Cube-Attack-Like Cryptanalysis of Keccak-Based Constructions Exploiting State Differences (Full Version)

Mohammad Vaziri[1] and Vesselin Velichkov[1]

The University of Edinburgh, United Kingdom
{mohammad.vaziri,vvelichk}@ed.ac.uk

Abstract. This paper presents an enhancement to cube-attack-like cryptanalysis by minimizing output-bit dependency on related key bits, thereby improving attack complexity. We construct two distinct initial states differing exclusively in predetermined bit positions. Through independent cube summation and state difference analysis, we observed reduced related key bits dependency for specific output bits. We validate our approach by targeting three Keccak keyed variants Ketje Minor, Keccak-MAC-512 and Keccak-MAC-384, developing a dedicated tool to recover all output-bit superpolies. Using our computational resources, we successfully attacked 4-round of Ketje Minor and 5-round of other variants, confirming both the method's validity and practical applicability. While the best known attacks on these structures reach 7-round, our results improve upon the 5-round.
We construct our initial state configurations based on the automated method proposed by Bi et al. in *Design, Codes and Cryptography* (2019), and compare our results with theirs. For the 4-round Ketje Minor, we reduce the time complexity from 2^{20} to $2^{16.8}$; for 5 round Keccak-MAC-512, from 2^{34} to $2^{31.2}$; and for 5 round Keccak-MAC-384, from $2^{27.6}$ to $2^{26.5}$.
Keywords: cube attack, Keccak Keyed Variants, Cube-Attack-Like Cryptanalysis

1 Introduction

Following collision attacks on MD5 and SHA-1 proposed by Wang et al.'s [WY05, WYY05], the U.S. National Institute of Standards and Technology (NIST) held a public competition in 2007 to choose a new cryptographic hash function standard (SHA-3). After five years of careful review, NIST picked Keccak [BDP+11] as the winner in 2012.

Besides being used as a regular hash function, Keccak can also work in keyed modes like message authentication codes (MAC) and stream ciphers. Additionally, the Keccak permutation has been used in other cryptographic designs, including authenticated encryption schemes such as Keyak [BDP+16b], Ketje [BDP+16a], and the pseudorandom function Kravatte [BDH+17].

With the growing use of lightweight devices, the demand for efficient cryptographic algorithms has increased substantially. In response, the National Institute of Standards and Technology (NIST) [Nat23] initiated a competition to standardize lightweight authenticated encryption with associated data (AEAD) and hashing schemes. On February 7, 2023, NIST selected Ascon [DEMS21] winning algorithm.

Another notable cryptographic algorithm submitted to NIST, is Xoodyak [DHP+20], which is designed based on [DHAK18] permutation. Xoodyak is designed by the Keccak team and has been one of the finalist of the NIST competition.

The first adaptation of cube attack [DS09] to Keccak keyed modes was developed by [DMP+15], who introduced cube-attack-like cryptanalysis where cube variables do not multiply in the first round. This key modification enables linearization of Keccak's first round and provided the first security evaluation of Keccak's keyed modes. Subsequent work by [HWX+17] proposed the conditional cube attack, exploiting Keccak's large state degree of freedom to identify conditional cube variables that avoid multiplication with ordinary cube variables through both first and second rounds.

To address the limited degrees of freedom in certain Keccak keyed modes and improve attack efficiency, researchers have pursued automated approaches for both cube-attack-like cryptanalysis and conditional cube attacks. Initial work by [BDL+19,SG18] independently developed mixed integer linear programming (MILP) model for cube-attack-like methods, while [LBDW17, SGSL18] advanced MILP modeling to improve conditional cube attacks. Subsequent innovations include [LDB+19]'s refined conditional attack variant and applications to Xoodyak by [ZLD+20, VV23], collectively expanding the framework's capabilities against Keccak based constructions.

Our contributions. This paper focuses on enhancing cube-attack-like cryptanalysis by reducing output-bit dependency on related key bits, thereby improving attack complexity. Building on configurations from [BDL+19], we construct two initial states that differ exclusively in predetermined bit positions. Through independent cube summation on both states and exploiting state differences, we observed reduced dependency on related key bits for specific output bits. We verify our approach by targeting three Keccak keyed variants Ketje Minor, Keccak-MAC-512, and Keccak-MAC-384, using by developing a tool [1] that recovers all output-bit superpolies.

Using our computational resources, we successfully attacked 4-round of Ketje Minor and 5-round of other variants. These results verify both the correctness and practical applicability of our method. However, due to the faster diffusion of variables in Ketje Minor a lightweight design, extending the attack to 5 rounds proved infeasible, as our computational resources encountered memory limitations.

It is worth mentioning that while the best known-attacks on our target structures reach 7 rounds, we specifically compare our results with the 5-round attacks from [BDL+19] for equivalent evaluation. Enhanced computational resources or tool optimizations could potentially extend our approach to 6-round attacks. The best achieved complexities (to the best of our knowledge) for each variant are:

- Ketje Minor: Using conditional cube attack, 2^{113} time complexity, targeting 7-round, proposed by [SGSL18].
- Keccak-MAC-512: Using cube-attack-like, 2^{111} time complexity, targeting 7-round, proposed by [SG18].

[1] The tool has been fully implemented, and all attacks have been empirically validated. The tool is available at: https://github.com/mohammadvaziri/Cube-Attack-Like-Cryptanalysis-of-Keccak-Based-Constructions-Exploiting-State-Differences.

– Keccak-MAC-384: Using conditional cube attack, 2^{75} time complexity, targeting 7-round, proposed by [LBDW17].

Table 1 presents a comparative analysis of our results against the results achieved by the [BDL+19] technique.

Table 1: Complexity comparison between our technique and [BDL+19] for 4-round attack on Ketje Minor and 5-round attacks on Keccak-MAC-512 and Keccak-MAC-384.

Target	$\|K\|$	Degree of freedom	Rounds	Time	Memory	Data	Reference
Ketje Minor	128	654	4	$2^{16.8}$ 2^{20}	2^{1} $2^{7.3}$	2^{10} $2^{15.3}$	Section 5.3 [BDL+19]
Keccak-MAC-512	128	447	5	$2^{31.2}$ 2^{34}	2^{7} 2^{15}	2^{28} 2^{31}	Section 5.1 [BDL+19]
Keccak-MAC-384	128	703	5	$2^{26.5}$ $2^{27.4}$	$2^{3.5}$ 2^{7}	$2^{21.5}$ 2^{24}	Section 5.2 [BDL+19]

Outline. The paper is organized as follows. Section 2 details the Keccak sponge construction and the specific keyed variants under evaluation. Section 3 systematically reviews cube attack variants, focusing on cube-attack-like cryptanalysis and its MILP-based automation. Section 4 presents our core technical contribution: a state difference propagation control method. Finally, Section 5 demonstrates practical cryptanalysis, applying our technique to three keyed Keccak variants (Ketje Minor, Keccak-MAC-512 and Keccak-MAC-384).

1.1 Notations

Throughout the paper, we will use the following notations:

A_0	initial state of the Keccak permutation,
$A_{i,\theta}$	internal state of Keccak after θ in the $(i+1)$-th round,
$A_{i,\rho}$	internal state of Keccak after ρ in the $(i+1)$-th round,
$A_{i,\pi}$	internal state of Keccak after π in the $(i+1)$-th round,
$A_{i,\chi}$	internal state of Keccak after χ in the $(i+1)$-th round,
A_i	output state of Keccak of the $(i+1)$-th round,
$(i,*,k)$	index of a column,
$(*,j,k)$	index of a row,
$(i,j,*)$	index of a lane,
(i,j,k)	index of a bit,
$A[i][j]$	the lane indexed by $(i,j,*)$ of state A,
$A[i][j][k]$	the bit indexed by (i,j,k) of state A.

2 Preliminaries

In the section, we will briefly introduce some necessary background for this paper.

2.1 The Keccak-*p* permutations

The Keccak-p permutations, denoted as Keccak-$p[b,n]$, are defined by two parameters: the width of the permutation in bits $b=25\times 2^l$ (where $l=0,...,6$) and the number of rounds n. The state of Keccak-$p[b,n]$ is represented as a three-dimensional array of bits $a[5][5][w]$, with $w=2^l$. Each bit is indexed by coordinates (x,y,z), where x and y are modulo 5 and z is modulo w.

The state can be visualized in several ways:

- A **slice** is a two-dimensional part $a[*][*][z]$.
- A **row** is a one-dimensional part $a[*][y][z]$.
- A **column** is $a[x][*][z]$.
- A **lane** is $a[x][y][*]$, also written as $a[x][y]$ for simplicity. At the lane level, the state becomes a 5×5 array.

These notations are visualized in Figure 1.

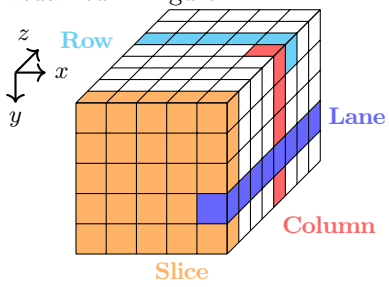


Fig. 1: The Keccak sate.

The round function of Keccak-$p[b,n]$ consists of five steps: θ, ρ, π, χ, and ι ($R=\iota\circ\chi\circ\pi\circ\rho\circ\theta$). These steps perform operations such as XOR, bitwise rotations, lane reordering, and nonlinear transformations. The round Function can be formalized as follows:

$$
\begin{aligned}
&\theta: A[x][y]=A[x][y]\bigoplus\textstyle\sum_{j=0}^{4}(A[x-1][j]\bigoplus(A[x+1][j]\lll 1)).\\
&\rho: A[x][y]=A[x][y]\lll r[x,y].\\
&\Pi: A[y][2x+3y]=A[x][y].\\
&\chi: A[x][y]=A[x][y]\oplus((\neg A[x+1][y])\wedge A[x+2][y]).\\
&\iota: A[0][0]=A[0][0]\oplus RC_{i_r}, \text{ where } RC_{i_r} \text{ is the } i_r\text{-th round constant.}
\end{aligned}
\tag{1}
$$

The operation θ is used for the state diffusion. If the variables in each column of the state has even parity, they bypass the diffusion to other columns caused by θ. This property is called column-parity-like kernel (CP-like kernel), and for the first time was exploited by [DMP+15] to control the diffusion of the cube variables through the first round of Keccak.

In the round function of Keccak, the only nonlinear operation is χ, with an algebraic degree of 2. As a result, following an r-round Keccak internal permutation, the maximum algebraic degree of the output polynomial is bounded by 2^r.

Due to the structure of the χ operation, any two variables placed in adjacent positions before χ will be multiplied together. To prevent variable multiplication in the first round, we strategically position variables in the initial state such that they avoid adjacent placement prior to χ. This approach, employed in previous cube attacks on Keccak constructions, ensures that the resulting polynomials maintain a degree of 1 with respect to the cube variables.

The operation ι is not considered in our analysis, as it does not influence the described attacks.

2.2 Keccak-MAC

The Keccak sponge function can be transformed into a Message Authentication Code (MAC) by prepending a secret key to the input message. This keyed version provides both data integrity and authentication guarantees. The security of Keccak-MAC depends on two fundamental requirements: preventing key disclosure and resisting forgery attempts.

In its basic single-block implementation (illustrated in Figure 2), Keccak-MAC processes a concatenated key-message pair through the sponge construction. The authentication tag output is conventionally 128 bits in length. The actual block size adapts to different Keccak variants for a capacity parameter $c=2n$, the block size becomes $1600-2n$ bits. Typically, both the secret key and generated tag occupy exactly two state lanes (128 bits total). Additional details can be found in [BDP+11]. In this work we only focus on Keccak-MAC-512 and Keccak-MAC-384.

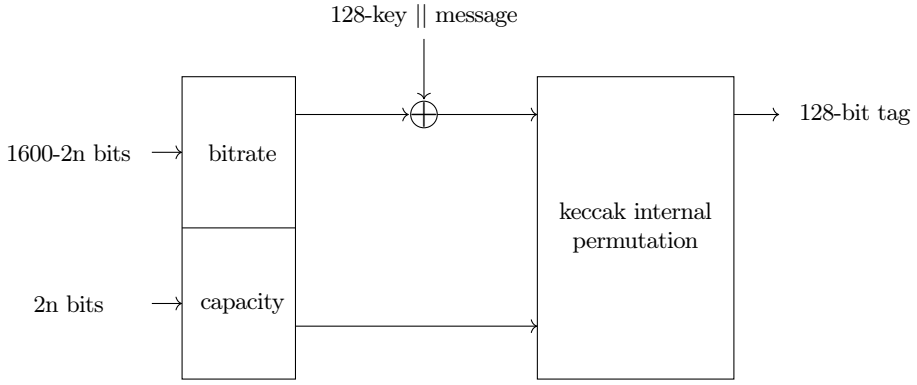


Fig. 2: Structure of Keccak-MAC-n

2.3 Ketje: Authenticated Encryption Scheme

Ketje is one of the 16 candidates in the 3rd round CAESAR competition is a group of authenticated encryption schemes based on the Keccak-p permutation. The initial versions, Ketje Jr and Ketje Sr, were introduced in Ketje V1. These versions use Keccak-p with state sizes of 200 and 400 bits, respectively. Later, Ketje Minor and Ketje Major were added in Ketje V2, using 800-bit and 1600-bit states. The main

change in Ketje V2 is the use of a modified permutation, called Keccak-p*, instead of the original Keccak-p used in V1.

Ketje uses a structure known as *MonkeyWrap* (Figure 3), which organizes the process into four stages:

- **Initialization:** The state is set up using the key, nonce, and padding. Then, a Keccak-p* permutation is applied for a number of rounds.
- **Associated Data Processing:** The associated data is divided into blocks of δ bits (except possibly the last one). Each block is padded, then added (via `XOR`) to the state. After each block, a single round of Keccak-p* is performed. If no associated data exists, a block derived from the empty string is still processed.
- **Message Processing:** The message is also split into δ-bit blocks. Each block is `XOR`ed with part of the state to produce the ciphertext, and then absorbed into the state.
- **Finalization:** A permutation with more rounds is applied to generate output bits. If the desired authentication tag length is not reached, additional rounds are used until enough bits are produced.

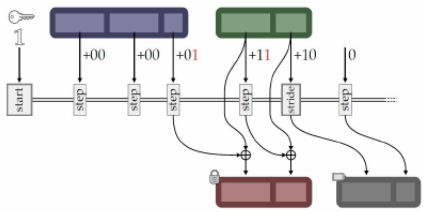


Fig. 3: Wrapping a header and a body with Monkey Wrap [BDP+16a]

For all four versions in Ketje V2, the number of rounds used during initialization is 12, one round is used per data block, and 6-round are used during finalization. Thus, the first ciphertext block is produced after at least 13-round. Most attacks described in the literature, including those in this work, reduce this round number to analyze security.

The parameters for each version of Ketje V2 are given in Table 2, showing differences in key size, internal state size, and output block size (δ).

Table 2: Parameters of Ketje V2 instances

Name	**Key size (bits)**	**Permutation**	δ	**Confidentiality Bound**
Ketje Jr	96	Keccak-p*[200]	16	min(96, $\|K\|$)
Ketje Sr	128	Keccak-p*[400]	32	min(128, $\|K\|$)
Ketje Minor	128	Keccak-p*[800]	128	min(128, $\|K\|$)
Ketje Major	128	Keccak-p*[1600]	256	min(128, $\|K\|$)

In Ketje V2, Ketje Minor and Ketje Major are introduced, where a twisted version of Keccak-p, denoted Keccak-p* is used. It reorders lanes before and after applying the original permutation to simplify certain operations. More details can be found in [BDP+16a].

3 Related Works

3.1 Cube Attacks

The cube attack is a key-recovery technique that uses chosen plaintexts. It was formally introduced in [DS09] and builds upon earlier methods such as higher-order differential cryptanalysis [Lai94].

This attack treats the output bits of a cipher as a Boolean polynomial $f:X^n \to \{0,1\}$ over the input bits. The main idea is that if f has a low algebraic degree d, then summing over all 2^{d-1} inputs of a specific subset of variables (called a cube), while keeping the other variables fixed, gives a function that is linear in the secret variables.

Theorem 1 (Dinur, Shamir [DS09]). *Given a Boolean polynomial $f:X^n \to \{0,1\}$ of degree d, let $t=x_0x_1\cdots x_{k-1}$ be a monomial with $0<k<d$. Then f can be written as*

$$f(x)=t\cdot P_t(x)+Q_t(x) \tag{2}$$

where $Q_t(x)$ contains no terms divisible by t, and $\deg(P_t)\leq d-k$. $P_t(x)$ is called superpoly of t. If we take the sum over the cube C_t defined by the k variables in t, we obtain

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

which has a degree of at most $d-k$. In particular, when $k=d-1$, the result is linear.

Because addition and subtraction are the same in $\mathbb{F}_2$, this cube sum behaves similarly to a derivative, reducing the polynomial's degree.

In Theorem 1, $P_t(x)$ denotes the *superpoly* of the term t. The original cube attack consists of two phases: preprocessing and online. In the preprocessing phase, a random cube set is selected to derive the superpoly (independent of secret variables). In the online phase, secret variables are fixed to compute the actual values of the obtained superpolies, and a linear system of equations is constructed. As highlighted in [DMP+15, DS09], implementing the original cube attack is computationally expensive because cube variables are randomly selected, and the resulting superpolies often lack useful information. In the following, the two phases of the cube attack are outlined: preprocessing and online.

Preprocessing Phase [DMP+15]. This phase is done once for a given cryptosystem and does not depend on the actual secret key.

Let $v=(v_1,...,v_{d-1})$ be the public variables (e.g., message bits or nonce), and $x=(x_1,...,x_n)$ the secret key variables. The output bit is represented by the polynomial $f(v,x)$. For a selected cube C_t over the public variables, we define the cube sum as:

$$\sum_{v\in C_t} f(v,x)=L(x)$$

where $L(x)$ is called the *superpoly*. If f has degree d, and the cube includes $d-1$ public variables, then $L(x)$ is a linear function of the secret variables:

$$L(x)=a_1x_1+...+a_nx_n+c$$

To determine $L(x)$, we compute:

- The constant term c as $\sum_{v\in C_t} f(v,0)$
- Each coefficient a_i by evaluating the sum $\sum_{v\in C_t} f(v,0,...,1_i,...,0)$

In general cases, we might need to estimate the degree d of $f(v,x)$ by testing cubes of different sizes. However, in the context of our attacks on Keccak, the degree is known, and such estimation is not necessary.

Online Phase [DMP+15]. This phase is carried out after the secret key has been fixed.

Here, the attacker chooses values for the public variables and computes:

$$\sum_{v\in C_t} f(v,x)=b_t$$

Since $L(x)$ was already determined in the preprocessing step, this gives a linear equation:

$$a_1x_1+...+a_nx_n+c=b_t$$

Repeating this process for different cubes C_t provides multiple linear equations. When enough independent equations are obtained to match the number of unknown key variables, the attacker can solve the system using Gaussian elimination.

3.2 Dinur et al.'s Cube-Attack-Like Cryptanalysis on Keccak

In traditional cube attacks, the cipher's structure is treated as a black-box system. Since the actual degree of the polynomial in the cube variables in higher rounds is unknown, the cube variables are typically selected at random. However, in the case of Keccak, the degree of the cube variables is explicitly known (for the r-th round, it is 2^r). This knowledge allows for a simplification of the cube attack procedure.

In [DMP+15], Dinur et al.'s. observe that by carefully selecting the positions of the cube variables in the initial state and utilizing the CP-like kernel property, they can prevent the multiplication of cube variables in the first round.

Consequently, for an attack on r-round of the Keccak keyed mode, the complexity of the cube attack depends solely on $S=2^{r-1}$ cube variables and T key bits that multiply with the cube variables in the first round.

In the preprocessing phase, for each possible value of the T key bits, the cube summation over S cube variables is computed and stored in a list L, indexed by the cube sum values. During the online phase, while the secret key bits remain fixed, a cube summation over the S cube variables is calculated. For each match found in L, the corresponding candidate values for the T related key bits are retrieved.

Therefore, the time complexity of the attack is 2^{S+T} for the preprocessing phase and 2^S for the online phase. Additionally, the overall memory and data complexity of the attack are 2^T and 2^S, respectively.

Trade-off Between Preprocessing and Online Phases. The basic attack described above involves an expensive preprocessing phase that dominates its time complexity. As presented in [DMP+15], Dinur *et al.* propose a complexity trade-off between the preprocessing and online phases, resulting in a more efficient attack.

For the specific case of 6-round reduced Keccak-MAC, their approach introduces 32-bit auxiliary variables during the preprocessing phase. These variables are constrained to equal the key bits in $A[0][0]$ within the same column. This constraint places half of $A[0][0]$ (a 32-bit key bits) along with the auxiliary variables set in the CP-like kernel.

This configuration provides two key advantages:

- Cube variables avoid multiplication with these key bits and auxiliary variables in the first round
- Only 32 key bits interact with cube variables after the first round

Consequently, just these 32 key bits affect the cube sums of the output after six rounds in the preprocessing phase.

In the online phase, this trade-off requires computing the cube sum for each possible value of the auxiliary variables. This technique modifies the time complexity of both preprocessing and online phases to $2^{S+T/2}$ and reduces the memory complexity to $2^{T/2}$. To calculate the data complexity, since the cube summation is performed over $T/2$ auxiliary variables in the online phase, the overall complexity increases to $2^{S+T/2}$.

Now let us formally define the following two key concepts:

Definition 1 (Related Key Bits). *The subset of secret key bits that multiply with cube variables during the first round.*

Definition 2 (Auxiliary Variables). *Variables constrained to equal specific key bits during the preprocessing phase, used to control their diffusion through the first θ operation.*

3.3 Automating Dinur et al.'s Technique with MILP

As discussed in Section 3.2, the complexity of the cube-attack-like cryptanalysis depends only on two parameters: the number of cube variables and the number of related key bits. To optimize the attack complexity while keeping the number of cube variables fixed, one can strategically select the positions of cube variables in the initial state. This careful positioning may reduce the number of related key bits, thereby improving the overall attack efficiency.

In the method proposed by [DMP+15], the selected cube variables are not optimal, which consequently leads to a non-optimal number of related key bits. To minimize the number of related key bits in cube-attack-like cryptanalysis, the works of [BDL+19] and [SG18] independently propose automated techniques based on MILP modeling.

The method presented in [BDL+19] enforces constraints by requiring all cube variables to be limited within the CP-like kernel. In contrast, the model introduced in [SG18] adopts a more flexible approach, permitting cube variable placement anywhere in the initial state without restriction to the CP-like kernel. This broader selection typically yields superior results.

In our attack framework presented in Section 4, we employ the MILP technique introduced by [BDL+19], primarily due to the availability of their implementation. We subsequently compare our results with their findings. The automation of Dinur et al.'s technique proposed by [BDL+19] can formally be described in Algorithm 1.

Algorithm 1: Automated Key Recovery via MILP-optimized cube attack

Input: : Initialized keyed Keccak construction
Output: Candidate secret key values
:

Initial Setup;
Load initial state
with cube variables and related keys obtained from MILP model in [BDL+19];
Configure auxiliary variables for half of the related keys;
    Align in same columns as related keys;
    Enforce non-multiplication with cube variables in first round;
3. Initialize sets;
    $G \leftarrow$ All guessing key values;
    $A \leftarrow$ All auxiliary variable values;
Preprocessing Phase;
for *each* $g \in G$ **do**
    Compute cube summation S_g for current key configuration;
    Store S_g in lookup table L indexed by summation value;
Online Phase;
for *each* $a \in A$ **do**
    Compute cube summation S_a;
    if S_a *exists in* L **then**
        Record candidate (g,a) pairs where $L[g] = S_a$;

3.3.1 MILP Model and Search Strategy proposed by [BDL+19]

In this section, we summarize the MILP model introduced by [BDL+19]. The goal is to search for linear cubes in Keccak-based keyed modes such that the number of related key bits is minimized.

For any bit in the initial state of Keccak, denoted $A[x][y][z]$, we define a binary MILP variable as follows:

$$A[x][y][z] = \begin{cases} 1 & \text{if the bit is a cube variable or a related key bit,} \\ 0 & \text{otherwise.} \end{cases}$$

To ensure that cube variables remain linear in the first round, we must prevent them from multiplying with each other in the χ step. This is expressed by the inequality

$$A[x_1][y_1][z_1] + A[x_2][y_2][z_2] \leq 1, \tag{3}$$

whenever two bits $A[x_1][y_1][z_1]$ and $A[x_2][y_2][z_2]$ would otherwise interact multiplicatively.

The diffusion of cube variables through the θ operation can be controlled using the CP-like kernel. This requires that the sum of cube variables in a column is constant (usually zero), which reduces their diffusion. For each column (x,z), the constraints are

$$\sum_{y=0}^{4} A[x][y][z] - 2d[x][z] \geq 0, \tag{4}$$

$$d[x][z] - A[x][i][z] \geq 0 \quad (0 \leq i \leq 4), \tag{5}$$

where $d[x][z]$ is an auxiliary variable that indicates whether column (x,z) contains cube variables. The number of independent cube variables in that column is then

$$\sum_{y=0}^{4} A[x][y][z] - d[x][z].$$

To ensure a sufficient number of cube variables for the attack (e.g., 2^n), we impose the global constraint

$$\sum_{x,y,z} A[x][y][z] - \sum_{x,z} d[x][z] = 2^n - 1. \tag{6}$$

If a cube variable $A[x_2][y_2][z_2]$ multiplies with a key bit $A[x_1][y_1][z_1]$, then the corresponding key bit must also be marked as "related." This is enforced by

$$A[x_1][y_1][z_1] - A[x_2][y_2][z_2] \geq 0. \tag{7}$$

The goal is to minimize the number of related key bits involved in the cube attack. Thus, the MILP model uses the objective function

$$\min \sum_{x,y,z} A[x][y][z]. \tag{8}$$

Combining the constraints 3–7 with the objective function 8 yields a complete MILP model. This model can be solved using a solver such as Gurobi [Gur13], which outputs candidate sets of cube variables with the minimal number of related key bits for a given round configuration.

4 Cube-Attack-Like Cryptanalysis Exploiting State Differences

In this section, we introduce our approach to improve cube-attack-like cryptanalysis. We explain how the dependency on related key bits in the superpolies of output bits can be reduced by exploiting state differences.

4.1 Basic Idea

As established in Section 3, optimizing the complexity of cube-attack-like cryptanalysis requires minimizing the number of related key bits involved in the attack. One potential research direction involves examining the superpolies of each output bit in

the cube summation to identify whether certain output bits depend on fewer related key bits. However, this approach proves ineffective for higher-round analysis due to the strong diffusion properties of the Keccak permutation.

Using our automated tool, we have systematically analyzed the superpolies of all output bits in several keyed variants of Keccak constructions. Our experiments demonstrate that after 5 rounds (requiring 16 cube variables for the attack), nearly all output bits exhibit dependence on almost every related key bit. To be more specific, the polynomial expression of each superpoly contains all related key bits

To obtain superpolies in the output bits that depend on fewer related key bits following cube summation, we observed that when considering two distinct configurations sharing identical cube variables and consequently the same related key bits, differing only in specific predetermined positions of the initial state, the following holds: By performing cube summation separately for each configuration and then computing their difference, the structural similarity between the two configurations causes cancellation of common terms in the superpolies. This differential analysis reveals that certain output bits depend on fewer related key bits compared to their counterparts in the non-differentiated case.

In the following, we present our methodology. To illustrate our technique, we focus on attacking 3-round Keccak-MAC-512.

For visualization clarity, the 3 dimensional state of the Keccak provided in Figure 1 is replaced with the figure presented in the Figure 4.

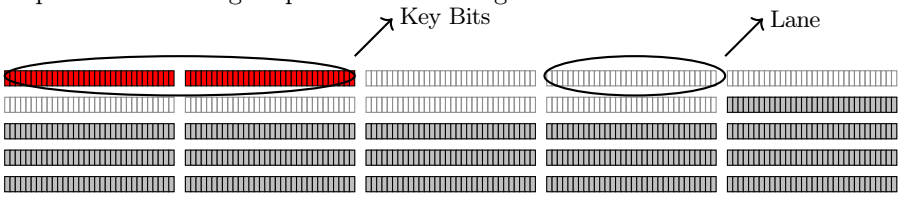


Fig. 4: Visualization of Keccak-MAC-512 state with key bits (red), padding (gray), and lane structure highlighted. The cube variable should be placed in white lanes. Each small square presents 2 bits.

We begin by outlining the attack procedure for 3-round Keccak-MAC-512. The attack requires 4 cube variables and employs the MILP model from [BDL+19], which identifies 9 related key bits.

The first state in Figure 5 illustrates the initial state configuration, where cube variables are uniquely colored for identification, and Related key bits are marked in red. As shown in Figure 5, state (b) demonstrates the spatial relationship between related key bit k_{42} and cube variable v_1 prior to the first χ operation. Their adjacent positioning ensures multiplication during the χ step, as highlighted in the figure.

State (c) in Figure 5, identifies the bit positions dependent on key bits after cube summation for 3-round Keccak-MAC-512. The 3-round attack serves as an illustrative example due to the limited number of cube variables and related key bits. As can be seen, there are just some limited bit positions with non-zero superpolies and also they depend on specific subsets of related key bits. This simplicity facilitates tracing the attack mechanism, though we emphasize that higher-round scenarios exhibit

considerably more complex behavior. We present this simplified case to demonstrate our technique's core principles.

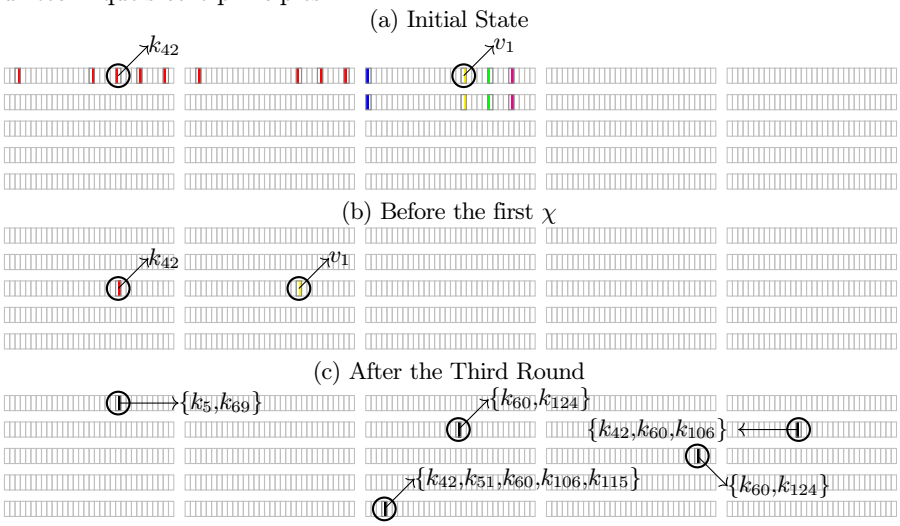


Fig. 5: (a) Initial state configuration for attacking 3-round Keccak-MAC-512.
(b) Detailed view of the interaction between k_{42} and v_1.
(c) Related key bits in non-zero superpolies in output bit positions after 3-round

To exploit state differences between two configurations, we proceed as follows. Let first configuration be defined by the initial state shown in Figure 5. For second configuration, we maintain the same initial state structure and introduce a constant difference at bit position $A_0[0][1][42]$ along with the same column containing related key bit k_{42}. For clarity, we set $A_0[0][1][42]=0$ in Configuration 1 and $A_0[0][1][42]=1$ in Configuration 2 (see Figure 6). This controlled modification enables precise analysis of state propagation differences.

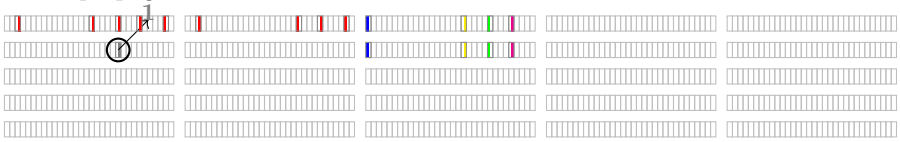


Fig. 6: Initial state of second configuration.

After the θ operation, the constant 1 in Configuration 2 propagates to produce $k_{42}+1$ in specific bit positions, replacing the original k_{42} term. This modification results in the multiplication of v_1 with $k_{42}+1$ rather than k_{42} during the subsequent χ operation.

By analyzing state differences after the first χ operation, we observe that only the cube variable v_1 appears at bit position $A_0[4][2][43]$. Figure 7 traces these state differences through multiple operations until the final round. The last state of Figure 7 specifically demonstrates how the non-zero superpolies depend on subsets of related key bits.

As shown in Figure 7, the superpolies for output bit positions $A_2[4][1][26]$ and $A_2[2][4][7]$ depend only on k_{60} and $\{k_{51},k_{60},k_{115}\}$, respectively, whereas in the first case (Figure 5), these superpolies depend on $\{k_{42},k_{60},k_{106}\}$ and $\{k_{42},k_{51},k_{60},k_{106},k_{115}\}$.

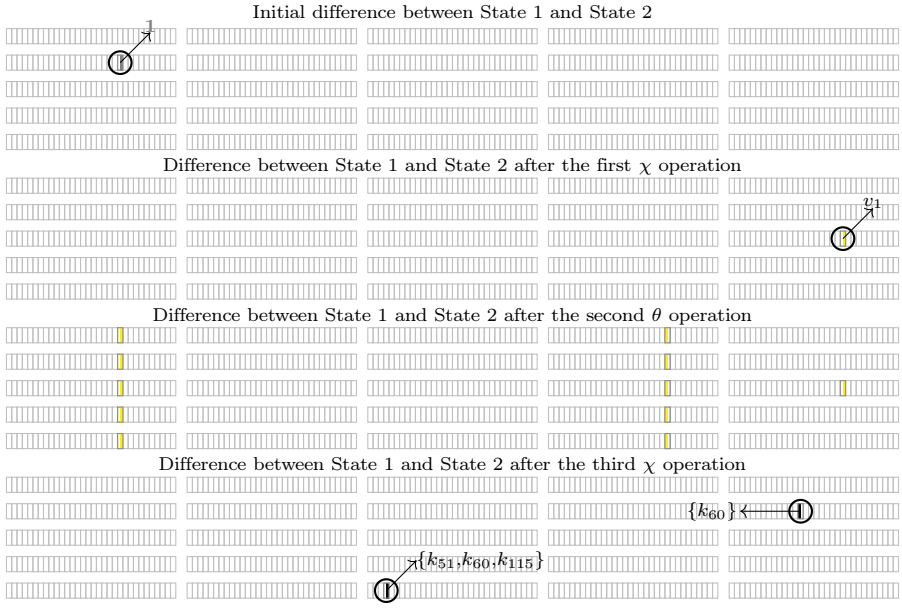


Fig. 7: State difference propagation between Configuration 1 (Figure 5) and Configuration 2 (Figure 6) across all rounds.

Additionally, the superpolies for bit positions $A_2[0][0][43]$, $A_2[2][1][35]$, and $A_2[3][2][57]$ are canceled after state difference exploitation, as they originally depend on $\{k_5,k_{69}\}$, $\{k_{60},k_{124}\}$, and $\{k_{60},k_{124}\}$ without such exploitation.

The limited number of cube variables in this analysis enables straightforward verification using computational algebra tools such as SageMath.

It should be noted that in our demonstrated examples, we exclusively include terms that contribute to the final superpolies in the cube summation. This focused representation eliminates non-essential components.

To further develop our analysis, we introduce two key concepts central to understanding state differences in our technique:

Definition 3 (Axis Cube Variable). *A cube variable that persists in the state difference pattern following the first χ operation.*

Definition 4 (Axis Related Key). *A related key bit that only multiplies with an axis cube variable after the first χ operation, and differs (one of them is negated in the states after passing θ) in two states for performing the state differences.*

In our proposed technique, the key strategy involves generating two distinct states that, while not identical, are as similar as possible. By doing so, when we compute the difference after independently performing the cube summation on each state, a significant number of terms in the corresponding superpolies cancel out. Consequently, the dependency of the superpolies on a smaller number of related key bits is expected.

As can be observed, the presence of axis cube variables introduces differences between the states. Consequently, the similarity between the two states depends on

the propagation of these axis cube variables. This implies that better control over the propagation of axis cube variables leads to greater similarity between the generated states. In the next section, we present a method to control the propagation of axis cube variables during the second round.

It should be noted that our technique employs only a single axis cube variable. This design choice stems from the observation that introducing additional axis cube variables would increase the dissimilarity between the two states.

4.2 Controlling the Propagation of Axis Cube Variable with Modifying the MILP Model Proposed by [BDL+19]

In the configuration setting described in Section 4.1, the axis cube variable is generated only at position $A_0[4][2][6]$ at the output of the first round. Therefore, the θ operation in the second round propagates this axis cube variable to 10 additional positions (see Figure 7). However, leveraging the CP-like kernel property of the θ operation, we can control the propagation by ensuring that axis cube variables remain within the same column after the χ operation at the first round. This configuration allows us to bypass the diffusion effect of θ in the second round. Below, we detail the methodology for achieving this.

To ensure the axis cube variable appears in two bit positions of the same column after the first round, we first analyze the initial positioning of v_0. The Keccak structure guarantees that when axis cube variables are placed in four specific bit positions of the initial state with each pair located in the same column (i.e., $A_0[x_0][y_0][z_0]$, $A_0[x_0][y_1][z_0]$, $A_0[x_1][y_2][z_1]$, and $A_0[x_1][y_3][z_1]$), two axis cube variables will appear in the same column after the first χ operation ($A_{0,\chi}[x_2][y_4][z_2]$, $A_{0,\chi}[x_2][y_5][z_2]$).

As a concrete example, when v_0 is initialized at positions $A_0[2][0][0]$, $A_0[2][1][0]$, $A_0[3][0][34]$, and $A_0[3][1][34]$, it propagates to multiple positions including $A_{0,\chi}[3][1][62]$ and $A_{0,\chi}[3][4][62]$ after the first round. In this configuration, the related key bit k_{65} multiplies with v_0 at position $A_{0,\chi}[3][1][62]$, while k_{60} multiplies with v_0 at $A_{0,\chi}[3][4][62]$. We designate k_{60} and k_{65} as axis related keys.

Crucially, to maintain v_0 exclusively in these output positions after the first round, we must prevent multiplication of axis related keys with any other cube variables. This constraint necessitates modifications to the MILP model proposed by [BDL+19], in Section 3.3.1. The following outlines the key steps in implementing these modifications.

1. Add the following constraints to the MILP model:

$$A_0[x_0][y_0][z_0] = A_0[x_0][y_1][z_0] = A_0[x_1][y_2][z_1] = A_0[x_1][y_3][z_1] = 1$$

2. Modify the constraint 6:

$$\sum_{x,y,z} A_0[x][y][z] - \sum_{x,z} d[x][z] = 2^{n-1} - 1$$

   by excluding the variables $A_0[x_0][y_0][z_0]$, $A_0[x_0][y_1][z_0]$, $A_0[x_1][y_2][z_1]$, $A_0[x_1][y_3][z_1]$, $d[x_0][z_0]$, and $d[x_1][z_1]$.
3. Solve the MILP model to obtain the solution file.

4. Implement the obtained configuration by:
   - Loading the configuration onto a state
   - Applying the transformation $\chi \circ \pi \circ \rho \circ \theta$
5. Verify whether the axis related keys multiply with any non-axis cube variables.
6. If such multiplication occurs:
   - Identify the initial position of the interfering cube variable (e.g., $A_0[x_3][y_6][z_3]$)
   - Add the constraint $A_0[x_3][y_6][z_3]=0$ to the MILP model
   - Return to Step 3

   Otherwise, output the obtained configuration as the final result.

The modified MILP model typically results in a greater number of related key bits compared to the conventional approach. Nevertheless, despite this increase in related key bits, our experiments reveal significant outcomes.

To clarify our technique, we present a concrete example for 4-round Keccak-MAC-512. We focus on the 4-round case because, in the 3-round scenario, our controlled propagation of the axis cube variable results in complete cancellation of all terms in the superpolies of the output bits. Consequently, for the 3-round case our technique becomes independent of any related key bits, making the 4-round case more illustrative for demonstrating our key-dependent approach.

Figure 8 illustrates the initial states of the two configurations obtained from our modified MILP model following the aforementioned steps. For these configurations, the model yields a minimum of 19 related key bits, which represents a 3-fold increase compared to the conventional attack proposed by [BDL+19].

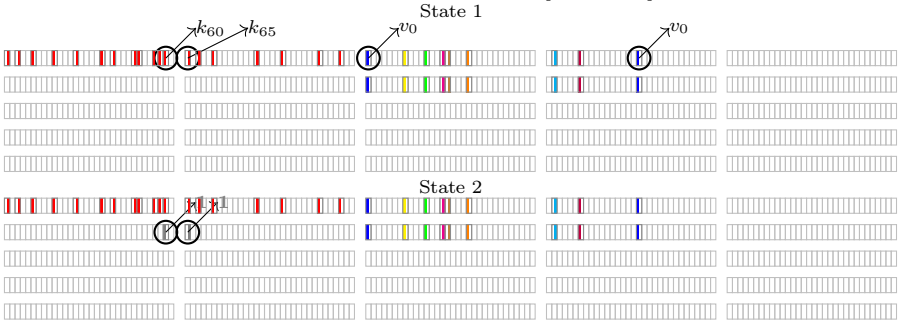


Fig. 8: Initial states of the two configurations to attack 4-round of Keccak-MAC-512, the axis cube variable is v_0 and the axis related key bits are k_{60} and k_{65}.

Figure 9 traces the evolution of state differences for the configurations shown in Figure 8 through successive rounds of operations. The final state of Figure 9 specifically illustrates how the non-zero superpolies depend on particular subsets of related key bits, demonstrating the controlled propagation achieved by our technique.

As shown in the final state of Figure 9, several state positions demonstrate particularly efficient key dependence. Specifically, the superpolies for positions $A_{3,\chi}[4][0][12]$, $A_{0,\chi}[2][1][7]$, $A_{0,\chi}[4][1][60]$, $A_{0,\chi}[3][2][55]$, and $A_{0,\chi}[3][3][41]$ each depend on no more than three related key bits: $\{k_{10},k_{27}\}$, $\{k_{27},k_{91}\}$, $\{k_{69}\}$, $\{k_{10},k_{69}\}$, and $\{k_{10},k_{27},k_{91}\}$ respectively. This constrained key dependence highlights a significant advantage of our technique.

To clarify our technique, we summarize the procedure in the Algorithm 2:

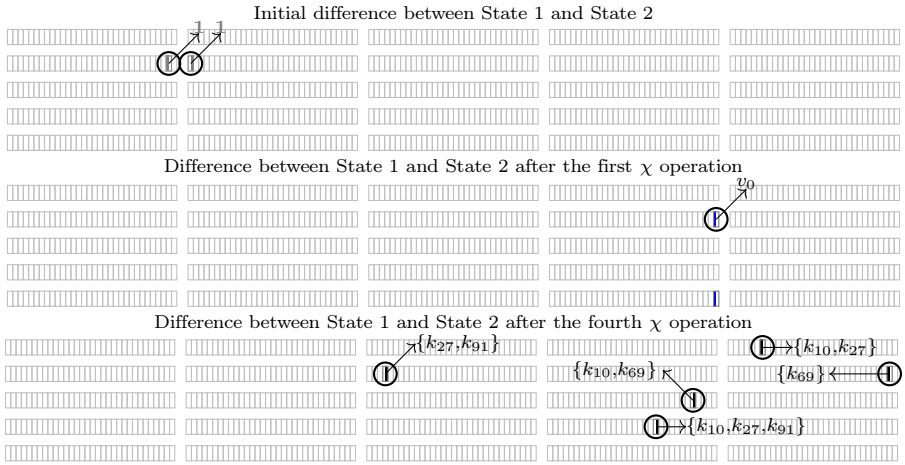


Fig. 9: State difference propagation between Configurations in (Figure 8) across some rounds.

Algorithm 2: Automated cube-attack-like Cryptanalysis with Exploiting State Differences

Input: : Initialized keyed Keccak construction
Output: Candidate secret key values
:
Initialization Phase;
  Load initial state
    with cube variables and related keys obtained from MILP model in [BDL+19];
  Designate axis cube variable and axis related key based on Section 4.2;
  Configure auxiliary variables:;
    Align in same columns as related keys;
    Enforce non-multiplication with cube variables in first round;
Analysis Phase;
  Configure two different initial states;
  Identify non-zero superpolies with minimal key dependency using proposed tool;
  Initialize sets:;
    $G \leftarrow$ All guessing key values;
    $A \leftarrow$ All auxiliary variable values;
Preprocessing Phase;
for *each* $g \in G$ **do**
  Compute differential cube summation for both initial states;
  Store result in lookup table L indexed by summation value;
Online Phase;
for *each* $a \in A$ **do**
  Compute differential cube summation S_a;
  if S_a *exists in* L **then**
    Retrieve candidate (g,a) pairs where $L[g] = S_a$;

4.3 Compute Limitations

In the technique we present, we have computationally verified the improvements. This required developing a dedicated tool capable of recovering all terms in all superpolies.

As evident, the number of terms grows exponentially with additional rounds, making superpoly recovery computationally infeasible beyond 4-round given our available computing server. However, we observed that tracking all terms is unnecessary and a significant portion can be eliminated to reduce computational overhead.

According to Equation 2, computing the superpoly requires only the polynomial expression of $t \cdot P_t(x)$. As this section explains, this includes exclusively terms containing the term t. More precisely, recovering $t \cdot P_t(x)$ requires only terms containing all cube variables; other terms cancel out during cube summation.

The χ operation's algebraic structure implies that the degree at round r is 2^r. Since in cube-attack-like cryptanalysis the first round is linearized, we only need to retain terms containing 2^{r-1} cube variables in round r. Terms with fewer variables do not contribute to $t \cdot P_t(x)$. To be more specific, the Boolean Algebraic Normal Form of χ can be represented as equation 9, where b_i represents the output bit corresponding to input a_i:

$$b_i = a_i + (a_{i+1} + 1) \cdot a_{i+2} \tag{9}$$

As evident from Equation 9, only the product term $a_{i+1} \cdot a_{i+2}$ generates terms containing 2^{r-1} cube variables in the r-th round. Consequently, for rounds beyond the first (which is linearized), the χ operation can be simplified to $b_i = a_{i+1} \cdot a_{i+2}$ while maintaining analytical precision. The original χ formulation must be retained for the first round.

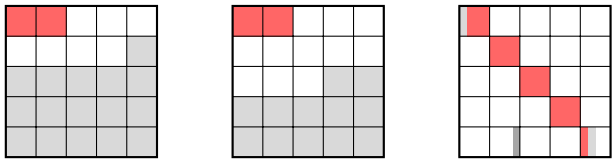
By implementing these optimizations, we significantly reduced the computational complexity of our tool. This enabled successful recovery of superpolies for all output bits up to 5-round in structures like Keccak-MAC-512, leveraging the available degree of freedom and our computing resources.

5 Applications

In this section, we evaluate our technique on four Keccak-based constructions: Keccak-MAC-512, Keccak-MAC-384, Ketje Minor. For Ketje Minor, we successfully verify the improvements up to 4 rounds, while for the other variants the improvements extend to 5 rounds.

In the following, we provide detailed evaluation results for keyed Keccak variants: Keccak-MAC-512, Keccak-MAC-384 and Ketje Minor. In Figure 10 red positions represent key bits; light gray areas indicate capacity bits for MAC structures and padded or encoded bits for Ketje structures; white positions correspond to nonce or message bits. This color scheme explicitly determines that cube variables must be placed exclusively in white areas.

All computations were performed on a server with the following specifications: **CPU:** Dual AMD EPYC 7302 processors (32 physical cores, 64 threads) operating at 3.0 GHz base frequency with 256 MB shared L3 cache, **Memory:** 503 GB of system RAM, **Storage:** 455 GB NVMe scratch disk, Operating System: Ubuntu 22.04 LTS, Compiler: `g++` version 11.4.0 with `-O3` optimization. We have implemented our tool in C++.



Keccak-MAC-512 Keccak-MAC-384 Ketje Minor

Fig. 10: The degrees of freedom in Keccak-MAC-512, Keccak-MAC-384, Ketje Minor.

5.1 Application to 5-round Keccak-MAC-512

Applying the automated technique from [BDL+19] to 5-round Keccak-MAC-512 yields 30 related key bits, requiring guessing of 15 key bits during preprocessing phase and 15 auxiliary variables in online phase. The time complexity for recovering these 30 related key bits is $2^{15} \times 2^{16} = 2^{31}$ for both preprocessing and online phases, with a total complexity of 2^{32} and memory complexity of 2^{15}.

For complete key recovery, we exploit the symmetric property of the permutation along the z-axis. By considering all i-bit rotations ($0 \leq i < 64$) in the z-axis, we focus on the case where $0 \leq i < 4$. This allows recovery of 120 independent key bits out of the total 128-bit key, leaving the remaining 8 bits for exhaustive search (2^8 complexity). Consequently, the total time complexity for full 128-bit key recovery becomes $4 \times 2^{32} + 2^8 \approx 2^{34}$, with memory complexity 2^{15}. The data complexity, because of doing cube summation over 15 auxiliary variables is $2^{15} \times 2^{16} = 2^{31}$.

To implement our technique, we first select the bit positions $A_0[2][0][0]$, $A_0[2][1][0]$, $A_0[3][0][34]$, and $A_0[3][1][34]$ to place the axis cube variable v_0. We designate k_{60} (located at $A_0[0][0][60]$) and k_{65} (located at $A_0[1][0][1]$) as the axis related keys. Following the methodology outlined in Section 4.2, our MILP model generated a configuration involving 33 related key bits.

For correct state difference computation in both preprocessing and online phases, we maintain fixed positions for the axis related keys across both phases. We partition the remaining related key bits into two distinct sets 18 guessed key bits for the preprocessing phase and 17 auxiliary variables for the online phase as follows:

$$\begin{aligned} &\text{guessed keys} = \{k_{60}, k_{65}, k_5, k_{18}, k_{25}, k_{27}, k_{33}, k_{41}, k_{43}, \\ &\quad k_{50}, k_{68}, k_{76}, k_{84}, k_{92}, k_{106}, k_{114}, k_{122}, k_{124}\} \\ &\text{keys corresponding to auxiliary variables} = \{k_{60}, k_{65}, k_{10}, k_{19}, k_{26}, \\ &\quad k_{28}, k_{34}, k_{42}, k_{49}, k_{58}, k_{66}, k_{69}, k_{83}, k_{91}, k_{98}, k_{107}, k_{121}\} \end{aligned} \tag{10}$$

Our analysis of state differences reveals distinct key dependencies in each phase in the preprocessing phase the bit position $A_{4,\chi}[4][3][15]$ exhibits dependence on only four key bits: $\{k_{27}, k_{76}, k_{92}, k_{106}\}$ and in the online Phase the corresponding bit position shows dependence on eight key bits corresponding to auxiliary variables: $\{k_{19}, k_{26}, k_{28}, k_{34}, k_{42}, k_{66}, k_{98}, k_{121}\}$.

Our cube summation process, operating on two similar states, incurs a computational cost of 2^{17} (compared to 2^{16} for single-state summation). The preprocessing phase requires $2^4 \times 2^{17} = 2^{21}$ operations for the 4 key bits. The online

phase follows the same principle as the preprocessing phase. This results in a base of $2^8 \times 2^{17} = 2^{25}$ operations for testing all combinations of the 8 key bits.

Since the lookup table in the preprocessing phase is constructed for a single bit ($A_{4,\chi}[4][3][15]$), false positives are mitigated by repeating the cube summation over 2^4 distinct plaintexts. Given that the bit $A_{4,\chi}[4][3][15]$ depends on four key bits in the preprocessing phase and eight key bits in the online phase, the process must be repeated for eight distinct plaintexts, with the corresponding results stored in the lookup table. Consequently, the time complexity in the preprocessing phase is $8 \times 2^{21} = 2^{24}$, while in the online phase it is $8 \times 2^{25} = 2^{28}$.

The described process recovers 12 related key bits initially. Through 9 rotations in the z-axis, we extend this to recover 108 key bits, with the remaining 20 bits determined via exhaustive search (2^{20} complexity). The complete attack requires:

- Time complexity: $9 \times (2^{24} + 2^{28}) + 2^{20} \approx 2^{31.2}$
- Memory complexity: $8 \times 2^4 \approx 2^7$
- Data complexity: $8 \times 2^8 \times 2^{17} \approx 2^{28}$

We emphasize that this process admits multiple optimization opportunities through different configurations. Systematic examination of alternative axis cube variable placements in the initial state may yield improved complexity results. Table 3 presents the configuration parameters used in our 5-round attack on Keccak-MAC-512.

Table 3: Configuration to attack on 5-round Keccak-MAC-512

Axis cube variable
$A_0[2][0][0] = A_0[2][1][0] = A[3][0][34] = A[3][1][34] = v_0$
Axis related Keys
$A[0][0][60] = k_{60}$, $A[1][0][1] = k_{65}$
Bit positions that differ in two states
$A_0[0][1][60]$, $A_0[1][1][1]$
Cube variables
$A_0[2][0][14] = A_0[2][1][14] = v_1, A_0[2][0][22] = A_0[2][1][22] = v_2, A_0[2][0][23] = A_0[2][1][23] = v_3,$ $A_0[2][0][29] = A_0[2][1][29] = v_4, A_0[2][0][37] = A_0[2][1][37] = v_5, A_0[2][0][38] = A_0[2][1][38] = v_6,$ $A_0[2][0][45] = A_0[2][1][45] = v_7, A_0[2][0][53] = A_0[2][1][53] = v_8, A_0[3][0][3] = A_0[3][1][3] = v_9,$ $A_0[3][0][11] = A_0[3][1][11] = v_{10}, A_0[3][0][12] = A_0[3][1][12] = v_{11}, A_0[3][0][19] = A_0[3][1][19] = v_{12}$ $A_0[3][0][26] = A_0[3][1][26] = v_{13}, A_0[3][0][27] = A_0[3][1][27] = v_{14}, A_0[3][0][35] = A_0[3][1][35] = v_{15}$
Other Related key bits
$k_{19}, k_{26}, k_{27}, k_{28}, k_{34}, k_{42}, k_{66}, k_{76}, k_{92}, k_{98}, k_{106}, k_{121}$
Guessed Key bits
$k_{27}, k_{76}, k_{92}, k_{106}$
Auxiliary variables
$A_0[0][1][19] = a_0, A_0[0][1][26] = a_1, A_0[0][1][28] = a_2, A_0[0][1][34] = a_3, A_0[0][1][42] = a_4,$ $A_0[1][1][2] = a_5, A_0[1][1][34] = a_6, A_0[1][1][57] = a_7,$

5.2 Application to 5-round Keccak-MAC-384

Applying the automated MILP technique from [BDL+19] to 5-round Keccak-MAC-384 yields a configuration with 15 related key bits.

The preprocessing phase requires $2^7 \times 2^{16} = 2^{23}$ operations, while the online phase needs $2^8 \times 2^{16} = 2^{24}$ computations, yielding a combined time complexity of $2^{24.5}$. For full 128-bit key recovery, we employ 8 rotational variants to extract 120 bits plus 2^8 exhaustive search, achieving the time complexity $7 \times 2^{24.5} + 2^{23} \approx 2^{27.4}$, the memory complexity 2^7 and the data complexity $2^{8+16} = 2^{24}$.

Following Section 4.2's methodology, our MILP model generated a 19 related key bits configuration. The preprocessing phase revealed the bit position $A_{4,\chi}[3][1][22]$ dependents on $\{k_3, k_{118}\}$, resulting in a preprocessing cost of $2^2 \times 2^{17} = 2^{19}$ operations.

In the online phase, the corresponding bit position depends on k_{39}, k_{96}, k_{106}, resulting in a computational cost of $2^3 \times 2^{17} = 2^{20}$ operations. Following the explanation in Section 5.1, the total time complexity for recovering 5 key bits is $3 \times (2^{19} + 2^{20}) \approx 2^{22.1}$. For full key recovery, the complexity increases to $22 \times 2^{22.1} + 2^{18} \approx 2^{26.5}$, with a memory complexity of $3 \times 2^2 = 2^{3.5}$ and a data complexity of $3 \times 2^{3+17} = 2^{21.5}$.

Table 4 summarizes the configuration parameters for our 5-round attack on Keccak-MAC-384.

Table 4: Configuration to attack on 5-round Keccak-MAC-384

Axis cube variable
$A_0[1][1][59] = A_0[1][2][59] = A[4][0][19] = A[4][1][19] = v_0$
Axis related Keys
$A[1][0][35] = k_{99}$, $A[1][0][60] = k_{124}$
Bit positions that differ in two states
$A_0[1][1][35]$, $A_0[1][1][60]$
Cube variables
$A_0[2][1][1] = v_1, A_0[2][2][1] = v_1, A_0[2][0][4] = v_2, A_0[2][1][4] = v_3, A_0[2][2][4] = v_2 + v_3,$ $A_0[2][0][7] = v_4, A_0[2][1][7] = v_5, A_0[2][2][7] = v_4 + v_5, A_0[2][0][10] = v_6, A_0[2][2][10] = v_6,$ $A_0[2][0][13] = v_7, A_0[2][2][13] = v_7, A_0[2][1][37] = v_8, A_0[2][2][37] = v_8, A_0[2][0][40] = v_9,$ $A_0[2][2][40] = v_9, A_0[2][0][43] = v_{10}, A_0[2][2][43] = v_{10}, A_0[4][0][0] = v_{11}, A_0[4][1][0] = v_{11}$ $A_0[4][0][16] = v_{12}, A_0[4][1][16] = v_{12}, A_0[4][0][22] = v_{13}, A_0[4][1][22] = v_{13}, A_0[4][0][26] = v_{14},$ $A_0[4][1][26] = v_{14}, A_0[4][0][42] = v_{15}, A_0[4][1][42] = v_{15}$
Other Related key bits
$k_3, k_{39}, k_{42}, k_{96}$
Guessed Key bits
k_3, k_{42}
Auxiliary variables
$A_0[0][1][39] = a_0, A_0[1][1][32] = a_1$

5.3 Application to 4-round Ketje Minor

For Ketje Minor, our computational resources were insufficient to recover 5-round superpolies due to RAM limitations. When applying the automated MILP technique from [BDL+19] to 4-round Ketje Minor, we obtained a configuration involving 21 related key bits. Through analysis, we identified output bit positions with reduced key-bit dependencies, most notably $A_{4,\chi}[1][3][4]$, which depends on only 10 key bits: $\{k_6, k_{43}, k_{56}, k_{59}, k_{65}, k_{68}, k_{69}, k_{77}, k_{78}, k_{106}\}$.

By dividing the 10 related key bits into 5 guessing key bits and 5 auxiliary variables, both preprocessing and online phases require $5 \times 2^5 \times 2^8 \approx 2^{15.3}$ operations

each, resulting in a combined time complexity of $2^{16.3}$. For complete 128-bit key recovery, we utilize 12 rotational variants to recover 120 key bits combined with a 2^8 exhaustive search, yielding final complexities of: time $12\times 2^{16.3}+2^8\approx 2^{20}$, memory $5\times 2^5\approx 2^{7.3}$, and data $5\times 2^{5+8}=2^{15.3}$.

For 4-round, we followed the technique proposed in Section 4 and observed that the bit position $A_{4,\chi}[3][3][7]$ dependents only on $\{k_{65},k_{69}\}$. The preprocessing complexity to recover 1 guessing-key is $2\times 2^9=2^{10}$, also the same for recovering 1 auxiliary variable in the online phase.

The total complexity for full key recovery is $58\times(2^{10}+2^{10})+2^{12}\approx 2^{16.8}$. The memory and data complexities are 2 and data $2^{1+9}=2^{10}$, respectively.

Table 5 summarizes the configuration parameters for our 4-round attack on Ketje Minor.

Table 5: Configuration to attack on 4-round Ketje Minor

Axis cube variable
$A_0[2][0][0]=A_0[2][1][0]=v_0$
Axis related Key
$A[2][2][18]=k_{74}$
Bit positions that differ in two states
$A_0[2][0][18]$
Cube variables
$A_0[2][1][9]=v_1,A_0[2][3][9]=v_1,A_0[2][1][28]=v_2,A_0[2][3][28]=v_2,A_0[2][1][31]=v_3,$ $A_0[2][3][31]=v_3,A_0[4][0][29]=v_4,A_0[4][1][29]=v_5,A_0[4][2][29]=v_6,$ $A_0[4][3][29]=v_4+v_5+v_6,A_0[4][1][30]=v_7,A_0[4][3][30]=v_7$
Other Related key bits
k_{65},k_{69}
Guessed Key bits
k_{65}
Auxiliary variables
$A_0[2][0][13]=a_0$

6 Conclusion and Future Works

In this paper, we have presented an improved cube-attack-like cryptanalysis technique that reduces output-bit dependency on related key bits, leading to more efficient attacks against Keccak-based constructions. Our method, building upon [BDL+19], demonstrates practical advantages through comprehensive attacks on 5-round versions of Keccak-MAC-512, Keccak-MAC-384, and 4-round of Ketje Minor. While current implementations are limited to 5-round, the technique shows potential for extension to 6-round given enhanced computational resources or tool improvement.

Future work may focus on:

- Extending the attack to higher rounds through tool optimizations and enhanced computational resources;
- Theoretical analysis of superpoly independence from specific key bits to establish formal security bounds, As the experiments suggest, complexity improvement on a higher number of rounds is expected;

– In all of our experiments, we also observed that none of the superpolies depend on axis related keys. Theoretically proving this or understanding the underlying logic may open a new research direction.

References

BDH+17. G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer. Farfalle: Parallel permutation-based cryptography. *IACR Trans. Symmetric Cryptol.*, 2017(4):1–38, 2017. URL: https://tosc.iacr.org/index.php/ToSC/article/view/801, doi:10.13154/tosc.v2017.i4.1-38.

BDL+19. W. Bi, X. Dong, Z. Li, R. Zong, and X. Wang. Milp-aided cube-attack-like cryptanalysis on keccak keyed modes. *Des. Codes Cryptogr.*, 87(6):1271–1296, 2019. doi:10.1007/S10623-018-0526-X.

BDP+11. G. Berton, J. Daemen, M. Peeters, G. V. Assche, and R. V. Keer. The keccak reference, 2011. URL: http://keccak.noekeon.org.

BDP+16a. G. Berton, J. Daemen, M. Peeters, G. V. Assche, and R. V. Keer. Ketje v2: A Submission to the CAESAR Competition (2016). CAESAR Competition Round 3 Submission, 2016. URL: http://competitions.cr.yp.to/round3/ketjev2.pdf.

BDP+16b. G. Berton, J. Daemen, M. Peeters, G. V. Assche, and R. V. Keer. Keyak v2: A Submission to the CAESAR Competition (2016). CAESAR Competition Round 3 Submission, 2016. URL: https://keccak.team/files/Keyakv2-doc2.2-diff.pdf.

DEMS21. C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, 2021. doi:10.1007/s00145-021-09398-9.

DHAK18. J. Daemen, S. Hoffert, G. V. Assche, and R. V. Keer. The design of Xoodoo and Xoofff. *IACR Trans. Symmetric Cryptol.*, 2018:1–38, 2018. doi:10.13154/tosc.v2018.i4.1-38.

DHP+20. J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer. Xoodyak: A lightweight cryptographic scheme. *IACR Trans. Symmetric Cryptol.*, 2020(S1):60–87, 2020. doi:10.13154/tosc.v2020.iS1.60-87.

DMP+15. I. Dinur, P. Morawiecki, J. Pieprzyk, M. Srebrny, and M. Straus. Cube attacks and cube-attack-like cryptanalysis on the round-reduced keccak sponge function. In *EUROCRYPT 2015*, volume 9056, pages 733–761. Springer, 2015. doi:10.1007/978-3-662-46800-5_28.

DS09. I. Dinur and A. Shamir. Cube attacks on tweakable black box polynomials. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009. doi:10.1007/978-3-642-01001-9_16.

Gur13. Gurobi Optimization, LLC. Gurobi optimizer reference manual. https://www.gurobi.com, 2013.

HWX+17. S. Huang, X. Wang, G. Xu, M. Wang, and J. Zhao. Conditional cube attack on reduced-round Keccak sponge function. In *Advances in Cryptology – EUROCRYPT 2017*, volume 10211 of *Lecture Notes in Computer Science*, pages 259–288. Springer, 2017. doi:10.1007/978-3-319-56614-6_9.

Lai94. X. Lai. Higher order derivatives and differential cryptanalysis. In *Communications and Cryptography: Two Sides of One Tapestry*, pages 227–233. Springer, 1994. doi:10.1007/978-1-4615-2694-0_23.

LBDW17. Z. Li, W. Bi, X. Dong, and X. Wang. Improved conditional cube attacks on Keccak keyed modes with MILP method. In *Advances in Cryptology – ASIACRYPT 2017*, volume 10624 of *Lecture Notes in Computer Science*, pages 99–127. Springer, 2017. doi:10.1007/978-3-319-70694-8_4.

LDB+19. Z. Li, X. Dong, W. Bi, K. Jia, X. Wang, and W. Meier. New conditional cube attack on Keccak keyed modes. *IACR Trans. Symmetric Cryptol.*, 2019(2):94–124, 2019. doi:10.13154/tosc.v2019.i2.94-124.

Nat23. National Institute of Standards and Technology (NIST). Lightweight Cryptography Standardization Process: NIST Selects Ascon. NIST Website, February 07, 2023. URL: https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon.

SG18. L. Song and J. Guo. Cube-attack-like cryptanalysis of round-reduced keccak using MILP. *IACR Trans. Symmetric Cryptol.*, 2018(3):182–214, 2018. doi:10.13154/tosc.v2018.i3.182-214.

SGSL18. L. Song, J. Guo, D. Shi, and S. Ling. New MILP modeling: Improved conditional cube attacks on Keccak-based constructions. In *Advances in Cryptology – ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 65–95. Springer, 2018. doi:10.1007/978-3-030-03329-3_3.

VV23. M. Vaziri and V. Velichkov. Conditional cube key recovery attack on round-reduced Xoodyak. In *ACNS Workshops*, volume 13907 of *Lecture Notes in Computer Science*, pages 43–62. Springer, 2023. doi:10.1007/978-3-031-41181-6_3.

WY05. X. Wang and H. Yu. How to break MD5 and other hash functions. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005. doi:10.1007/11426639_2.

WYY05. X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005. doi:10.1007/11535218_2.

ZLD+20. H. Zhou, Z. Li, X. Dong, K. Jia, and W. Meier. Practical key-recovery attacks on round-reduced Ketje Jr, Xoodoo-AE and Xoodyak. *Comput. J.*, 63(8):1231–1246, 2020. doi:10.1093/comjnl/bxz152.