Finding Ordinary Cube Variables for Keccak-MAC with Greedy Algorithm

Fukang Liu

Shanghai Key Laboratory of Trustworthy Computing, School of Computer Science and Software Engineering, 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 key-independent bit conditions. Firstly, we choose some good candidates for ordinary cube variables with key-independent conditions, which do not multiply with the chosen conditional cube variables in the first two rounds. Then, we construct inequalities of these candidates by considering their relations after one round. In this way, we can greatly reduce the number of inequalities and therefore can solve them without a solver. Based on our new way to recover the 128-bit key, the conditional cube attack on 7-round Keccak-MAC-128/256/384 is improved to 2^{71} and we can attack 6-round Keccak-MAC-512 with at most 2^{40} calls of 6-round Keccak internal permutation.

Keywords: 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 [3,5,11,16], conditional cube attack [8,9,12], linear structures for preimage attack [7], one/two/three-round connector for collision attack [4,10,13].

At Eurocrypt 2017, Huang et al. presented the conditional cube attack [8] on round-reduced Keccak keyed modes based on the pioneer work, i.e., cube attack [6,5] and cube tester [1]. Cube tester was first proposed by Aumasson et al. [1], which aims 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 used to mount key-recovery attack. In fact, conditional cube tester can be viewed as a key-dependent distinguisher. To make the conditional cube tester work, Huang et al. developed a theorem and defined two types cube variables as conditional cube variables and ordinary cube variables based on their relations in the first two rounds. Specifically, the relations are explained as follows.

1. *Conditional cube variables can not multiply with each other in the first two rounds.*
2. *Ordinary cube variables can not multiply with each other in the first round.*
3. *Ordinary cube variables can not multiply with conditional cube variables in the first two rounds.*

The theorem to help confirm the number of each type of the cube variables in order to establish a conditional cube tester is specified as follows, whose proof is much 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 $v_0 v_1 ... v_{p+q-1}$ will not appear in the output polynomials of $(n+2)$-round Keccak sponge function.*

Based on the new discovery, they successfully mounted key-recovery attack 5/6/7-round Keccak-MAC-512/384/256 by establishing a conditional cube tester with $p = 1$. The reason why they could not reach more rounds for Keccak-MAC-512/384 was that they could not find enough ordinary cube variables. Later, an MILP-based method was proposed at Asiacrypt 2017 to find more ordinary cube variables for Keccak-MAC-512/384 [9]. However, there are too many key-dependent conditions used to slow down the propagation of the ordinary cube variables in [9], thus making the time complexity of the key-recovery attack not optimal. In order to reduce the key-dependent conditions, [12] developed a new MILP method to find enough cube variables with as few key-dependent conditions as possible. The modeling in [12] seems sophisticated at the first glance. However, it is quite 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. 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].

In a word, the original method to find enough ordinary cube variables with greedy algorithm in [8] is developed as an MILP problem by Li et al. [9] and Song et al. [12] so as to find more cube variables, which is much based on the modeling and some mathematical tools. In this paper, we present a straightforward and simple method to find comparably enough ordinary cube variables for Keccak-MAC-512/384 by considering the potentially useful key-independent bit conditions.

Outline. A brief introduction of Keccak internal permutation and the construction of Keccak-MAC-n will be presented in Section 2. Then, we will show our method to find enough ordinary cube variables for Keccak-MAC-384 and Keccak-MAC-512 in Section 3 and Section 4 respectively. Next, a slightly improved key-recovery method will be given in Section 5. The difference between our work and previous work is explained in Section 6. At last, we summarize the paper in Section 7.

1.1 Our Contributions

In this paper, we present an alternative method to find ordinary cube variables for Keccak-MAC-512/384. Firstly, 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 construct inequalities of the candidates and solve them based on some observation rather than a solver since their scale is small. Of course, they can be solved using a solver as well. With this method, enough ordinary cube variables without key-dependent bit conditions can be found to attack 6-round Keccak-MAC-512 and 7-round Keccak-MAC-384.

Moreover, we observe that there are many redundant iterations in z-axis of the conditional cube tester in [8]. Then, we give an optimal way to recover the key for Keccak-MAC-256/128, which is twice faster than [8]. Combining this observation with the discovered 63 ordinary cube variables for Keccak-MAC-384, the time complexity to mount key-recovery attack on 7-round Keccak-MAC-384 is improved to 2^{71} from 2^{75} by using conditional cube tester. Moreover, we also give an optimal way to recover the 128-bit key for 6-round Keccak-MAC-512 with at most 2^{40} calls of 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 of 6-round Keccak internal permutation in [12]. We summarize some related results in Table 1.

Table 1. Related Results of Keccak-MAC

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

2 Description of Keccak-MAC

The Keccak-p 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 $\mathbf{R}$ n_r times. The state A can be seen 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.

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

The construction of Keccak-MAC-n is illustrated in Figure 1. In this paper, we also only consider a single block like [5,8,9,12]. Moreover, 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.

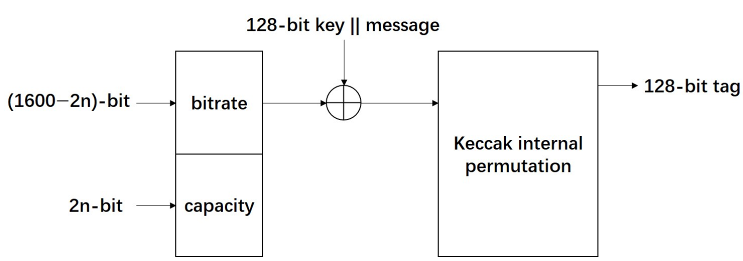

Fig. 1. Construction of Keccak-MAC-n

For Keccak-MAC-n, n $\in \{128, 256, 384, 512\}$, the 128-bit key is placed at $A^0[0][0]$ and $A^0[1][0]$ as marked in red in Figure 2. Specifically, k_i is placed at $A^0[0][0][i]$ and k_{i+64} is placed at $A^0[1][0][i]$, where $0 \leq i \leq 63$.

Observation 1. Based on the definition of θ operation, $A^0_\theta[3][i] = A^0[3][i] \oplus \bigoplus_{y=0}^{4} A^0[2][y] \oplus \bigoplus_{y=0}^{4}(A^0[4][y] \lll 1)$ for $0 \leq i \leq 4$. Therefore, the value of $A^0_\theta[3][i]$ is independent of the 128-bit key. In other words, if we add bit conditions on $A^0_\theta[3][i]$, all of them are key-independent.

Then, we consider the influence of $\pi \circ \rho$ operation as shown in Figure 3.

Observation 2. After $\pi \circ \rho$ operation, $A^0_\theta[2][i]$ and $A^0_\theta[4][k]$ are next to $A^0_\theta[3][j]$ in each row.

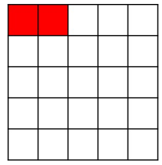

Fig. 2. Position of key

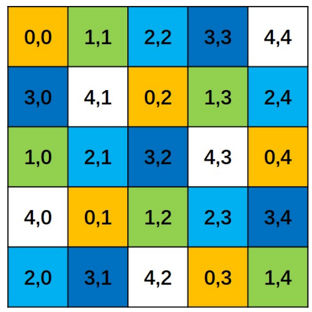

Fig. 3. $\pi \circ \rho$ operation

The above two observations are quite important to determine good candidates for ordinary cube variables. In the next two sections, we will expand on how to determine them.

3 Finding Ordinary Cube Variables for Keccak-MAC-384

In this section, the procedure to find enough ordinary cube variables for Kecccak-MAC-384 will be described in detail.

3.1 Determining Candidates for Keccak-MAC-384

The initial state of Keccak-MAC-384 is shown in Figure 4 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 to slow down its propagation. Then, the ordinary cube variables are set in the CP kernel.

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 don't impose bit conditions. For these 64 possible variables, 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.

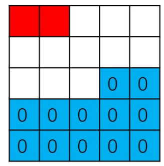

Fig. 4. Keccak-MAC-384

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 $A_{\theta}^{0}[3][k]$ $(0 \leq k \leq 4)$ to slow down the propagation of the variables. To remove the redundant conditions, we impose 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, 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.

3.2 Discussion

Imposing some bit conditions on $A_{\theta}^{0}[3][k]$ $(0 \leq k \leq 4)$ 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 $A_{\theta}^{0}[3][k_0]$ is $A_{\theta}^{0}[3][k_0][i] = 0$. However, for the fifth column, the bit condition on a certain bit j of $A_{\theta}^{0}[3][k_1]$ is $A_{\theta}^{0}[3][k_1][j] = 1$. If $i = j$ and $k_0 = k_1$, 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 imposed 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 imposed a condition.

However, in fact, the second case can be processed in a simple way. After the candidates are determined, if a contradiction in the second case is detected,

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: $A^0_\theta[3][2][46] = 0$ 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_{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: $A^0_\theta[3][0][59] = 1$ i=20: $A^0_\theta[3][0][12] = 1$ i=25: $A^0_\theta[3][0][17] = 1$ i=35: $A^0_\theta[3][4][6] = 1, A^0_\theta[3][0][27] = 1$ i=41: $A^0_\theta[3][0][33] = 1$	i=8: $A^0_\theta[3][0][0] = 1$ i=22: $A^0_\theta[3][0][14] = 1$ i=30: $A^0_\theta[3][4][1] = 1, A^0_\theta[3][0][22] = 1$ i=38: $A^0_\theta[3][4][9] = 1$ i=57: $A^0_\theta[3][0][49] = 1$																	

$A[2][0][i] = A[2][2][i] + c$																					
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: $A^0_\theta[3][3][23] = 0$ i=15: $A^0_\theta[3][1][22] = 0$ i=30: $A^0_\theta[3][1][37] = 0$ i=38: $A^0_\theta[3][1][45] = 0$ i=46: $A^0_\theta[3][1][53] = 0$ i=57: $A^0_\theta[3][1][0] = 0$	i=14: $A^0_\theta[3][1][21] = 0, A^0_\theta[3][3][36] = 0$ i=20: $A^0_\theta[3][3][42] = 0$ i=33: $A^0_\theta[3][3][55] = 0$ i=40: $A^0_\theta[3][1][47] = 0$ i=52: $A^0_\theta[3][1][59] = 0$ i=62: $A^0_\theta[3][3][20] = 0$																			

$A[2][1][i] = A[2][2][i] + c$														
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: $A^0_\theta[3][2][46] = 0, A^0_\theta[3][3][23] = 0$ i=18: $A^0_\theta[3][2][63] = 0$ i=56: $A^0_\theta[3][3][14] = 0$	i=14: $A^0_\theta[3][3][36] = 0$ i=20: $A^0_\theta[3][3][42] = 0$ i=62: $A^0_\theta[3][3][20] = 0$												

it implies that two ordinary variables multiplies with each other in the first round. For example, supposing $A^0_\theta[3][0][t]$ is imposed 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 in $A[3][0][t]$ in the first round. Please refer to $\pi \circ \rho$ operation in Figure 3. Therefore, the second case is equivalent to the case that two ordinary cube variables multiply with each other in the first round. Actually, we can also derive it from **Observation 1** and **2**. Thus, we do not have to process the second bad case and we only need focus on the relation of the candidates in the first round as well as the contradiction caused by conditions.

3.3 Constructing Inequalities

The inequalities of candidates are constructed based on 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 inequalities obtained are presented 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 the inequalities 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 inequalities, we can obtain at most 28 variables. Moreover, there are 56 fully free variables, i.e., there are no inequalities on them. We have to stress that this is not the unique way to determine the final candidates and obviously our method is much inspired from greedy algorithm. Of course, these inequalities can be solved with a solver as well. However, it can not help derive the properties implied in these inequalities.

Table 3. Inequalities 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. And 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 63 ordinary cube variables is shown 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 inequalities (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	$A^0_\theta[1][4][60] = 1, A^0_\theta[1][0][5] = 1$.
Key-independent conditions for v_0	$A^0_\theta[3][1][7] = 0, A^0_\theta[3][2][45] = 0$.
Other key-independent conditions for ordinary cube variables	Refer to Table 3 according to the chosen variables.

4 Finding Ordinary Cube Variables for Keccak-MAC-512

In the same way as we deal with Keccak-MAC-384, we found 32 candidates for ordinary cube variables as displayed in Table 5.

The inequalities obtained 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 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

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

<table>
<tr><td colspan="20">$A[2][0][i] = A[2][1][i] + c$</td></tr>
<tr><td>i</td><td>1</td><td>8</td><td>12</td><td>14</td><td>15</td><td>20</td><td>23</td><td>25</td><td>28</td><td>41</td><td>42</td><td>43</td><td>45</td><td>50</td><td>52</td><td>53</td><td>61</td><td>62</td><td>63</td></tr>
<tr><td>Variable</td><td>v_1</td><td>v_2</td><td>v_3</td><td>v_4</td><td>v_5</td><td>v_6</td><td>v_7</td><td>v_8</td><td>v_9</td><td>v_{10}</td><td>v_{11}</td><td>v_{12}</td><td>v_{13}</td><td>v_{14}</td><td>v_{15}</td><td>v_{16}</td><td>v_{17}</td><td>v_{18}</td><td>v_{19}</td></tr>
<tr><td>Condition</td><td colspan="14">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: $A^0_\theta[3][1][6] = 0, A^0_\theta[3][2][44] = 0$</td><td colspan="5">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: $A^0_\theta[3][1][59] = 0$</td></tr>
</table>

<table>
<tr><td colspan="14">$A[3][0][i] = A[3]1][i] + c$</td></tr>
<tr><td>i</td><td>3</td><td>4</td><td>9</td><td>13</td><td>15</td><td>23</td><td>30</td><td>35</td><td>39</td><td>40</td><td>46</td><td>56</td><td>57</td></tr>
<tr><td>Variable</td><td>v_{20}</td><td>v_{21}</td><td>v_{22}</td><td>v_{23}</td><td>v_{24}</td><td>v_{25}</td><td>v_{26}</td><td>v_{27}</td><td>v_{28}</td><td>v_{29}</td><td>v_{30}</td><td>v_{31}</td><td>v_{32}</td></tr>
</table>

variables become one possible variable. Then, the goal becomes how to find these possible variables.

Suppose $A^0_\theta[i][j][k]$ contains a variable, then after χ operation, three bits will contain this variable. Based on the definition of χ operation, among the three bits, one bit will always contain this variable and the other two bits contains 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 propagates to the second round based on our tracing algorithm in Appendix A. 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 independent-key conditions, once the key-dependent bits are detected, the corresponding variable should not be chosen as a candidate.

Based on the above method by tracing the influenced bits in the first two round, 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 the 5 bits of A^1_π will multiply with v_0 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 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.

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}	

Based on this table, we can find at most three possible ordinary cube variables. One choice is as follows:

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

According to the Table 6, adding $A[2][0][37] = A[2][1][37] = v_{e_2}$ to the above variables and converting the bit condition $A^0_\theta[3][1][53] = 0$ into $A^0_\theta[3][1][53] = 1$ 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-th bit and another key-independent bit condition to allow the propagation to the 1109-th bit of A^1_π.

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 by a solver. However, to mount key-recovery attack on 6-round Keccak-MAC-512, 31 ordinary cube variables are needed. 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 $2^5 \times 12$.

5 Recovering the Key

In this section, we will introduce a new slightly improved way to recover 128-bit key for Keccak-MAC by removing redundant iterations of conditional cube tester. In [8], 64 iterations in z-axis of the conditional cube tester are used to recover 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-256 for instance, after 31 iterations in z-axis, 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}.

Table 7. Candidates for Keccak-MAC-512 with One Key-dependent Bit Condition

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

Since 63 ordinary cube variables have been found for Keccak-MAC-384 as displayed in Table 4, we can recover the 128-bit key for Keccak-MAC-384 in the sane way as for Keccak-MAC-256, whose time complexity is 2^{71}.

For key-recovery attack on 6-round Keccak-MAC-512 using the conditional cube tester, we choose $A[2][0][11] = A[2][1][11]$ in Table 7 as the ordinary cube variable with one key-dependent bit condition $A_\theta^0[1][4][7] = 1$, 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.

Table 8. The Order to Recover the 93 bits of Key with Conditional Cube Tester

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

Therefore, the total time complexity becomes $2^{35} \times 31 + 2^{35} = 2^{40}$. However, once $A[2][0][19] = A[2][1][19]$ is chosen as the ordinary cube variable with one key-dependent bit condition, the time complexity is estimated as $\lceil \frac{128}{3} \rceil \times 2^{2^5+3} = \lceil \frac{128}{3} \rceil \times 2^{35} \approx 2^{40}$ in [12], which is greater than 2^{40}. Thus, our new way to recover the 128-bit key can reach the optimal time complexity by choosing a good ordinary cube variable with one key-dependent bit condition.

6 Comparison with Previous Work

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

As for [9], it seems that the key-independent bit conditions have been considered. However, it is strange that [9] 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, [9] 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 [9] did not make full use of the key-independent bit conditions.

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 is presented, which is almost the same with what we find. However, it is strange that there are 18 key-independent bit conditions to slow down the propagation of the ordinary cube variables. According to our method, 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. In fact, we can reach the minimum key-independent bit conditions, which is 8+3=11. Thus, we guess that [12] did not observe the redundancy in key-independent bit conditions.

Moreover, our method does not rely on any mathematic tool nor the sophisticated modeling used in [9,12]. Although our method shares many similarities with the core idea in [9,12], we find ordinary cube variables with a different method, which is much inspired from greedy algorithm.

At last, we present a new way to recover the 128-bit key by observing that many iterations of the conditional cube tester are redundant, thus slightly improving the time complexity to recover 128-bit key for Keccak-MAC-128/256/384. By choosing a different ordinary cube variable with one key-dependent bit condition, we can reach the optimal time complexity to attack 6-round Keccak-MAC-512.

7 Conclusion

In this paper, we present a new method to find ordinary cube variables by making full use of key-independent bit conditions. Our method is simple and much inspired from greedy algorithm, which does not require sophisticated modeling nor usage of mathematical tools. Moreover, based on our method, the property of the constructed inequalities can be considered, while it is not clear in previous work by using a solver. As our method shows, the reason why [9,12] can find enough ordinary cube variables for Keccak-MAC is due to many useful key-independent bit conditions to slow down the propagation of the ordinary cube variables. With our new method to recover the key by removing redundant iterations of the conditional cube tester, the time complexity of conditional cube attack on 7-round Keccak-MAC-128/256/384 are all improved to 2^{71} and we can attack 6-round Keccak-MAC-512 with at most 2^{40} calls of 6-round Keccak internal permutation.

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 S-traus. 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 polynomial-s. 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. Cryptology ePrint Archive, Report 2018/810, 2018. `https://eprint.iacr.org/2018/810`.
12. Ling Song, Jian Guo, Danping Shi, and San Ling. New milp modeling: Improved conditional cube attacks to keccak-based constructions. Cryptology ePrint Archive, Report 2017/1030, 2017. `https://eprint.iacr.org/2017/1030`.
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.
16. Chen-Dong Ye and Tian Tian. New insights into divide-and-conquer attacks on the round-reduced keccak-mac. Cryptology ePrint Archive, Report 2018/059, 2018. `https://eprint.iacr.org/2018/059`.

A Tracing the Influenced Positions of One Variable

Since θ, ρ, π are all linear transformations, we can use a linear transformation matrix M_0 to express the three consecutive operations $\pi \circ \rho \circ \theta$. Based on the definitions of the three operations, we can know that for each row of M_0, there are only 11 non-zero elements, whose values are all 1. Besides, since the two consecutive operations $\pi \circ \rho$ is equivalent to a permutation of bit positions, we also use an array M_1 to express it. In this paper, we only focus on Keccak-MAC-512/384 and therefore the size of M_0 is 1600×1600 and M_1 is 1600. As explained above, there are only 11 non-zero elements in each row of M_0. Thus, we use another smaller matrix SM_0 to record M_0 with size 1600×11. Specifically, the positions of the non-zero elements in each row of M_0 is recorded in the same row of SM_0. Besides, we also introduce a bit vector X to represent the 1600-bit state A with $A[x][y][z] = X[(5x + y) \times 64 + z]$.

After a variable is set to $A[x][y][z]$, we firstly consider how it propagates through θ operation. If this variable is set in the CP kernel, then only $A^0_\theta[x][y][z]$ contains this variable after θ operation. Otherwise, 11 bits of A^0_θ will contain this variable, which are $A^0_\theta[x-1][i][z+1]$, $A^0_\theta[x+1][i][z] (0 \leq i \leq 4)$ and $A^0_\theta[x][y][z]$. Then, we calculate the corresponding positions of the influenced bits in X, i.e., $A[x][y][z] = X[(5x+y) \times 64 + z]$. Suppose the influenced bit positions of X after θ operation are stored in an array XP.

For each element $XP[i]$ in XP, we calculate how it propagates through $\pi \circ \rho$ operation with M_1, which is $M_1[XP[i]]$. For each $M_1[XP[i]]$, three bits will be influenced through χ operation if without bit conditions to slow down the propagation. If some proper bit conditions are added, then the propagation through χ will be slowed down and we only record the influenced bit positions.

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

Input: EP, SM_0. **Output:** $finalPosition$

```
for row in (0...1599) do
    for col in (0...10) do
        if SM_0[row][col] == EP[i] then
            finalPosition.push_back(row)
            break
```

After knowing the influenced bit positions of $XP[i]$ through $\chi \circ \pi \circ \rho$, which will be stored in an array EP, we show how to trace its propagation in the second round. For each element $EP[i]$ in EP, we calculate how it propagates through $\pi \circ \rho \circ \theta$ in the second round with SM_0 based on Algorithm 1. The output $finalPosition$ will record the influenced bit positions.

Once knowing how a variable propagates in the first tow rounds with or without bit conditions to slow down this propagation according to the above statement, it is quite easy to determine the relation, i.e., multiply or not, of different variables in the first two rounds. Thus, we omit it.