29]+ A[0][1][28]+ A[0][1][29]+ A[1][1][28]+ A[2][1][27]+ A[2][1][28]
	+ A[4][1][29]+ A[0][2][28]+ A[0][2][29]+ A[1][2][28]+ A[2][2][27]+ A[2][2][28]+ 1
	A[0][0][29]= A[1][0][29]+ A[2][0][28]+ A[0][1][29]+ A[2][1][28]+ A[0][2][29]+ A[2][2][28]+ 1
	A[0][0][30]= A[1][0][29]+ A[4][0][30]+ A[1][1][29]+ A[4][1][30]+ A[1][2][29]+ 1
	A[0][0][34]= A[2][0][33]+ A[0][1][34]+ A[1][1][34]+ A[2][1][33]+ A[0][2][34]+ A[2][2][33]
	A[0][0][39]= A[4][0][37]+ A[0][1][39]+ A[4][1][37]+ A[0][2][39]+ 1
	A[0][0][40]= A[3][0][41]+ A[0][1][40]+ A[3][1][41]+ A[4][1][41]+ A[0][2][40]+ 1
	A[0][0][42]= A[2][0][41]+ A[0][1][42]+ A[0][2][42]
	A[0][0][43]= A[2][0][42]+ A[0][1][43]+ A[1][1][43]+ A[2][1][42]+ A[0][2][43]+ A[2][2][42]+ 1
	A[0][0][46]= A[1][2][46]+ A[2][0][45]+ A[0][1][46]+ A[2][1][45]+ A[0][2][46]+ A[2][2][45]+ 1
	A[0][0][48]= A[1][0][48]+ A[2][0][47]+ A[0][1][48]+ A[2][1][47]+ A[0][2][48]+ A[2][2][47]+ 1
	A[0][0][49]= A[2][0][48]+ A[2][0][49]+ A[3][0][48]+ A[0][1][49]+ A[2][1][48]
	+ A[3][1][48]+ A[0][2][49]+ A[2][2][48]
	A[0][0][60]= A[2][0][59]+ A[0][1][60]+ A[2][1][59]+ A[0][1][60]+ A[2][2][59]+ 1
	A[0][0][63]= A[3][0][0]+ A[0][1][63]+ A[3][1][0]+ A[0][2][63]+ 1
	A[1][0][8]= A[3][0][7]+ A[1][1][8]+ A[3][1][7]+ A[1][2][8]
	A[1][0][22]= A[0][1][23]+ A[1][1][22]+ A[1][2][22]+ A[2][2][24]+ 1
	A[1][0][23]= A[4][0][24]+ A[0][1][24]+ A[1][1][23]+ A[4][1][24]+ A[1][2][23]+ 1
	A[1][0][25]= A[3][0][24]+ A[1][1][25]+ A[3][1][24]+ A[1][2][25]+ 1
	A[1][0][28]= A[1][0][29]+ A[2][0][28]+ A[4][0][29]+ A[0][1][29]+ A[1][1][28]+
	A[2][1][28]+ A[4][1][29]+ A[0][2][29]+ A[1][2][28]+ A[2][2][28]
	A[1][0][44]= A[3][0][43]+ A[1][1][44]+ A[3][1][43]+ A[1][2][44]
	A[1][0][45]= A[3][0][44]+ A[1][1][45]+ A[3][1][44]+ A[1][2][45]
	A[1][0][49]= A[2][0][49]+ A[3][0][48]+ A[3][1][48]+ 1
	A[1][0][50]= A[4][0][51]+ A[0][1][51]+ A[4][4][51]+ 1
	A[1][0][51]= A[3][0][50]+ A[1][1][51]+ A[3][1][50]+ A[1][2][51]+ A[2][2][51]
	A[1][0][59]= A[4][0][60]+ A[1][1][59]+ A[4][1][60]+ A[1][2][59]
	A[2][0][2]= A[4][0][1]+ A[2][1][2]+ A[4][1][1]+ A[2][2][2]
	A[2][0][4]= A[2][1][4]+ A[2][2][4]
	A[2][0][5]= A[4][0][4]+ A[2][1][5]+ A[4][1][4]+ A[2][2][5]
	A[2][0][7]= A[4][0][6]+ A[2][1][7]+ A[3][1][7]+ A[4][1][6]+ A[2][2][7]
	A[2][0][22]= A[4][0][21]+ A[2][1][22]+ A[4][1][21]+ A[2][2][22]
	A[2][0][25]= A[4][0][24]+ A[2][1][25]+ A[4][0][24]+ A[2][2][25]
	A[2][0][30]= A[4][0][29]+ A[2][1][30]+ A[3][1][30]+ A[4][1][29]+ A[2][2][30]
	A[2][0][31]= A[4][0][30]+ A[2][1][31]+ A[3][1][31]+ A[4][1][30]+ A[2][2][31]
	A[2][0][38]= A[4][0][37]+ A[2][1][38]+ A[4][1][37]+ A[2][2][38]
	A[2][0][39]= A[3][0][41]+ A[2][1][39]+ A[3][1][41]+ A[4][1][41]+ A[2][2][39]
	A[2][0][50]= A[4][0][49]+ A[2][1][50]+ A[3][1][50]+ A[4][1][49]+ A[2][2][50]
	A[2][0][51]= A[2][2][51]+ 1
	A[2][0][62]= A[3][0][0]+ A[1][1][63]+ A[2][1][62]+ A[3][1][0]+ A[2][2][62]
	A[2][0][63]= A[4][0][62]+ A[2][1][63]+ A[4][1][62]+ A[2][2][63]
	A[3][0][22]= A[4][0][24]+ A[0][1][24]+ A[3][1][22]+ A[4][1][24]
	A[3][0][40]= A[4][0][37]+ A[3][1][40]+ A[4][1][37]+ A[4][1][40]
	A[3][0][49]= A[4][0][51]+ A[0][1][51]+ A[3][1][49]+ A[4][1][51]+ A[2][2][50]+ 1
	A[4][0][2]= A[4][1][2]+ 1
	A[4][0][22]= A[3][1][23]+ A[4][1][22]+ A[2][2][23]
	A[4][0][23]= A[4][1][23]+ A[2][2][24]
	A[4][0][50]= A[2][1][51]+ A[3][1][51]+ A[4][1][50]+ 1
	A[0][1][20]= A[2][0][19]+ A[2][1][19]+ A[2][2][19]+ 1
	A[1][2][40]= 1 A[4][1][0]= 1 A[1][2][19]= 1
	A[1][2][20]= 1 A[1][1][40]= 1 A[2][1][8]= 0 A[1][1][15]= 1

Table 12. Conditions to Distinguish Keccak-384

Conditional Cube Variables
A[0][0][3]=A[1][2][2]=v_0, A[0][1][3]=A[2][1][33]=A[2][2][33]=v_0+v_{25}, A[1][0][2]=v_0+v_{17},
A[0][0][6]=A[1][0][5]=A[1][2][5]=v_1, A[0][1][6]=A[2][1][36]=A[2][2][36]=v_1+v_{17},
A[0][0][9]=A[0][1][9]=A[2][1][39]=A[2][2][39]=A[1][0][8]=A[1][2][8]=v_2,
A[0][0][11]=A[1][0][10]=v_3, A[0][1][11]=A[2][1][41]=A[2][2][41]=v_3+v_{18}, A[1][2][10]=v_3+v_{16},
A[0][0][14]=A[2][1][44]=A[2][2][44]=A[1][0][13]=A[1][2][13]=v_4, A[0][1][14]=$v_4+v_{16}+v_{26}$,
A[0][0][16]=v_5, A[0][1][16]=A[2][1][46]=A[2][2][46]=v_5+v_{19}, A[1][0][15]=$v_5+v_{20}+v_{27}$, A[1][2][15]=v_5+v_{27},
A[0][0][19]=A[1][0][18]=A[1][2][18]=v_6, A[0][1][19]=A[2][1][49]=A[2][2][49]=v_6+v_{20},
A[0][0][21]=A[0][1][21]=A[2][1][51]=A[2][2][51]=v_7, A[1][0][20]=$v_7+v_{21}+v_{28}$, A[1][2][20]=v_7+v_{28},
A[0][0][22]=A[2][1][52]=A[2][2][52]=A[1][0][21]=v_8, A[0][1][22]=A[1][2][21]=v_8+v_{14},
A[0][0][24]=A[1][0][23]=A[1][2][23]=v_9, A[0][1][24]=A[2][1][54]=A[2][2][54]=v_9+v_{21},
A[0][0][27]=A[2][1][57]=A[2][2][57]=A[1][0][26]=A[1][2][26]=v_{10}, A[0][1][27]=$v_{10}+v_{28}$,
A[0][0][29]=A[1][0][28]=v_{11}, A[0][1][29]=A[2][1][59]=A[2][2][59]=$v_{11}+v_{22}$, A[1][2][28]=$v_{11}+v_{15}$,
A[0][1][32]=$v_{12}+v_{15}+v_{29}$, A[0][0][32]=A[2][1][62]=A[2][2][62]=A[1][0][31]=A[1][2][31]=v_{12},
A[0][0][62]=A[1][2][61]=v_{13}, A[0][1][62]=A[2][1][28]=A[2][2][28]=$v_{13}+v_{23}$, A[1][0][61]=$v_{13}+v_{24}$,
A[3][1][6]=A[3][2][6]=A[1][3][21]=A[0][1][25]=A[0][3][25]=v_{14},
A[3][1][13]=A[3][2][13]=$v_{15}+v_{29}$, A[1][3][28]=A[0][3][32]=v_{15},
A[3][1][59]=A[3][2][59]=$v_{16}+v_{26}$, A[1][3][10]=A[0][3][14]=v_{16}
A[1][3][2]=A[4][0][40]=A[4][2][40]=A[0][3][6]=v_{17},
A[1][0][7]=A[4][0][45]=A[4][2][45]=$v_{18}+v_{26}$, A[1][3][7]=A[0][3][11]=v_{18}
A[1][0][12]=A[1][3][12]=A[4][0][50]=A[4][2][50]=A[0][3][16]=v_{19},
A[1][3][15]=A[0][3][19]=v_{20}, A[4][0][53]=A[4][2][53]=$v_{20}+v_{27}$,
A[1][3][20]=A[0][3][24]=v_{21}, A[4][0][58]=A[4][2][58]=$v_{21}+v_{28}$,
A[1][0][25]=A[4][0][63]=A[4][2][63]=$v_{22}+v_{29}$, A[1][3][25]=A[0][3][29]=v_{22},
A[1][0][58]=A[1][3][58]=A[4][0][32]=A[4][2][32]=A[0][3][62]=v_{23},
A[1][3][61]=A[4][0][35]=A[4][2][35]=A[0][1][1]=A[0][3][1]=A[2][1][31]=A[2][2][31]=v_{24},
A[1][0][63]=A[1][3][63]=A[4][0][37]=A[4][2][37]=A[0][3][3]=v_{25},
A[1][2][7]=A[0][2][14]=v_{26}, A[0][2][22]=A[3][1][3]=A[3][2][3]=v_{27}
A[0][2][27]=A[3][1][8]=A[3][2][8]=v_{28}, A[1][2][25]=A[0][2][32]=v_{29}, A[2][0][55]=A[2][1][55]=v_{30}
A[0][2][60]=A[0][3][60]=v_{31}, A[0][1][37]=A[0][3][37]=v_{32}

Table 13. Conditional Cube Variables to Distinguish Keccak-224