Finding Ordinary Cube Variables for Keccak-MAC with Greedy Algorithm

Fukang Liu, Zhenfu Cao, and Gaoli Wang

Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, Shanghai, China
`liufukangs@163.com`

Abstract. In this paper, we introduce an alternative method to find ordinary cube variables for Keccak-MAC by making full use of the key-independent bit conditions. First, we select some potential candidates for ordinary cube variables by properly adding key-independent bit conditions, which do not multiply with the chosen conditional cube variables in the first two rounds. Then, we carefully determine the ordinary cube variables from the candidates to establish the conditional cube tester. Finally, based on our new method to recover the 128-bit key, the conditional cube attack on 7-round Keccak-MAC-128/256/384 is improved to 2^{71} and 6-round Keccak-MAC-512 can be attacked with at most 2^{40} calls to 6-round Keccak internal permutation. It should be emphasized that our new approach does not require sophisticated modeling. As far as we know, it is the first time to clearly reveal how to utilize the key-independent bit conditions to select ordinary cube variables for Keccak-MAC.

Keywords: hash function, Keccak, Keccak-MAC, ordinary cube variables, conditional cube attack

1 Introduction

In 2007, the U.S. National Institute of Standards and Technology (NIST) announced a public contest aiming at the selection of a new standard for a cryptographic hash function after Wang et al. made a break-through in MD-SHA hash family [14,15]. After five years of intensive scrutiny, Keccak was selected as the new SHA-3 standard [2].

Due to the low algebraic degree of a Keccak round, algebraic cryptanalysis has been deeply studied for Keccak, including cube attack [5], cube-attack-like cryptanalysis [3,5,11], conditional cube attack [8,9,12], linear structures for preimage attack [7], one/two/three-round connector for collision attack [4,10,13].

Recently, the application of cube attack on Keccak keyed mode has attracted researchers' interest and several results have been obtained [3,5,8,9,11,12]. Cube attack was first proposed by Dinur and Shamir at Eurocrypt 2009 [6], where a primitive is treated as a black-box polynomial in terms of plaintext and secret key. The first application of cube attack to Keccak keyed mode was presented at

Eurocrypt 2015 [5]. Two years later, at Eurocrypt 2017, Huang et al. introduced the conditional cube attack [8] on round-reduced Keccak keyed modes based on the pioneer work, i.e. cube attack [5,6] and cube tester [1]. Cube tester was first proposed by Aumasson et al. [1], aiming at detecting the non-random behaviour e.g. the cube sums are always equal to zero. Conditional cube tester detects a non-random behaviour (the cube sums are zero) only when some conditions hold. Therefore, once the key is involved in the conditions, conditional cube tester can be utilized to mount key-recovery attack. Indeed, conditional cube tester can be viewed as a key-dependent distinguisher.

At Eurocrypt 2017, Huang et al. firstly applied the conditional cube tester to mount key-recovery attack on 5/6/7-round Keccak-MAC-512/384/256 [8]. Later at Asiacrypt 2017, an MILP-based method [9] was proposed to identify good parameters for the conditional cube tester. Therefore, the conditional cube attack on Keccak-MAC-512/384 was extended by one more round. However, it seems that the modelling in [9] did not capture all factors influencing the performance of attack. Consequently, by taking more factors into consideration, Song et al. developed a new general MILP approach for Keccak-based primitives at Asiacrypt 2018 [12] and presented many applications. Despite that Song et al. claimed that 64-dimensional cube variables with only 2 key-dependent bit conditions were found, the details of the 64-dimensional cube variables were not reported in [12]. For the new modeling in [12], it seems sophisticated at the first glance. However, since more factors are taken into account, it is more general and powerful to mount new or improved attack on many Keccak-based constructions.

Due to the limited number of bits of Keccak-MAC-512 that can be controlled for an attacker, it is very difficult to find 64-dimensional cube variables under the conditional cube attack framework proposed by Huang et al. [8]. However, cube-attack-like cryptanalysis works quite well for Keccak-MAC-512 and attack on 7-round Keccak-MAC-512 was first achieved in [3], which was later slightly improved in [11].

Up until now, the improvement for [8] are all based on the MILP approach [9,12], which sometimes requires sophisticated modeling. This motivates us to consider whether there exist other simple approaches to find sufficient cube variables to establish the conditional cube tester.

Our Contributions In this paper, we present an alternative method to find ordinary cube variables for Keccak-MAC-512/384. First, we observe that there are many potentially useful key-independent conditions to slow down the propagation of ordinary cube variables, which will help determine the candidates for ordinary cube variables. Then, we introduce a clever way to choose the ordinary cube variables from the candidates by considering their relations in the first round. With such a method, sufficient ordinary cube variables can be discovered to establish the conditional cube tester for 6-round Keccak-MAC-512 and 7-round Keccak-MAC-384. Meanwhile, the number of key-dependent bit conditions is minimum. It should be stressed that we do not use any specific greedy algorithms but use the idea of greedy algorithm. Specifically, we determine a small region of potential candidates at first and then select the final candidates from these potential candidates. As far as we know, it is the first time to clearly reveal how to utilize the key-independent bit conditions to select ordinary cube variables for Keccak-MAC.

Moreover, we observe that there are many unnecessary iterations of the conditional cube tester to recover the full key in [8]. Therefore, an optimal procedure to recover the full key for 7-round Keccak-MAC-256/128 based on the conditional cube tester in [8] is proposed and the new key-recovery attack is twice faster. Such an optimal approach is applied to the newly discovered 64-dimensional cube variables for 7-round Keccak-MAC-384. Consequently, conditional cube attack on 7-round Keccak-MAC-384 is improved to 2^{71} from 2^{75}. By carefully choosing the order to recover the full key, we can recover the 128-bit key for 6-round Keccak-MAC-512 with at most 2^{40} calls to 6-round Keccak internal permutation, while it costs $\lceil\frac{128}{3}\rceil \times 2^{2^5+3} = \lceil\frac{128}{3}\rceil \times 2^{35} \approx 2^{40.4}$ calls in [12]. The results are summarized in Table 1.

Table 1. Related results of Keccak-MAC

Attack Type	Capacity	Rounds	Time	Ref.
Conditional Cube Attack	256/512	7	2^{72}	[8]
	768	7	2^{75}	[9]
	1024	6	$2^{40.4}$	[12]
	256/512/768	7	2^{71}	Sect. 4
	1024	6	2^{40}	Sect. 5
Cube-attack-like Cryptanalysis	1024	7	$2^{112.6}$	[3]
	1024	7	2^{111}	[11]

Organization The rest of the paper is organized as follows. The preliminaries of this paper will be presented in section 2. In section 3, our tracing algorithm will be introduced. Then, we will show our method to find enough ordinary cube variables for Keccak-MAC-384 and Keccak-MAC-512 in section 4 and section 5 respectively. Next, a slightly improved key-recovery method will be given in section 6. The difference between our work and previous work is explained in section 7. At last, we summarize the paper in section 8.

2 Preliminaries

In this section, we will introduce the details of Keccak-MAC and some related techniques such as cube tester and conditional cube tester.

2.1 Description of Keccak-MAC

Keccak is a family of hash functions and Keccak-MAC is based on Keccak. The Keccak internal permutations denoted by Keccak-p$[b, n_r]$ are specified by two parameters, which are the width of permutation in bits b and the number of rounds n_r. There are many choices for b, i.e. $b = 25\times 2^l$ with $l \in \{0, 1, 2, 3, 4, 5, 6\}$. Keccak-p$[b, n_r]$ works on a b-bit state A and iterates an identical round function $\mathbf{R}$ for n_r times. The state A can be viewed as a three-dimensional array of bits, namely $A[5][5][w]$ with $w = 2^l$. The expression $A[x][y][z]$ represents the bit with (x, y, z) coordinate. At lane level, $A[x][y]$ represents the w-bit word located at the x^{th} column and the y^{th} row. In this paper, the coordinates are considered within modulo 5 for x and y and within modulo w for z. The round function $\mathbf{R}$ consists of five operations $\mathbf{R} = \iota \circ \chi \circ \pi \circ \rho \circ \theta$ as follows.

$$\begin{aligned}
&\theta : A[x][y] = A[x][y] \oplus (\bigoplus_{y'=0}^{4} A[x-1][y']) \oplus (\bigoplus_{y'=0}^{4} (A[x+1][y'] \lll 1)).\\
&\rho : A[x][y] = A[x][y] \lll r[x, y].\\
&\pi : A[y][2x + 3y] = A[x][y].\\
&\chi : A[x][y] = A[x][y] \oplus (\overline{A[x+1][y]} \wedge A[x+2][y]).\\
&\iota : A[x][y] = A[x][y] \oplus RC.
\end{aligned}$$

According to the above definition of θ operation, 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 [2], this property is called **column parity kernel**, **CP kernel** for short.

The construction of Keccak-MAC-n is illustrated in Figure 1. For the sake of convenience, we denote the state A after θ, ρ, and π in round i ($i \geq 0$) by A^i_θ, A^i_ρ and A^i_π respectively. The input state of round i is denoted by A^i. The 128-bit key is denoted by k, where k_i represents the i^{th} bit of k.

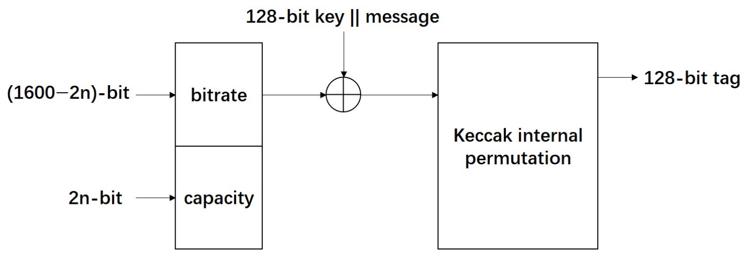

Fig. 1. Construction of Keccak-MAC-n

For Keccak-MAC-n, where $n \in \{128, 256, 384, 512\}$, the size of the internal state is 1600 bits and the 128-bit key is placed at $A^0[0][0]$ and $A^0[1][0]$. Specifically, k_i is placed at $A^0[0][0][i]$ and k_{i+64} is placed at $A^0[1][0][i]$, where $0 \leq i \leq 63$. Therefore, we can obtain Observation 1.

Observation 1 *Since*

$$A_\theta^0[3][i] = A^0[3][i] \oplus (\bigoplus_{y=0}^{4} A^0[2][y]) \oplus (\bigoplus_{y=0}^{4}(A^0[4][y] \lll 1))$$

for $0 \le i \le 4$, $A_\theta^0[3][i]$ *is independent of the 128-bit key. In other words, if we add bit conditions on* $A_\theta^0[3][i]$ *, all of them are key-independent.*

Then, we consider the influence of $\pi \circ \rho$ operation as shown in Figure 2. Consequently, Observation 2 can be obtained.

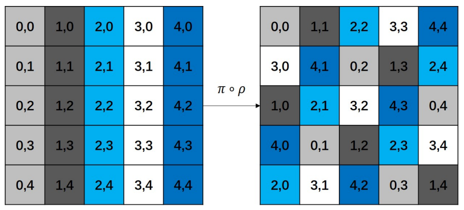

Fig. 2. $\pi \circ \rho$ operation

Observation 2 *After* $\pi \circ \rho$ *operation,* $A_\theta^0[2][i]$ *and* $A_\theta^0[4][t]$ *are next to* $A_\theta^0[3][j]$ *in each row, where* $(i, j, t) \in \{(2,3,4), (4,0,1), (1,2,3), (3,4,0), (0,1,2)\}$.

Our approach to determine the candidates for ordinary cube variables is heavily based on the two observations.

2.2 Cube Tester

Cube tester was first proposed by Aumasson et al. at FSE 2009 [1] after Dinur et al. introduced cube attack at Eurocrypt 2009 [6]. Different from standard cube attack, which aims at key extraction, cube tester performs non-randomness detection. In our paper, we only concentrate on a specific non-random behaviour, i.e. the cube sums are zero. To describe cube tester, we first recall the concept of cube attack as follows.

Theorem 2. [6] *Given a polynomial* $f : \{0,1\}^n \to \{0,1\}$ *of degree d. Suppose* $0 < k < d$ *and t denotes the monomial* $x_0 \ldots x_{k-1}$. *Then, f can be written as*

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

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

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

If there exists such a cube C_t that the following equation always hold, then C_t can be viewed as one type of cube tester [1], i.e. the sum over it always equals zero.

$$\sum_{x' \in C_t} f = \sum_{x' \in C_t} f(x', x_k, \ldots, x_{n-1}) = P_t(x_k, \ldots, x_{n-1}) = 0.$$

For example, consider the following polynomial f:

$$f(x_0, x_1, x_2, x_3) = x_0x_1 + x_1x_2 + x_2x_4 + x_1x_3 + x_1x_2x_4.$$

Then, the following equation always hold:

$$\sum_{(x_0, x_3) \in \{0,1\}^2} f(x_0, x_1, x_2, x_3) = 0.$$

The reason is that that none of the monomial in $f(x_0, x_1, x_2, x_3)$ is divisible by x_0x_3. However, if we sum f over all values of (x_1, x_2), then we can obtain the following equation:

$$\sum_{(x_1, x_2) \in \{0,1\}^2} f(x_0, x_1, x_2, x_3) = 1 + x_4.$$

That is, the sum is dependent on the value of x_4.

2.3 Conditional Cube Tester

The concept of conditional cube tester was firstly proposed by Huang et al. [8] at Eurocrypt 2017. Their goal is to construct a key-dependent distinguisher. Therefore, they have to overcome the obstacle of how to involve the key information into the distinguisher. Motivated by this, they firstly classify the cube variables into two types: conditional cube variable and ordinary cube variable. The classification is based on the multiplying relations of the cube variables in the first two rounds as follows.

- *Conditional cube variables can not multiply with each other after the second round.*
- *Ordinary cube variables can not multiply with each other after the first round.*
- *Ordinary cube variables can not multiply with conditional cube variables after the second round.*

Then, they develop a theorem to confirm the number of each type of the cube variables in order to establish a conditional cube tester, as specified below, whose proof is based on the relations of the cube variables in the first two rounds as above.

Theorem 1. [8] *For (n+2)-round Keccak sponge function ($n > 0$), if there are p conditional cube variables v_0, v_1, ..., v_{p-1} and $q = 2^{n+1} - 2p + 1$ ordinary* *cube variables v_p, v_{p+1}, ..., v_{p+q-1}, then the term $v_0v_1...v_{p+q-1}$ will not appear in the output polynomials of $(n+2)$-round Keccak sponge function.*

To make conditional cube tester work, it is essential to introduce some conditions which will influence the above multiplying relations between the conditional cube variable and ordinary cube variable in the first two round. Specifically, only when all the introduced conditions hold will their multiplying relations be satisfied, thus making **Theorem 1** work. Among the introduced conditions, the key-independent conditions can always be satisfied by controlling the input. For the key-dependent conditions, whether they are satisfied is detected by the conditional cube tester based on **Theorem 1**. To be more specific, the attack procedure can be briefly divided into three steps.

Step 1: Except for the cube variables in the input state, the attacker assigns a random value to the remaining part of it, while keeping the key-independent bit conditions satisfied.

Step 2: The attacker starts to exhaust all possible values of the cube variables and calculates the sum of all outputs.

Step 3: If the sum is zero, then the attacker knows that the key-dependent bit conditions are satisfied with an overwhelming probability and therefore he can extract some equations for the involved key bits. Otherwise, the attacker knows the key-dependent bit conditions do not hold. For this case, the attacker will flip some bits involved in the key-dependent bit conditions and located in the controllable part of the input state. Then, he goto Step 2 again.

The above procedure is only used to extract a small number of equations for the key bits. To recover the full key, the attacker will repeat the above procedure by changing the parameters of the conditions cube tester to extract more equations for the key bits. Finally, the attacker can solve the obtained equation system to recover some key bits. The remaining key bits can be recovered by brute force.

3 Tracing Algorithm

Several algorithms to determine the relations of cube variables in the first two rounds have been presented in [8]. In this section, we introduce a new method to achieve the same goal. We do not claim that our new method have any advantages over [8]. The purpose to use this new method is only to suit our programming. Before introducing how to determine the candidates for ordinary cube variables, we firstly describe how to trace the propagation of a variable in A^0_θ to A^1_π.

Since θ, ρ, π are all linear transformations, an equivalent linear transformation matrix $M \in F_2^{1600\times1600}$ can be derived to express these three consecutive operations $\pi\circ\rho\circ\theta$. From the definitions of the three operations, it can be known that for each row of M, there are only 11 non-zero elements, whose values are all 1. To reduce the size of M, we can only record the positions of M where the corresponding value is 1 in a smaller matrix SM of size 1600×11. Specifically, suppose $M[i][J] = 1$ ($J \in \{j_0, \ldots, j_{10}\}$), then we construct a smaller matrix SM where $SM[i][t] = j_t$ for $0 \leq t \leq 10$. Moreover, since the operation $\pi \circ \rho$ is equivalent to a permutation of bit positions, an equivalent permutation P of size 1600 can be derived to express it.

To make the tracing algorithm more explicit, we should consider the internal state as a boolean vector denoted by V rather than a three-dimensional array. In addition, assume the internal state is an 1600-bit variable. For other sizes of the internal state, the procedure to trace the propagation is similar. For the sake of convenience, we denote the state V after θ, ρ, and π in round i ($i \geq 0$) by V_θ^i, V_ρ^i and V_π^i respectively. The input state of round i is denoted by V^i.

Now we describe how to trace the propagation of the variable in A_θ^0 to A_π^1.

step 1. Suppose $A_\theta^0[x][y][z]$ contains a variable, we record $t_0 = (5x + y) \times 64 + z$.

step 2. Calculate how the variable in $V_\theta^0[t_0]$ propagates through $\pi \circ \rho$ operation with P. Consequently, we record $t_1 = P[t_0]$.

step 3. According to the definition of χ, after $\iota \circ \chi$ operation, three bits of V^1 will contain the variable from $V_\pi^0[t_1]$. We denote the corresponding three bit positions by t_2, t_3 and t_4. Among the three bits, one bit will always contain this variable. The other two bits contain this variable depending on bit conditions. We classify these three bits into three types. The first type is the bit that always contains the variable. The second type is the bit that contains the variable depending on a key-independent bit condition. The third type is the bit that contains the variable depending on a key-dependent bit condition. Then, for each of the three bits, we trace how the variable in $V^1[pos]$ ($pos \in \{t_2, t_3, t_4\}$) propagates to V_π^1 with Algorithm 1. The bit positions of V_π^1 containing the variable from $V^1[pos]$ are stored in the the array $finalPosition$.

Algorithm 1 Tracing the influenced bit positions after $\pi \circ \rho \circ \theta$ operation

```
1: for $row$ in (0...1599) do
2:     for $col$ in (0...10) do
3:         if $SM[row][col] = pos$ then
4:             $finalPosition.push\_back(row)$
5:             break
```

Up until now, the propagation of the variable in A_θ^0 to A_π^1 is known, i.e. the bit positions of A_π^1 containing the variable from A_θ^0 are known and are classified into three types. At last, we only need focus on how the cube variable in A^0 propagates to A_θ^0, which can be easily finished by considering the influence of θ operation.

Once knowing and recording how a variable propagates in the first two rounds with or without bit conditions to slow down this propagation, it is quite easy to determine their multiplying relations in the first two rounds. For example, suppose we know that $A_\pi^0[x][y][z]$ contains a variable v' and $A_\pi^0[x-1][y][z]$

contains a different variable v'', then v'' will multiply with v' after the first round. In the same way, suppose we know that $A_{\pi}^{1}[x][y][z]$ contains a variable v' and $A_{\pi}^{1}[x-1][y][z]$ contains a different variable v'', then v'' will multiply with v' after the second round.

4 Finding Ordinary Cube Variables for Keccak-MAC-384

In this section, we will expand on the procedure to find sufficient ordinary cube variables for Keccak-MAC-384. First, the potential candidates for ordinary cube variables will be determined by carefully adding key-independent bit conditions to slow down its propagation. Then, we consider the multiplying relations of these candidates after the first round and deduce some contradictions. As will be shown, from these contradictions, we can efficiently determine how many ordinary cube variables can eventually survive.

4.1 Determining Candidates for Keccak-MAC-384

The initial state of Keccak-MAC-384 is shown in Figure 3 with 12 lanes set to 0. In the same way as [8,9,12], $A[2][0][0] = A[2][1][0] = v_0$ is chosen as the conditional cube variable with four bit conditions ($A_{\theta}^{0}[1][4][60] = 1$, $A_{\theta}^{0}[1][0][5] = 1$, $A_{\theta}^{0}[3][1][7] = 0$, $A_{\theta}^{0}[3][2][45] = 0$) to slow down its propagation. Then, the ordinary cube variables are set in the CP kernel. The complete procedure is as follows.

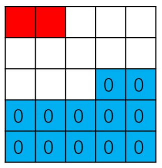

Fig. 3. Initial state of Keccak-MAC-384

- For the first column, we exhaust all 64 possible variables $A[0][1][i] = A[0][2][i]$ ($0 \leq i \leq 63$). Based on **Observation 1** and **2**, if we add bit conditions to slow down the propagation of the variables in this case, all of them are key-dependent bit conditions. Therefore, we do not add bit conditions. For each of these 64 possible variables, the tracing algorithm is applied to determine its multiplying relation with the chosen conditional cube variable in the first two rounds. Only those are selected as candidates that they do not multiply with v_0 in the first two rounds.
- For the second column, we exhaust all 64 possible variables $A[1][1][i] = A[1][2][i]$ ($0 \leq i \leq 63$) and process in the same way as the first column.

- For the third column, we exhaust 63×3 possible variables $A[2][0][i] = A[2][1][i]$, $A[2][0][i] = A[2][2][i]$ and $A[2][1][i] = A[2][2][i]$ $(1 \leq i \leq 63)$. Based on **Observation 1** and **2**, we can add key-independent bit conditions on $A_\theta^0[3][t]$ $(0 \leq t \leq 4)$ to slow down the propagation of the variables. To remove the redundant conditions, we add a condition only when it is necessary. In other words, if such a condition is not added and the variable satisfies the required relation with v_0 in the first two rounds, this condition is not necessary and redundant. Moreover, if such a condition is added, the variable still does not satisfy the requirement, we filter this variable.
- For the forth column, we exhaust all 64 possible variables $A[3][0][i] = A[3][1][i]$ $(0 \leq i \leq 63)$ and process in the same way as the first column since there are no key-independent bit conditions to slow the propagation of variables.
- For the fifth column, we exhaust 64 possible variables $A[4][0][i] = A[4][1][i]$ $(0 \leq i \leq 63)$. Based on **Observation 1** and **2**, we can add key-independent bit conditions to slow down the propagation of variables as the third column.

The candidates found with our method are presented in Table 2.

4.2 Discussion

Adding some bit conditions on $A_\theta^0[3][t]$ $(0 \leq t \leq 4)$ as described above will cause the following bad cases.

Case 1: Contradiction of conditions will occur. Specifically, for the third column, the bit condition on a certain bit i of $A_\theta^0[3][t_0]$ is $A_\theta^0[3][t_0][i] = 0$. However, for the fifth column, the bit condition on a certain bit j of $A_\theta^0[3][t_1]$ is $A_\theta^0[3][t_1][j] = 1$. If $i = j$ and $t_0 = t_1$, the contradiction of conditions is detected. In other words, we can not choose both of their corresponding variables as the final ordinary cube variables. Moreover, if $A_\theta^0[3][y_0][z_0]$ and $A_\theta^0[3][y_1][z_0]$ are added on different bit conditions for $y_0 > 1, y_1 > 1$, this is also a contradiction since $A[3][y][z_0]$ is set to a constant 0 for Keccak-MAC-384 for $y > 1$.

Case 2: Contradiction between conditions and ordinary cube variables will occur. Specifically, for the forth column, some of $A[3][0][i] = A[3][1][i]$ $(0 \leq i \leq 63)$ will be chosen as candidates. The bad case is that $A[3][0][t] = A[3][1][t]$ is chosen as a candidate and $A_\theta^0[3][0][t]$ or $A_\theta^0[3][1][t]$ is added on a condition.

Indeed, the second case can be processed in a simple way. After the candidates are determined, if a contradiction in the second case is detected, it implies that two ordinary variables multiplies with each other in the first round. For example, supposing $A_\theta^0[3][0][t]$ is added on a condition and $A[3][0][t] = A[3][1][t]$ is chosen as a candidate, it implies a variables set in $A[2][4]$ or $A[4][1]$ is chosen as a candidate, which will multiply with the variable set to $A[3][0][t]$ after the first round. This can be seen from the $\pi \circ \rho$ operation in Figure 2. Thus, the second case is equivalent to the case that two ordinary cube variables multiply with

Table 2. Candidates for Keccak-MAC-384, where c is an adjustable constant over GF(2) for each variable.

$A[0][1][i] = A[0][2][i] + c$									
i	15	22	28	34	37	46	47	58	59
Variable	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9

$A[1][1][i] = A[1][2][i] + c$											
i	7	15	20	26	30	38	39	40	52	54	57
Variable	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}	v_{17}	v_{18}	v_{19}	v_{20}

$A[2][0][i] = A[2][1][i] + c$																			
i	1	8	12	14	15	20	23	25	28	41	42	43	45	50	52	53	61	62	63
Variable	v_{21}	v_{22}	v_{23}	v_{24}	v_{25}	v_{26}	v_{27}	v_{28}	v_{29}	v_{30}	v_{31}	v_{32}	v_{33}	v_{34}	v_{35}	v_{36}	v_{37}	v_{38}	v_{39}
Condition	i=1: $A_\theta^0[3][2][46] = 0$ i=15: $A_\theta^0[3][1][22] = 0$ i=25: $A_\theta^0[3][1][32] = 0$ i=50: $A_\theta^0[3][2][31] = 0$ i=63: $A_\theta^0[3][1][6] = 0$, $A_\theta^0[3][2][44] = 0$										i=14: $A_\theta^0[3][1][21] = 0$ i=23: $A_\theta^0[3][2][4] = 0$ i=42: $A_\theta^0[3][1][49] = 0$ i=52: $A_\theta^0[3][1][59] = 0$								

$A[3][0][i] = A[3][1][i] + c$													
i	3	4	9	13	15	23	30	35	39	40	46	56	57
Variable	v_{40}	v_{41}	v_{42}	v_{43}	v_{44}	v_{45}	v_{46}	v_{47}	v_{48}	v_{49}	v_{50}	v_{51}	v_{52}

$A[4][0][i] = A[4][1][i] + c$																			
i	3	5	8	10	12	14	20	22	25	30	31	35	38	41	47	57	58	62	63
Variable	v_{53}	v_{54}	v_{55}	v_{56}	v_{57}	v_{58}	v_{59}	v_{60}	v_{61}	v_{62}	v_{63}	v_{64}	v_{65}	v_{66}	v_{67}	v_{68}	v_{69}	v_{70}	v_{71}
Condition	i=3: $A_\theta^0[3][0][59] = 1$ i=20: $A_\theta^0[3][0][12] = 1$ i=25: $A_\theta^0[3][0][17] = 1$ i=35: $A_\theta^0[3][4][6] = 1, A_\theta^0[3][0][27] = 1$ i=41: $A_\theta^0[3][0][33] = 1$										i=8: $A_\theta^0[3][0][0] = 1$ i=22: $A_\theta^0[3][0][14] = 1$ i=30: $A_\theta^0[3][4][1] = 1, A_\theta^0[3][0][22] = 1$ i=38: $A_\theta^0[3][4][9] = 1$ i=57: $A_\theta^0[3][0][49] = 1$								

$A[2][0][i] = A[2][2][i] + c$																					
i	1	5	6	14	15	16	20	21	27	30	33	38	39	40	41	46	51	52	57	61	62
Variable	v_{72}	v_{73}	v_{74}	v_{75}	v_{76}	v_{77}	v_{78}	v_{79}	v_{80}	v_{81}	v_{82}	v_{83}	v_{84}	v_{85}	v_{86}	v_{87}	v_{88}	v_{89}	v_{90}	v_{91}	v_{92}
Condition	i=1: $A_\theta^0[3][3][23] = 0$ i=15: $A_\theta^0[3][1][22] = 0$ i=30: $A_\theta^0[3][1][37] = 0$ i=38: $A_\theta^0[3][1][45] = 0$ i=46: $A_\theta^0[3][1][53] = 0$ i=57: $A_\theta^0[3][1][0] = 0$										i=14: $A_\theta^0[3][1][21] = 0, A_\theta^0[3][3][36] = 0$ i=20: $A_\theta^0[3][3][42] = 0$ i=33: $A_\theta^0[3][3][55] = 0$ i=40: $A_\theta^0[3][1][47] = 0$ i=52: $A_\theta^0[3][1][59] = 0$ i=62: $A_\theta^0[3][3][20] = 0$										

$A[2][1][i] = A[2][2][i] + c$														
i	1	11	14	15	18	19	20	24	41	52	56	58	61	62
Variable	v_{93}	v_{94}	v_{95}	v_{96}	v_{97}	v_{98}	v_{99}	v_{100}	v_{101}	v_{102}	v_{103}	v_{104}	v_{105}	v_{106}
Condition	i=1: $A_\theta^0[3][2][46] = 0, A_\theta^0[3][3][23] = 0$ i=18: $A_\theta^0[3][2][63] = 0$ i=56: $A_\theta^0[3][3][14] = 0$									i=14: $A_\theta^0[3][3][36] = 0$ i=20: $A_\theta^0[3][3][42] = 0$ i=62: $A_\theta^0[3][3][20] = 0$				

each other in the first round. Benefiting from this new property, we do not have to process the second bad case and only need concentrate on the relation of the candidates in the first round as well as the contradiction caused by conditions.

4.3 Deducing Contradictions

The contradictions of candidates are deduced from two cases. The first case is that variables multiply with each other in the first round. The second case is that there is contradiction of conditions. The contradictions deduced are displayed in Table 3. In this table, $v_i\{v_{j_0},...,v_{j_n}\}$ means v_i can not be chosen with any of $\{v_{j_0},...,v_{j_n}\}$ as the final candidates at the same time. We count the times that each variable appears in these contradictions and do not choose the one which appears more than one time as marked in red and blue. However, although some variables appear two times as marked in green in this table, we can still choose them. Therefore, for the obtained contradictions, at most 28 variables can be derived. Moreover, there are 56 fully free variables, i.e. there are no contradictions on them.

Table 3. Contradictions of candidates

$v_1\{v_{70}\}$	$v_2\{v_{54}, v_{63}\}$	$v_3\{v_{19}\}$	$v_5\{v_{59}\}$	$v_7\{v_{62}\}$
$v_8\{v_{12}, v_{53}, v_{66}\}$	$v_{11}\{v_{77}\}$	$v_{12}\{v_{79}\}$	$v_{13}\{v_{80}\}$	$v_{15}\{v_{84}\}$
$v_{16}\{v_{85}\}$	$v_{17}\{v_{86}, v_{101}\}$	$v_{20}\{v_{104}\}$	$v_{22}\{v_{44}\}$	$v_{27}\{v_{46}\}$
$v_{29}\{v_{47}\}$	$v_{34}\{v_{52}\}$	$v_{37}\{v_{41}\}$	$v_{41}\{v_{57}, v_{91}\}$	$v_{43}\{v_{74}\}$
$v_{45}\{v_{63}, v_{77}\}$	$v_{46}\{v_{65}\}$	$v_{48}\{v_{67}\}$	$v_{49}\{v_{82}\}$	$v_{50}\{v_{84}\}$

Observe that we consider the third column under three cases, which will cause two problems. Specifically, if $A[2][0][t] = A[2][1][t] + c$, $A[2][0][t] = A[2][2][t] + c$ and $A[2][1][t] = A[2][2][t] + c$ are chosen simultaneously, only two variables rather than three variables can be obtained. In this case, we should change the variables as $A[2][0][t] = v_{x_0}, A[2][1][t] = v_{x_1}, A[2][2][t] = v_{x_0} + v_{x_1} + c$. This is due to that the ordinary cube variables are set in the CP kernel. According to Table 2, there are 8 possible values for t and they are $\{1, 14, 15, 20, 41, 52, 61, 62\}$. Therefore, for the worst case, we can finally obtain $28 + 56 - 8 = 76$ ordinary cube variables, which is much larger than the required number (63) to mount key-recovery attack on 7-round Keccak-MAC-384.

On the other hand, if two of $A[2][0][t] = A[2][1][t] + c$, $A[2][0][t] = A[2][2][t] + c$, $A[2][1][t] = A[2][2][t] + c$ are chosen simultaneously, we should change the variables as $A[2][0][t] = v_{x_0}, A[2][1][t] = v_{x_1}, A[2][2][t] = v_{x_0} + v_{x_1} + c$.

One choice of the 64-dimensional cube variables to establish the conditional cube tester is displayed in Table 4.

Table 4. One choice of ordinary cube variables for Keccak-MAC-384

Free ordinary cube variables (56-6=50 in total)	v_4, v_6, v_9, v_{10}, v_{14}, v_{18}, v_{21}, v_{23}, v_{24}, v_{25}, v_{26}, v_{28}, v_{30}, v_{31}, v_{32}, v_{33}, v_{35}, v_{36}, v_{38}, v_{39}, v_{40}, v_{42}, v_{51}, v_{55}, v_{56}, v_{58}, v_{60}, v_{61}, v_{64}, v_{68}, v_{69}, v_{71}, v_{72}, v_{73}, v_{75}, v_{76}, v_{78}, v_{81}, v_{83}, v_{87}, v_{88}, v_{89}, v_{90}, v_{92}, v_{93}, v_{94}, v_{95}, v_{96}, v_{97}, v_{98}, v_{99}, v_{100}, v_{102}, v_{103}, v_{105}, v_{106}. $\{v_{21}, v_{72}, v_{93}\}$, $\{v_{24}, v_{75}, v_{95}\}$, $\{v_{25}, v_{76}, v_{96}\}$ $\{v_{26}, v_{78}, v_{99}\}$, $\{v_{35}, v_{89}, v_{102}\}$ and $\{v_{38}, v_{92}, v_{106}\}$ provide two variables respectively.
Ordinary cube variables derived from contradictions (13 in total)	v_1, v_{54}, v_{63}, v_3, v_5, v_7, v_{53}, v_{66}, v_{11}, v_{79}, v_{13}, v_{15}, v_{16}
Conditional cube variable	v_0
Key-dependent conditions	$A_\theta^0[1][4][60] = 1, A_\theta^0[1][0][5] = 1.$
Key-independent conditions for v_0	$A_\theta^0[3][1][7] = 0, A_\theta^0[3][2][45] = 0.$
Other key-independent conditions for ordinary cube variables	Refer to Table 3 according to the chosen variables.

5 Finding Ordinary Cube Variables for Keccak-MAC-512

Although 32-dimensional cube variables have been found with MILP to establish the 6-round conditional cube tester for Keccak-MAC-512 and the time complexity is practical, we want to explain how to apply our method to achieve the same goal. This is for a better understanding of the differences between our method and others based on MILP. Now, we expand on how to find sufficient cube variables for Keccak-MAC-512.

In a similar way for Keccak-MAC-384, 32 candidates for ordinary cube variables are discovered as displayed in Table 5. The corresponding contradictions are as follows.

$$v_2\{v_{24}\},\ v_7\{v_{26}\},\ v_9\{v_{27}\},\ v_{14}\{v_{32}\},\ v_{17}\{v_{21}\}.$$

Therefore, there will be $32 - 5 = 27$ possible ordinary cube variables in total if the ordinary cube variables are set only in the CP kernel. As a result, we can not mount key-recovery attack on 6-round Keccak-MAC-512, which requires 31 ordinary cube variables if only v_0 is chosen to be the conditional cube variable.

Based on [12], the variables which multiply with v_0 only in the second round can be leveraged as well. For an intuitive example, suppose one variable v_{x_0} multiplies with v_0 only in the second round and the multiplying bit position is p_0. If another variable v_{x_1} multiplies with v_0 only in the second round and the multiplying bit position is p_0 as well, then setting $v_{x_0} = v_{x_1}$ will cause the already filtered two variables become one possible variable. Then, the goal becomes how to find these possible variables.

Suppose $A_\theta^0[i][j][k]$ contains a variable, then after χ operation, three bits will contain this variable. Based on the definition of χ operation, among the three

Table 5. Candidates for Keccak-MAC-512, where c is an adjustable constant over GF(2) for each variable.

$A[2][0][i] = A[2][1][i] + c$																			
i	1	8	12	14	15	20	23	25	28	41	42	43	45	50	52	53	61	62	63
Variable	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}	v_{17}	v_{18}	v_{19}
Condition	i=1: $A^0_\theta[3][2][46] = 0$ i=15: $A^0_\theta[3][1][22] = 0$ i=25: $A^0_\theta[3][1][32] = 0$ i=50: $A^0_\theta[3][2][31] = 0$ i=63: $A^0_\theta[3][1][6] = 0, A^0_\theta[3][2][44] = 0$														i=14: $A^0_\theta[3][1][21] = 0$ i=23: $A^0_\theta[3][2][4] = 0$ i=42: $A^0_\theta[3][1][49] = 0$ i=52: $A^0_\theta[3][1][59] = 0$				

$A[3][0][i] = A[3]1][i] + c$													
i	3	4	9	13	15	23	30	35	39	40	46	56	57
Variable	v_{20}	v_{21}	v_{22}	v_{23}	v_{24}	v_{25}	v_{26}	v_{27}	v_{28}	v_{29}	v_{30}	v_{31}	v_{32}

bits, one bit will always contain this variable and the other two bits contain this variable depending on the conditions. We classify the three bits into three types.

Type-1: It always contains this variable.
Type-2: It contains this variable depending on a key-independent bit condition.
Type-3: It contains this variable depending on a key-dependent bit condition.

Then, we trace how the three bits propagate to the second round with the tracing algorithm. Specifically, we trace the **Type-1** bit and record the influenced bits of A^1_π multiplying with v_0 in the second round. For the **Type-2** and **Type-3** bits, we process in the same way. The recorded bits for **Type-1**, **Type-2** and **Type-3** are defined as core bits, independent-key bits and key-dependent bits. Since our focus is the minimal key-dependent conditions, once the key-dependent bits are detected, the corresponding variable should not be chosen as a candidate.

With the above method, we reconsider the filtered ordinary cube variables set in the CP kernel. Besides, the variables set to a single bit are also considered. The final result obtained is displayed in Table 6.

For a better understanding of this table, we take the variable $A[3][1][8]$ as instance. For the first column, it means $A[3][1][8]$ is set to be a variable. For the second column, it means 5 bits of A^1_π will multiply with v_0 only in the second round. For the third column, $\{656,1003\}$ means the two bits of A^1_π, i.e. $A^1_\pi[0][2][16]$ and $A^1_\pi[0][3][43]$, will multiply with v_0 only in the second round depending on the same key-independent bit condition. The last column means $A[3][1][8]$ can not be chosen as a variable with any of v_1 and v_{31} in Table 5 simultaneously.

According to Table 6, at most three more possible ordinary cube variables can be obtained. One choice is as follows:

$$\begin{aligned}
&A[3][0][58] = A[3][1][58] = A[2][0][24] = A[2][1][24] = v_{e_0},\\
&A[3][0][61] = v_{e_1}, A[3][1][61] = v_{e_2},\\
&A[2][0][26] = A[2][1][26] = v_{e_3}, v_{e_3} = v_{e_2} + v_{e_1}\\
&A[2][0][46] = A[2][1][46] = v_{e_2}.\\
&\text{Condition}: A^0_\theta[3][3][20] = 0, A^0_\theta[3][4][21] = 0, A^0_\theta[3][1][53] = 0.
\end{aligned}$$

According to Table 6, adding $A[2][0][37] = A[2][1][37] = v_{e_2}$ to the above variables and converting the bit condition $A_\theta^0[3][1][53] = 0$ into $A_\theta^0[3][1][53] = 1$ is also possible. However, it can not help improve the number of possible variables. In fact, there are many interesting cases. For example, if $A[3][0][60] = A[3][1][60]$ does not multiply with v_{16} in the first round, we can obtain one more candidate. For the third row, if {652,1109} does not depend on the same condition, then we can add one key-independent bit condition to prevent the propagation to the 652-nd bit and another key-independent bit condition to allow the propagation to the 1109-th bit of A_π^1.

Table 6. Possible candidates for Keccak-MAC-512

Possible Variables	Core Bits	Key-independent Bits	Contradictions
$A[2][0][4] = A[2][1][4]$	1540		
$A[2][0][5] = A[2][1][5]$	1109	{652,1109}	
$A[2][0][9] = A[2][1][9]$	848,467	{656,1003}	
$A[2][0][13] = A[2][1][13]$	652,1109		
$A[2][0][16] = A[2][1][16]$	1472	515	v_{25}
$A[2][0][24] = A[2][1][24]$	515		
$A[2][0][26] = A[2][1][26]$	665		
$A[2][0][29] = A[2][1][29]$	71,1032	241	
$A[2][0][33] = A[2][1][33]$	491		v_{29}
$A[2][0][35] = A[2][1][35]$	1131,42	1242	
$A[2][0][37] = A[2][1][37]$	1040		
$A[2][0][46] = A[2][1][46]$	903	1040	
$A[2][0][51] = A[2][1][51]$	767,1160		
$A[2][0][54] = A[2][1][54]$	1510		
$A[2][0][57] = A[2][1][57]$	170	205	
$A[2][0][60] = A[2][1][60]$	1280	1540	v_{20}
$A[3][0][41] = A[3][1][41]$	113		
$A[3][0][43] = A[3][1][43]$	848		
$A[3][0][50] = A[3][1][50]$	42		v_{12}
$A[3][0][58] = A[3][1][58]$	515		
$A[3][0][60] = A[3][1][60]$	665		v_{16}
$A[3][0][61] = A[3][1][61]$	903		
$A[3][1][8]$	170,848,467,1382,1003	{656,1003},{903},{1237}	v_1, v_{31}
$A[3][0][32]$	491,903,1382	{13},{848},{775}	v_{29}
$A[3][0][61]$	665	{42},{1348}	
$A[3][1][61]$	903,665	{42},{1348}	

Then we test whether v_{e_i} $(0 \leq i \leq 3)$ multiplies with each other in the first round and check whether the three bit conditions to slow down the propagation of v_{e_1} and v_{e_2} are contradict with the conditions in Table 5. It is shown that the three variables are all valid. Therefore, we can obtain at most $32 - 5 + 3 = 30$ ordinary cube variables without key-dependent bit conditions. It reveals in a way why [12] can only discover the same number of such ordinary variables with a solver. However, to mount key-recovery attack on 6-round Keccak-MAC-512, we need 31 ordinary cube variables. Thus, we try to search ordinary cube variables set in the CP kernel with only one key-dependent bit condition, which satisfy the required relation with v_0 and the chosen $32 + 4 = 36$ candidates for ordinary cube variables. Our searching result is displayed in Table 7. Thus, there are many possible choices for 31 ordinary cube variables, i.e. at least $2^5 \times 12$. The verification can be found at `https://github.com/Crypt-CNS/Keccak_ConditionalCubeAttack.git`.

Table 7. Candidates for Keccak-MAC-512 with one key-dependent bit condition

Variable	Conditions
$A[2][0][11] = A[2][1][11]$	$A^0_\theta[1][4][7] = 1$
$A[2][0][19] = A[2][1][19]$	$A^0_\theta[1][4][15] = 1$
$A[2][0][21] = A[2][1][21]$	$A^0_\theta[1][0][26] = 1$, $A^0_\theta[3][2][2] = 0$
$A[2][0][22] = A[2][1][22]$	$A^0_\theta[1][0][27] = 1$
$A[2][0][30] = A[2][1][30]$	$A^0_\theta[3][1][37] = 0$, $A^0_\theta[1][0][35] = 1$
$A[2][0][34] = A[2][1][34]$	$A^0_\theta[1][0][39] = 1$, $A^0_\theta[3][2][15] = 0$
$A[2][0][44] = A[2][1][44]$	$A^0_\theta[3][1][51] = 0$, $A^0_\theta[1][0][49] = 1$
$A[2][0][56] = A[2][1][56]$	$A^0_\theta[1][4][52] = 1$, $A^0_\theta[3][1][63] = 0$
$A[3][0][12] = A[3][1][12]$	$A^0_\theta[4][1][20] = 0$
$A[3][0][20] = A[3][1][20]$	$A^0_\theta[4][2][36] = 0$
$A[3][0][29] = A[3][1][29]$	$A^0_\theta[2][4][60] = 1$
$A[3][0][34] = A[3][1][34]$	$A^0_\theta[2][4][1] = 1$

6 Recovering Full Key

In this section, a new slightly improved way to recover 128-bit key for Keccak-MAC is presented by removing unnecessary iterations of conditional cube tester. In [8], 64 iterations of the conditional cube tester were used to recover the 128-bit key for Keccak-MAC-256. For each iteration, it costs $2^{64+2} = 2^{66}$ to recover 2-bit key. Observe that once there are only a few key bits to be recovered, there is no need to iterate the conditional cube tester since each iteration is costly and only 2 bits are recovered.

Taking Keccak-MAC-128/256 for instance, for the 64-dimensional cube variable [8], after 31 iterations in z-axis of the conditional cube tester, 62 bits of key can be recovered. Then, the remaining 66 bits can be recovered by brute force. Therefore, the time complexity is improved to $2^{66} \times 31 + 2^{66} = 2^{71}$ from 2^{72}. Similarly, for the 64-dimensional cube variables in Table 4, we can recover the 128-bit key for 7-round Keccak-MAC-384 with time complexity 2^{71}.

For the conditional cube attack on 6-round Keccak-MAC-512, we choose $A[2][0][11] = A[2][1][11]$ in Table 7 as the ordinary cube variable with one key-dependent bit condition $A^0_\theta[1][4][7] = 1$, while $A[2][0][19] = A[2][1][19]$ is chosen in [12]. For our choice, only 31 iterations in z-axis is enough. Then, $3 \times 31 = 93$ bits can be recovered with time complexity $2^{32+3} \times 31 = 2^{35} \times 31$. The remaining $128 - 93 = 35$ bits can be recovered by brute force. The order to recover 93 bits of key with conditional cube tester is shown in Table 8. Therefore, the total time complexity becomes $2^{35} \times 31 + 2^{35} = 2^{40}$. However, the time complexity is estimated as $\lceil \frac{128}{3} \rceil \times 2^{2^5+3} = \lceil \frac{128}{3} \rceil \times 2^{35} = 2^{40.4}$ in [12], which implies 64 iterations of the conditional cube tester are used to recover the 128-bit key.

Table 8. The order to recover 93 bits of key with conditional cube tester

$(k_0, k_{53}, k_{62} \oplus k_{126})$, $(k_1, k_{54}, k_{63} \oplus k_{127})$, $(k_2, k_{55}, k_0 \oplus k_{64})$, $(k_3, k_{56}, k_1 \oplus k_{65})$,
$(k_4, k_{57}, k_2 \oplus k_{66})$, $(k_5, k_{58}, k_3 \oplus k_{67})$, $(k_6, k_{59}, k_4 \oplus k_{68})$, $(k_7, k_{60}, k_5 \oplus k_{69})$,
$(k_8, k_{61}, k_6 \oplus k_{70})$, $(k_9, k_{62}, k_7 \oplus k_{71})$, $(k_{10}, k_{63}, k_8 \oplus k_{72})$, $(k_{22}, k_{11}, k_{20} \oplus k_{84})$,
$(k_{23}, k_{12}, k_{21} \oplus k_{85})$, $(k_{24}, k_{13}, k_{22} \oplus k_{86})$, $(k_{25}, k_{14}, k_{23} \oplus k_{87})$, $(k_{26}, k_{15}, k_{24} \oplus k_{88})$,
$(k_{27}, k_{16}, k_{25} \oplus k_{89})$, $(k_{28}, k_{17}, k_{26} \oplus k_{90})$, $(k_{29}, k_{18}, k_{27} \oplus k_{91})$, $(k_{30}, k_{19}, k_{28} \oplus k_{92})$,
$(k_{31}, k_{20}, k_{29} \oplus k_{93})$, $(k_{32}, k_{21}, k_{30} \oplus k_{94})$, $(k_{44}, k_{33}, k_{42} \oplus k_{106})$, $(k_{45}, k_{34}, k_{43} \oplus k_{107})$,
$(k_{46}, k_{35}, k_{44} \oplus k_{108})$, $(k_{47}, k_{36}, k_{45} \oplus k_{109})$, $(k_{48}, k_{37}, k_{46} \oplus k_{110})$, $(k_{49}, k_{38}, k_{47} \oplus k_{111})$,
$(k_{50}, k_{39}, k_{48} \oplus k_{112})$, $(k_{51}, k_{40}, k_{49} \oplus k_{113})$, $(k_{52}, k_{41}, k_{50} \oplus k_{114})$.

7 Comparison with Previous Work

Our work is heavily based on [8]. However, Huang et al. did not consider the potentially useful key-independent bit conditions to slow down the propagation of ordinary cube variables [8].

As for [9], it seems that the key-independent bit conditions have been considered. However, it is strange that Li et al. found 63 ordinary cube variables with 6 key-dependent bit conditions for Keccak-MAC-384, while we can find much more ordinary cube variables without key-dependent bit conditions, i.e. at least 76 variables. Besides, Li et al. only found 25 ordinary cube variables set in the CP kernel for Keccak-MAC-512, while we can find $32 - 5 = 27$ ordinary cube variables set in the CP kernel. Therefore, we guess that the key-independent bit conditions were not fully leveraged in [9].

As for [12], minimum key-dependent bit conditions is considered in the model. In that paper, one instance of 31 ordinary cube variables for Keccak-MAC-512 was presented, which is almost the same with what we found. However, it is strange that there are 18 key-independent bit conditions to slow down the propagation of the ordinary cube variables. With our approach, there are at most $10 + 3 + 1 = 14$ key-independent bit conditions for ordinary cube variables. If we choose the same cube variables as [12], only $9 + 3 = 12$ key-independent bit conditions are sufficient. Indeed, we can reach the minimum key-independent bit conditions, which is $8 + 3 = 11$. Thus, we guess the redundancy in key-independent bit conditions are not well processed in the modeling in [12]. It should be noted that the redundancy of key-independent bit conditions will not affect the time complexity to recover the key. However, from the scientific point, if there is a more accurate answer, why not choose it?

In addition, a new slightly improved approach to recover the 128-bit key is introduced. This is based on the observation that many iterations of the conditional cube tester are costly once a few bits of key are left. Consequently, we improve the conditional cube attack on 7-round Keccak-MAC-128/256/384 and 6-round Keccak-MAC-512.

8 Conclusion

An algorithm to search ordinary cube variables for Keccak-MAC is developed. The first step is to identify a small region of potential candidates by making full use of the key-independent bit conditions. Then, these candidates are further filtered according to their relations after the first round with an efficient approach. In this way, sufficient ordinary cube variables can be discovered to establish the conditional cube tester. Combined with the new slightly improved way to recover the key, the time complexity of the conditional cube attack on 7-round Keccak-MAC-128/256/384 and 6-round Keccak-MAC-512 are improved to 2^{71} and 2^{40} respectively.

Acknowledgement We thank the anonymous reviewers of IWSEC 2019 for their insightful comments and suggestions. Fukang Liu and Zhenfu Cao are supported by National Natural Science Foundation of China (Grant No.61632012, 61672239). Gaoli Wang is supported by the National Natural Science Foundation of China (No. 61572125) and National Cryptography Development Fund (No. MMJJ20180201).

References

1. Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, pages 1–22, 2009.
2. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, 2011. `http://keccak.noekeon.org`.
3. Wenquan Bi, Xiaoyang Dong, Zheng Li, Rui Zong, and Xiaoyun Wang. MILP-aided cube-attack-like cryptanalysis on Keccak keyed modes. Cryptology ePrint Archive, Report 2018/075, 2018. `https://eprint.iacr.org/2018/075`.
4. Itai Dinur, Orr Dunkelman, and Adi Shamir. New attacks on Keccak-224 and Keccak-256. In *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, pages 442–461, 2012.
5. 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 *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, pages 733–761, 2015.

6. Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 278–299, 2009.
7. Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced Keccak. In *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*, pages 249–274, 2016.
8. Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional cube attack on reduced-round Keccak sponge function. In *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*, pages 259–288, 2017.
9. Zheng Li, Wenquan Bi, Xiaoyang Dong, and Xiaoyun Wang. Improved conditional cube attacks on Keccak keyed modes with MILP method. In *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*, pages 99–127, 2017.
10. Kexin Qiao, Ling Song, Meicheng Liu, and Jian Guo. New collision attacks on round-reduced Keccak. In *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*, pages 216–243, 2017.
11. Ling Song and Jian Guo. Cube-attack-like cryptanalysis of round-reduced Keccak using MILP. *IACR Trans. Symmetric Cryptol.*, 2018(3):182–214, 2018.
12. Ling Song, Jian Guo, Danping Shi, and San Ling. New MILP modeling: Improved conditional cube attacks on Keccak-based constructions. In *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*, pages 65–95, 2018.
13. Ling Song, Guohong Liao, and Jian Guo. Non-full sbox linearization: Applications to collision attacks on round-reduced Keccak. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 428–451, 2017.
14. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 17–36, 2005.
15. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 19–35, 2005.