Preimage Attacks on Reduced Troika with Divide-and-Conquer Methods

Fukang Liu[1,3], Takanori Isobe[2,3]

[1] East China Normal University, Shanghai, China
liufukangs@163.com
[2] National Institute of Information and Communications Technology, Japan
[3] University of Hyogo, Hyogo, Japan
takanori.isobe@ai.u-hyogo.ac.jp

Abstract. Troika is a recently proposed sponge-based hash function for IOTA's ternary architecture and platform, which is developed by CYBERCRYPT. In this paper, we introduce the preimage attack on 2 and 3 rounds of Troika with a divide-and-conquer approach. Instead of directly matching a given hash value, we propose equivalent conditions to determine whether a message is the preimage before computing the complete hash value. As a result, for the two-round hash value that can be generated with one block, we can search the preimage only in a valid space and efficiently enumerate the messages which can satisfy most of the equivalent conditions with a guess-and-determine technique. For the three-round preimage attack, an MILP-based method is applied to separate the one-block message space into two parts in order to obtain the best advantage over brute force. Our experiments show that the time complexity of the preimage attack on 2 (out of 24) rounds of Troika can be improved to 3^{79}, which is 3^{164} times faster than the brute force. For the preimage attack on 3 (out of 24) rounds of Troika, we can obtain an advantage of $3^{25.7}$ over brute force. In addition, how to construct the second preimage for two-round Troika in seconds is presented as well. Our attacks do not threaten the security of Troika.

Keywords: hash function, Troika, preimage, guess-and-determine, divide-and-conquer, MILP

1 Introduction

IOTA and CYBERCRYPT announced a new lightweight ternary cryptographic hash function named Troika as well as the competition for cryptanalysts to evaluate Troika with a € 200,000 prize pool for breaking its round-reduced variants on December 20, 2018 [1]. The motivation to design Troika is to develop suitable new lightweight hash function for the ternary architecture of the IOTA protocol. Since the announcement of this competition, practical collisions for one/two-round Troika with two blocks have been found by Virginie Lallemand. The one-round preimage challenge was solved by Håvard Raddum, John-Petter Indrøy and Morten Øygarden.

Troika [3] is a hash function $h : F_3^* \to F_3^{243}$ mapping arbitrary-length inputs to hash values of 243 trits. It follows the sponge construction with a rate of 243 and a capacity of 486 trits, yielding a total state of 729 trits, as shown in Fig. 1. Furthermore, the rate part of the state of Troika is overwritten by the input instead of added to it, in order to enable distributed hashing where only the capacity part of the state (486 trits) needs to be sent instead of the entire state (729 trits). Troika has to satisfy the following three requirements in order to be considered secure.

- Preimage resistance: No preimage attack of non-negligible success probability with a complexity of less than 3^{243} queries.
- Second preimage resistance: No second preimage attack of non-negligible success probability with a complexity of less than 3^{243} queries.
- Collision resistance: No collision attack of non-negligible success probability with a complexity of less than $3^{243/2}$ queries.

Although Troika shares many similarities with Keccak [4], which is the winner of SHA-3, the nonlinear transform is placed before the linear transform in Troika. Moreover, the algebraic degree of one-round Troika is 4 while it is 2 for Keccak. Cryptanalysts are obviously aware of the low algebraic degree of one-round Keccak. As a result, the linearizing techniques are widely exploited in the collision attack and preimage attack on Keccak [5,7,8,10,11]. However, the disadvantage of such linearizing techniques is the fast consumption of degree of freedom.

Considering the high algebraic degree of one-round Troika, it is not wise to use similar linearizing techniques since the degree of freedom will be faster utilized. Therefore, we will use a different strategy to achieve linearization without consuming degree of freedom. In addition, we observe that the length of hash value is almost equal to the length of one-block message, i.e. the padding rule must be satisfied. This motivates us to investigate whether it is possible to search the preimage only in a smaller potential space when the preimage can be generated with one block. As will be shown, invalid preimages can be efficiently discarded and no degree of freedom are consumed with our method.

Our Contributions Firstly, we propose equivalent conditions to pre-determine whether a message is the preimage of a given hash value. As a consequence, when the hash value can be derived from one block, the search for the preimage of two-round Troika can be limited in a much smaller space, which can be further accelerated with a guess-and-determine approach. Indeed, it is expected that our algorithm to find the preimage of two-round Troika can be applied to arbitrary hash value, as shown in our partially solving the two-round preimage challenge [1], though it is difficult to give an accurate estimation of the time complexity. Moreover, we can construct several second preimages for arbitrary messages in seconds for two-round Troika.

For the preimage attack on three rounds of Troika, the variables set at the rate part of input state can be seprated into two parts with an MILP-based method, one of which is used to verify some equivalent conditions. Only those conditions are satisfied will we start guessing the values for the variables in another part. Due to the sufficient diffusion of three-round Troika permutation, we expect our approach can be applied to arbitrary hash value. All our results are displayed in Table 1.

Table 1: Summary of preimage and collision attack on Troika

Attack Type	Rounds	Time	Generic	Ref.
Collision	1	practical	$3^{243/2}$	[1]
	2	practical	$3^{243/2}$	[1]
Preimage	1	practical	3^{243}	[1]
	2	3^{79}	3^{243}	Sec. 4
	3	$3^{217.3}$	3^{243}	Sec. 5
Second Preimage	2	3^{6}	3^{243}	Sec. 4.6

Organization The paper is organized as follows. The description of Troika is presented at Section 2. Then, we introduce how to derive equivalent conditions to match a given hash value in Section 3. The preimage attack on two and three rounds of Troika are displayed in Section 4 and Section 5 respectively. Finally, the paper is summarized in Section 6.

2 Description of Troika

The hash function Troika $h : F_3^* \rightarrow F_3^{243}$ maps arbitrary-length inputs to hash values of 243 trits [3]. It should follow the sponge construction with a rate of 243 and a capacity of 486 trits, yielding a total state of 729 trits as shown in Fig. 1. The state is initially initialized with all zeros. A message $m \in F_3^*$ is firstly padded with a trit "1" and non-negative number of "0" until the trit length of the padded message becomes multiple of 243. Then, the padded message is divided into n blocks of 243 trits each. Each block will be loaded in the rate part before processed. Formally, Troika operates on a state $A \in F_3^{729}$, which is organized as a $9 \times 3 \times 27$ cuboid of trits $A \in F_3^{9\times3\times27}$.

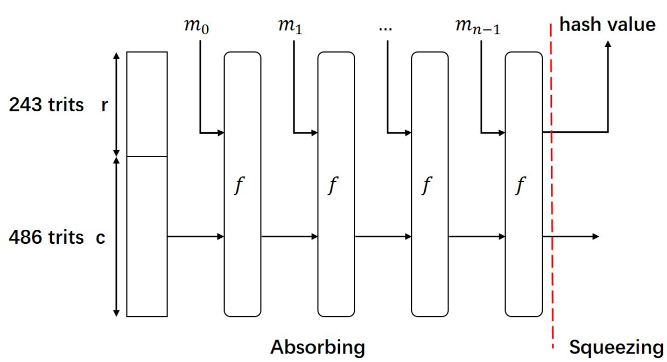

Fig. 1: Overview of Troika's Sponge Structure

The individual trits of the state are identified as $A[x][y][z]$ via their x, y, z coordinates where $0 \le x < 9$, $0 \le y < 3$ and $0 \le z < 27$. as illustrated in Fig. 2. $A[\cdot][y][z]$ composed of 9 trits is called a row of A, $A[x][\cdot][z]$ composed of 3 trits is called a column, $A[x][y][\cdot]$ composed of 27 trits is called a lane, $A[\cdot][\cdot][z]$ composed of 27 trits is called a slice, and $A[\cdot][y][\cdot]$ composed of 243 trits is called a plane. The rate part is $A[\cdot][\cdot][z]$ $(0 \leq z < 9)$ and the capacity part is $A[\cdot][\cdot][z]$ $(9 \leq z < 27)$.

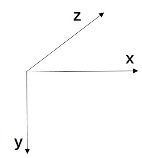

Fig. 2: Coordinate

The internal permutation of Troika consists of 24 rounds. Each round is composed of five operations: **SubTrytes**, **ShiftRows**, **ShiftLanes**, **AddColumnParity** and **AddRoundConstant**, where only **SubTrytes** is the nonlinear transform.

SubTrytes The SubTrytes mapping consists of the application of a 3-trit S-box $S : F_3^3 \rightarrow F_3^3$ to each tryte of the state as follows:

$$(a_2, a_1, a_0) \leftarrow S(9A[3i][y][z] + 3A[3i+1][y][z] + A[3i+2][y][z]),$$
$$(A[3i][y][z], A[3i+1][y][z], A[3i+2][y][z]) \leftarrow (a_2, a_1, a_0),$$

where $0 \leq i < 3$, $0 \leq y < 3$, $0 \leq z < 27$ and $t_i \in F_3$ $(0 \leq i \leq 2)$. The lookup table of the S-box is specifed in Table 2.

Table 2: Lookup table for the tryte S-box

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
$S(x)$	6	25	17	5	15	10	4	20	24	0	1	2	9	12	26	18	16	14	3	13	23	7	11	12	8	21	19

ShiftRows The ShiftRows provides diffusion along the x-axis in each row by shifting entire trytes cyclically to the right as follows:

$$A[x][0][z] \leftarrow A[x][0][z], A[x][1][z] \leftarrow A[(x-3)][1][z], A[x][2][z] \leftarrow A[(x-6)][2][z],$$

where $0 \leq x < 9$ and $0 \leq z < 27$.

ShiftLanes ShiftLanes is to provide diffusion along the z-axis in each lane by shifting trits cyclically to the right as follows:

$$A[x][y][z] \leftarrow A[x][y][(z - r[x][y])\%27],$$

where $0 \leq x < 9$, $0 \leq y < 3$ and $0 \leq z < 27$. The specification of $r[x][y]$ can be referred to Table 3.

Table 3: Specification of rotational constants $r[x][y]$

	x=0	x=1	x=2	x=3	x=4	x=5	x=6	x=7	x=8
y=0	19	13	21	10	24	15	2	9	3
y=1	14	0	6	5	1	25	22	23	2
y=2	7	17	26	12	8	18	16	11	4

AddColumnParity AddColumnParity provides diffusion along columns by adding to each column $A[x][\cdot][z]$ the parities of the two adjacent columns $A[x-1][\cdot][z]$ and $A[x+1][\cdot][z+1]$, where indices are taken modulo their respective dimensions:

$$A[x][y][z] \leftarrow A[x][y][z] + \Sigma_{y'=0}^{2} A[x-1][y'][z] + \Sigma_{y'=0}^{2} A[x+1][y'][z+1],$$

where $0 \leq x < 9$, $0 \leq y < 3$ and $0 \leq z < 27$.

AddRoundConstant The operation AddRoundConstant only works on the first plane $A[\cdot][0][\cdot]$ in each round. Suppose RC_i represents the round constant in round i, which is a vector of size 243 then, the internal state A is updated as follows:

$$A[x][0][z] \leftarrow A[x][0][z] + RC_i[x+9z],$$

where $0 \leq x < 9$ and $0 \leq z < 27$.

For convenience, we denote these five operations by ST, SR, SL, AP and AC respectively and define $L = AP \circ SL \circ SR$ and $L^{-1} = SR^{-1} \circ SL^{-1} \circ AP^{-1}$. For simplicity, we denote the input state of round i by A^i $(0 \leq i \leq 23)$. The states after ST, SR, SL, AP and AC in round i are denoted by A^i_{ST}, A^i_{SR}, A^i_{SL}, A^i_{AP} and A^i_{AC} respectively. Obviously, the state A can be viewed as a trit vector of size 729 as well. When it is viewed as a trit vector, $A[x][y][z]$ will correspond to the $(27z+9y+x)$-th trit in the vector. The complete description of Troika can be found at [3].

3 Equivalent Conditions to Find the Preimage

In this section, we introduce equivalent conditions to determine whether an input state is the preimage of a given hash value. Given a hash value of $(t+1)$-round $(0 \leq t \leq 23)$ Troika permutation, 243 trits in the rate part of A^t_{AC} are constants. Set variables to the remaining 486 trits in the capacity part of A^t_{AC} and construct an equation system

$$L^{-1} \cdot A^t_{AC} = A^t_{ST}.$$

Note that such an equation system must have solutions to A^t_{AC}. Otherwise, it is impossible to obtain the given hash value. Therefore, we define a space S satisfying the following two constraints:

Constraint 1. For each A^t_{ST} belonging to S, the equation system $L^{-1} \cdot A^t_{AC} = A^t_{ST}$ must have solutions to A^t_{AC}.

Constraint 2. For those A^t_{ST} not belonging to S, the equation system $L^{-1} \cdot A^t_{AC} = A^t_{ST}$ must not have solutions to A^t_{AC}.

Obviously, $A^t_{ST} \in S$ is a necessary but not sufficient condition to obtain the $(t+1)$-round preimage of the given hash value with one block. This is due to that the capacity part of the input state is fixed. However, when we start from a random input state A^0 with a correct fixed capacity part and compute forward until A^t_{ST}, the corresponding A^0 must be the preimage of the given hash value if $A^t_{ST} \in S$. As a result, the equivalent condition to match a given hash value with one block can be stated as follows.

The Equivalent Condition. *To find the preimage of* $(t+1)$*-round Troika, when starting from a random input state with a correct fixed capacity part, the preimage is found only when* A^t_{ST} *belongs to a specific space* S *satisfying* ***Constraint 1*** *and* ***Constraint 2****.*

3.1 Deriving the Space S

Let $A^t_{AC} = (C||V)$, where C is a 243-trit constant dependent on the hash value and V is a 486-trit variable. Then, the equation becomes

$$L^{-1} \cdot (C||V) = L^{-1} \cdot (C||0) + L^{-1} \cdot (0||V) = A^t_{ST}.$$

Let $T = A^t_{ST} - L^{-1} \cdot (C||0)$, we have

$$L^{-1} \cdot (0||V) = T.$$

Define a matrix SL^{-1}, where $SL^{-1}[i][j] = L^{-1}[i][j+243]$ for $(0 \leq i < 729, 0 \leq j < 486)$, we obtain

$$SL^{-1} \cdot V = T.$$

Suppose there is a space TS, which is used to store all valid T that make the equation system $SL^{-1} \cdot V = T$ have solutions to V. Then, the space S used to store all valid A^t_{ST} can be trivially derived since $A^t_{ST} = T + L^{-1} \cdot (C||0)$.

The space TS can be easily calculated based on Gauss elimination. A similar example is explained in Appendix A. Then, a linear equation system ET in terms of T can be derived to store all valid values of T which can make $SL^{-1} \cdot V = T$ have solutions to V. Apply Gauss elimination to ET, the solution structure of T can be determined. Such a structure is good for attackers since it reveals that some trits of T are fixed as shown in Table 8 (see Appendix A), implying that the some trits of A^t_{ST} must be constants in order to match a given hash value. The space TS is obviously the set of T satisfying the conditions in Table 8.

Taking into account the equivalent condition to determine whether an input state is the preimage, instead of computing until A^{t+1}, we can only compute until A^t_{ST} and check whether these conditions on A^t_{ST} hold. If they do not hold, such an input state must not be the preimage and we can try another input state. Such a strategy is ultimately exploited in our preimage attack on two/three rounds of Troika.

To make this paper clear, we define some terms. A tryte is called a **conditional tryte** if this tryte can not take arbitrary values. A trit is called a **conditional trit** if its value is fixed to a constant. A condition is called a **single-tryte condition** if only one tryte is involved in it. A condition is called a **multi-tryte condition** if more than one tryte are involved in it. A condition is called a **single-trit/two-trit/three-trit condition** if it is imposed on a conditional tryte, where one/two/three trits of this tryte are fixed to constants. According to Table 8, there are 162 conditional trytes, 216 conditional trits, 162 single-tryte conditions, 115 single-trit conditions (marked in black), 40 two-trit conditions (marked in blue), 7 three-trit conditions (marked in red) and 27 multi-tryte conditions.

4 Preimage Attack on Two-Round Troika

To find the preimage for two-round Troika, according to Table 8, there are 7 three-trit conditions and 40 two-trit conditions on A^1_{ST}. If we can guess the message in a proper way to ensure these conditions always hold, an advantage over brute force is achieved. This motivates us to investigate the property of an S-box.

4.1 Linearizing the Inputs of an S-box

Denote the input and output of an S-box by $(x_0, x_1, x_2) \in F_3^3$ and $(y_0, y_1, y_2) \in F_3^3$ respectively. If (y_0, y_1, y_2) is a constant, then (x_0, x_1, x_2) is a constant as well. When two trits of (y_0, y_1, y_2) are fixed, there are $3 \times 3^2 = 27$ patterns for (y_0, y_1, y_2) since it take values from F_3^3. We list all these 27 cases in Table 7 (see Appendix A). Based on this table, we observe **Property 1**.

Property 1. *When two trits of the output of an S-box are fixed, at least one linear equation of its corresponding input can be derived. In other words, the two-trit condition on the output hold with a probability of at least* 3^{-1} *if the inputs are linearized with such linear equations.*

4.2 Naive Preimage Attack on Two-Round Troika

Since there are 7 three-trit conditions and 40 two-trit conditions on A^1_{ST}, if we linearize the corresponding inputs of the S-box in A^1 based on Table 7, the probability that these conditions hold is improved to at least 3^{-40} from $3^{-21-80} = 3^{-101}$.

Observe that the nonlinear transform (**SubTrytes**) in the first round can be fully peeled off. In other words, we start from the state A^0_{ST} and set variables V_1 to $A^0_{ST}[\cdot][\cdot][z]$ $(0 \le z \le 8)$. After linearizing some inputs of the S-box in A^1 as discussed above, there are at least $3 \times 7 + 40 = 61$ linear equations in terms of A^1 in order to satisfy the 7 three-trit conditions and 40 two-trit conditions. Since A^1 is linear with V_1, these linear equations are converted to the linear equations in terms of V_1 and form a linear equation system. Then, we can arbitrary choose V_1 from the solution space of this linear equation system and test whether the hash value is matched. In this way, we can gain an advantage of at least 3^{61} over brute force to find the preimage of two-round Troika.

4.3 Improved Preimage Attack on Two-Round Troika

Only the three-trit and two-trit conditions on A^1_{ST} are exploited in the above naive two-round preimage attack. Indeed, the single-trit conditions on A^1_{ST} can be utilized as well to significantly improve the attack.

In the same way, we start from the middle state A^0_{ST} and set variables at $A^0_{ST}[\cdot][\cdot][z]$ $(0 \leq z \leq 8)$. Formally, let $A^0_{ST} = (V_1\|P)$, where P is a 486-trit constant representing the capacity part of A^0_{ST} and V_1 is a 243-trit variable. Consider the following relation:

$$L \cdot (V_1\|P) = L \cdot (V_1\|0) + L \cdot (0\|P) = A^0_{AP}.$$

Let $V_0 = A^0_{AP} - L \cdot (0\|P)$, we have

$$L \cdot (V_1\|0) = V_0.$$

To leverage all the single-tryte conditions on A^1_{ST}, we can firstly compute all valid inputs of the S-box for the corresponding conditional trytes in A^1_{ST}. For example, there is a single-trit condition on $(A^1_{ST}[0][2][0], A^1_{ST}[1][2][0], A^1_{ST}[2][2][0])$ (see Table 8). As a result, the tryte $(A^1[0][2][0], A^1[1][2][0], A^1[2][2][0])$ can only take 9 possible values, thus resulting that $(A^0_{AP}[0][2][0], A^0_{AP}[1][2][0], A^0_{AP}[2][2][0])$ can only take 9 possible values as well. Note that $V_0 = A^0_{AP} - L \cdot (0\|P)$ and $L \cdot (0\|P)$ is a constant for a fixed capacity part of the input state. Therefore, the corresponding tryte in V_0 can also only take 9 possible values. Similarly, for each conditional tryte in A^1_{ST}, we store the valid values for the corresponding tryte in V_0 in a two-dimensional dynamic array PV.

However, due to the non-full diffusion of L, there are 15 trits in A^0_{AP} as listed below, which only depend on P. Therefore, before storing each valid value, we firstly check whether it is contradictory with the values of these 15 trits. Only those values that are consistent with these 15 trits will be stored. If there is no valid value for a specific conditional tryte, it implies that such a fixed capacity P can never lead to the given hash value. In this case, it is essential to generate another value for the capacity part of A^0 by compressing random messages until there is at least one valid value for each conditional tryte in A^1_{ST}.

$$\begin{gathered}
A^0_{AP}[8][1][5], A^0_{AP}[6][1][7], A^0_{AP}[6][2][7], A^0_{AP}[7][1][12], A^0_{AP}[7][1][13],\\
A^0_{AP}[7][1][14], A^0_{AP}[7][1][15], A^0_{AP}[3][1][16], A^0_{AP}[3][1][17], A^0_{AP}[3][1][18],\\
A^0_{AP}[3][0][19], A^0_{AP}[3][1][19], A^0_{AP}[3][0][20], A^0_{AP}[3][1][20], A^0_{AP}[2][1][26].
\end{gathered}$$

According to the equivalent condition in Section 3, if we can find a solution V_1 such that $V_0 = L\cdot(V_1\|0)$ can be contained in PV, then we ensure all the single-tryte conditions on A^1_{ST}. Since only the 162 single-tryte conditions are considered at this phase, we only need compute the corresponding 162 trytes in V_0. Consequently, we only need to focus on the $162 \times 3 = 486$ linear equations between V_1 and V_0. Denote the equation system composed of these 486 linear equations by S_1: $SL \cdot V_1 = V_0'$, where SL is the coefficient matrix of size 486×243 and V_0' is of size 486×1. Note that all valid values for the trytes in V_0' have been stored in PV.

With Gauss elimination, it is easy to derive a linear equation system S_0 in terms of V_0', which is used to store all valid values of V_0' that makes the equation system S_1:

$SL \cdot V_1 = V_0'$ have solutions to V_1. If we can find a value for V_0' such that it is not only a solution of S_0 and but also contained in PV, then V_1 is found to satisfy all the single-tryte conditions. In next parts, we will expand on how to find such V_0' with a guess-and-determine approach. Before searching for such V_0', a preprocessing phase is necessary to pre-determine whether it can be found.

Note that all single-trit/two-trit/three-trit conditions have been taken into account. Therefore, for different trytes in V_0', the number of their valid values stored in PV will be different. Specifically, some trytes in V_0' can only take a unique value if they correspond to a three-trit condition. Some trytes in V_0' can only take at most 3 values if they correspond to a two-trit condition. And some trytes in V_0' can take at most 9 values if they correspond to a single-trit condition. For the trytes taking 1 or 3 values, it has been discussed previously that at least 61 linear equations in terms of V_0' can be derived.

Therefore, after obtaining S_0, we derive linear equations for each tryte of V_0' based on its valid values stored in PV as far as possible and add them to S_0. Once new equations are introduced in S_0, more variables in S_0 will become fixed. As a result, it is possible to remove invalid values for some trytes in V_0' from PV, which can be proceeded by checking whether each valid value is contained in the solution space of the updated S_0 via Gauss elimination. As invalid values are removed, the number of valid values for some trytes will decrease, thus having the potential to be linearized. Such a procedure is repeated until S_0 becomes stable, which means the size of the solution space S_0 will not be changed when adding the derived linear equations to it.

Observe that it is possible that none valid values for some trytes in V_0' are left after removing operation. In this case, it implies that such a fixed capacity P can never lead to the given hash value and it is necessary to generate another P by compressing arbitrary message.

On the whole, the above procedure can be illustrated with Figure 3. First of all, we initialize PV as stated above. At this phase, we will remove the invalid values from PV based on the 15 constant trit values, which are computed from a fixed value for the capacity part of the input. If no valid value for a conditional tryte is left, we conclude it is impossible to generate the given hash value with only one block when the capacity part of input takes such a fixed value. Otherwise, we start initializing the equation system S_0 and record its size of solution space. Next, we will repeat updating PV and S_0 until the size of the solution space of S_0 becomes unchanged. At this phase, it is possible that there is no valid value for a conditional tryte in PV after updating it. Then, we will again conclude that it is infeasible. Once a stable S_0 is obtained, it implies we cannot remove invalid values from PV anymore and PV also becomes stable. Therefore, we can start the guess-and-determine phase to find the preimage.

4.4 Guess-and-Determine Method to Find the Preimage

After obtaining the stable linear equation system S_0, whose coefficient matrix is the row simplest form matrix, instead of naively exhausting the solution space of S_0 and checking whether it is contained in PV, we use a guess-and-determine technique to find V_0' which is not only contained in PV but also contained in the solution space of S_0.

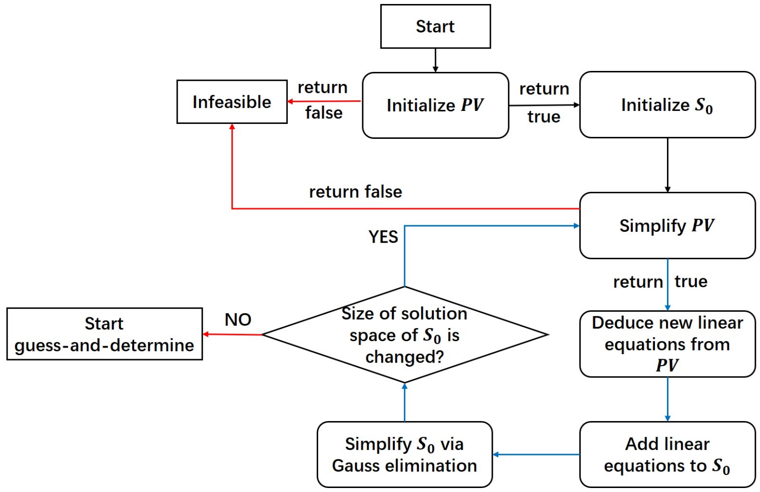

Fig. 3: Illustration of the procedure to obtain a stable S_0

As is known, for the coefficient matrix of S_0, each non-zero row will correspond to an equation

$$\alpha_0 V'_0[0] + \alpha_1 V'_0[1] + \cdots + \alpha_{485} V'_0[485] = \alpha_{486},$$

where $\alpha_i \in F_3$ $(0 \leq i \leq 486)$. Since it corresponds to a non-zero row, there must exists $\alpha_i \neq 0$ $(0 \leq i \leq 485)$. Suppose $\alpha_i \neq 0$ and $\alpha_j = 0$ $(j < i \leq 485)$, if there exists $\alpha_k \neq 0$ $(i < k \leq 485)$, then we define $V'_0[k]$ as the free variable in the equation system S_0. Moreover, we define this equation as the the equation on the trit $V'_0[i]$.

Note that the coefficient matrix of S_0 is the row simplest form matrix. If we guess the values for V'_0 in the order that

$$\begin{aligned}&(V'_0[485], V'_0[484], V'_0[483]) \rightarrow (V'_0[482], V'_0[481], V'_0[480]) \rightarrow\\ &(V'_0[3i+2], V'_0[3i+1], V'_0[3i]) \rightarrow \cdots \rightarrow (V'_0[2], V'_0[1], V'_0[0]),\end{aligned}$$

we can always verify the equations on $(V'_0[3i+2], V'_0[3i+1], V'_0[3i])$ $(0 \leq i \leq 161)$ when $(V'_0[3i+2], V'_0[3i+1], V'_0[3i])$ $(0 \leq i \leq 161)$ is guessed. In other words, when choosing a valid value from PV for the tryte $(V'_0[3i+2], V'_0[3i+1], V'_0[3i])$ $(0 \leq i \leq 161)$, we can verify whether it is contained in the solution space of S_0 before guessing the remaining free variables. If it is not contained, such a guess for this tryte is obviously wrong. Following such an order to guess, there are several advantages over simply exhausting the solution space of S_0 when properly using the guess-and-determine technique below.

How to Guess. Firstly, note that each tryte of V'_0 can take at most 9 possible values. If all the three trits in a tryte are free variables, we have to try 27 possible values of this tryte when simply exhausting the solution space of S_0. However, if we only choose valid values from PV for the three trits, we only need to guess at most 9 times, thus obtaining an advantage of at least 3^1. If two trits in a tryte are free variables, we can obtain an advantage by guessing values from PV for this tryte when the number of valid values for this tryte stored in PV is smaller than 9, which is possible to occur. If only one trit in a tryte is a variable, advantages can be gained when the number of valid values for this tryte stored in PV is smaller than 3. Otherwise, we simply guess this free variable and determine the whole tryte and then check whether it is contained in the corresponding PV. The last case is that no trit in a tryte is a variable. In this case, according to the guess order, the value for this tryte can be computed based on the corresponding three equations on them. Then, we simply check whether the computed value for this tryte is contained in the corresponding PV, which can be finished in 1 time.

Local Test. When guessing a value for a tryte from PV, it is necessary to check whether such a guessed value is contained in the solution space of S_0. This can be efficiently checked by verifying the equations on this tryte due to the guess order. If a guessed value can not pass the test, i.e. the equations on this tryte do not hold, there is no need to move ahead to next tryte from this guessed value, thus reducing the search space further more.

Look-ahead and fast backtracking. Although local test can provide early stop in a way, it is possible to occur that one value of a first guessed tryte will always lead to a contradiction for a later guessed tryte. In this case, there will be a lot of unnecessary backtracking if the two trytes locate far from each other since the guess order is predetermined. To improve the efficiency of looking ahead, we can construct a table for each tryte, which is used to record the trytes to be checked when this tryte is guessed. Only when all trytes in the recorded table can pass the local test can we move ahead to guess another tryte. In this case, for each checked tryte, the index of the first valid value in PV are recorded in order to remove redundant operations when the search actually reach these trytes.

Although look-ahead can be used to achieve faster early stop, another bad case may occur, which causes many unnecessary backtracking. Specifically, we can ensure that there is at least one valid value for a later guessed tryte with the look-ahead strategy. However, when we actually reach this tryte, we have to look ahead from this tryte as far as possible. It is possible that there is no valid value for this tryte that can pass look-ahead. As a result, backtracking starts. However, there will be many unnecessary backtracking if the value of this tryte is only influenced by a pre-guessed tryte that locates far from it. Obviously, if we can immediately backtrack to this pre-guessed tryte, the backtracking between this tryte and the pre-guessed tryte is removed, thus further reducing the search space on the whole. To achieve efficiency of fast backtracking, we construct a table for each tryte, which is used to record the tryte to be backtracked when this tryte fails to move ahead.

4.5 Complexity Evaluation

When a valid V_0' is found with the guess-and-determine approach, start exhausting the solution space of $SL \cdot V_1 = V_0'$ and check whether the given hash value can be generated with V_1. The size of the solution space of $SL \cdot V_1 = V_0$ is 3^6 based on our analysis, which is only related to the fixed coefficient matrix SL.

According to our guess-and-determine technique above, the found V_0' can only ensure the single-tryte conditions on A^1_{ST}. Therefore, each solution of the equation system $SL \cdot V_1 = V_0'$ can also only ensure the single-tryte conditions on A^1_{ST}. For the multi-tryte conditions on A^1_{ST}, they are not taken into account in our guess-and-determine technique to find a valid V_0'. To remove unnecessary enumeration of V_1 for each found V_0', when a valid V_0' is found, we firstly check some multi-tryte conditions composed of the trits that can be computed based on the fully determined V_0'. There are 8 such multi-tryte conditions. Only when they are satisfied will we start exhausting the solution space of $SL \cdot V_1 = V_0'$. Consequently, the time complexity to find a preimage with one block is equivalent to the time complexity to enumerate all valid V_0' with our guess-and-determine technique.

To calculate the time complexity to enumerate all valid V_0', we firstly omit the influence of local test, look-ahead and fast backtracking and only focus on the size of solution space if adopting our method to guess values for each tryte. Initialize a counter $cnt = 0$. After a stable S_0 is obtained, we check the positions of free variables. Suppose there are f_0 free variables in a tryte of V_0' and the number of valid values for this tryte stored in PV is f_1. For each of the 162 trytes of V_0', update cnt based on the following relations between f_0 and f_1.

$$cnt = \begin{cases} cnt & (\text{if } f_0 = 0) \\ cnt + log_3(f_1) & (\text{if } f_0 = 1 \text{ and } f_1 \leq 3) \\ cnt + 1 & (\text{if } f_0 = 1 \text{ and } 3 < f_1 \leq 9) \\ cnt + log_3(f_1) & (\text{if } f_0 = 2) \\ cnt + log_3(f_1) & (\text{if } f_0 = 3) \end{cases}$$

We generate hundreds of thousands of hash values used as the inputs to the program with random one-block messages. After a stable equation system S_0 and PV are obtained, start computing cnt. Among all these values for cnt, the largest one is $cnt = 92$. However, the effect of local test, look-ahead and fast backtracking has not been taken into account. It is reasonable to estimate that these early stop strategies can at least reduce the whole search space by a factor of 3^{13} according to our experiments. Specifically, when it is computationally feasible, we exhaust all possible values from the first guessed tryte to a certain later tryte. Then, record the total trying times in order to enumerate all valid solutions until this tryte. Meanwhile, the number of valid solutions is recorded as well. Suppose the total trying times is 3^{cnt_0}, the number of valid solutions is 3^{cnt_1} and the search space is 3^{cnt_2} without early stop strategy. In this way, we can reduce the whole search space by a factor of at least $3^{cnt_2 - cnt_1}$. As a consequence, for the hash value of two-round Troika that can be generated with one block, the time complexity to find its preimage is upper bounded by $3^{92-13} = 3^{79}$, which is 3^{164} times faster than brute force. Due to this significant advantage over brute force as well as our algorithm to predetermine whether a hash value can be generated with one block, it is expected that our algorithm can be applied to arbitrary hash values.

Attempt to solve the two-round preimage challenge. For the two-round preimage challenge [1], our algorithm shows that one block is not sufficient to generate this hash value. Therefore, we append random message blocks before the last block to generate a suitable capacity part for the last block. Such a capacity part can pass the test of our algorithm to determine whether it is potential to match the given hash value by using the degree of freedom of the last block. Then, the guess-and-determine technique will be applied to enumerate all valid V_0' and the corresponding V_1. The appended message block M_{app} we found is shown in Table 4. With such an appended message block, the two-round preimage challenge can be partially solved. Specifically, we found a solution M_{last} for the last block in seconds. There are only 40 different trits between the two-round preimage challenge and the hash value computed from $M_{app}||M_{last}$, as displayed in Table 4.

4.6 Second Preimage Attack on Two-Round Troika

To find the preimage of two-round Troika with one block, two linear equation systems S_0 in terms of V_0' and S_1: $SL \cdot V_1 = V_0'$ are constructed. The goal is to find a valid V_0'. After it is found, start exhausting the solution space of S_1: $SL \cdot V_1 = V_0'$. However, when given a message M_0 and its corresponding hash value H_0 after two-round Troika permutation, the corresponding value for V_0' computed from M_0 is known! To find the second preimage for (M_0, H_0), we simply set V_0' the same with that computed from M_0. Then, we start exhausting the solution space S_1: $SL \cdot V_1 = V_0'$. Note that there is at least one solution to V_1 that can lead to the hash value H_0, which exactly corresponds to M_0. However, our program suggests that there are several V_1 that can lead to the same hash value H_0. For the sake of correctness, we generate many random messages and compute the corresponding hash value. Our program suggests that there are always several V_1 which can lead to the same given hash value. Since the size of the solution space of S_1 is 3^6, the time complexity to find the second preimage is upper bounded by 3^6. To support our approach, we randomly generate a value for M_0 and compute the corresponding hash value H_0. Then we found that there are many second preimages for H_0. Due to the space limit, we only list 6 second preimages (M_1, M_2, M_3, M_4, M_5, M_6) in Table 9 (see Appendix A).

5 Preimage Attack on Three-Round Troika

The preimage attack on two-round Troika can be viewed as the interaction between two linear equation systems. As the attacked round increases, it is almost impossible to establish similar linear equation systems. However, we can still construct two interacting systems to find the preimage of three-round Troika. Such an idea is much inspired from the cube-attack-like cryptanalysis of Keccak-MAC by Dinur et al. [6].

Note that an equivalent condition to match a given hash value has been proposed in Section 3. Specifically, when starting from a state with a correct fixed capacity part, matching a three-round hash value is equivalent to satisfying the 243 trit conditions on A^2_{ST} as displayed in Table 8. The main technique is to separate the 243 variables set at $A^0_{ST}[\cdot][\cdot][z]$ $(0 \leq z \leq 8)$ into two parts PA_1 and PA_2. Then, exhaust all possible values of the variables at PA_1 and compute some trytes in A^2_{ST}. Only when the equivalent conditions on these trytes hold can we start exhausting all possible values

Table 4: Partially solving the two-round preimage challenge

Two-round preimage challenge
10022220211101201100100111010021122102121221022020112 1
1111112110002211121020121212121210202102111122021222 12
1111122210201120112001122222020100200100220221010202 02
2200120110120100001111111201020112222120110221210111 22
12111111100120100221211001 2

M_{app}
2021122010100112102021102002100102221020000112010120 21
0221111100122020111121212201000102021222011112101201 02
2010221002001210111021011121020012211012210111021201 00
0002212120111020012112111202121101020112201110210202 12
0111011221012120110002100 21

M_{last}
2111100122210000100200002202002122011021201122020220 00
2121022222100100221000122220201102121010100012111110 00
1101202120122221002221021021011101002100000211011102 11
2120111110212100112211221212212111022212012222112021 00
0011210000011121212100221 00

Hash value computed from $M_{app}\|\|M_{last}$
2002222022110120111010011101012112210022122102212011 21
1111112110002211121020121212121210202102111122021222 12
1111122210201120112001122222020100200100220221010202 02
2100120100020100021011111221021112022111110020211111 02
2221010020021002211112112 00

of the variables in PA_2. When all variables in PA_1 and PA_2 are guessed, the one-block message is fixed and we can determine whether it is the preimage of the three-round hash value.

As has been mentioned in [3], after three-round Troika permutation, the computation of one S-box requires the knowledge of all S-boxes in the first round. Therefore, it is reasonable to assume that three-round Troika provides sufficient diffusion and the 243 trits of the three-round hash value are independent from each other. In other words, suppose the capacity part of A^0_{ST} is fixed and there are 243 variables at $A^0_{ST}[\cdot][\cdot][z]$ $(0 \leq z \leq 8)$, we expect that one hash value only corresponds to one value of these 243 variables. Note that matching a given hash value is equivalent to satisfying the 243 conditions on A^2_{ST} when starting from a correct fixed capacity part. Suppose the 243 trit conditions are not independent from each other, it may occur that more than one values of the variables can make the all the 243 trit conditions on A^2_{ST} hold, suggesting that one hash value may correspond to more than one value of the 243 variables. Consequently, we can assume the 243 conditions on A^2_{ST} are independent based on the assumption that three-round Troika provides sufficient diffusion.

If there are t_0 trit conditions on A^2_{ST} that can be tested by only guessing all the t_1 variables at PA_1, based on the assumption that these trit conditions are independent, we can expect that only $3^{t_1-t_0}$ valid values are left for these t_1 variables after 3^{t_1} computations. Then, for each of the $3^{t_1-t_0}$ valid values, exhaust the remaining $(243-t_1)$ variables and compute the three-round hash value. As a result, with time complexity $3^{t_1} + 3^{243-t_0}$, we can exhaust all possible one-block messages. Suppose the given hash value can be generated with only one block, the preimage must be found. When it can not be generated with only one block, we can append random blocks before the last block to generate a valid capacity part of the last block and exhaust all possible values of the last block with the above method. As a result, with at most $3 \times (3^{t_1} + 3^{243-t_0})$ computations, we can expect to find the preimage of three-round Troika.

Based on the above analysis, achieving the optimal time complexity is equivalent to finding the optimal separation of the 243 variables. As will be shown, such a problem can be solved with MILP (Mixed-Integer Linear Programming), which was firstly introduced to cryptanalysis in [9].

5.1 Finding Optimal Separation with MILP

Our attack starts from the middle state A^0_{ST} and the 243 variables $(v_0, ..., v_{242})$ are set at $A^0_{ST}[\cdot][\cdot][z]$ $(0 \leq z \leq 8)$, i.e. $v_{27z+9y+x} = A^0_{ST}[x][y][z]$. First of all, for each conditional tryte CT_i $(0 \leq i \leq 161)$ at A^2_{ST}, record the corresponding variables in $(v_0, ..., v_{242})$ that need knowing in order to compute this conditional tryte, which can be easily finished with the linear transform matrix L. Suppose the recorded variables for the conditional tryte CT_i are $(v_{j_0}, v_{j_1}, ..., v_{j_r})$, then we construct the following inequalities to ensure that when CT_i needs to be determined, each of $(v_{j_0}, v_{j_1}, ..., v_{j_r})$ must be guessed:

$$TV_i - VV_{j_s} \leq 0,\ s \in \{0, 1, .., r\},$$

where $TV_i = 1$ $(0 \leq i \leq 161)$ denotes that the tryte CT_i needs to be determined and $VV_i = 1$ $(0 \leq i \leq 242)$ denotes that the variable v_i belongs to PA_1, while $TV_i = 0$

$(0 \leq i \leq 161)$ denotes that the tryte CT_i does not need to be determined and $VV_i = 0$ $(0 \leq i \leq 242)$ denotes that the variable v_i belongs to PA_2. In this way, as many as 16305 inequalities can be derived.

The objective function of the MILP model is set as

$$\text{MAX} \sum_{i=0}^{161} c_i \cdot TV_i,$$

where c_i denotes the number of conditonal trits in the conditional tryte CT_i.

To ensure that the number of variables in PV_1 is not too large, we adaptively add the following inequality to the constraints

$$\sum_{i=0}^{242} VV_i \leq bd,$$

where bd is used to constrain the number of variables in PA_1. To obtain optimal time complexity of the preimage attack on three-round Troika, bd should be as small as possible while the objective function should be as large as possible. Therefore, we adaptively choose values for bd and record the results of the objective function returned by the Gurobi solver [2]. Some results are displayed in Table 5.

Table 5: Results for different *bd*

bd	Result of obj.	Time complexity of attack
124	5	$3 \times (3^{124} + 3^{238})$
160	10	$3 \times (3^{160} + 3^{233})$
170	13	$3 \times (3^{170} + 3^{230})$
190	18	$3 \times (3^{190} + 3^{225})$
200	20	$3 \times (3^{200} + 3^{223})$
210	24	$3 \times (3^{210} + 3^{219})$
215	27	$3 \times (3^{215} + 3^{216})$

Based on the results displayed in Table 5, the optimal value for bd is 215. For $bd = 215$, the corresponding separation of the variables $(v_0, ..., v_{242})$ and the conditional trits in A^2_{ST} to be checked are listed in Table 6. Therefore, the time complexity of the preimage attack on three-round Troika is $3^{217.3}$, which is $3^{25.7}$ times faster than brute force.

6 Conclusion & Future Work

By discovering some equivalent conditions to pre-determine whether a message is the preimage of a given hash value, invalid messages can be filtered at an early stage and the search can be limited to a smaller potential space. To speed up the search in this potential smaller space for two-round preimage attack, two interacting linear equation systems

Table 6: The optimal separation of variables to achieve best time complexity

PA_1
v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_8, v_9, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}, 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}, v_{33}, v_{34}, v_{35}, v_{36}, v_{38}, v_{39}, v_{40}, v_{41}, v_{42}, v_{43}, v_{44}, v_{45}, v_{46}, v_{47}, v_{48}, v_{51}, v_{52}, v_{53}, v_{54}, v_{55}, v_{57}, v_{58}, v_{59}, v_{60}, v_{63}, v_{64}, v_{65}, v_{66}, v_{67}, v_{69}, v_{70}, v_{71}, v_{72}, v_{73}, v_{74}, v_{75}, v_{76}, v_{77}, v_{78}, v_{79}, v_{80}, v_{82}, v_{83}, v_{84}, v_{85}, v_{86}, v_{87}, v_{88}, v_{89}, v_{90}, v_{91}, v_{92}, v_{93}, v_{94}, v_{95}, v_{97}, v_{98}, v_{99}, v_{100}, v_{101}, v_{102}, v_{103}, v_{104}, v_{105}, v_{106}, v_{107}, v_{108}, v_{109}, v_{110}, v_{111}, v_{113}, v_{114}, v_{115}, v_{116}, v_{117}, v_{118}, v_{119}, v_{120}, v_{121}, v_{122}, v_{123}, v_{124}, v_{125}, v_{126}, v_{128}, v_{129}, v_{130}, v_{131}, v_{132}, v_{133}, v_{134}, v_{135}, v_{137}, v_{138}, v_{140}, v_{141}, v_{142}, v_{143}, v_{144}, v_{145}, v_{146}, v_{147}, v_{148}, v_{149}, v_{150}, v_{151}, v_{152}, v_{153}, v_{155}, v_{156}, v_{157}, v_{158}, v_{159}, v_{160}, v_{161}, v_{162}, v_{163}, v_{165}, v_{166}, v_{167}, v_{168}, v_{170}, v_{171}, v_{172}, v_{173}, v_{174}, v_{175}, v_{177}, v_{178}, v_{179}, v_{180}, v_{181}, v_{182}, v_{183}, v_{185}, v_{187}, v_{188}, v_{190}, v_{192}, v_{193}, v_{194}, v_{195}, v_{197}, v_{198}, v_{199}, v_{200}, v_{201}, v_{202}, v_{203}, v_{204}, v_{205}, v_{206}, v_{207}, v_{208}, v_{209}, v_{210}, v_{212}, v_{213}, v_{214}, v_{215}, v_{216}, v_{217}, v_{218}, v_{220}, v_{221}, v_{222}, v_{223}, v_{224}, v_{225}, v_{226}, v_{227}, v_{228}, v_{229}, v_{230}, v_{232}, v_{233}, v_{234}, v_{235}, v_{236}, v_{237}, v_{238}, v_{239}, v_{240}, v_{241}, v_{242}.
27 conditional trits on A^2_{ST} to be checked
$A^2_{ST}[4][0][3]$, $A^2_{ST}[5][2][3]$, $A^2_{ST}[6][0][4]$, $A^2_{ST}[8][0][4]$, $A^2_{ST}[7][1][4]$, $A^2_{ST}[2][0][6]$, $A^2_{ST}[1][1][6]$, $A^2_{ST}[2][1][6]$, $A^2_{ST}[4][0][7]$, $A^2_{ST}[3][1][7]$, $A^2_{ST}[4][1][7]$, $A^2_{ST}[5][1][7]$, $A^2_{ST}[4][0][8]$, $A^2_{ST}[3][1][8]$, $A^2_{ST}[4][1][8]$, $A^2_{ST}[5][1][8]$, $A^2_{ST}[5][2][8]$, $A^2_{ST}[4][0][9]$, $A^2_{ST}[3][1][9]$, $A^2_{ST}[4][1][9]$, $A^2_{ST}[5][1][9]$, $A^2_{ST}[5][0][13]$, $A^2_{ST}[5][1][13]$, $A^2_{ST}[4][2][13]$, $A^2_{ST}[0][1][23]$, $A^2_{ST}[1][2][23]$, $A^2_{ST}[2][2][23]$.

are constructed. Then, a guess-and-determine technique involving fast cutting branches to efficiently enumerate valid solutions for one of the equation systems is proposed. Consequently, the time complexity to find the preimage of two-round Troika with one block is at most 3^{79}, which is 3^{164} times faster then brute force. Moreover, with the divide-and-conquer method, the one-block message space is separated in an optimal way with MILP so as to achieve optimal time complexity of the preimage attack on three-round Troika. As a result, the preimage of three-round Troika can be found with time complexity $3^{217.3}$.

Our algorithm shows that the second preimage for two-round Troika can be found with pretty small time complexity. In other words, we can efficiently enumerate several two-round differential characteristics which can lead to a collision for two-round Troika with only one block. To construct a collision for three-round Troika, we have placed the obtained two-round differential characteristic in the last two rounds and computed backward by one round to obtain the actual input difference. However, there is always difference in the capacity part of the input, which implies that we cannot find a three-round differential characteristic to generate a collision with only one block. We also have tested whether the obtained two-round differential characteristics for collision can be extended to three rounds. However, it is shown that there is always difference in the rate part of the output. Our future work is to improve the strategy to search the (second) preimage for two-round Troika and see whether it is possible to actually solve the three-round collision challenge and two-round preimage challenge.

Acknowledgement We thank the anonymous reviewers of IWSEC 2019 for their insightful comments and suggestions. We also thank the Troika Group for the discussion. Fukang Liu is supported by Invitation Programs for Foreigner-based Researchers of the National Institute of Information and Communications Technology (NICT). Takanori Isobe is supported by Grant-in-Aid for Scientific Research (B) (KAKENHI 19H02141) for Japan Society for the Promotion of Science.

References

1. Cybercrypt. `https://www.cyber-crypt.com/troika-challenge/`.
2. Gurobi. `https://www.gurobi.com/`.
3. Troika: a ternary hash function, 2018. `https://www.cyber-crypt.com/wp-content/uploads/2018/12/20181221.iota_.troika-reference.v1.0.1.pdf`.
4. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, 2011. `http://keccak.noekeon.org`.
5. Itai Dinur, Orr Dunkelman, and Adi Shamir. New attacks on Keccak-224 and Keccak-256. In *FSE 2012*, pages 442–461, 2012.
6. 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 *EUROCRYPT 2015*, pages 733–761, 2015.
7. Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced Keccak. In *ASIACRYPT 2016*, pages 249–274, 2016.
8. Ting Li, Yao Sun, Maodong Liao, and Dingkang Wang. Preimage attacks on the round-reduced Keccak with cross-linear structures. *IACR Trans. Symmetric Cryptol.*, 2017(4):39–57, 2017.
9. Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *Inscrypt 2011*, pages 57–76, 2011.
10. Kexin Qiao, Ling Song, Meicheng Liu, and Jian Guo. New collision attacks on round-reduced Keccak. In *EUROCRYPT 2017*, pages 216–243, 2017.
11. Ling Song, Guohong Liao, and Jian Guo. Non-full sbox linearization: Applications to collision attacks on round-reduced Keccak. In *CRYPTO 2017*, pages 428–451, 2017.

A Some Tables and Example

Suppose there is a linear equation system $E_0 \cdot X = Y$, where $X \in F_3^3$ and $Y \in F_3^4$ and

$$E_0 = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 2 \\ 0 & 1 & 1 \\ 1 & 0 & 2 \end{pmatrix}. \tag{1}$$

The goal is to compute the space YS to store all valid Y which can make the equation system $E_0 \cdot X = Y$ have solutions to X for each $Y \in SY$. Construct a new matrix $E_1 = (E_0, E)$ where E is an identity matrix of size 4×4. Then, apply Gauss elimination to E_1 to obtain $E_1' = (E_0', E')$ where E_0' becomes the staircase matrix.

$$E_1 = \begin{pmatrix} 2 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 2 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow E_1' = \begin{pmatrix} 2 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 1 & 1 \end{pmatrix}. \tag{2}$$

Therefore, YS is actually the solution space of the following equation system 3.

$$E_1' = \begin{pmatrix} 2 & 0 & 1 & 1 \end{pmatrix} \cdot Y = \begin{pmatrix} 0 \end{pmatrix}. \tag{3}$$

Table 7: Linearizing the input of an S-box

Output (y_0, y_1, y_2)	Inputs (x_0, x_1, x_2)	Equations
(-,0,0)	(1,0,0), (1,1,0), (1,2,0)	$x_0 = 1, x_2 = 0.$
(-,0,1)	(0,1,2), (1,0,1), (2,2,2)	$x_0 + x_1 = 1.$
(-,0,2)	(0,2,1), (1,0,2), (2,1,1)	$x_0 - x_1 = 1.$
(-,1,0)	(2,0,0), (2,1,2), (2,2,1)	$x_0 = 2, x_1 + x_2 = 0.$
(-,1,1)	(0,2,0), (1,1,1), (2,0,1)	$x_0 + x_1 = 2.$
(-,1,2)	(0,1,0), (1,2,2), (2,0,2)	$x_0 - x_1 = 2.$
(-,2,0)	(0,0,0), (0,1,1), (0,2,2)	$x_0 = 0, x_1 - x_2 = 0.$
(-,2,1)	(0,0,1), (1,2,1), (2,1,0)	$x_0 + x_1 = 0.$
(-,2,2)	(0,0,2), (1,1,2), (2,2,0)	$x_0 - x_1 = 0.$
(0,-,0)	(0,0,0), (1,0,0), (2,0,0)	$x_1 = 0, x_2 = 0.$
(0,-,1)	(0,2,0), (1,0,1), (2,1,0)	$x_0 - x_1 = 1.$
(0,-,2)	(0,1,0), (1,0,2), (2,2,0)	$x_0 + x_1 = 1.$
(1,-,0)	(0,1,1), (1,1,0), (2,1,2)	$x_1 = 1, x_0 + x_2 = 1.$
(1,-,1)	(0,1,2), (1,2,1), (2,0,1)	$x_0 - x_1 = 2.$
(1,-,2)	(0,0,2), (1,2,2), (2,1,1)	$x_0 + x_1 = 0.$
(2,-,0)	(0,2,2), (1,2,0), (2,2,1)	$x_1 = 2, x_0 - x_2 = 1.$
(2,-,1)	(0,0,1), (1,1,1), (2,2,2)	$x_0 - x_1 = 0.$
(2,-,2)	(0,2,1), (1,1,2), (2,0,2)	$x_0 + x_1 = 2.$
(0,0,-)	(1,0,0), (1,0,1), (1,0,2)	$x_0 = 1, x_1 = 0.$
(0,1,-)	(0,1,0), (0,2,0), (2,0,0)	$x_2 = 0.$
(0,2,-)	(0,0,0), (2,1,0), (2,2,0)	$x_2 = 0.$
(1,0,-)	(0,1,2), (1,1,0), (2,1,1)	$x_1 = 1, x_0 - x_2 = 1.$
(1,1,-)	(1,2,2), (2,0,1), (2,1,2)	$x_0 + x_1 - x_2 = 1.$
(1,2,-)	(0,0,2), (0,1,1), (1,2,1)	$x_0 - x_1 - x_2 = 1.$
(2,0,-)	(0,2,1), (1,2,0), (2,2,2)	$x_1 = 2, x_0 + x_2 = 1.$
(2,1,-)	(1,1,1), (2,0,2), (2,2,1)	$x_0 - x_1 + x_2 = 1.$
(2,2,-)	(0,0,1), (0,2,2), (1,1,2)	$x_0 + x_1 + x_2 = 1.$

Table 8: Conditions on T

Slice: 0	Slice: 1	Slice: 2	Slice: 3	Slice: 4
−−− −−− 0−0 00− −−− −00 −−0 0−− −−−	−−− −−− 0−0 00− −−− −00 −−0 −−0 −−−	−−− −−− 0−0 000 −−− −0− −−0 −−0 −−−	−−− −0− 0−0 −00 −−− −0− −−0 −−0 −−−	−−− −0− 0−0 −00 −0− −0− −−− −−0 −−−
Slice: 5	**Slice: 6**	**Slice: 7**	**Slice: 8**	**Slice: 9**
−−− −0− 0−− −00 00− −0− −−− −−0 −−−	−−0 −0− −−− −00 00− −0− −−− −−0 −−−	−−0 −0− −−− −−0 000 −0− −−− −−0 −−−	0−0 −0− −−− −−0 000 −−− −−− −−0 −−−	0−0 −0− −−− −−0 000 −−− −−− −−− −−0
Slice: 10	**Slice: 11**	**Slice: 12**	**Slice: 13**	**Slice: 14**
0−0 −0− −−− −−− 000 −−− −−− −0− −−0	0−0 −−− −−− −−− 000 −−− 0−− −0− −−0	0−0 −−0 −−− −−− 0−0 −−− 0−− −0− −−0	0−0 −−0 −−− −−− −−0 0−− 0−− −0− −−0	00− −−0 −−− −−− −−0 0−− 0−− −0− −−0
Slice: 15	**Slice: 16**	**Slice: 17**	**Slice: 18**	**Slice: 19**
00− −−0 −−− −−− −−− 0−− 0−− −0− 0−0	−0− −−0 −−− −−− −−− 0−− 00− −0− 0−0	−0− 0−0 −−− −−− −−− 0−− 00− −0− 0−−	−0− 0−0 −0− −−− −−− 0−− 00− −−− 0−−	−0− 0−0 −0− −−− −−− 0−− −0− −−− 00−
Slice: 20	**Slice: 21**	**Slice: 22**	**Slice: 23**	**Slice: 24**
−0− 0−− −0− −−− −−− 0−− −0− 0−− 00−	−0− 0−− −0− −−− −−− −−0 −0− 0−− 00−	−−− 0−− −0− 0−− −−− −−0 −0− 0−− 00−	−−− 0−− −0− 0−− −−− −−0 −00 0−− −0−	−−− 0−− −00 0−− −−− −−0 −−0 0−− −0−
Slice: 25	**Slice: 26**			
−−− −−− 000 0−− −−− −−0 −−0 0−− −0−	−−− −−− 0−0 00− −−− −−0 −−0 0−− −0−			

Multi-tryte conditions

$T[7] + T[364] + T[531] + T[694] = 0$, $T[25] + T[308] + T[512] + T[572] = 0$,
$T[48] + 2T[582] = 0$, $T[52] + T[226] + T[328] + T[678] = 0$,
$T[71] + 2T[380] = 0$, $T[75] + T[419] + T[459] + T[609] = 0$,
$T[90] + 2T[678] = 0$, $T[98] + T[293] + T[407] + T[595] = 0$,
$T[117] + T[380] + T[672] + T[705] = 0$, $T[128] + 2T[419] = 0$,
$T[143] + 2T[419] = 0$, $T[155] + T[170] + T[446] + T[709] = 0$,
$T[168] + 2T[531] = 0$, $T[195] + T[390] + T[545] + T[558] = 0$,
$T[199] + T[308] + T[512] + T[572] = 0$, $T[232] + 2T[595] = 0$,
$T[259] + T[535] + T[582] + T[622] = 0$, $T[266] + 2T[380] = 0$,
$T[281] + 2T[545] = 0$, $T[301] + T[308] + T[512] + T[572] = 0$,
$T[337] + 2T[709] = 0$, $T[363] + 2T[531] = 0$, $T[432] + 2T[582] = 0$,
$T[485] + 2T[545] = 0$, $T[508] + 2T[595] = 0$, $T[645] + 2T[678] = 0$,
$T[667] + 2T[709] = 0$.

Table 9: Instances of second preimage

M_0	010111010120012002222211020001011221011201111110102021 0112201121021120010120001102200002110222121112220221200 022220002100110000012011202010212212112211201212002111 021102000112202120120010200202000102120211210212012222 002210111200011200202200101
H_0	201122002102220012201001121112002110102101210010010000 211020121011111222011201021220212210011022020101011220 202010222210112101212020102111202112211000220021012122 1222200000020222021210102021010122120122111122221022201 110011212021210221220022111
M_1	010111010022012002222211020001011221110201111110102021 0112201121021120010120001102200002110222121112210221200 022102011100110000012011202010110212112211201212002111 021102000112202120120010200202000102120211210212012222 002210111200011200202200101
M_2	010111010022012002222211020001011221110201111110102021 0112201121021120010120001102200002110222121112210221200 022102011100110000110011202010110201112211201212002111 021102000112202120120010200202000102120211210212012222 002210111200011200202200101
M_3	010111010022012002222211020001011221110201111110102021 0112201121021120010120001102200002110222121112220221200 022102002100110000012011202010110212112211201212002111 021102000112202120120010200202000102120211210212012222 002210111200011200202200101
M_4	010111010120012002222211020001011221011201111110102021 0112201121021120010120001102200002110222121112210221200 022220011100110000012011202010212212112211201212002111 021102000112202120120010200202000102120211210212012222 002210111200011200202200101
M_5	010111010120012002222211020001011221011201111110102021 0112201121021120010120001102200002110222121112220221200 022220002100110000110011202010212201112211201212002111 021102000112202120120010200202000102120211210212012222 002210111200011200202200101
M_6	010111010120012002222211020001011221212201111110102021 0112201121021120010120001102200002110222121112220221200 022220002100110000110011202010011201112211201212002111 021102000112202120120010200202000102120211210212012222 002210111200011200202200101