Meet-in-the-Middle Preimage Attacks on Sponge-based Hashing

Lingyue Qin[1], Jialiang Hua[2], Xiaoyang Dong[2], Hailun Yan[3], and Xiaoyun Wang[2,4,5]

[1] BNRist, Tsinghua University, Beijing, China
`qinly@tsinghua.edu.cn`
[2] Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China
`{huajl18,xiaoyangdong,xiaoyunwang}@tsinghua.edu.cn`
[3] School of Cryptology, Univesity of Chinese Academy of Sciences, Beijing, China
`hailun.yan@ucas.ac.cn`
[4] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education,Shandong University, Jinan, China
[5] School of Cyber Science and Technology, Shandong University, Qingdao, China

Abstract. The Meet-in-the-Middle (MitM) attack has been widely applied to preimage attacks on Merkle-Damgård (MD) hashing. In this paper, we introduce a generic framework of the MitM attack on sponge-based hashing. We find certain bit conditions can significantly reduce the diffusion of the unknown bits and lead to longer MitM characteristics. To find good or optimal configurations of MitM attacks, e.g., the bit conditions, the neutral sets, and the matching points, we introduce the bit-level MILP-based automatic tools on `Keccak`, `Ascon` and `Xoodyak`. To reduce the scale of bit-level models and make them solvable in reasonable time, a series of properties of the targeted hashing are considered in the modelling, such as the linear structure and CP-kernel for `Keccak`, the Boolean expression of Sbox for `Ascon`. Finally, we give an improved 4-round preimage attack on `Keccak-512/SHA3`, and break a nearly 10 years' cryptanalysis record. We also give the first preimage attacks on 3-/4-round `Ascon-XOF` and 3-round `Xoodyak-XOF`.

Keywords: MitM · Automatic Tool · `Keccak/SHA3` · `Ascon`· `Xoodyak`

1 Introduction

The Meet-in-the-Middle (MitM) attack proposed by Diffie and Hellman in 1977 [21] is a generic technique for cryptanalysis of symmetric-key primitives. The essence of the MitM attack is actually an efficient way to exhaustively search a space for the right candidate based on the birthday attack, i.e., dividing the whole space into two independent subsets (also known as neutral sets) and then finding matches from the two subsets. Suppose $E_K(\cdot)$ to be a block cipher whose size is n-bit such that $C = E_K(P) = F_{K_2}(F_{K_1}(P))$, where $K = K_1\|K_2$ has n bits, and K_1 and K_2 are independent key materials of $n/2$ bits. For a given plaintext-ciphertext pair (P, C), a naive exhaust search attack needs a time complexity 2^n to find the key. However, the birthday-paradox based MitM attack computes independently $F_{K_1}(P)$ and $F^{-1}_{K_2}(C)$ with independent guesses of K_1 and K_2, and searches collision between $F_{K_1}(P)$ and $F^{-1}_{K_2}(C)$ to find the K with a time complexity about $2^{n/2}$. In the past decades, the MitM attack has been widely applied to the cryptanalysis on block ciphers [49,29,12,39] and hash functions [57,2,33]. In the meantime, various techniques have been introduced to improve the framework of MitM attack, such as internal state guessing [29], splice-and-cut [2], initial structure [57], bicliques [11], 3-subset MitM [12], indirect-partial matching [2,57], sieve-in-the-middle [14], match-box [31], dissection [23], differential-aided MitM [40,30], etc. Till now, the MitM attack and its variants have broken MD4 [43,33], MD5 [57], KeeLoq [38], HAVAL [4,58], GOST [39], GEA-1/2 [7,1], etc.

At CRYPTO 2011 and 2016, several ad-hoc automatic tools [13,18] were proposed for MitM attacks. At IWSEC 2018, Sasaki [55] introduced MILP-based MitM attacks on GIFT block cipher. At EUROCRYPT 2021, Bao et al. [5] introduced the MILP-based automatic search framework for MitM preimage attacks on `AES`-like hashing, whose compression function is built from `AES`-like block cipher or permutations. At CRYPTO 2021, Dong et al. [27] further extended Bao et al.'s model into key-recovery and collision attacks. At CRYPTO 2022, Schrottenloher and Stevens [59] simplified the language of the automatic model and applied it in both classic and quantum settings. Bao et al. [6] considered the MitM attack in a view of the superposition states.

When applying to hash functions, most of the MitM attacks targeted on Merkle-Damgård [50,16] domain extender, whose compression function is usually built from a block cipher and PGV hashing modes [53], such as Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO) and Miyaguchi-Preneel (MP). The goal of the MitM attack is to find a sequence of internal states which satisfy a closed computational path: there is a relation between the value before the first round and the value after the last round in previous applications of MitM attacks. For example, when attacking block-cipher based hashing modes (DM, MMO, MP, etc.), the closed computation path is computed across the first and last rounds via the feed-forward mechanism of the hashing modes. While when attacking block ciphers, the closed computation path is linked via an encryption/decryption oracle. As shown in Figure 1, one starts by separating the path in two chunks (splice-and-cut): the backward chunk and the forward chunk depending on different neutral sets. Both chunks form independent computation paths. One then finds a partial match between them at certain round.

When considering the new hashing mode, i.e., sponge-based hashing, there is no feed-forward mechanism anymore, e.g. `Keccak`. We have to try novel ways to build the so-called closed computational path for MitM attack. Most preimage attacks on sponge-based hashing focus on the analysis on `Keccak`/`SHA-3` with linearization technique. At ASIACRYPT 2016, Guo et al. [34] introduced the *linear structure* technique to linearise several rounds of `Keccak` and derived upto 4-round preimage attacks. Later, Guo et al.'s attacks were improved by the cross-linear structure [45] at ToSC 2017 and *allocating approach* [44] at EURO-

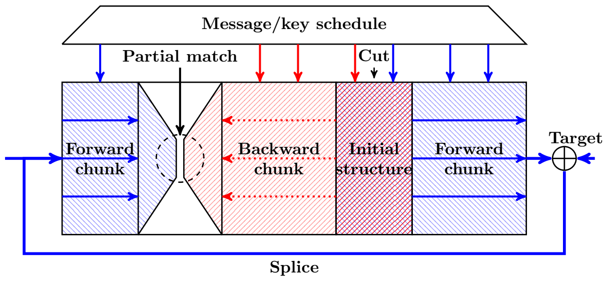

Fig. 1: The closed computation path of the MitM attack

CRYPT 2019. Further improvements in this line were proposed in [54,36,47,46]. At EUROCRYPT 2021, Dinur [22] gave the preimage attacks on Keccak by solving multivariate equation systems. Additionally, theoretical preimage attacks marginally better than exhaustive attacks were studied in [8,51].

At INDOCRYPT 2011, an MitM attack on 2-round Keccak is given by Naya-Plasencia et al. [52]. However, their MitM attack is different from what we are considering. In [52], after computing the inverse of one round Keccak from the target, partial internal states are known. Then, Naya-Plasencia et al. divide the message block into many independent parts and compute forward independently for each part until the known internal states, and filter the messages. Naya-Plasencia et al.'s attack is more like a divide-and-conquer and the birthday attack is not used.

Our Contributions.

We apply the birthday-paradox based MitM attack[6], which has been widely used to attack Merkle-Damgård [50,16] hashing with PGV modes as well as block ciphers, to the sponge-based hash functions. Additionally, by applying bit conditions, the diffusion of the two neutral sets can be controlled and reduced, and therefore lead to longer MitM characteristics. Finally, we propose a generic MitM framework with conditions for sponge-based hash functions.

To apply our framework to Keccak, Ascon-XOF and Xoodyak-XOF, we have to search sound configurations for MitM attack, including the choice of the two neutral sets, the bit conditions, the matching points etc. As Keccak, Ascon are bit-level hashing, we introduce the bit-level MILP-based automatic tools to detect those configurations. Different from the previous byte-level MitM MILP models [5,27,59], the bit-level modelling usually leads to huge scale MILP models and makes it hard to solve in reasonable time. Therefore, we explore detailed properties of dedicated ciphers to reduce the models. For Keccak, we apply the linear structures in starting states and CP-kernel properties in matching phase. For Ascon, the Boolean expressions of the Sbox are explored in the starting

[6]The Demirci-Selçuk MitM attacks [17,28,19,20,10] are not considered in this paper, which is a quite different technique.

states and matching points. In previous modellings [5,27], cells depending on both two neutral sets are always regarded as useless and unknown in the MitM attack. The unknown cells can significantly reduce the number of known cells when propagating (somewhat like polluting), since any known cells will become unknown by operating with the unknown cells. Inspired by the *indirect-partial matching* technique [2,57], the cells depending on the additions of the two neutral sets are also useful and should not be regarded as unknown in the automatic searching models. Therefore, we introduce new constraints for those kinds of cells and reduce the polluting speed of the unknown cells.

At last, we derive a better 4-round preimage attack on `Keccak/SHA3-512` than Morawiecki et al.'s rotational cryptanalysis [51] at FSE 2013, that breaks their nearly 10 years' record. While previous preimage attack on `Keccak-512` with linear structure techniques [34] (including improvements with various techniques [47,35,54]) only reaches 3 rounds. For `Ascon-XOF`, the first 3-round and 4-round preimage attacks are given. For `Xoodyak-XOF`, the first 3-round preimage attack is given. A summary of the related results are given in Table 1.

Table 1: A Summary of the Attacks. Lin. Stru.: Linear Structure. MitM: MitM Attack. Diff.: Differential. †: this attack ignores the padding bits.

Target	Attacks	Methods	Rounds	Time	Memory	Ref.
`Keccak-512`	Preimage	Lin.Stru.	2	2^{384}	-	[34]
		Lin.Stru.	2	2^{321}	-	[54]
		Lin.Stru.	2	2^{270}	-	[47]
		Lin.Stru.	2	2^{252}	-	[35]
		Lin.Stru.	3	2^{482}	-	[34]
		Lin.Stru.	3	2^{475}	-	[54]
		Lin.Stru.	3	2^{452}	-	[47]
		Lin.Stru.	3	2^{426}	-	[35]
		Rotational	4	2^{506}	-	[51]
		MitM	4	$2^{504.58}$	2^{108}	Sect. 4
	Collision	Diff.	2	Practical	-	[52]
		Diff.	3	Practical	-	[24]
`Xoodyak-XOF`	Preimage	Neural	1	-	-	[48]
		MitM	3	$2^{125.06}$	2^{97}	Sect. 5
`Ascon-XOF`	Preimage	MitM	3	$2^{120.58}$	2^{39}	Sect. E
		MitM	4	$2^{126.4}$	2^{45}	Sect. 6.2
		Algebraic†	6	$2^{127.3}$	-	[26]
	Collision	Diff.	2	2^{103}	-	[32]

Comparison to Schrottenloher and Stevens's MitM attack. At CRYPTO 2022, Schrottenloher and Stevens [59] introduced preimage attacks on `SPHINCS+-Haraka` [3], which is sponge-based hashing with permutation `Haraka` [42]. Their MitM attack computes from the two ends, i.e., the known inner part and the target, to the middle matching part. Combining with the guess-and-determine technique, they derived a 3.5-round (out of 5 rounds) quantum preimage attack on `SPHINCS+-Haraka`. As stated in [59, Section 3.1], their frameworks do not lead to interesting results on Ascon [26]. The reason may be that for Keccak or Ascon the inverse of one round is not as easy as Haraka. Our framework mainly uses the forward computation, and leads to novel results on both Keccak and Ascon.

2 Preliminaries

In the section, we give some brief descriptions of the Meet-in-the-Middle attack, the sponge-based hash function, the Keccak-f permutation, Ascon-Hash and Ascon-XOF, Xoodyak and Xoodoo permutation.

2.1 The Meet-in-the-Middle Attack

Since the pioneering works on preimage attacks on Merkle–Damgård hashing, e.g. MD4, MD5, and HAVAL [43,57,2,33], techniques such as *splice-and-cut* [2], *initial structure* [57] and *indirect-partial matching* have been invented to significantly improve the MitM approach. As shown in Figure 1, in the MitM attack, the compression function is divided at certain intermediate rounds (initial structure) into two chunks. One chunk is computed forward (named as forward chunk), and the other is computed backward (named as backward chunk). One of them is computed across the first and last rounds via the feed-forward mechanism of the hashing mode, and they end at a common intermediate round (partial matching point) and form a closed computation path of the MitM attack. In each of the chunks, the computation involves at least one distinct message word (or a few bits of it), such that they can be computed over all possible values of the involved message word(s) independently from the message word(s) involved in the other chunk (the distinct words are called neutral words). In the initial structure, the two chunks overlapped and the neutral words for both chunks appear simultaneously, but still, the computations of the two chunks on the neutral words are independent. The highlevel framework is in Figure 1, which can be divided into three configurations:

1. The chunk separation – the positions of initial structure and matching points.
2. The neutral sets – the selection on the two neutral sets (denoted as 🟥 or 🟦 sets), which determines the degree of freedom (DoF) for each chunk.
3. The matching – the deterministic relation used for matching, which determines the filtering ability (degree of matching, DoM).

After setting up the configurations, the basic attack procedure goes as follows.

1. Choose constants for the initial structure.
2. For all 2^{d_1} values of 🟥 neutral set, compute backward from the initial structure to the matching points to generate a table L_1, whose indices are the values for matching, and the elements are the values of 🟥 neutral set.
3. Similarly, build L_2 for 2^{d_2} values of 🟦 neutral set with forward computation.
4. Check whether there is an m-bit match on indices between L_1 and L_2.
5. For the pairs surviving the partial match, check for a full-state match. Steps 1-5 will be repeated until we find a full match.

The attack complexity. Denote the size of the target by h, and the number of bits for the match by m. An MitM episode is performed with time $2^{\max(d_1,d_2)} + 2^{d_1+d_2-m}$ and the total time complexity of the attack is:

$$2^{h-(d_1+d_2)} \cdot (2^{\max(d_1,d_2)} + 2^{d_1+d_2-m}) \simeq 2^{h-\min(d_1,d_2,m)}. \tag{1}$$

To illustrate how the MitM attack works, we detail the 7-round attack on AES-hashing of Sasaki [56] in Supplementary Material A as an example.

2.2 The Sponge-based Hash Function

The sponge construction [9] shown in Figure 2 takes a variable-length message as input and produces a digest of any desired length. The b-bit internal state is composed of an outer part of r bits and an inner part of c bits, where r is the rate and c is the capacity. To evaluate the sponge function, one proceeds in three phases with an inner permutation f:

1. **Initialization:** Initialize the b-bit state with the given value (all 0's for `Keccak`) before proceeding the message blocks.
2. **Absorbing:** The message is padded and split into blocks of r bits. Absorb each r-bit block M_i by XORing into the internal state.
3. **Squeezing:** Produce the digest.

We named the hash functions with sponge construction as the sponge-based hash functions, e.g. `Keccak` [9], `Ascon` [26], `Xoodyak` [15], to name a few.

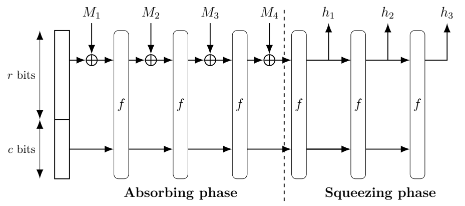

Fig. 2: The sponge construction

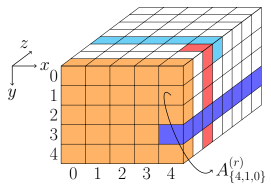

Fig. 3: The `Keccak` state

2.3 The `Keccak`-f Permutations

The `Keccak` hash function family [9] specifies 7 `Keccak` permutations, denoted `Keccak`-$f[b]$, where $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ is the width of the permutation. In this paper, we focus on `Keccak`-$f[1600]$, where the state A is arranged as 5×5 64-bit lanes as depicted in Figure 3. Let $A^{(r)}_{\{x,y,z\}}$ denote the bit located at the x-th column, y-th row and z-th lane in the round r ($r \geq 0$), where $0 \leq x \leq 4,\ 0 \leq y \leq 4,\ 0 \leq z \leq 63$. For `Keccak` in the rest of this paper, all the coordinates are considered modulo 5 for x and y and modulo 64 for z. The function `Keccak`-$f[1600]$ consists of 24 rounds which consists of five operations $\iota \circ \chi \circ \pi \circ \rho \circ \theta$. Denote the internal states of round r as

$$A^{(r)} \xrightarrow{\theta} \theta^{(r)} \xrightarrow{\rho} \rho^{(r)} \xrightarrow{\pi} \pi^{(r)} \xrightarrow{\chi} \chi^{(r)} \xrightarrow{\iota} A^{(r+1)}.$$

Operations in each round are:

$$\begin{aligned}
\theta: &\quad \theta^{(r)}_{\{x,y,z\}} = A^{(r)}_{\{x,y,z\}} \oplus \sum\nolimits_{y'=0}^{4} (A^{(r)}_{\{x-1,y',z\}} \oplus A^{(r)}_{\{x+1,y',z-1\}}),\\
\rho: &\quad \rho^{(r)}_{\{x,y,z\}} = \theta^{(r)}_{\{x,y,z-\gamma[x,y]\}},\\
\pi: &\quad \pi^{(r)}_{\{y,2x+3y,z\}} = \rho^{(r)}_{\{x,y,z\}},\\
\chi: &\quad \chi^{(r)}_{\{x,y,z\}} = \pi^{(r)}_{\{x,y,z\}} \oplus (\pi^{(r)}_{\{x+1,y,z\}} \oplus 1) \cdot \pi^{(r)}_{\{x+2,y,z\}},\\
\iota: &\quad A^{(r+1)} = \chi^{(r)} \oplus RC_r,
\end{aligned} \tag{2}$$

where $\gamma[x,y]$'s are constants given in Table 4 of Supplementary Material B, RC_r is round-dependent constant.

The `Keccak` and SHA3 Hash Function. The `Keccak` hash function follows the sponge construction. For `Keccak`$[r,c,d]$, the capacity is c, the bitrate is r and the diversifier is d. NIST standardized four SHA3-l versions ($l \in 224, 256, 384, 512$), where $c = 2l$ and $r = 1600 - 2l$. The only difference of `Keccak` and SHA3 is the padding rule. The padding rule for `Keccak` is padding the message with '10*1', which is a single bit 1 followed by the minimum number of 0 bits followed by a single bit 1, to make the whole length to a multiple of $(1600 - 2l)$. For SHA3, the message is padded with '0110*1'. However, the padding rule does not affect the final time complexity of our attack.

2.4 `Ascon-Hash` and `Ascon-XOF`

The `Ascon` family [26] includes the hash functions `Ascon-Hash` and `Ascon-Hasha` as well as the extendable output functions `Ascon-XOF` and `Ascon-XOFa` with sponge-based modes of operations.

`Ascon Permutation.` The inner permutation applies 12 round functions to a 320-bit state. The state A is split into five 64-bit words, and denote $A^{(r)}_{\{x,y\}}$ to be the x-th (column) bit of the y-th (row) 64-bit word, where $0 \le y \le 4$, $0 \le x \le 63$. The round function consists of three operations p_C, p_S and p_L. Denote the internal states of round r as $A^{(r)} \xrightarrow{p_S \circ p_C} S^{(r)} \xrightarrow{p_L} A^{(r+1)}$.

- **Addition of Constants** p_C**:** $A^{(r)}_{\{*,2\}} = A^{(r)}_{\{*,2\}} \oplus RC_r$.
- **Substitution Layer** p_S**:** For each x, this step updates the columns $A^{(r)}_{\{x,*\}}$ using the 5-bit Sbox. Assume the S-box maps $(a_0, a_1, a_2, a_3, a_4) \in \mathbb{F}_2^5$ to $(b_0, b_1, b_2, b_3, b_4) \in \mathbb{F}_2^5$, where a_0 is the most significant bit. The algebraic

normal form (ANF) of the Sbox is as follows:

$$\begin{aligned}
b_0 &= a_4a_1 + a_3 + a_2a_1 + a_2 + a_1a_0 + a_1 + a_0,\\
b_1 &= a_4 + a_3a_2 + a_3a_1 + a_3 + a_2a_1 + a_2 + a_1 + a_0,\\
b_2 &= a_4a_3 + a_4 + a_2 + a_1 + 1,\\
b_3 &= a_4a_0 + a_4 + a_3a_0 + a_3 + a_2 + a_1 + a_0,\\
b_4 &= a_4a_1 + a_4 + a_3 + a_1a_0 + a_1.
\end{aligned} \tag{3}$$

– **Linear Diffusion Layer p_L:**

$$\begin{aligned}
A^{(r+1)}_{\{*,0\}} &\leftarrow S^{(r)}_{\{*,0\}} \oplus (S^{(r)}_{\{*,0\}} \ggg 19) \oplus (S^{(r)}_{\{*,0\}} \ggg 28),\\
A^{(r+1)}_{\{*,1\}} &\leftarrow S^{(r)}_{\{*,1\}} \oplus (S^{(r)}_{\{*,1\}} \ggg 61) \oplus (S^{(r)}_{\{*,1\}} \ggg 39),\\
A^{(r+1)}_{\{*,2\}} &\leftarrow S^{(r)}_{\{*,2\}} \oplus (S^{(r)}_{\{*,2\}} \ggg 1) \oplus (S^{(r)}_{\{*,2\}} \ggg 6),\\
A^{(r+1)}_{\{*,3\}} &\leftarrow S^{(r)}_{\{*,3\}} \oplus (S^{(r)}_{\{*,3\}} \ggg 10) \oplus (S^{(r)}_{\{*,3\}} \ggg 17),\\
A^{(r+1)}_{\{*,4\}} &\leftarrow S^{(r)}_{\{*,4\}} \oplus (S^{(r)}_{\{*,4\}} \ggg 7) \oplus (S^{(r)}_{\{*,4\}} \ggg 41).
\end{aligned}$$

`Ascon-Hash` and `Ascon-XOF`. The state A is composed of the outer part with 64 bits $A_{\{*,0\}}$ and the inner part 256 bits $A_{\{*,i\}}$ $(i = 1, 2, 3, 4)$. For `Ascon-Hash`, the output size is 256 bits, and the security claim is 2^{128}. For `Ascon-XOF`, the output can have arbitrary length and the security claim against preimage attack is $\min(2^{128}, 2^l)$, where l is the output length. In this paper, we target on `Ascon-XOF` with a 128-bit hash value and a 128-bit security claim against preimage attack.

2.5 `Xoodyak` and `Xoodoo` Permutation

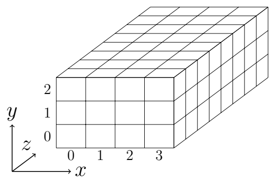

Fig. 4: Toy version of the `Xoodoo` state. The order in y is opposite to `Keccak`

Internally, `Xoodyak` makes use of the `Xoodoo` permutation [15], whose state (shown in Figure 4) bit denoted by $A^{(r)}_{\{x,y,z\}}$ is located at the x-th column, y-th row and z-th lane in the round r, where $0 \leq x \leq 3,\ 0 \leq y \leq 2,\ 0 \leq z \leq 31$. For `Xoodoo`, all the coordinates are considered modulo 4 for x, modulo 3 for y and modulo 32 for z. The permutation consists of the iteration of a round function $R = \rho_{\text{east}} \circ \chi \circ \iota \circ \rho_{\text{west}} \circ \theta$. The number of rounds is a parameter, which is 12 in `Xoodyak`. Denote the internal states of the round r as

$$A^{(r)} \xrightarrow{\theta} \theta^{(r)} \xrightarrow{\rho_{\text{west}}} \rho^{(r)} \xrightarrow{\iota} \iota^{(r)} \xrightarrow{\chi} \chi^{(r)} \xrightarrow{\rho_{\text{east}}} A^{(r+1)}.$$

$$\begin{aligned}
\theta: \quad & \theta^{(r)}_{\{x,y,z\}} = A^{(r)}_{\{x,y,z\}} \oplus \sum\nolimits_{y'=0}^{2} (A^{(r)}_{\{x-1,y',z-5\}} \oplus A^{(r)}_{\{x-1,y',z-14\}}),\\
\rho_{\text{west}}: \quad & \rho^{(r)}_{\{x,0,z\}} = \theta^{(r)}_{\{x,0,z\}},\ \rho^{(r)}_{\{x,1,z\}} = \theta^{(r)}_{\{x-1,1,z\}},\ \rho^{(r)}_{\{x,2,z\}} = \theta^{(r)}_{\{x,2,z-11\}},\\
\iota: \quad & \iota^{(r)}_{\{0,0,z\}} = \rho^{(r)}_{\{0,0,z\}} \oplus RC_r,\ \text{where } RC_r \text{ is round-dependent constant,} \qquad (4)\\
\chi: \quad & \chi^{(r)}_{\{x,y,z\}} = \iota^{(r)}_{\{x,y,z\}} \oplus (\iota^{(r)}_{\{x,y+1,z\}} \oplus 1) \cdot \iota^{(r)}_{\{x,y+2,z\}},\\
\rho_{\text{east}}: \quad & A^{(r+1)}_{\{x,0,z\}} = \chi^{(r)}_{\{x,0,z\}},\ A^{(r+1)}_{\{x,1,z\}} = \chi^{(r)}_{\{x,1,z-1\}},\ A^{(r+1)}_{\{x,2,z\}} = \chi^{(r)}_{\{x-2,2,z-8\}}.
\end{aligned}$$

`Xoodyak` can serve as a `XOF`, i.e. `Xoodyak-XOF`, which offers arbitrary output length l. The preimage resistance is $\min(2^{128}, 2^l)$. We target on `Xoodyak-XOF` with output of 128-bit hash value and 128-bit absorbed message block.

3 Meet-in-the-Middle Attack on Sponge-based Hashing

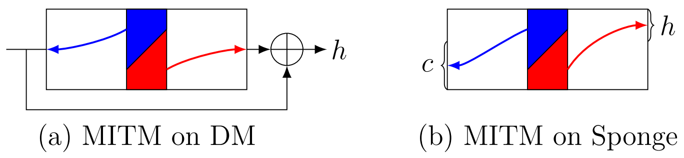

Fig. 5: Differences in MitM attack on PGV and sponge hash functions

The essence of the Meet-in-the-Middle attack is actually an efficient way to exhaustively search a space for the right one based on the birthday attack. Taken the MitM attack on DM construction (Figure 5(a)) as an example, suppose the size of the internal state is n, the size of the output is h ($n \geq h$). In the perspective of exhaust search attack, one chooses a random internal state to verify if it leads to the given h-bit target. After searching a space of 2^h internal states, one will find the preimage. In the MitM attacks, as shown in Figure 5(a), the attacker starts from the internal state in the middle, which is divided into two independent forward and backward chunks (marked in red and blue, respectively). One computes the two chunks independently until the matching point to filter the wrong internal states. The detials are given in Section 2.1.

When considering the sponge-based hashing, if we start from some similar internal states in the middle (as shown in Figure 5(b)) to search preimage with the given target, the internal state has to satisfy not only the target in forward computation, but also the c-bit inner part in backward computation. In other words, we have to search a space with $2^{(h+c)}$ internal states to meet the target and the inner part. Taking the complexity of exhaustive search (i.e., 2^h) into consideration, it may be not a good idea to search a $(h+c)$-bit space even with MitM. For sponge-based hashing, we do not follow the conventional start-from-the-middle way to drive the MitM attack. We try to search a more compact space to find the preimage. In fact, we choose to search the r-bit outer part. With the known c-bit inner part, a search of h-bit subspace of the outer part (if $r > h$) with MitM method is enough to find the preimage. The highlevel framework of the MitM approach is shown in Figure 6 and highlighted in the green box. Since we start from the outer part and try to satisfy the h-bit target, only forward computations are involved. Like the MitM attack in Section 2.1, we need to specify the configurations: the two neutral sets of the outer part, the two independent forward computation chunks, the matching points. We may partially solve the inverse of the permutation from the h-bit target to get some internal bits. Thereafter, by forward computing the two independent chunks until those internal bits, the deterministic relations on the two neutral sets were established, which act as the matching point.

3.1 The Conditions in the MitM Attack

The key point of the MitM is to extend the number of rounds of the independent computation path for blue or red neutral words. For `Keccak`, we have $\chi : b_i = a_i \oplus (a_{i+1} \oplus 1) \cdot a_{i+2}$. Supposing a_{i+1} is blue neutral word and a_i is red neutral word, then b_i depends on both blue and red neutral words if $a_{i+2} = 1$, otherwise b_i only depends on the red neutral words a_i. That is what we called "conditions".

Setting conditions to control the characteristic can trace back to Wang et al.'s collision attacks with message modification techniques [60,61]. Then, conditions are applied to enhance the probability of the differentials, i.e., the conditional differential cryptanalysis [41]. Later, conditions are used to reduce the diffusion of the cube variables in dynamic cube attack [25] and conditional cube attacks [37]. In MitM attack, the conditions were used to build MitM attacks [57,2] on MD/SHA hashing with ARX structure. For modular addition $X + Y = Z$ ($X, Y, Z \in \mathbb{F}_2^{32}$), particularly the computation of i-th and $(i+1)$-th bits, assume that the carry from $(i-1)$-th bit to i-th bit is 0. Then, the $(i+1)$-th bit of Z is computed as $Z_{\{i+1\}} = X_{\{i+1\}} \oplus Y_{\{i+1\}} \oplus X_{\{i\}} \cdot Y_{\{i\}}$. When $X_{\{i+1\}}$ is blue neutral word and $Y_{\{i\}}$ is red neutral word, the idea of making $X_{\{i\}} = 0$ as a condition so that $Z_{\{i+1\}}$ is only affected by blue neutral word.

In this paper, we try to apply conditions to reduce the diffusion of the red/blue neutral words, and expect to find longer MitM characteristics. In our MitM attack on sponge-based hashing, the conditions usually depend on bits from both inner part (capacity) and outer part (rate). In order to modify certain conditions, we have to modify bits from the inner part. Therefore, as shown in Figure 6, we place the MitM attack in the processing of the last message block and modify the conditions determined by inner part by randomly changing the first several message blocks (e.g. M_1 in Figure 6). Suppose there are μ conditions only determined by the inner part[7], the probability to find one right M_1 satisfying all the conditions is about $2^{-\mu}$.

[7]Note that, if the conditions are determined by both outer part and inner part, then for given inner part, it is possible to change the message block (i.e., M_2 in Figure 6) to modify the conditions.

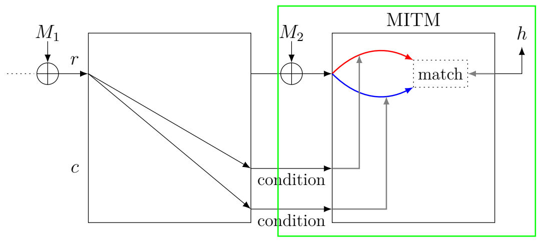

Fig. 6: Framework of the MitM attack on sponge-based hashing

Once we find one right M_1, we assign arbitrary values to all bits except those neutral bits for M_2. Then, an MitM episode is performed:

1. Suppose the two neutral sets of the outer part are of 2^{d_1} and 2^{d_2} values, respectively, which are marked by the 🟥 and 🟦 color. For each of 2^{d_1} values, compute forward to the matching points.
2. For each of 2^{d_2} values, compute forward to the matching points.
3. Compute backward with the known h-bit target to the matching points to derive an m-bit matching.
4. Filter states.

The complexity of the MitM episode is $2^{\max(d_1,d_2)}+2^{d_1+d_2-m}$, which actually checks $2^{d_1+d_2}$ M_2. In order to find a preimage of h, we have to repeat the episode for $2^{h-(d_1+d_2)}$ times. After the conditions in inner part are satisfied, suppose there are 2^{η} non-neutral bits in M_2 that provide 2^{η} MitM episodes. If $\eta+d_1+d_2<h$, we have to find $2^{h-(\eta+d_1+d_2)}$ M_1, which all satisfy the conditions in the inner part of the last permutation. The total time complexity is

$$2^{h-(\eta+d_1+d_2)}\cdot 2^{\mu}+2^{h-(d_1+d_2)}\cdot(2^{\max(d_1,d_2)}+2^{d_1+d_2-m}). \qquad (5)$$

4 MitM Preimage Attack on `Keccak`

This section first gives some techniques and properties in previous preimage attacks on `Keccak`. Then we propose our MILP model for the MitM attack on `Keccak`. As an application, we mount a 4-round preimage attack on `Keccak`-512.

4.1 Preliminaries on `Keccak`

Most previous preimage attacks on `Keccak`/`SHA3` are with the linearization technique. The *linear structure* technique allows to linearize the underlying permutation of `Keccak` for several rounds. In our attack, we also apply the *linear structure* technique proposed by Guo et al. [34] to linearize one round of `Keccak`, in order to speed up the search.

Linear Structure. We give an example to explain the *linear structure* technique. As shown in Fig. 7, the variables $v_{0,z}$ and $v_{1,z}$ $(0 \leq z \leq 63)$ are allocated as $A^{(0)}_{\{0,0,z\}} = v_{0,z}$, $A^{(0)}_{\{0,1,z\}} = v_{0,z} \oplus c_{0,z}$, $A^{(0)}_{\{2,0,z\}} = v_{1,z}$, $A^{(0)}_{\{2,1,z\}} = v_{1,z} \oplus c_{1,z}$, where $c_{0,z}$ and $c_{1,z}$ $(0 \leq z \leq 63)$ are constants. After the θ operation, the variables will not diffuse. After the π operation, any two variables are not adjacent in a row, and all outputs of $A^{(1)}$ are linear. To further reduce the diffusion of the variables in χ operation, one can add restricted constraints to the value of constant bits. For example, for the row $\pi^{(0)}_{\{*,0,z\}}$, setting two bit conditions $\pi^{(0)}_{\{1,0,z\}} = 0$ and $\pi^{(0)}_{\{4,0,z\}} = 1$, the other bits except $A^{(1)}_{\{0,0,z\}}$ in $A^{(1)}_{\{*,0,z\}}$ will be constants. Those conditions can be satisfied by modifying the message block and inner part.

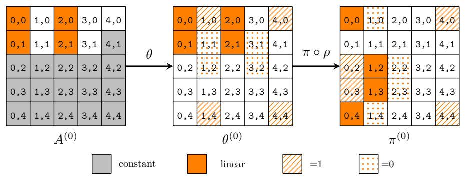

Fig. 7: The linear structure of 1-round `Keccak`-512

Properties of the Sbox χ. Guo et al. proposed several properties of the Sbox χ, which help to mount preimage attacks on `Keccak` [34]. Those properties can be also applied in our MitM attack, so we give a brief introduction in the following. Assume $\chi : \mathbb{F}_2^5 \to \mathbb{F}_2^5$ maps $(a_0, a_1, a_2, a_3, a_4)$ to $(b_0, b_1, b_2, b_3, b_4)$ as

$$b_i = a_i \oplus (a_{i+1} \oplus 1) \cdot a_{i+2}, \tag{6}$$

where all indices are modulo 5. The inverse operation χ^{-1} is

$$a_i = b_i \oplus (b_{i+1} \oplus 1) \cdot (b_{i+2} \oplus (b_{i+3} \oplus 1) \cdot b_{i+4}). \tag{7}$$

Property 1. [34] When there are three known consecutive output bits, two linear equations of the input bits can be constructed. E.g., assuming that (b_0, b_1, b_2) are known, two linear equations on (a_0, a_1, a_2, a_3) are constructed as

$$b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2, \quad b_1 = a_1 \oplus (b_2 \oplus 1) \cdot a_3. \tag{8}$$

4.2 MILP Model of the MitM Preimage Attack on `Keccak`

In previous MILP models [5,27] of the MitM attack, each bit can take one of the four colors (⬛, 🟥, 🟦, and ⬜). Generally, the ⬜ bits depending on both 🟥 and 🟦, are unknown and useless in the MitM attack. In our model, bits whose Boolean expression depending on the addition of 🟥 and 🟦 (not multiplied) can also be used in our MitM attack, which is known as *the indirect-partial matching technique* [57,2] and ignored by previous automatic models [5,27]. Therefore, we introduce another color, i.e., 🟩. In our MILP models, there are five colors (⬛, 🟥, 🟦, 🟩, and ⬜). We first inroduce a new efficient encoding scheme of those colors. Applying the *linear structure* technique, we can skip the first round and constuct the model from the second round. Then we model the attribute propagation of the five colors over the five operaions in each round of `Keccak`. We also model the matching phase with the CP-kernel for `Keccak`. With all above works, we bulid an automatic MILP model for the MitM preimage attack on `Keccak`.

Encoding Scheme. Since there are 5 colors to encode in the MILP model, the previous 2-bit encoding method [5,27] is not suitable and we introduce a new 3-bit encoding scheme, i.e., each bit is represented by three 0-1 variables $(\omega_0, \omega_1, \omega_2)$:

- `Gray` ⬛: $(1,1,1)$, global constant bits,
- `Red` 🟥: $(0,1,1)$, bits determined by ⬛ bits and 🟥 bits of starting state,
- `Blue` 🟦: $(1,1,0)$, bits determined by ⬛ bits and 🟦 bits of starting state,
- `Green` 🟩: $(0,1,0)$, bits determined by ⬛ bits, 🟥 bits and 🟦 bits, but the expression does not contain the product of 🟥 and 🟦 bits,
- `White` ⬜: $(0,0,0)$, bits dependent on the product of 🟦 and 🟥 bits.

We set ω_1 to 0 for ⬜ and to 1 for any other color (⬛, 🟥, 🟦, 🟩). So ⬜ bit can be quickly detected by the value of ω_1. Then we set ω_0 to 1 for (⬛, 🟦) and to 0 for other color (🟥, 🟩, ⬜). Similarly for ω_2.

Modelling the Starting State with the Linear Structure Technique. In the starting state, each of the 1600 bits takes one color of ⬛, 🟥 and 🟦. We allocate variables $\alpha_{\{x,y,z\}}$ and $\beta_{\{x,y,z\}}$ for the bit with index $\{x,y,z\}$, where $\alpha_{\{x,y,z\}} = 1$ if and only if the bit is 🟦 and $\beta_{\{x,y,z\}} = 1$ if and only if the bit is 🟥. Therefore, we can compute the initial DoF by $\lambda_{\mathcal{B}} = \sum \alpha_{\{x,y,z\}}, \lambda_{\mathcal{R}} = \sum \beta_{\{x,y,z\}}$. For `Keccak`, we apply the 1-round restricted linear structure as the example given in Section 4.1. Denote the starting state after XORing message block by $A^{(0)}$. The bits in $A^{(0)}_{\{0,0,z\}}$, $A^{(0)}_{\{0,1,z\}}$, $A^{(0)}_{\{2,0,z\}}$ and $A^{(0)}_{\{2,1,z\}}$ can be colored as 🟦 or 🟥. And the remaining bits of $A^{(0)}$ need to be ⬛. In order to control the diffusion of θ operation, $A^{(0)}_{\{0,0,z\}}$ and $A^{(0)}_{\{0,1,z\}}$ should be the same color and the $A^{(0)}_{\{0,0,z\}} \oplus A^{(0)}_{\{0,1,z\}}$ should be constant, which consumes one degree of freedom. Similarly for $A^{(0)}_{\{2,0,z\}}$ and $A^{(0)}_{\{2,1,z\}}$. Thereafter, the coloring pattern keeps the same over the first θ operation. Then with the conditions set in $\pi^{(0)}$ as introduced in Section 4.1, we can omit the first χ operation and construct the model from $A^{(1)}$ only considering the linear operation $\pi \circ \rho$ from $A^{(0)}$.

Modelling the Attribute Propagation. The round function of `Keccak` consists of five operations θ, ρ, π, χ and ι. The linear operations ρ and π only change the position of each bit of the state. The operation ι can be ignored because it will not change the coloring pattern.

Modelling the θ operation. At first, we give the rule of `XOR` with an arbitrary number of inputs under the new coloring scheme. We name the rule of by `XOR-RULE`, which involves five rules:

1. `XOR-RULE-1`: If the inputs have (0,0,0) □ bit , the output is □.
2. `XOR-RULE-2`: If the inputs are all (1,1,1) ■ bits , the output is ■.
3. `XOR-RULE-3`: If the inputs have (1,1,0) 🟦 (≥ 1) and ■ (≥ 0) bits, the output will be 🟦 without consuming DoF, or ■ by consuming one DoF of 🟦.
4. `XOR-RULE-4`: If the inputs have (0,1,1) 🟥 (≥ 1) and ■ (≥ 0) bits, the output will be 🟥 without consuming DoF, or ■ by consuming one DoF of 🟥.
5. `XOR-RULE-5`: If the inputs have (0,1,0) 🟩 bits, or have at least two kinds of 🟦, 🟥 and 🟩 bits:
   (a) the output can be 🟩 without consuming DoF.
   (b) the output can be 🟦 (or 🟥) by consuming one DoF of 🟥 (or 🟦).
   (c) the output can be ■ by consuming one DoF of 🟥 and one DoF of 🟦.

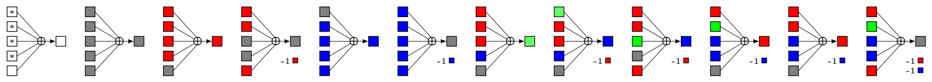

Fig. 8: `5-XOR-RULE` ("*" represents the bit can be any color)

We give some valid coloring patterns of 5 inputs of `XOR`, which are named by `5-XOR-RULE`, as shown in Figure 8. Similar to previous MitM attacks [5], we can use some new variables to identify which rule is applied in different cases. We define three 0-1 variables ν_i ($i \in \{0,1,2\}$), where $\nu_0 = 1$ if and only if all the ω_0's of the 5 input bits are 1, similar to the cases $i = 1, 2$. The above five rules can be represented by (ν_0, ν_1, ν_2):

1. $(\nu_0, \nu_1, \nu_2) = (*, 0, *)$, `XOR-RULE-1` is applied.
2. $(\nu_0, \nu_1, \nu_2) = (1, 1, 1)$, `XOR-RULE-2` is applied.
3. $(\nu_0, \nu_1, \nu_2) = (1, 1, 0)$, `XOR-RULE-3` is applied.
4. $(\nu_0, \nu_1, \nu_2) = (0, 1, 1)$, `XOR-RULE-4` is applied.
5. $(\nu_0, \nu_1, \nu_2) = (0, 1, 0)$, `XOR-RULE-5` is applied.

Taking $(\nu_0, \nu_1, \nu_2) = (1, 1, 0)$ as an example. $\nu_1 = 1$ means that all the ω_1's of the input bits are 1 and there is no □ bit. $\nu_0 = 1$ means that there only may have the ■ or 🟦. $\nu_2 = 1$ means that there must have 🟦 or 🟩 or □. Based on the above analysis, we can deduce that when $(\nu_0, \nu_1, \nu_2) = (1, 1, 0)$, there only have 🟦 and ■ and the number of 🟦 is greater than or equal to one, where `XOR-RULE-3` is applied. Denote the output bit as $(\omega_0^O, \omega_1^O, \omega_2^O)$ and the consumed DoF of 🟥 bits and 🟦 bits are $(\delta_{\mathcal{R}}, \delta_{\mathcal{B}})$, we can derive

$$\begin{cases} \omega_0^O - \nu_0 \geq 0, \quad -\omega_0^O + \nu_1 \geq 0, \\ \omega_1^O - \nu_1 = 0, \\ \omega_2^O - \nu_2 \geq 0, \quad -\omega_2^O + \nu_1 \geq 0, \end{cases} \qquad \begin{cases} \delta_{\mathcal{R}} - \omega_0^O + \nu_0 = 0, \\ \delta_{\mathcal{B}} - \omega_2^O + \nu_2 = 0. \end{cases} \tag{9}$$

In the θ operation, the expression of the output bit is XORing 11 input bits. If we directly compute the XORing value of the 11 input bits, we may double counting the consumption of DoF. For example, given (x, z), we compute $\theta^{(r)}_{\{x,0,z\}}$ and $\theta^{(r)}_{\{x,1,z\}}$ by $\theta^{(r)}_{\{x,y,z\}} = A^{(r)}_{\{x,y,z\}} \oplus \sum_{y'=0}^{4} (A^{(r)}_{\{x-1,y',z\}} \oplus A^{(r)}_{\{x+1,y',z-1\}})$. If bits in the common formula $\sum_{y'=0}^{4} (A^{(r)}_{\{x-1,y',z\}} \oplus A^{(r)}_{\{x+1,y',z-1\}})$ are only determined by 🟥 bits, and $A^{(r)}_{\{x,0,z\}}$, $A^{(r)}_{\{x,1,z\}}$ are ⬛ bits, we can let the summation bit of $\sum_{y'=0}^{4} (A^{(r)}_{\{x-1,y',z\}} \oplus A^{(r)}_{\{x+1,y',z-1\}})$ be ⬛ by consuming only one DoF of 🟥. Thereafter, the two output bits $\theta^{(r)}_{\{x,0,z\}}$, $\theta^{(r)}_{\{x,1,z\}}$ will be ⬛. However, if we directly set the two output bits $\theta^{(r)}_{\{x,0,z\}}$, $\theta^{(r)}_{\{x,1,z\}}$ to be ⬛ by the `XOR-RULE` individually, we have to consume 2 DoF of 🟥. To solve this problem, we depose the θ operation to three steps in our model, as described in the following expressions:

$$
\begin{aligned}
C^{(r)}_{\{x,z\}} &= A^{(r)}_{\{x,0,z\}} \oplus A^{(r)}_{\{x,1,z\}} \oplus A^{(r)}_{\{x,2,z\}} \oplus A^{(r)}_{\{x,3,z\}} \oplus A^{(r)}_{\{x,4,z\}},\\
D^{(r)}_{\{x,z\}} &= C^{(r)}_{\{x-1,z\}} \oplus C^{(r)}_{\{x+1,z-1\}},\\
\theta^{(r)}_{\{x,y,z\}} &= A^{(r)}_{\{x,y,z\}} \oplus D^{(r)}_{\{x,z\}}.
\end{aligned}
$$

At first, we compute the coloring pattern of $C^{(r)}_{\{x,z\}}$. Then, we compute the coloring pattern of $D^{(r)}_{\{x,z\}}$ and compute $\theta^{(r)}_{\{x,y,z\}}$ at last.

Modelling the χ operation. For the χ operation in the round 0, we add conditions to control the diffusion of the 🟥 or 🟦 and the first χ is omitted. For the χ operation from round 1, we build the `SBOX-RULE`. The χ operation maps $(a_0, a_1, a_2, a_3, a_4)$ to $(b_0, b_1, b_2, b_3, b_4)$. According to Equ. (6), $b_i = a_i \oplus (a_{i+1} \oplus 1) \cdot a_{i+2}$. Hence, for each output bit b_i, we determine its color by (a_i, a_{i+1}, a_{i+2}):

1. If there are ⬜ bits in (a_i, a_{i+1}, a_{i+2}), the output is ⬜.
2. If there are all ⬛ bits, the output is ⬛.
3. If there are only 🟦 (≥ 1) and ⬛ (≥ 0) bits, the output will be 🟦.
4. If there are only 🟥 (≥ 1) and ⬛ (≥ 0) bits, the output will be 🟥.
5. If there are 🟩, or more than two kinds of 🟦, 🟥 and 🟩 bits in (a_i, a_{i+1}, a_{i+2}):
   (a) if a_{i+1} and a_{i+2} are all 🟦 (or 🟥), the output is 🟩.
   (b) if a_{i+1} or a_{i+2} is ⬛, the output is 🟩.
   (c) if a_{i+1} and a_{i+2} are of arbitrarily two kinds of 🟦, 🟥, 🟩, the output is ⬜.

The rules `SBOX-RULE` restrict the coloring pattern of $(a_i, a_{i+1}, a_{i+2}, b_i)$ to the subset of $\mathbb{F}_2^{12}$, which is described by the linear inequalities by using the convex hull computation. Some valid coloring patterns are shown in Figure 9.

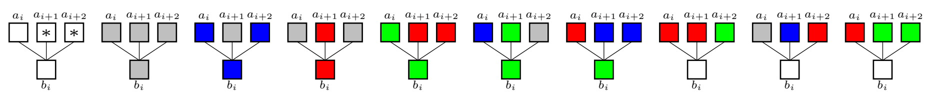

Fig. 9: `SBOX-RULE` for `Keccak` ("*" represents the bit can be any color)

Modelling Matching Phase. Suppose the first 512 bits of $A^{(r+1)}$ are the hash value. In order to attack more rounds, we try to compute certain bits or relations in $A^{(r)}$ by the hash value in $A^{(r+1)}$, to act as the matching points.

Leaked linear relations of $A^{(r)}$. From the 512-bit hash, we know $A^{(r+1)}_{\{*,0,*\}}$ and the first 3 lanes of $A^{(r+1)}_{\{*,1,*\}}$. From $A^{(r+1)}_{\{*,0,*\}}$, we deduce $\pi^{(r)}_{\{*,0,*\}}$ from Equ. (7). Applying the inverse of the operations ρ and π to $\pi^{(r)}_{\{*,0,*\}}$, we can deduce

$$\theta^{(r)}_{\{x,x,z\}} = \pi^{(r)}_{\{x,0,z+\gamma[x,x]\}}, \forall\ 0 \le x \le 4, 0 \le z \le 63. \tag{10}$$

In addition, according to Equ. (8), two linear equations can be deduced from the first three bits of each row of $A^{(r+1)}_{\{*,1,*\}}$, which are given as

$$\begin{aligned} A^{(r+1)}_{\{0,1,z\}} &= \pi^{(r)}_{\{0,1,z\}} \oplus (A^{(r+1)}_{\{1,1,z\}} \oplus 1) \cdot \pi^{(r)}_{\{2,1,z\}}, \\ A^{(r+1)}_{\{1,1,z\}} &= \pi^{(r)}_{\{1,1,z\}} \oplus (A^{(r+1)}_{\{2,1,z\}} \oplus 1) \cdot \pi^{(r)}_{\{3,1,z\}}. \end{aligned}$$

Then applying the inverse of ρ and π, the linear equations are transformed to

$$\begin{aligned} A^{(r+1)}_{\{0,1,z\}} &= \theta^{(r)}_{\{3,0,z-\gamma[3,0]\}} \oplus (A^{(r+1)}_{\{1,1,z\}} \oplus 1) \cdot \theta^{(r)}_{\{0,2,z-\gamma[0,2]\}}, \\ A^{(r+1)}_{\{1,1,z\}} &= \theta^{(r)}_{\{4,1,z-\gamma[4,1]\}} \oplus (A^{(r+1)}_{\{2,1,z\}} \oplus 1) \cdot \theta^{(r)}_{\{1,3,z-\gamma[1,3]\}}. \end{aligned} \tag{11}$$

With the known value $\theta^{(r)}_{\{x,x,z\}}$ $(0 \le x \le 4, 0 \le z \le 63)$ by (10), we add the same known values (underlined) to both sides of (11), which are

$$\begin{aligned} & A^{(r+1)}_{\{0,1,z\}} \oplus \underline{\theta^{(r)}_{\{3,3,z-\gamma[3,0]\}}} \oplus \underline{(A^{(r+1)}_{\{1,1,z\}} \oplus 1)} \cdot \underline{\theta^{(r)}_{\{0,0,z-\gamma[0,2]\}}} \\ & = \theta^{(r)}_{\{3,0,z-\gamma[3,0]\}} \oplus \underline{\theta^{(r)}_{\{3,3,z-\gamma[3,0]\}}} \oplus \underline{(A^{(r+1)}_{\{1,1,z\}} \oplus 1)} \cdot (\theta^{(r)}_{\{0,2,z-\gamma[0,2]\}} \oplus \underline{\theta^{(r)}_{\{0,0,z-\gamma[0,2]\}}}) \\ & A^{(r+1)}_{\{1,1,z\}} \oplus \underline{\theta^{(r)}_{\{4,4,z-\gamma[4,1]\}}} \oplus \underline{(A^{(r+1)}_{\{2,1,z\}} \oplus 1)} \cdot \underline{\theta^{(r)}_{\{1,1,z-\gamma[1,3]\}}} \\ & = \theta^{(r)}_{\{4,1,z-\gamma[4,1]\}} \oplus \underline{\theta^{(r)}_{\{4,4,z-\gamma[4,1]\}}} \oplus \underline{(A^{(r+1)}_{\{2,1,z\}} \oplus 1)} \cdot (\theta^{(r)}_{\{1,3,z-\gamma[1,3]\}} \oplus \underline{\theta^{(r)}_{\{1,1,z-\gamma[1,3]\}}}). \end{aligned} \tag{12}$$

According to the CP-kernel property [9] of operation θ, we deduce

$$\begin{aligned} \theta^{(r)}_{\{3,0,z-\gamma[3,0]\}} \oplus \theta^{(r)}_{\{3,3,z-\gamma[3,0]\}} &= A^{(r)}_{\{3,0,z-\gamma[3,0]\}} \oplus A^{(r)}_{\{3,3,z-\gamma[3,0]\}}, \\ \theta^{(r)}_{\{0,2,z-\gamma[0,2]\}} \oplus \theta^{(r)}_{\{0,0,z-\gamma[0,2]\}} &= A^{(r)}_{\{0,2,z-\gamma[0,2]\}} \oplus A^{(r)}_{\{0,0,z-\gamma[0,2]\}}, \\ \theta^{(r)}_{\{4,1,z-\gamma[4,1]\}} \oplus \theta^{(r)}_{\{4,4,z-\gamma[4,1]\}} &= A^{(r)}_{\{4,1,z-\gamma[4,1]\}} \oplus A^{(r)}_{\{4,4,z-\gamma[4,1]\}}, \\ \theta^{(r)}_{\{1,3,z-\gamma[1,3]\}} \oplus \theta^{(r)}_{\{1,1,z-\gamma[1,3]\}} &= A^{(r)}_{\{1,3,z-\gamma[1,3]\}} \oplus A^{(r)}_{\{1,1,z-\gamma[1,3]\}}. \end{aligned} \tag{13}$$

Combining (12) and (13), there are linear relations on bits in $A^{(r)}$ with the known hash value of $A^{(r+1)}$, which will be used in our matching points:

$$\begin{aligned} & A^{(r)}_{\{3,0,z-\gamma[3,0]\}} \oplus A^{(r)}_{\{3,3,z-\gamma[3,0]\}} \oplus (A^{(r+1)}_{\{1,1,z\}} \oplus 1) \cdot (A^{(r)}_{\{0,2,z-\gamma[0,2]\}} \oplus A^{(r)}_{\{0,0,z-\gamma[0,2]\}}) \\ & \qquad = A^{(r+1)}_{\{0,1,z\}} \oplus \theta^{(r)}_{\{3,3,z-\gamma[3,0]\}} \oplus (A^{(r+1)}_{\{1,1,z\}} \oplus 1) \cdot \theta^{(r)}_{\{0,0,z-\gamma[0,2]\}}, \end{aligned} \tag{14}$$

$$\begin{aligned} & A^{(r)}_{\{4,1,z-\gamma[4,1]\}} \oplus A^{(r)}_{\{4,4,z-\gamma[4,1]\}} \oplus (A^{(r+1)}_{\{2,1,z\}} \oplus 1) \cdot (A^{(r)}_{\{1,3,z-\gamma[1,3]\}} \oplus A^{(r)}_{\{1,1,z-\gamma[1,3]\}}) \\ & \qquad = A^{(r+1)}_{\{1,1,z\}} \oplus \theta^{(r)}_{\{4,4,z-\gamma[4,1]\}} \oplus (A^{(r+1)}_{\{2,1,z\}} \oplus 1) \cdot \theta^{(r)}_{\{1,1,z-\gamma[1,3]\}}. \end{aligned} \tag{15}$$

Observation 1 (Conditions in Matching Points of Keccak) *In* (14)*, if four bits* $(A^{(r)}_{\{3,0,z-\gamma[3,0]\}}, A^{(r)}_{\{3,3,z-\gamma[3,0]\}}, A^{(r)}_{\{0,2,z-\gamma[0,2]\}}, A^{(r)}_{\{0,0,z-\gamma[0,2]\}})$ *in* $A^{(r)}$ *satisfy the following two conditions, there is a 1-bit filter:*

(1) There has no □ *in* $(A^{(r)}_{\{3,0,z-\gamma[3,0]\}}, A^{(r)}_{\{3,3,z-\gamma[3,0]\}}, A^{(r)}_{\{0,2,z-\gamma[0,2]\}}, A^{(r)}_{\{0,0,z-\gamma[0,2]\}})$.

(2) $(A^{(r)}_{\{3,0,z-\gamma[3,0]\}}, A^{(r)}_{\{3,3,z-\gamma[3,0]\}})$ *is of* (🟥,🟦), (🟥,🟩), (🟦,🟩), (⬛,🟩)*, or* (🟩,🟩)*, or opposite order .*

We introduce a binary variable $\delta_{\mathcal{M}}$ to represent whether there is a filtering. Similarly to the `XOR-RULE`, we add three 0-1 variables ν_i $(i \in \{0,1,2\})$, where $\nu_i = 1$ $(i = 0,2)$ if and only if all ω_i's of $(A^{(r)}_{\{3,0,z-\gamma[3,0]\}}, A^{(r)}_{\{3,3,z-\gamma[3,0]\}})$ are 1, and $\nu_1 = 1$ if and only if all ω_1's of $(A^{(r)}_{\{3,0,z-\gamma[3,0]\}}, A^{(r)}_{\{3,3,z-\gamma[3,0]\}}, A^{(r)}_{\{0,2,z-\gamma[0,2]\}}, A^{(r)}_{\{0,0,z-\gamma[0,2]\}})$ are 1. We can derive

$$\begin{cases} \nu_1 - \delta_{\mathcal{M}} \geq 0, \quad -\nu_0 - \delta_{\mathcal{M}} + 1 \geq 0, \quad -\nu_2 - \delta_{\mathcal{M}} + 1 \geq 0, \\ \nu_0 - \nu_1 + \nu_2 + \delta_{\mathcal{M}} \geq 0. \end{cases}$$

The Objective Function. Let $l_{\mathcal{R}}$, and $l_{\mathcal{B}}$ be the accumulated consumption of DoF of 🟦 and 🟥, i.e., $l_{\mathcal{R}} = \sum \delta_{\mathcal{R}}$ and $l_{\mathcal{B}} = \sum \delta_{\mathcal{B}}$. Therefore, we can get $\mathrm{DoF}_{\mathcal{R}} = \lambda_{\mathcal{R}} - l_{\mathcal{R}}$, $\mathrm{DoF}_{\mathcal{B}} = \lambda_{\mathcal{B}} - l_{\mathcal{B}}$. We also have $\mathrm{DoM} = \sum \delta_{\mathcal{M}}$. According to the time complexity given by Equ. (1), we need to maximize the value of $\min\{\mathrm{DoF}_{\mathcal{R}}, \mathrm{DoF}_{\mathcal{B}}, \mathrm{DoM}\}$ to find the optimal attacks. We introduce an auxiliary variable v_{obj}, impose the following constraints, and maximize v_{obj},

$$\{v_{obj} \leq \mathrm{DoF}_{\mathcal{R}}, v_{obj} \leq \mathrm{DoF}_{\mathcal{B}}, v_{obj} \leq \mathrm{DoM}\}. \tag{16}$$

4.3 MitM Preimage Attack on 4-Round Keccak-512

We follow the framework in Figure 6 to perform the attack with two message blocks (M_1, M_2). The MitM attack is placed at the 2nd block. We construct an MILP model for `Keccak-512` following Section 4.2. The source code is given in . . . . By solving with our MILP model, we mount a 4-round MitM preimage attack on `Keccak-512`, shown in Figure 10,16,17,18, which contains 3 additional symbols:

- ⬛, 🟦: there consumes one degree of freedom of 🟥 to let the bit be ⬛ or 🟦.
- ⬛: there consumes one degree of freedom of 🟦 to let the bit be ⬛.
- 🟩: 🟩 bits used for matching.

Conditions in the Linear Structure. State $A^{(0)}$ contains 16 🟦 bits and 216 🟥 bits. We take the similar strategy with [47] to get 1-round linear structure of `Keccak`. We introduce 116 binary variables $v = \{v_0, v_1, \cdots, v_{115}\}$ and 116 binary variables $c = \{c_0, c_1, \cdots, c_{115}\}$. Those variables v_i's and c_i's are placed at the $16 + 216 = 232$ 🟦 and 🟥 bits in $A^{(0)}$ in Figure 10,16,17,18. For example, we set $A^{(0)}_{\{0,0,0\}} = v_0$ and $A^{(0)}_{\{0,1,0\}} = v_0 \oplus c_0$. When we choose $c \in \mathbb{F}_2^{116}$ to be arbitrary constant, the θ operation will act as identity with regard to the 🟦 and 🟥 bits in $A^{(0)}$. To reduce the diffusion of χ in round 0, we need to set some bits conditions in $\pi^{(0)}$ to be constants. According to (6), if a_{i+1} is 🟦 or 🟥, we need to set $a_{i+2} = 0$ and $a_i = 1$. So there are totally $232 \times 2 = 464$ conditions on $\pi^{(0)}$. After the inverse of $\rho \circ \pi$, we determine the positions of bit conditions in $\theta^{(0)}$. Taking the state $\theta^{(0)}_{\{*,*,1\}}$ as an example, there are six bit conditions as

$$\theta^{(0)}_{\{1,0,1\}} = 1,\ \theta^{(0)}_{\{1,2,1\}} = 0,\ \theta^{(0)}_{\{1,4,1\}} = 1, \theta^{(0)}_{\{3,1,1\}} = 0,\ \theta^{(0)}_{\{3,2,1\}} = 0,\ \theta^{(0)}_{\{4,4,1\}} = 1. \tag{17}$$

Above conditions can be converted to those on $A^{(0)}$. The bits in $A^{(0)}$ can be divided into two parts: the bits determined by the outer part (i.e., they can be modified by directly changing the absorbed M_2), and the bits determined by inner part. The equations in Equ. (17) can be transformed to

$$\begin{aligned}
&\boldsymbol{A^{(0)}_{\{1,0,1\}}} \oplus \boldsymbol{c_2} \oplus A^{(0)}_{\{0,2,1\}} \oplus A^{(0)}_{\{0,3,1\}} \oplus A^{(0)}_{\{0,4,1\}} \oplus \boldsymbol{c_1} \oplus A^{(0)}_{\{2,2,0\}} \oplus A^{(0)}_{\{2,3,0\}} \oplus A^{(0)}_{\{2,4,0\}} = 1,\\
&A^{(0)}_{\{1,2,1\}} \oplus \boldsymbol{c_2} \oplus A^{(0)}_{\{0,2,1\}} \oplus A^{(0)}_{\{0,3,1\}} \oplus A^{(0)}_{\{0,4,1\}} \oplus \boldsymbol{c_1} \oplus A^{(0)}_{\{2,2,0\}} \oplus A^{(0)}_{\{2,3,0\}} \oplus A^{(0)}_{\{2,4,0\}} = 0,\\
&A^{(0)}_{\{1,4,1\}} \oplus \boldsymbol{c_2} \oplus A^{(0)}_{\{0,2,1\}} \oplus A^{(0)}_{\{0,3,1\}} \oplus A^{(0)}_{\{0,4,1\}} \oplus \boldsymbol{c_1} \oplus A^{(0)}_{\{2,2,0\}} \oplus A^{(0)}_{\{2,3,0\}} \oplus A^{(0)}_{\{2,4,0\}} = 1,\\
&\boldsymbol{A^{(0)}_{\{3,1,1\}}} \oplus \boldsymbol{c_3} \oplus A^{(0)}_{\{2,2,1\}} \oplus A^{(0)}_{\{2,3,1\}} \oplus A^{(0)}_{\{2,4,1\}} \oplus \boldsymbol{A^{(0)}_{\{4,0,0\}}} \oplus A^{(0)}_{\{4,1,0\}} \oplus A^{(0)}_{\{4,2,0\}} \oplus A^{(0)}_{\{4,3,0\}} \oplus A^{(0)}_{\{4,4,0\}} = 0,\\
&A^{(0)}_{\{3,2,1\}} \oplus \boldsymbol{c_3} \oplus A^{(0)}_{\{2,2,1\}} \oplus A^{(0)}_{\{2,3,1\}} \oplus A^{(0)}_{\{2,4,1\}} \oplus \boldsymbol{A^{(0)}_{\{4,0,0\}}} \oplus A^{(0)}_{\{4,1,0\}} \oplus A^{(0)}_{\{4,2,0\}} \oplus A^{(0)}_{\{4,3,0\}} \oplus A^{(0)}_{\{4,4,0\}} = 0,\\
&A^{(0)}_{\{4,4,1\}} \oplus \boldsymbol{A^{(0)}_{\{3,0,1\}}} \oplus \boldsymbol{A^{(0)}_{\{3,1,1\}}} \oplus A^{(0)}_{\{3,2,1\}} \oplus A^{(0)}_{\{3,3,1\}} \oplus A^{(0)}_{\{3,4,1\}} \oplus \boldsymbol{c_0} \oplus A^{(0)}_{\{0,2,0\}} \oplus A^{(0)}_{\{0,3,0\}} \oplus A^{(0)}_{\{0,4,0\}} = 1,
\end{aligned} \tag{18}$$

where $\boldsymbol{c_0} = \boldsymbol{A^{(0)}_{\{0,0,0\}}} \oplus \boldsymbol{A^{(0)}_{\{0,1,0\}}}$, $\boldsymbol{c_1} = \boldsymbol{A^{(0)}_{\{2,0,0\}}} \oplus \boldsymbol{A^{(0)}_{\{2,1,0\}}}$, $\boldsymbol{c_2} = \boldsymbol{A^{(0)}_{\{0,0,1\}}} \oplus \boldsymbol{A^{(0)}_{\{0,1,1\}}}$, $\boldsymbol{c_3} = \boldsymbol{A^{(0)}_{\{2,0,1\}}} \oplus \boldsymbol{A^{(0)}_{\{2,1,1\}}}$. Given an inner part, the 464 conditions on state $A^{(0)}$ will be a linear system of the $576 - 116 = 460$ variables of M_2 (marked by bold). We compute the rank of the coefficient matrix of the linear system is 232. In other words, through some linear transformations, there are 232 equations out of the total 464 only determined by the bits of inner part. For example, combining the 2nd and 3rd equations in Equ. (18), we can deduce an equation between two inner part bits $A^{(0)}_{\{1,2,1\}} \oplus A^{(0)}_{\{1,4,1\}} = 1$. We have to randomly test 2^{232} M_1 to compute the inner part satisfying the 232 equations. Then for a right inner part, there are $2^{460-232} = 2^{228}$ solutions of M_2, which make all the 464 equations hold. For each solution of M_2, the ⬛ and $c \in \mathbb{F}_2^{116}$ in the outer part will be fixed. Then with the fixed inner part, we can conduct the MitM episodes to filter states.

Consumed Degrees of Freedom. As shown in Figure 10,16,17,18, after adding 464 conditions and consuming 108 🟥 and 8 🟦 degrees of freedom in round 0, we can derive the coloring pattern in $A^{(1)}$. The remaining degrees of freedom for 🟥 and 🟦 are 108 and 8, respectively. We give two examples to explain the consumption of degrees of freedom in the computation from $A^{(1)}$ to $A^{(3)}$.

1. For $\theta^{(1)}_{\{0,0,12\}}$ marked by ⬛ in Figure 11 (part of Figure 10), we set an equation of 🟥 to a constant, which means consuming one DoF of 🟥 to let $\theta^{(1)}_{\{0,0,12\}}$ be

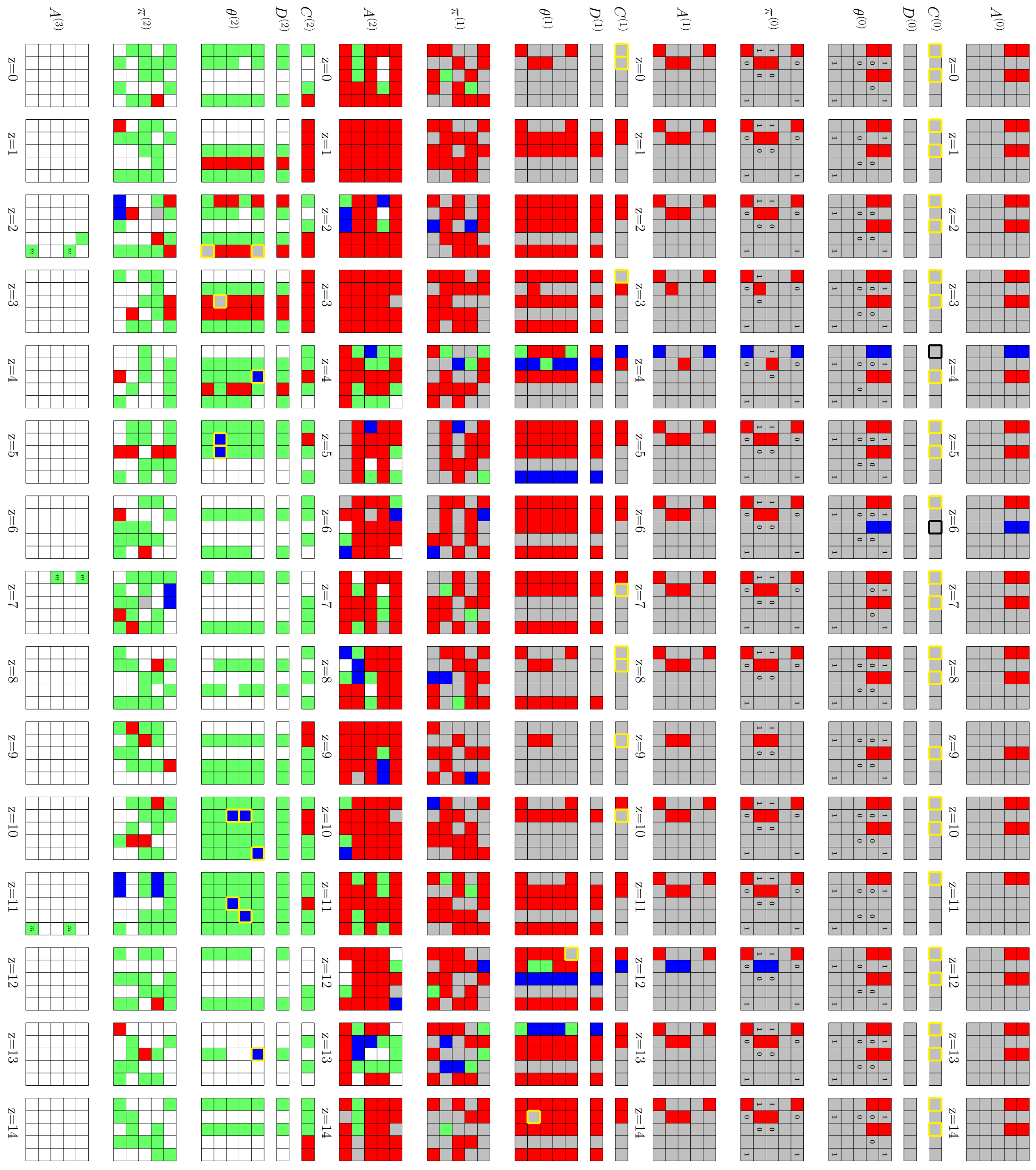

Fig. 10: The MitM preimage attack on 4-round Keccak-512 (part 1)

▨, as listed below:

$$\begin{aligned} A^{(1)}_{\{0,0,12\}} \oplus A^{(1)}_{\{4,0,12\}} \oplus A^{(1)}_{\{4,1,12\}} \oplus A^{(1)}_{\{4,2,12\}} \oplus A^{(1)}_{\{4,3,12\}} \oplus A^{(1)}_{\{4,4,12\}} \\ \oplus A^{(1)}_{\{1,0,11\}} \oplus A^{(1)}_{\{1,1,11\}} \oplus A^{(1)}_{\{1,2,11\}} \oplus A^{(1)}_{\{1,3,11\}} \oplus A^{(1)}_{\{1,4,11\}} = const., \end{aligned} \tag{19}$$

where the bits marked by red are 🟥 and others marked by black are ▨.

2. For $\theta^{(2)}_{\{1,3,5\}}$ marked by 🟦 in Figure 12, we set an equation of 🟥 to a constant, which means consuming one DoF of 🟥 to let $\theta^{(2)}_{\{1,3,5\}}$ be 🟦. Since there have

$$\begin{aligned} \theta^{(2)}_{\{1,3,5\}} =& A^{(2)}_{\{1,3,5\}} \oplus D^{(2)}_{\{1,5\}} = A^{(2)}_{\{1,3,5\}} \oplus C^{(2)}_{\{0,5\}} \oplus C^{(2)}_{\{2,4\}}, \\ C^{(2)}_{\{0,5\}} =& A^{(2)}_{\{0,0,5\}} \oplus A^{(2)}_{\{0,1,5\}} \oplus A^{(2)}_{\{0,2,5\}} \oplus A^{(2)}_{\{0,3,5\}} \oplus A^{(2)}_{\{0,4,5\}}, \\ C^{(2)}_{\{2,4\}} =& A^{(2)}_{\{2,0,4\}} \oplus A^{(2)}_{\{2,1,4\}} \oplus A^{(2)}_{\{2,2,4\}} \oplus A^{(2)}_{\{2,3,4\}} \oplus A^{(2)}_{\{2,4,4\}}. \end{aligned}$$

We can set all the 🟥 involved to be constant:

$$\begin{aligned} A^{(2)}_{\{1,3,5\}} \oplus A^{(2)}_{\{0,0,5\}} \oplus A^{(2)}_{\{0,1,5\}} \oplus A^{(2)}_{\{0,3,5\}} \oplus A^{(2)}_{\{0,4,5\}} \\ A^{(2)}_{\{2,0,4\}} \oplus A^{(2)}_{\{2,1,4\}} \oplus A^{(2)}_{\{2,2,4\}} \oplus A^{(2)}_{\{2,3,4\}} \oplus A^{(2)}_{\{2,4,4\}} = const. \end{aligned} \tag{20}$$

Then, we have $\theta^{(2)}_{\{1,3,5\}} = A^{(2)}_{\{0,2,5\}} \oplus const.$

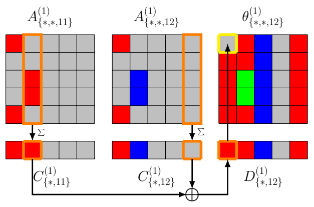

Fig. 11: Example (1) of consumed DoF

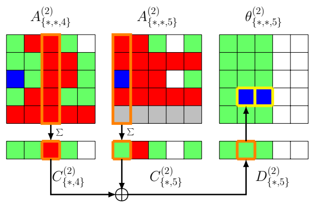

Fig. 12: Example (2) of consumed DoF

There are totally 100 ▨ and 🟦 in internal states between $A^{(1)}$ and $\theta^{(2)}$, which means that the accumulated consumed degree of freedom of 🟥 is 100. Denote the 100-bit constants (i.e. constants such as (19) and (20)) as $c_{\mathcal{R}} \in \mathbb{F}_2^{100}$. At last, the numbers of remaining degrees of freedom for 🟥 and 🟦 are both 8 bits.

Matching Strategy with Green Bits. According to Obs. 1, we can use $A^{(3)}$ to count the number of matching equations, i.e. $m = 8$. We give an example matching Equ. (21) (marked by 🟩 in $A^{(3)}$ with $z = 2$ and $z = 41$ in Figure 10) according to Equ. (15), which satisfies the matching conditions of Obs. 1:

$$A^{(3)}_{\{4,1,2\}} \oplus A^{(3)}_{\{4,4,2\}} \oplus (A^{(4)}_{\{2,1,22\}} \oplus 1) \cdot (A^{(3)}_{\{1,3,41\}} \oplus A^{(3)}_{\{1,1,41\}}) \oplus A^{(4)}_{\{1,1,22\}} \oplus \theta^{(3)}_{\{4,4,2\}} \oplus (A^{(4)}_{\{2,1,22\}} \oplus 1) \cdot \theta^{(3)}_{\{1,1,41\}} = 0. \tag{21}$$

With known bits in $A^{(4)}$ and $\theta^{(3)}$, the left part of Equ. (21) can be written as Boolean expression (denoted as $f_{\mathcal{M}}$) on 🟥, 🟦 and ▨ bits of the starting state. Therefore, denote $f_{\mathcal{M}} = f_{\mathcal{R}} \oplus f_{\mathcal{B}} \oplus f_{\mathcal{G}}$, where $f_{\mathcal{R}}$ only contains monomials on 🟥/▨

bits, $f_{\mathcal{B}}$ contains monomials on 🟦/◼ bits, and $f_{\mathcal{G}}$ contains monomials on ◼ bits and constants. With the given ◼ bits, we can compute the value $f_{\mathcal{R}} \oplus f_{\mathcal{G}} = f'_{\mathcal{M}}$ with forward computing `Keccak` permutation by setting all the 🟦 bits as 0. Similarly, we get $f_{\mathcal{B}} \oplus f_{\mathcal{G}} = f''_{\mathcal{M}}$ by setting 🟥 bits as 0. By setting all 🟥 and 🟦 bits as 0, we get $f_{\mathcal{G}} = f'''_{\mathcal{M}}$. Therefore, we compute $f_{\mathcal{R}} = f'_{\mathcal{M}} \oplus f'''_{\mathcal{M}}$ and $f_{\mathcal{B}} = f''_{\mathcal{M}} \oplus f'''_{\mathcal{M}}$. Then, we can derive the matching equation from Equ. (21) as $f_{\mathcal{R}} = f_{\mathcal{B}} \oplus f_{\mathcal{G}}$.

MitM Attack on 4-round `Keccak-512`. In our attack Algorithm 1 in Figure 10,16,17,18, we first precompute inversely from the target $A^{(4)}$ to $A^{(3)}$, and derive 128 Boolean equations with similar form with Equ. (21). Among them, 8 Boolean equations act as the matching points in the MitM phase, and the other 120 Boolean equations are used to further filter the partial matched states.

Following the framework in Figure 6, we use two message blocks (M_1, M_2) to build the attack as Algorithm 1. In Line 5, once we find solutions of M_2, we can perform 2^{100} MitM episodes in Line 14 to 25 for each of the 2^{228} solutions. For each MitM episode, $2^8 \times 2^8$ internal states are exhausted. Suppose there needs 2^x possible values of M_1. To find a 512-bit target preimage, we need $2^{x-232+228+100+8+8} = 2^{512}$, i.e., $x = 400$. The steps of Algorithm 1 are analyzed below:

- In Line 4, the time complexity is 2^{400} 4-round `Keccak` and $2^{400} \times 464^3$ bit operations to solve the linear system (the time to solve a system of n linear equations is about $O(n^3)$).
- In Line 9, the time complexity is $2^{400-232+228} \times \frac{1}{4} = 2^{394}$ 4-round `Keccak`.
- We describe the way to use Equ. (21) as the matching point. In the concrete `Keccak` attack, we can not set the 108 🟥 bits to be 0 when computing $f_{\mathcal{B}} + f_{\mathcal{G}} = f''_{\mathcal{M}}$ and $f_{\mathcal{G}} = f'''_{\mathcal{M}}$. This is because, the actual size of the 🟥 and 🟦 neutral sets is both 2^8, not 2^{108}. The 100 consumed DoF of 🟥 bits of $A^{(1)}$ are used to make 100 internal bits (denoted as $c_{\mathcal{R}} \in \mathbb{F}_2^{100}$) to be ◼/🟦, so that the remaining 🟥 set of size 2^8 can be computed independent to the 🟦 set.
  We detail the method to derive similar matching equation like "$f_{\mathcal{R}} = f_{\mathcal{B}} \oplus f_{\mathcal{G}}$" or "$f_{\mathcal{B}} = f_{\mathcal{R}} \oplus f_{\mathcal{G}}$" for the two 2^8 🟥/🟦 sets. With fixed ◼ bits in $A^{(1)}$ and $c_{\mathcal{R}} \in \mathbb{F}_2^{100}$, there are 2^8 🟥 bits stored in $U[c_{\mathcal{R}}]$, which is derived in Line 10 of Algorithm 1 following Dong et al.'s method [27]. Setting 🟦 in $A^{(1)}$ as 0, for each element of $U[c_{\mathcal{R}}]$, compute forward the `Keccak` permutation to the 8 matching bit equations (e.g. Equ. (21)) to get 8 $f'_{\mathcal{M}} = f_{\mathcal{R}} \oplus f_{\mathcal{G}}$ for matching. Randomly pick an element e of $U[c_{\mathcal{R}}]$ and set 🟦 in $A^{(1)}$ as 0, to compute the 8 matching equations $f'''_{\mathcal{M}} = f_{\mathcal{G}} + Const(e)$, where $Const(e)$ is determined by e. That is, for each of the 2^8 🟦, compute $f''_{\mathcal{M}} = f_{\mathcal{B}} + f_{\mathcal{G}} + Const(e)$ by setting the 108 🟥 $A^{(1)}$ bits as e. Therefore, we get $f''_{\mathcal{M}} + f'''_{\mathcal{M}} = f_{\mathcal{B}} = f_{\mathcal{R}} \oplus f_{\mathcal{G}} = f'_{\mathcal{M}}$ as filter. To dive into details, we refer the readers to Line 10 to Line 25.
  In Line 10, the time complexity is $2^{396+108} \times \frac{2}{4} = 2^{503}$ 4-round `Keccak`, since only two rounds from $A^{(1)}$ to $A^{(3)}$ are needed to compute to derive the ◼/🟦 bits and the matching points.
- In Line 13, the time complexity is $2^{396+100} \times \frac{2}{4} = 2^{495}$ 4-round `Keccak`.

Algorithm 1: Preimage Attack on 4-round `Keccak-512`

```
Precompute inversely from the target to $A^{(3)}$, and derive 128 Boolean
  equations of similar form with Equ. (21)
/* Among them, 8 Boolean equations act as the matching points in
   the MitM phase. The other 120 Boolean equations are used to
   further filter the partial matched states.                    */
for $2^x$ values of $M_1$ do
    Compute the inner part of the 2nd block and solve the system of 464
      linear equations
    if the equations have solutions   /* with probability of $2^{-232}$   */
     then
        for each of the $2^{228}$ solutions of $M_2$ do
            /* With $x = 400$, there are $2^{400-232+228} = 2^{396}$ iterations */
            Compute the ⬛ bits in $A^{(1)}$
            Traversing the $2^{108}$ values of 🟥 in $A^{(1)}$ while fixing 🟦 as 0, compute
              forward to determine 100-bit ⬛/🟦 bits (denoted as $c_{\mathcal{R}} \in \mathbb{F}_2^{100}$),
              and the 8-bit matching point, e.g., in (21), i.e., compute eight
              $f'_{\mathcal{M}} = f_{\mathcal{R}} \oplus f_{\mathcal{G}}$. Build the table $U$ and store the 108-bit 🟥 bits of
              $A^{(1)}$ as well as the 8-bit matching point in $U[c_{\mathcal{R}}]$.
            /* This method to solve the nonlinear constrained neutral
               words is borrowed from Dong et al. [27].                  */
            for $c_{\mathcal{R}} \in \mathbb{F}_2^{100}$ do
                Randomly pick a 108-bit 🟥 $e \in U[c_{\mathcal{R}}]$, and set 🟦 in $A^{(1)}$ as 0,
                  compute to the matching point to get eight
                  $f'''_{\mathcal{M}} = f_{\mathcal{G}} + Const(e)$
                for $2^8$ values in $U[c_{\mathcal{R}}]$ do
                    Restore the values of 🟥 of $A^{(1)}$ and the corresponding
                      matching point (i.e., eight $f_{\mathcal{R}} \oplus f_{\mathcal{G}} = f'_{\mathcal{M}}$) in a list $L_1$
                      (indexed by matching point)
                end
                for $2^8$ values of 🟦 do
                    Set the 108-bit 🟥 in $A^{(1)}$ as $e$. Compute to the matching
                      point to get eight $f''_{\mathcal{M}} = f_{\mathcal{B}} + f_{\mathcal{G}} + Const(e)$. Together
                      with $f'''_{\mathcal{M}}$, compute $f_{\mathcal{B}} = f''_{\mathcal{M}} + f'''_{\mathcal{M}}$ and store 🟦 in $L_2$
                      indexed by matching point.
                end
                for values matched between $L_1$ and $L_2$ do
                    Compute $A^{(3)}$ from the matched 🟥 and 🟦 bits
                    if $A^{(3)}$ satisfy the 120 precomputed Boolean
                      equations   /* Probability of $2^{-120}$              */
                     then
                        if it leads to the given hash value then
                            Output the preimage
                        end
                    end
                end
            end
        end
    end
end
```

- In Line 15, this step is just to retrieve $U[c_{\mathcal{R}}]$ to restore it in L_1 with matching point as index. Suppose one access to the table is equivalent to one Sbox application. The time complexity is $2^{396+100+8} \times \frac{1}{4\times 320} = 2^{493.36}$ 4-round `Keccak`, since there are 4×320 Sboxes for 4-round `Keccak`.
- In Line 18, the time complexity is $2^{396+100+8} \times \frac{2}{4} = 2^{503}$ 4-round `Keccak`.
- In Line 21, the time is $2^{396+100+8+8-8} \times \frac{2}{4} = 2^{503}$ 4-round `Keccak`.
- In Line 22, $A^{(3)}$ is checked against the 120 Boolean equations precomputed in Line 1, which acts as a filter of 2^{-120}. After the filter, the time of the final check against the target h is $2^{396+100+8-120} = 2^{384}$ 4-round `Keccak`.

The total complexity is $2^{400}+2^{400}\times 464^3+2^{394}+2^{503}+2^{495}+2^{493.36}+2^{503}+2^{503}+2^{384} \approx 2^{504.58}$ 4-round `Keccak`. The memory to store U is 2^{108}. We also give an experiment of the MitM episode of 3-round `Keccak-512` in the Supplementary Material C.

Remark on padding rule. The last message block has at least 2-bit padding (i.e., '11') for `Keccak` and 4-bit padding (i.e., '0111') for SHA3. Therefore, we have $x = 402$ for `Keccak`-512 and $x = 404$ for SHA3-512. However, it only increases the negligible part $2^{400} \times 464^3$ in Line 4 to $2^{402} \times 464^3$ for `Keccak`-512 and $2^{404} \times 464^3$ for SHA3-512. Therefore the final time complexity is still $2^{504.58}$ 4-round `Keccak`-512 or SHA3-512 considering the padding. The memory is 2^{108} to store U.

5 MitM Preimage Attack on `Xoodyak-XOF`

The `Xoodyak` is based on `Xoodoo` permutation, which is inspired by `Keccak`-f permutation. The MILP model for `Xoodyak` is similar with that for `Keccak` in Section 4.2. The size of absorbed message block of `Xoodyak-XOF` is 128 bits. We target on `Xoodyak-XOF` with output of 128-bit hash (like the case for `Ascon-XOF`). In this section, we list the differences in the MILP model and give an MitM preimage attack on 3-round `Xoodyak-XOF`.

5.1 MILP Model of the MitM Preimage Attack on `Xoodyak-XOF`

For `Xoodyak`, without the help of the *linear structure* technique, the situations of adding conditions to the state before the Sbox χ are more complex. We give the details of the condition rules and the matching rule for `Xoodyak` in the following.

Modelling the χ operation with conditions in Round 0. The Sbox χ of `Xoodyak` is different from that of `Keccak` at the sizes of inputs and outputs, which acts on a column $A^{(r)}_{\{x,*,z\}}$. Assume χ operation maps $(a_0, a_1, a_2) \in \mathbb{F}_2^3$ in to $(b_0, b_1, b_2) \in \mathbb{F}_2^3$ as $b_i = a_i \oplus (a_{i+1} \oplus 1) \cdot a_{i+2}$. In round 0, all the operations between the starting state $A^{(0)}$ and χ are linear, thus the inputs only have 🟥, 🟦, 🟩 and ■. We can add conditions on ■ bits to control the diffusion. The different rules from the `SBOX-RULE` for `Keccak` are listed below:

1. If there are two ◼ bits and one 🟥/🟦/🟩 bit in (a_0, a_1, a_2), we can add one or two conditions to make one or two outputs to be ◼. Without losing generality, suppose a_i is 🟥 or 🟦 or 🟩 and both a_{i+1} and a_{i+2} are ◼ bits:
   (a) the color of b_i is always same with a_i.
   (b) $a_{i+2}=1$, b_{i+1} will be ◼; otherwise, the color of b_{i+1} will be same with a_i.
   (c) $a_{i+1}=0$, b_{i+2} will be ◼; otherwise, the color of b_{i+2} will be same with a_i.
2. If there is only one ◼ bit and the other two are among (🟥, 🟦)/(🟥, 🟩)/(🟦, 🟩), we add conditions to reduce the number of 🟩 in the output. If a_i is ◼:
   (a) b_i is always be □.
   (b) $a_i=0$, the color of b_{i+1} will be same with a_{i+1} and b_{i+2} will be 🟩.
   (c) $a_i=1$, b_{i+1} will be 🟩 and the color of b_{i+2} will be same with a_{i+2}.
   (d) without conditions on a_i, both b_{i+1} and b_{i+2} will be 🟩.

The rules can be described by a system of linear inequalities by using the convex hull computation. Some valid coloring patterns are shown in Figure 13.

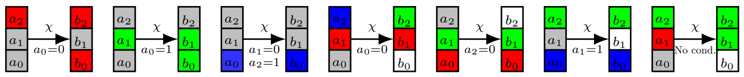

Fig. 13: Some valid coloring patterns with conditions for χ

Modelling the Matching Degrees. Suppose the 128 bits in the top plane $A^{(r+1)}_{\{*,2,*\}}$ of $A^{(r+1)}$ is the hash value. We can easily compute the top plane of state $\chi^{(r)}_{\{*,2,*\}}$ from the hash value by the inverse of ρ_{east}. Each bit of $\chi^{(r)}_{\{*,2,*\}}$ is computed by $b_2 = a_2 \oplus (a_0 \oplus 1) \cdot a_1$, where (a_0, a_1, a_2) comes from the input column of each Sbox in $\iota^{(r)}$. Hence, we deduce the deterministic relations of $\iota^{(r)}$ to count the DoMs.

Observation 2 (Conditions in Matching Points of `Xoodyak`) *If (a_0, a_1, a_2) satisfy the following conditions, we say there is a 1-bit matching:*

1. *There is no □ bit in (a_0, a_1, a_2).*
2. *There is no the product of 🟦 and 🟥, concretely, (a_0, a_1) should not be (🟦, 🟥) or (🟥, 🟩) or (🟦, 🟩) or (🟩, 🟩), or opposite order.*
3. *In fact, (a_0, a_1, a_2) should be (◼, $*$, 🟩) or ($*$, ◼, 🟩) or (🟥, 🟥, 🟩) or (🟦, 🟦, 🟩) or (🟥, 🟥, 🟦) or (🟦, 🟦, 🟥), where '$*$' is 🟥 or 🟦 or 🟩 or ◼. We exclude several cases such as (🟦, ◼, 🟥), since it is a filter if ◼ $= 1$, but not for ◼ $= 0$.*

5.2 MitM Preimage Attack on 3-round `Xoodyak-XOF`

We also follow the similar framework in Figure 6 and perform the attack with two message blocks (M_1, M_2), where M_2 has two padding bits '10'. The MitM attack is placed in the 2nd block. Solving with our MILP model for `Xoodyak`, we get a 3-round MitM preimage attack as shown in Figure 23 in Supplementary Material D. Similar to `Keccak`, we decompose the θ operation into three steps, where $C^{(r)}_{\{x,z\}} = \sum_{y'=0}^{2} A^{(r)}_{\{x,y',z\}}$, $D^{(r)}_{\{x,z\}} = C^{(r)}_{\{x-1,z-5\}} \oplus C^{(r)}_{\{x-1,z-14\}}$, and $\theta^{(r)}_{\{x,y,z\}} = A^{(r)}_{\{x,y,z\}} \oplus D^{(r)}_{\{x,z\}}$. The starting state $A^{(0)}$ contains 4 🟦 bits and 97 🟥 bits. There are totally 53 conditions on ⬛ bits of $\iota^{(0)}$, which are listed in Table 2. In the computation from $A^{(0)}$ to $\iota^{(2)}$, the accumulated consumed degree of freedom of 🟥 is 93 and there is no DoF of 🟦 consumed. Therefore, $\text{DoF}_{\mathcal{B}} = 4$, $\text{DoF}_{\mathcal{R}} = 97 - 93 = 4$. The degree of matching is counted by the deterministic relations of $\iota^{(2)}$ according to Observation 2, we get DoM $= 4$, where

$$\chi^{(2)}_{\{2,2,4\}} = \iota^{(2)}_{\{2,2,4\}} \oplus (\iota^{(2)}_{\{2,0,4\}} \oplus 1) \cdot \iota^{(2)}_{\{2,1,4\}},\ \chi^{(2)}_{\{1,2,10\}} = \iota^{(2)}_{\{1,2,10\}} \oplus (\iota^{(2)}_{\{1,0,10\}} \oplus 1) \cdot \iota^{(2)}_{\{1,1,10\}},$$
$$\chi^{(2)}_{\{1,2,19\}} = \iota^{(2)}_{\{1,2,19\}} \oplus (\iota^{(2)}_{\{1,0,19\}} \oplus 1) \cdot \iota^{(2)}_{\{1,1,19\}},\ \chi^{(2)}_{\{2,2,22\}} = \iota^{(2)}_{\{2,2,22\}} \oplus (\iota^{(2)}_{\{2,0,22\}} \oplus 1) \cdot \iota^{(2)}_{\{2,1,22\}}. \quad (22)$$

In the starting state $A^{(0)}$, there are $128 - 2 - 97 - 4 = 25$ ⬛ bits which can be modified by changing M_2. Similar to the conditions of `Keccak`, the 53 conditions can form a linear system taking these 25 bits of M_2 as variables and the 256 bits inner part as constants. The rank of the coefficient matrix is 13. Therefore, we have to randomly test $2^{(53-13)} = 2^{40}$ M_1 to satisfy the 40 equations only determined by the inner part. Then for a right inner part, there are $2^{25-13} = 2^{12}$ solutions of M_2, which make all the 53 equations hold. The attack is similar to the attack on `Keccak`, and we list it in the Algorithm 3 in Supplementary Material D. In the MitM episode in Line 10 to 22, a space of 2^{4+4} is searched. In order to search a 128-bit preimage, we have to search a space of $2^{x-40+12+93+8} = 2^{128}$, i.e., $x = 55$. Each step of Algorithm 3 is analyzed below:

- In Line 2, the time complexity is $2^{55} \times 53^3 = 2^{72.2}$ bit operations and 2^{55} 3-round `Xoodyak`.
- In Line 7, the time complexity is $2^{27+97} \times \frac{128+128+4}{128\times 3} = 2^{123.44}$ 3-round `Xoodyak`. The fraction $\frac{128+128+4}{128\times 3}$ is because that in the last round only 4 Sboxes with matching points are computed, while there are totally 128×3 Sboxes applications in the 3-round `Xoodyak`.
- In Line 9, the time is $2^{27+93} \times \frac{128+128+4}{128\times 3} = 2^{119.44}$ 3-round `Xoodyak`.
- In Line 11, the time is $2^{27+93+4} \times \frac{1}{320} = 2^{115.68}$ 3-round `Xoodyak`. This step is just to retrieve the values $U[c_{\mathcal{R}}]$ and restore it in L_1. Assuming one table access is about one Sbox application, we get the fraction $\frac{1}{320}$.
- In Line 14, the time is $2^{27+93+4} \times \frac{128+128+4}{128\times 3} = 2^{123.44}$ 3-round `Xoodyak`.
- In Line 17, the time complexity is $2^{27+93+4} \times \frac{2}{3} = 2^{123.42}$ 3-round `Xoodyak`.
- In Line 18, we only compute 5 Sboxes with $\iota^{(2)}$ to gain a filter of 2^{-5}, whose time complexity is $2^{27+93+4} \times \frac{5}{128\times 3} = 2^{117.74}$ 3-round `Xoodyak`.
- In Line 21, we check the remaining states with the remaining $128-4-5 = 119$ Sboxes, which is $2^{27+93+4-5} \times \frac{119}{384} = 2^{117.31}$ 3-round `Xoodyak`.

The total complexity of the 3-round attack is $2^{72.2} + 2^{55} + 2^{123.44} + 2^{119.44} + 2^{115.68} + 2^{123.44} + 2^{123.42} + 2^{117.74} + 2^{117.31} \approx 2^{125.06}$ 3-round `Xoodyak-XOF`, and the memory to store U is 2^{97}.

$$\iota^{(0)}_{\{0,0,3\}} = 0; \iota^{(0)}_{\{0,1,3\}} = 1; \iota^{(0)}_{\{1,0,3\}} = 0; \iota^{(0)}_{\{2,0,3\}} = 0; \iota^{(0)}_{\{3,1,4\}} = 1; \iota^{(0)}_{\{3,2,5\}} = 1;$$
$$\iota^{(0)}_{\{2,0,6\}} = 0; \iota^{(0)}_{\{2,1,6\}} = 1; \iota^{(0)}_{\{3,0,6\}} = 0; \iota^{(0)}_{\{3,1,6\}} = 1; \iota^{(0)}_{\{2,0,7\}} = 1; \iota^{(0)}_{\{3,1,7\}} = 0; \iota^{(0)}_{\{3,2,7\}} = 1;$$
$$\iota^{(0)}_{\{2,2,8\}} = 0; \iota^{(0)}_{\{2,0,8\}} = 1; \iota^{(0)}_{\{3,0,10\}} = 0; \iota^{(0)}_{\{3,0,11\}} = 0; \iota^{(0)}_{\{3,1,11\}} = 1; \iota^{(0)}_{\{1,0,12\}} = 0; \iota^{(0)}_{\{1,1,12\}} = 1;$$
$$\iota^{(0)}_{\{0,0,13\}} = 0; \iota^{(0)}_{\{0,1,13\}} = 1; \iota^{(0)}_{\{3,0,13\}} = 0; \iota^{(0)}_{\{3,1,13\}} = 1; \iota^{(0)}_{\{2,0,14\}} = 0; \iota^{(0)}_{\{2,0,15\}} = 0;$$
$$\iota^{(0)}_{\{3,0,15\}} = 0; \iota^{(0)}_{\{3,1,15\}} = 1; \iota^{(0)}_{\{0,1,19\}} = 1; \iota^{(0)}_{\{3,0,19\}} = 0; \iota^{(0)}_{\{2,1,20\}} = 0; \iota^{(0)}_{\{3,0,20\}} = 0; \iota^{(0)}_{\{3,1,20\}} = 1;$$
$$\iota^{(0)}_{\{0,0,21\}} = 1; \iota^{(0)}_{\{2,2,21\}} = 0; \iota^{(0)}_{\{3,1,21\}} = 0; \iota^{(0)}_{\{3,2,21\}} = 1; \iota^{(0)}_{\{2,0,22\}} = 1; \iota^{(0)}_{\{2,2,22\}} = 0; \iota^{(0)}_{\{3,1,22\}} = 0;$$
$$\iota^{(0)}_{\{1,0,23\}} = 0; \iota^{(0)}_{\{1,1,23\}} = 1; \iota^{(0)}_{\{0,0,28\}} = 0; \iota^{(0)}_{\{3,0,28\}} = 0; \iota^{(0)}_{\{0,0,29\}} = 0; \iota^{(0)}_{\{0,1,29\}} = 1;$$
$$\iota^{(0)}_{\{3,0,29\}} = 0; \iota^{(0)}_{\{3,1,29\}} = 1; \iota^{(0)}_{\{0,0,30\}} = 0; \iota^{(0)}_{\{2,2,30\}} = 0; \iota^{(0)}_{\{2,0,31\}} = 1; \iota^{(0)}_{\{2,2,31\}} = 0; \iota^{(0)}_{\{3,1,31\}} = 0$$

Table 2: Bit Conditions in 3-round Attack on `Xoodyak-XOF`

6 MitM Preimage Attack on `Ascon-XOF`

In this section, we list the details of the MILP model for `Ascon-XOF` different from that for `Keccak`, and give an MitM preimage attack on 4-round `Ascon-XOF`.

6.1 MILP Model of the MitM Preimage Attack on `Ascon-XOF`

We do not model `Ascon` with 🟩 bit, since the Sbox of `Ascon` is much more complex than `Keccak`, and the output of the Sbox will be unknown if there is 🟩 bit in the input in most cases. The 🟩 bit can hardly improve the modelling of `Ascon`. The other modellings are similar to `Keccak`, we only list the differences.

Modelling the Starting State with Conditions. In the starting state $A^{(0)}$, the 64-bit outer part can be 🟦 or 🟥 bits, while the last 256-bit inner part is of ⬛. As shown in the first two cases in Figure 14, p_S maps $(a_0, a_1, a_2, a_3, a_4)$ to $(b_0, b_1, b_2, b_3, b_4)$, where the a_0 in the outer part could be 🟥 or 🟦 and (a_1, a_2, a_3, a_4) in the inner part are always ⬛. When a_0 is 🟥 and others are ⬛, all the output bits excluding b_2 should be 🟥 according to Equ. (3) (case 1 in Figure 14). However, if we add some conditions on (a_1, a_2, a_3, a_4), for example $a_1 = 1$ and $a_3 + a_4 = 1$, then according to Equ. (3), b_0 and b_3 can also be ⬛ (case 2 in Figure 14). We add conditions on the bits in the inner part of $A^{(0)}$ to control the diffusion of 🟥 and 🟦 over p_S. We name the rule by `CondSBOX-RULE`:

1. Condition on a_1: when $a_1 = 1$, b_0 is ⬛ and the color of b_4 is the same with a_0; when $a_1 = 0$, b_4 is ⬛ and the color of b_0 is the same with a_0.
2. Condition on $a_3 + a_4$: when $a_3 + a_4 = 1$, b_3 is ⬛; when $a_3 + a_4 = 0$, the color b_3 is the same with a_0.

Some valid coloring patterns of `CondSBOX-RULE` are shown in Figure 14.

Modelling the Matching Degree. We target on `Ascon-XOF` with a 128-bit hash value, which needs two output blocks. Suppose the 64-bit word $A^{(r+1)}_{\{0,*\}}$ of $A^{(r+1)}$ is the first 64-bit hash value of the first output block. We can easily compute the first 64 bits of state $S^{(r)}$ from the hash value by the inverse of p_L.

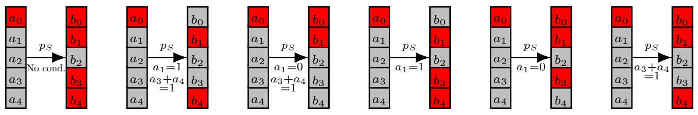

Fig. 14: Some valid red coloring patterns of `CondSBOX-RULE` (Similar to blue bits)

Each of the first 64 bits of $S^{(r)}$ is computed by $b_0 = a_4a_1 + a_3 + a_2a_1 + a_2 + a_1a_0 + a_1 + a_0$ by Equ. (3), where $(a_0, a_1, a_2, a_3, a_4)$ comes from the inputs of each Sbox in $A^{(r)}$. Hence, we deduce the deterministic relations of $A^{(r)}$ to count the degree of freedom of matching.

Observation 3 (Conditions in Matching Points of `Ascon`) *If* $(a_0, a_1, a_2, a_3, a_4)$ *satisfy the following conditions, we say there is a 1-bit matching:*

1. *There is no* □ *bit in* $(a_0, a_1, a_2, a_3, a_4)$.
2. *There are* 🟦 *and* 🟥 *bits in* $(a_0, a_1, a_2, a_3, a_4)$.
3. *There are no product of* 🟦 *and* 🟥, *concretely,* (a_1, a_4) *should not be* (🟦,🟥) *or* (🟥,🟦), *and the same to* (a_1, a_2) *and* (a_0, a_1).

6.2 MitM Preimage Attack on 4-round `Ascon-XOF`

Applying the MILP model, we find a 4-round MitM preimage attack as shown in Figure 24 in Supplementary Material E. The starting state $A^{(0)}$ contains 2 🟦 bits and 45 🟥 bits. There are totally 50 conditions on ▨ of $A^{(0)}$, which are listed in Table 3. In the computation from $A^{(0)}$ to $A^{(3)}$, the accumulated consumed degrees of freedom of 🟥 is 43 and there is no DoF of 🟦 consumed. Therefore, $\mathrm{DoF}_{\mathcal{B}} = 2$, $\mathrm{DoF}_{\mathcal{R}} = 45 - 43 = 2$. The two matching bit equations (DoM = 2) are derived by $A^{(3)}$ with Observation 3, which are:

$$\begin{cases} A^{(3)}_{\{4,4\}} \cdot A^{(3)}_{\{4,1\}} + A^{(3)}_{\{4,3\}} + A^{(3)}_{\{4,2\}} \cdot A^{(3)}_{\{4,1\}} + A^{(3)}_{\{4,2\}} + A^{(3)}_{\{4,1\}} \cdot A^{(3)}_{\{4,0\}} + A^{(3)}_{\{4,1\}} + A^{(3)}_{\{4,0\}} = S^{(3)}_{\{4,0\}}, \\ A^{(3)}_{\{48,4\}} \cdot A^{(3)}_{\{48,1\}} + A^{(3)}_{\{48,3\}} + A^{(3)}_{\{48,2\}} \cdot A^{(3)}_{\{48,1\}} + A^{(3)}_{\{48,2\}} + A^{(3)}_{\{48,1\}} \cdot A^{(3)}_{\{48,0\}} + A^{(3)}_{\{48,1\}} + A^{(3)}_{\{48,0\}} = S^{(3)}_{\{48,0\}}. \end{cases} \tag{23}$$

$A^{(0)}_{\{1,1\}} = 1; A^{(0)}_{\{3,1\}} = 1; A^{(0)}_{\{4,1\}} = 0; A^{(0)}_{\{5,1\}} = 1; A^{(0)}_{\{6,1\}} = 0; A^{(0)}_{\{8,1\}} = 0; A^{(0)}_{\{9,1\}} = 1;$
$A^{(0)}_{\{12,1\}} = 1, A^{(0)}_{\{12,3\}} + A^{(0)}_{\{12,4\}} = 1; A^{(0)}_{\{15,1\}} = 1; A^{(0)}_{\{18,1\}} = 1,\ A^{(0)}_{\{18,3\}} + A^{(0)}_{\{18,4\}} = 1;$
$A^{(0)}_{\{19,1\}} = 0; A^{(0)}_{\{22,1\}} = 1,\ A^{(0)}_{\{22,3\}} + A^{(0)}_{\{22,4\}} = 1; A^{(0)}_{\{23,1\}} = 1; A^{(0)}_{\{26,1\}} = 1;$
$A^{(0)}_{\{28,1\}} = 1,\ A^{(0)}_{\{28,3\}} + A^{(0)}_{\{28,4\}} = 1; A^{(0)}_{\{29,1\}} = 1; A^{(0)}_{\{30,1\}} = 0,\ A^{(0)}_{\{30,3\}} + A^{(0)}_{\{30,4\}} = 1;$
$A^{(0)}_{\{31,1\}} = 1; A^{(0)}_{\{32,1\}} = 1; A^{(0)}_{\{33,1\}} = 1; A^{(0)}_{\{35,1\}} = 1,\ A^{(0)}_{\{35,3\}} + A^{(0)}_{\{35,4\}} = 1; A^{(0)}_{\{38,1\}} = 1;$
$A^{(0)}_{\{39,1\}} = 1,\ A^{(0)}_{\{39,3\}} + A^{(0)}_{\{39,4\}} = 1; A^{(0)}_{\{40,1\}} = 0,\ A^{(0)}_{\{40,3\}} + A^{(0)}_{\{40,4\}} = 1; A^{(0)}_{\{41,1\}} = 0;$
$A^{(0)}_{\{44,1\}} = 0; A^{(0)}_{\{45,1\}} = 0; A^{(0)}_{\{47,1\}} = 0,\ A^{(0)}_{\{47,3\}} + A^{(0)}_{\{47,4\}} = 1; A^{(0)}_{\{48,1\}} = 0;$
$A^{(0)}_{\{50,1\}} = 0,\ A^{(0)}_{\{50,3\}} + A^{(0)}_{\{50,4\}} = 1; A^{(0)}_{\{51,1\}} = 1; A^{(0)}_{\{53,1\}} = 0;$
$A^{(0)}_{\{54,1\}} = 1,\ A^{(0)}_{\{54,3\}} + A^{(0)}_{\{54,4\}} = 1; A^{(0)}_{\{59,1\}} = 0,\ A^{(0)}_{\{59,3\}} + A^{(0)}_{\{59,4\}} = 1;$
$A^{(0)}_{\{60,1\}} = 0,\ A^{(0)}_{\{60,3\}} + A^{(0)}_{\{60,4\}} = 1; A^{(0)}_{\{61,1\}} = 0,\ A^{(0)}_{\{61,3\}} + A^{(0)}_{\{61,4\}} = 1$

Table 3: Bit Conditions in 4-round Attack on `Ascon-XOF`

Algorithm 2: Preimage Attack on 4-round `Ascon-XOF`

```
Inversely precompute $S^{(3)}_{\{*,0\}}$ with the 64-bit hashing value
for $2^x$ values of $(M_1, M_2)$ do
    Compute the inner part of the 3rd block
    if the conditions in Table 3 are satisfied   /* probability of $2^{-50}$   */
     then
        for $2^{64-2-45-2} = 2^{15}$ values of the ■ bits in $M_3$
        /* There are 2-bit padding in $M_3$ fixed globally   */
         do
            Compute the ■ bits in $A^{(0)}$ to $A^{(1)}$, except those ▣ bits
            Traversing the $2^{45}$ values for 🟥 in $A^{(0)}$ while fixing 🟦 as 0, compute
              the 43-bit ▣/🟦 bits as $c_{\mathcal{R}} \in \mathbb{F}_2^{43}$ and store the 45-bit 🟥 of $A(0)$
              and two matching bits (i.e., $A^{(3)}_{\{4,1\}} \cdot A^{(3)}_{\{4,0\}} + A^{(3)}_{\{4,1\}} + A^{(3)}_{\{4,0\}} + S^{(3)}_{\{4,0\}}$,
              $A^{(3)}_{\{48,1\}} \cdot A^{(3)}_{\{48,0\}} + A^{(3)}_{\{48,1\}} + A^{(3)}_{\{48,0\}} + S^{(3)}_{\{48,0\}}$, which are red parts of
              (23)) in table $U[c_{\mathcal{R}}]$.
            for $c_{\mathcal{R}} \in \mathbb{F}_2^{43}$ do
                for $2^2$ values in $U[c_{\mathcal{R}}]$ do
                    Retrieve the 45-bit 🟥 of $A^{(0)}$ and the two matching bits,
                      store the 45-bit 🟥 in $L_1$ indexed by the two matching bits
                end
                for $2^2$ values of 🟦 do
                    Compute to the matching point (blue parts of (23)) and
                      store 2-bit 🟦 of $A^{(0)}$ in $L_2$ indexed by the matching point
                end
                for values matched between $L_1$ and $L_2$ do
                    Compute the first 2 Sboxes of $S^{(3)}$
                    if the 2 Sboxes satisfy the precomputed $S^{(3)}_{\{*,0\}}$
                    /* probability of $2^{-2}$. This step is to avoid
                       computing all Sboxes of $S^{(3)}$ and only use
                       partial Sboxes to filter first.   */
                     then
                        if it leads to the given hash value then
                            Output the preimage
                        end
                    end
                end
            end
        end
    end
end
```

Following the framework in Figure 6, we choose (M_1, M_2) to make the 50 conditions hold, and perform the MitM attack with the 3rd message block M_3. The 4-round attack is given in Algorithm 2. In Line 11 to Line 26, a subspace of 2^{2+2} is traversed. In Algorithm 2, to find a 128-bit preimage, we exhaust $2^{x-50+15+43+4} = 2^{128}$, i.e., $x = 116$. The steps of Alg. 2 are analyzed below:

- In Line 3, the time complexity is $2^{116} \times 2 = 2^{117}$ 4-round `Ascon`.
- In Line 6, there are 15 free bits in M_3 by excluding the 2-bit padding, 45-bit 🟥 and 2-bit 🟦 bits.
  In Line 9, the time complexity is $2^{116-50+15} \times \frac{1}{4} = 2^{79}$ 4-round `Ascon`.
- In Line 10, for each of the 2^{45} 🟥 bits in $A^{(0)}$, we need compute $45+56+20 = 121$ out of the total $64 \times 4 = 256$ Sbox applications (4 rounds) to build U. Hence, the time of Line 10 is $2^{116-50+15+45} \times \frac{121}{256} = 2^{124.9}$ 4-round `Ascon`.
- As U stores red values and matching points, in Line 13, building L_1 is just to retrieve the values in $U[c_{\mathcal{R}}]$. If one table access is about one Sbox application, the time of Line 13 is $2^{116-50+15+43+2} \times \frac{1}{256} = 2^{118}$ 4-round `Ascon`.
- In Line 16, given 🟦 bits of $A^{(0)}$, the time to compute the matching points is $2+5+6 = 13$ Sbox applications. Therefore, the time of Line 16 is $2^{116-50+15+43+2} \times \frac{13}{256} = 2^{121.7}$ 4-round `Ascon`.
- In Line 19, we need to compute the first two Sboxes of $A^{(3)}$, i.e., $5 \times 2 = 10$ bits. For the linear layer, each output bit depends on 3 input bits. Therefore, to compute the 10 bits of $A^{(3)}$, we have to know $10 \times 3 = 30$ bits of $S^{(2)}$, which are assumed to be involved in 30 Sboxes of $S^{(2)}$. Since in Line 9, the Sboxes determined by ■ bits are already computed in round 0, we totally have to compute $47+64+30+2 = 143$ Sboxes to compute the first 2 Sboxes of $S^{(3)}$. The time of Line 19 is $2^{116-50+15+43+2} \times \frac{143}{256} = 2^{125.16}$ 4-round `Ascon`.
- In Line 23, the time is $2^{116-50+15+43+2-2} = 2^{124}$ 4-round `Ascon`.

The total time complexity is $2^{117} + 2^{79} + 2^{124.9} + 2^{118} + 2^{121.7} + 2^{125.16} + 2^{124} \approx 2^{126.4}$ 4-round `Ascon`. The memory is 2^{45} to store U. In addition, we also give a 3-round MitM preimage attack on `Ascon-XOF` in Supplementary Material E with time complexity of $2^{120.58}$ 3-round `Ascon` and memory complexity of 2^{39}. An experiment on the MitM episode is given in Supplementary Material F.

7 Conclusion

In this paper, we give the framework of the MitM attack on sponge-based hashing. To find good attacks, we build bit-level MILP based automatic tools for MitM attacks on `Keccak-512`, `Ascon-XOF`, and `Xoodyak-XOF`. Although the birthday-paradox MitM attack has been widely applied to block ciphers or MD-based hash functions since 1977, this is the first attempt to apply it to `Keccak`, etc. Our attacks lead to improved or first preimage attacks on reduced-round `Keccak-512`, `Ascon-XOF`, and `Xoodyak-XOF`.

Discussion. Similar to previous preimage attacks [34,44], our attack on `Keccak` also uses the linearization-based techniques. In previous linearization-based preimage attack [34,44], all the variables should not be multiplied with each other, so that one can build linear equations on those variables and the target bits. In our MitM attack, the variables are divided into two independent sets. Within each set, the variables can be multiplied with each other. However, any variable from one set should not be multiplied with the variables from the other set. For Keccak, to attack more rounds, we use a one-round linear structure to skip the MILP programming of the first round and accelerate the MILP model. It should be noted that the linear structure used in this paper is just a technique to accelerate the search. Moreover, by using the linear structure, the search only covers a small fraction of the whole space of the solutions, which may be not the optimal MitM attack at all. For other instances of Keccak, it is open problem to apply one or two-round linear structures in the search for MitM attacks.

References

1. Dor Amzaleg and Itai Dinur. Refined cryptanalysis of the GPRS ciphers GEA-1 and GEA-2. *IACR Cryptol. ePrint Arch.*, page 424, 2022.
2. Kazumaro Aoki and Yu Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In *SAC 2008*, volume 5381, pages 103–119. Springer, 2008.
3. Jean-Philippe Aumasson, Daniel J Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, et al. SPHINCS+: Submission to the NIST post-quantum project. 2019.
4. Jean-Philippe Aumasson, Willi Meier, and Florian Mendel. Preimage attacks on 3-pass HAVAL and step-reduced MD5. In *SAC 2008*, volume 5381, pages 120–135.
5. Zhenzhen Bao, Xiaoyang Dong, Jian Guo, Zheng Li, Danping Shi, Siwei Sun, and Xiaoyun Wang. Automatic search of meet-in-the-middle preimage attacks on AES-like hashing. In *EUROCRYPT 2021, Part I*, volume 12696, pages 771–804.
6. Zhenzhen Bao, Jian Guo, Danping Shi, and Yi Tu. Superposition meet-in-the-middle attacks: Updates on fundamental security of AES-like hashing. Cryptology ePrint Archive, Paper 2021/575, 2021. https://eprint.iacr.org/2021/575.
7. Christof Beierle, Patrick Derbez, Gregor Leander, Gaëtan Leurent, Håvard Raddum, Yann Rotella, David Rupprecht, and Lukas Stennes. Cryptanalysis of the GPRS encryption algorithms GEA-1 and GEA-2. In *EUROCRYPT 2021, Proceedings, Part II*, volume 12697, pages 155–183. Springer, 2021.
8. Daniel J. Bernstein. Second preimages for 6 (7?(8??)) rounds of Keccak. *NIST mailing list*, 2010.
9. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document. *Submission to NIST (Round 2)*, 3(30):320–337, 2009.
10. Alex Biryukov, Patrick Derbez, and Léo Perrin. Differential analysis and meet-in-the-middle attack against round-reduced TWINE. In *FSE 2015*, pages 3–27.
11. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In *ASIACRYPT 2011, Proceedings*, pages 344–371.
12. Andrey Bogdanov and Christian Rechberger. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In *SAC 2010*, volume 6544, pages 229–240. Springer, 2010.
13. Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic search of attacks on round-reduced AES and applications. In *CRYPTO 2011, Proceedings*, volume 6841, pages 169–187. Springer, 2011.

14. Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. Sieve-in-the-middle: Improved MITM attacks. In *CRYPTO 2013, Proceedings, Part I*, pages 222–240.
15. Joan Daemen, Seth Hoffert, Michaël Peeters, G Van Assche, and R Van Keer. Xoodyak, a lightweight cryptographic scheme. 2020.
16. Ivan Damgård. A design principle for hash functions. In *CRYPTO '89*, pages 416–427.
17. Hüseyin Demirci and Ali Aydin Selçuk. A meet-in-the-middle attack on 8-round AES. In *FSE 2008*, pages 116–126. Springer, 2008.
18. Patrick Derbez and Pierre-Alain Fouque. Automatic search of meet-in-the-middle and impossible differential attacks. In *CRYPTO 2016, Proceedings, Part II*, volume 9815, pages 157–184. Springer, 2016.
19. Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved key recovery attacks on reduced-round AES in the single-key setting. In *EUROCRYPT 2013, Proceedings*, volume 7881, pages 371–387. Springer, 2013.
20. Patrick Derbez and Léo Perrin. Meet-in-the-middle attacks and structural analysis of round-reduced prince. In *FSE 2015*, pages 190–216. Springer, 2015.
21. Whitfield Diffie and Martin E. Hellman. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, 1977.
22. Itai Dinur. Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In *EUROCRYPT 2021, Proceedings, Part I*, volume 12696, pages 374–403. Springer, 2021.
23. Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In *CRYPTO 2012*, volume 7417, pages 719–740.
24. Itai Dinur, Orr Dunkelman, and Adi Shamir. Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. In *FSE 2013*, volume 8424, pages 219–240. Springer, 2013.
25. Itai Dinur and Adi Shamir. Breaking grain-128 with dynamic cube attacks. In *FSE 2011*, volume 6733, pages 167–187. Springer, 2011.
26. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.
27. Xiaoyang Dong, Jialiang Hua, Siwei Sun, Zheng Li, Xiaoyun Wang, and Lei Hu. Meet-in-the-middle attacks revisited: Key-recovery, collision, and preimage attacks. In *CRYPTO 2021, Proceedings, Part III*, volume 12827, pages 278–308. Springer.
28. Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved single-key attacks on 8-round AES-192 and AES-256. In *ASIACRYPT 2010, Proceedings*, volume 6477, pages 158–176. Springer, 2010.
29. Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved meet-in-the-middle attacks on reduced-round DES. In *INDOCRYPT 2007, Proceedings*, volume 4859, pages 86–100. Springer, 2007.
30. Thomas Espitau, Pierre-Alain Fouque, and Pierre Karpman. Higher-order differential meet-in-the-middle preimage attacks on SHA-1 and BLAKE. In *CRYPTO 2015, Proceedings, Part I*, volume 9215, pages 683–701. Springer, 2015.
31. Thomas Fuhr and Brice Minaud. Match box meet-in-the-middle attack against KATAN. In *FSE 2014*, pages 61–81, 2014.
32. David Gérault, Thomas Peyrin, and Quan Quan Tan. Exploring differential-based distinguishers and forgeries for ASCON. *IACR Trans. Symmetric Cryptol.*, 2021(3):102–136, 2021.

33. Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In *ASIACRYPT 2010, Proceedings*, volume 6477, pages 56–75.
34. Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to crypt-analysis of round-reduced Keccak. In *ASIACRYPT 2016, Proceedings, Part I*, volume 10031, pages 249–274, 2016.
35. Le He, Xiaoen Lin, and Hongbo Yu. Improved preimage attacks on round-reduced Keccak-384/512 via restricted linear structures. Cryptol. ePrint Arch., Paper, 2022/788.
36. Le He, Xiaoen Lin, and Hongbo Yu. Improved preimage attacks on 4-round Keccak-224/256. *IACR Trans. Symmetric Cryptol.*, 2021(1):217–238, 2021.
37. Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional cube attack on reduced-round Keccak sponge function. In *EURO-CRYPT 2017, Proceedings, Part II*, volume 10211, pages 259–288, 2017.
38. Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel. A practical attack on KeeLoq. In *EUROCRYPT 2008, Proceedings*, volume 4965, pages 1–18. Springer, 2008.
39. Takanori Isobe. A single-key attack on the full GOST block cipher. *J. Cryptol.*, 26(1):172–189, 2013.
40. Simon Knellwolf and Dmitry Khovratovich. New preimage attacks against reduced SHA-1. In *CRYPTO 2012, Proceedings*, volume 7417, pages 367–383.
41. Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of NLFSR-based cryptosystems. In *ASIACRYPT 2010 Proceedings*, volume 6477, pages 130–145. Springer, 2010.
42. Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka v2 - efficient short-input hashing for post-quantum applications. *IACR Trans. Symmetric Cryptol.*, 2016(2):1–29, 2016.
43. Gaëtan Leurent. MD4 is not one-way. In *FSE 2008*, volume 5086, pages 412–428.
44. Ting Li and Yao Sun. Preimage attacks on round-reduced Keccak-224/256 via an allocating approach. In *EUROCRYPT 2019, Proceedings, Part III*, volume 11478, pages 556–584. Springer, 2019.
45. 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.
46. Xiaoen Lin, Le He, and Hongbo Yu. Improved preimage attacks on 3-round Keccak-224/256. *IACR Trans. Symmetric Cryptol.*, 2021(3):84–101, 2021.
47. Fukang Liu, Takanori Isobe, Willi Meier, and Zhonghao Yang. Algebraic attacks on round-reduced Keccak. In *ACISP 2021, Proceedings*, volume 13083, pages 91–110.
48. Guozhen Liu, Jingwen Lu, Huina Li, Peng Tang, and Weidong Qiu. Preimage attacks against lightweight scheme xoodyak based on deep learning. In *Future of Information and Communication Conference*, pages 637–648. Springer, 2021.
49. Stefan Lucks. Attacking triple encryption. In *FSE '98, Proceedings*, volume 1372, pages 239–253. Springer, 1998.
50. Ralph C. Merkle. A certified digital signature. In *CRYPTO 1989, Proceedings*, pages 218–238, 1989.
51. Pawel Morawiecki, Josef Pieprzyk, and Marian Srebrny. Rotational cryptanalysis of round-reduced Keccak. In *FSE 2013,*, volume 8424, pages 241–262.
52. María Naya-Plasencia, Andrea Röck, and Willi Meier. Practical analysis of reduced-round Keccak. In *INDOCRYPT 2011*, volume 7107, pages 236–254.
53. Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *CRYPTO '93*, volume 773, pages 368–378.

54. Mahesh Sreekumar Rajasree. Cryptanalysis of round-reduced Keccak using non-linear structures. In *INDOCRYPT 2019*, volume 11898, pages 175–192.
55. Yu Sasaki. Integer linear programming for three-subset meet-in-the-middle attacks: Application to GIFT. In *IWSEC 2018*, volume 11049, pages 227–243.
56. Yu Sasaki. Meet-in-the-middle preimage attacks on AES hashing modes and an application to whirlpool. In *FSE 2011*, pages 378–396. Springer, 2011.
57. Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In *EUROCRYPT 2009, Proceedings*, volume 5479, pages 134–152.
58. Yu Sasaki and Kazumaro Aoki. Preimage attacks on 3, 4, and 5-pass HAVAL. In *ASIACRYPT 2008, Proceedings*, volume 5350, pages 253–271. Springer, 2008.
59. André Schrottenloher and Marc Stevens. Simplified MITM modeling for permutations: New (quantum) attacks. *IACR Cryptol. ePrint Arch.*, page 189, 2022.
60. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *CRYPTO 2005, Proceedings*, volume 3621, pages 17–36. Springer, 2005.
61. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *EUROCRYPT 2005, Proceedings*, volume 3494, pages 19–35. Springer, 2005.

Supplementary Material

A An Example of the MitM Attack

We take the MitM preimage attack on 7-round AES-hashing in [56] as an example.

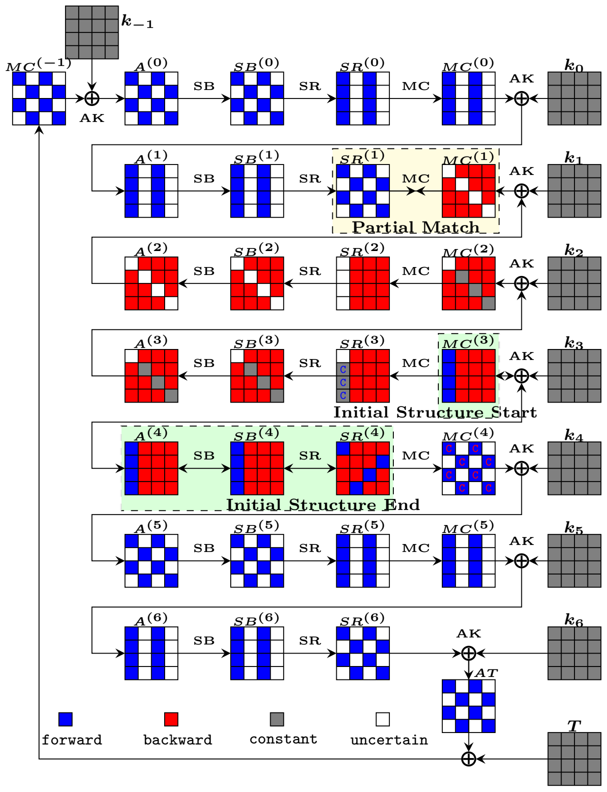

Fig. 15: The MitM preimage attack on 7-round AES-hashing.

Denote the internal states of round r as

$$A^{(r)} \xrightarrow{SB} SB^{(r)} \xrightarrow{SR} SR^{(r)} \xrightarrow{MC} MC^{(r)} \xrightarrow{AK} A^{(r+1)}.$$

$A^{(r)}_{\{i\}}$ represents the i-th ($0 \leq i \leq 15$) byte of state $A^{(r)}$ numbered from up to bottom, left to right. $A^{(r)}_{\{i-j\}}$ represents the i-th byte to j-th byte of state $A^{(r)}$.

Chunk Separation: As shown in Figure 15, the initial structure involves a few consecutive starting steps, i.e. $\{MC^{(3)}, A^{(4)}, SB^{(4)}, SR^{(4)}\}$. The $MC^{(3)}_{\{0-3\}}$

are chosen as neutral bytes (marked by blue) for the forward chunk and the $SR^{(4)}_{\{1-6,8,9,11,12,14,15\}}$ (marked by red) are chosen as neutral bytes for the backward chunk. Results from two chunks will match at $SR^{(1)}$ and $MC^{(1)}$ for a partial match.

Constraints on Initial Structure: To make the initial structure work, one needs to add 3 constraints on the neutral bytes for the forward chunk $MC^{(3)}_{\{0-3\}}$ to avoid the impacts on the backward chunk. The bytes $SR^{(3)}_{\{1,2,3\}}$ can be predetermined constant values as follows:

$$\begin{bmatrix} \mathtt{c}_0 = 9 \cdot MC^{(3)}_{\{0\}} \oplus e \cdot MC^{(3)}_{\{1\}} \oplus b \cdot MC^{(3)}_{\{2\}} \oplus d \cdot MC^{(3)}_{\{3\}} \\ \mathtt{c}_1 = d \cdot MC^{(3)}_{\{0\}} \oplus 9 \cdot MC^{(3)}_{\{1\}} \oplus e \cdot MC^{(3)}_{\{2\}} \oplus b \cdot MC^{(3)}_{\{3\}} \\ \mathtt{c}_2 = b \cdot MC^{(3)}_{\{0\}} \oplus d \cdot MC^{(3)}_{\{1\}} \oplus 9 \cdot MC^{(3)}_{\{2\}} \oplus e \cdot MC^{(3)}_{\{3\}} \end{bmatrix}. \tag{24}$$

There are 2^8 values of $MC^{(3)}_{\{0-3\}}$ when the constants $\mathtt{c}_0, \mathtt{c}_1, \mathtt{c}_2$ are determined. Similarly, adding 8 constraints on the neutral bytes for the backward chunk to avoid the impacts on 8 bytes $MC^{(4)}_{\{0,2,5,7,8,10,13,15\}}$.

Matching through MC: According to the property of the MC operation, the match is tested column by column. There are totally five bytes known in each column of $SR^{(1)}$ and $MC^{(1)}$. So there has one byte matching for each column. Taking the match for first column as an example. The $SR^{(1)}_{\{0,2\}}$ are deduced in the forward computation and $MC^{(1)}_{\{1,2,3\}}$ are deduced in the backward computation. There has

$$\begin{aligned} & d \cdot SR^{(1)}_{\{0\}} \oplus e \cdot SR^{(1)}_{\{2\}} \\ =& d \cdot (b \cdot MC^{(1)}_{\{1\}} \oplus d \cdot MC^{(1)}_{\{2\}} \oplus 9 \cdot MC^{(1)}_{\{3\}}) \oplus e \cdot (9 \cdot MC^{(1)}_{\{1\}} \oplus e \cdot MC^{(1)}_{\{2\}} \oplus b \cdot MC^{(1)}_{\{3\}}). \end{aligned} \tag{25}$$

Forward and Backwork Computation: The forward computation list contains the blue neutral bits in $MC^{(3)}$ to $SR^{(1)}$. When accounting for the constraints, one can compute the neutral bytes in the forward chunk by traversing 2^8 possible values of $MC^{(3)}_{\{0-3\}}$. Then store $MC^{(3)}_{\{0-3\}}$ in table L_1 indexed by the value of $SR^{(1)}$ as the left part of Equ. (25) (i.e. $d \cdot SR^{(1)}_{\{0\}} \oplus e \cdot SR^{(1)}_{\{2\}}$). Similarly, the backward computation list contains the red neutral bits in $SR^{(4)}$ to $MC^{(1)}$. Store them in table L_2 indexed by the value of $MC^{(1)}$ as the right part of Equ. (25). Then one can use L_1 and L_2 for a 32-bit partial match on the indices.

B Figures of 4-round MitM Preimage Attack on Keccak-512

The offset $\gamma[x,y]$ in `Keccak` round function is given in Table 4.

	$x=0$	$x=1$	$x=2$	$x=3$	$x=4$
$y=0$	0	1	62	28	27
$y=1$	36	44	6	55	20
$y=2$	3	10	43	25	39
$y=3$	41	45	15	21	8
$y=4$	18	2	61	56	14

Table 4: The offset $\gamma[x,y]$ in the ρ operation for `Keccak`

The figures of the 4-round MitM attack on `Keccak`-512 are given as Figure 10,16,17,18.

C An Experiment on 3-round Keccak-512

We give an experimental attack on 3-round `Keccak-512` in Figure 19,20,21,22 to verify our method. In the attack, the starting state $A^{(0)}$ have 16 🟥 and 16 🟦 bits, which are given in Table 5. The consumes of degree of freedom happen at the first θ operation due to the linear structure, which consuming 8 🟥 and 8 🟦. Therefore, $\text{DoF}_{\mathcal{B}} = 16-8 = 8$, $\text{DoF}_{\mathcal{R}} = 16-8 = 8$. In Table 6, there are totally 64 conditions on $\theta^{(0)}$. The degree of matching is counted by the deterministic relations of $A^{(2)}$ according to Observation 1, we get DoM $= 11$ and list the equations for filtering as Equ. (26).

red bit	$A^{(0)}_{\{0,0,2\}}, A^{(0)}_{\{0,1,2\}}, A^{(0)}_{\{2,0,13\}}, A^{(0)}_{\{2,1,13\}}, A^{(0)}_{\{0,0,17\}}, A^{(0)}_{\{0,1,17\}}, A^{(0)}_{\{2,0,32\}}, A^{(0)}_{\{2,1,32\}}, A^{(0)}_{\{2,0,37\}}, A^{(0)}_{\{2,1,37\}}, A^{(0)}_{\{2,0,40\}}, A^{(0)}_{\{2,1,40\}}, A^{(0)}_{\{0,0,48\}}, A^{(0)}_{\{0,1,48\}}, A^{(0)}_{\{0,0,63\}}, A^{(0)}_{\{0,1,63\}}$
blue bit	$A^{(0)}_{\{0,0,8\}}, A^{(0)}_{\{0,1,8\}}, A^{(0)}_{\{0,0,9\}}, A^{(0)}_{\{0,1,9\}}, A^{(0)}_{\{0,0,13\}}, A^{(0)}_{\{0,1,13\}}, A^{(0)}_{\{0,0,41\}}, A^{(0)}_{\{0,1,41\}}, A^{(0)}_{\{2,0,45\}}, A^{(0)}_{\{2,1,45\}}, A^{(0)}_{\{2,0,49\}}, A^{(0)}_{\{2,1,49\}}, A^{(0)}_{\{2,0,62\}}, A^{(0)}_{\{2,1,62\}}, A^{(0)}_{\{2,0,1\}}, A^{(0)}_{\{2,1,1\}}$

Table 5: The red bits and blue bits in $A^{(0)}$ of 3-round experiment on `Keccak-512`

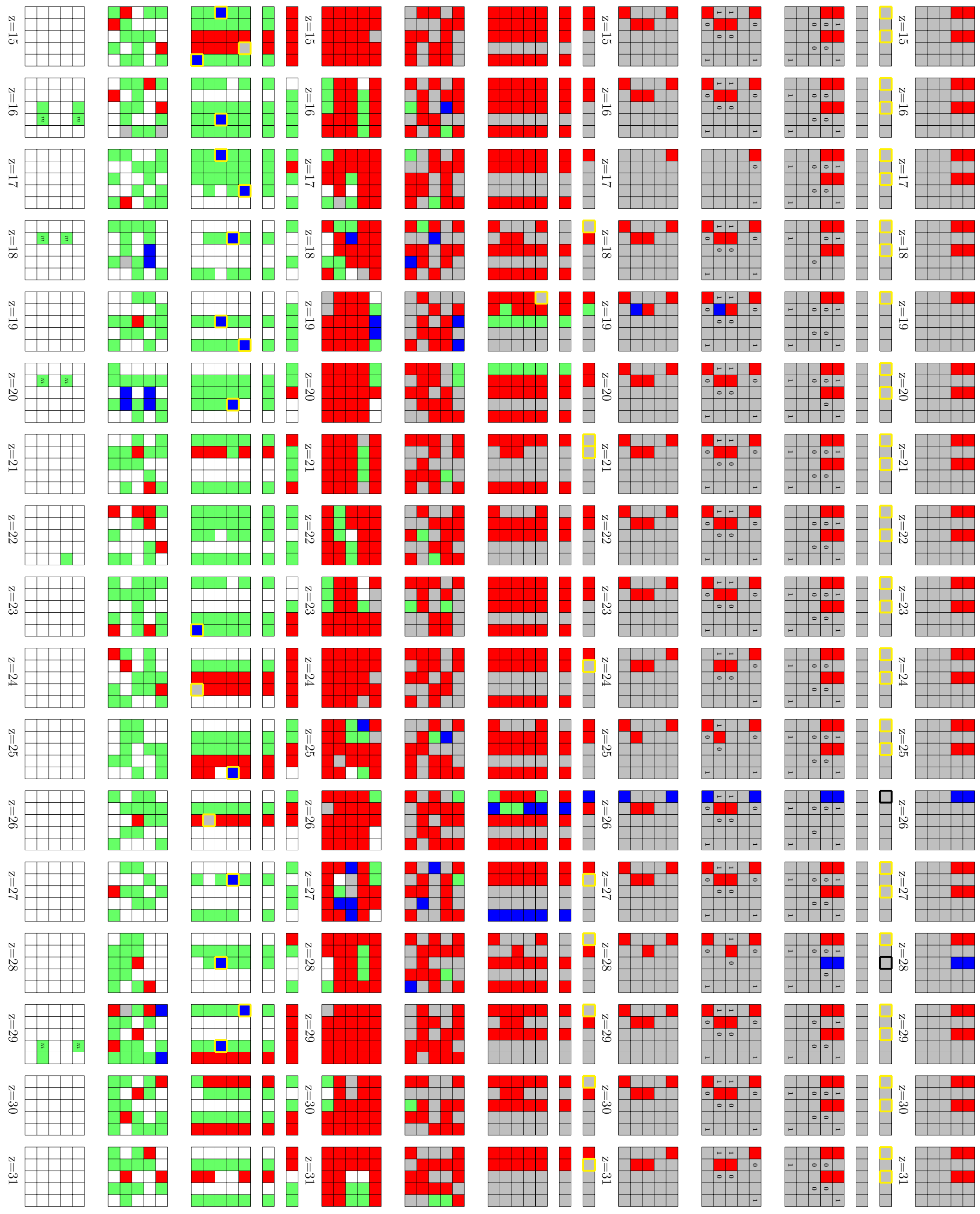

Fig. 16: The MitM preimage attack on 4-round Keccak-512 (part II)

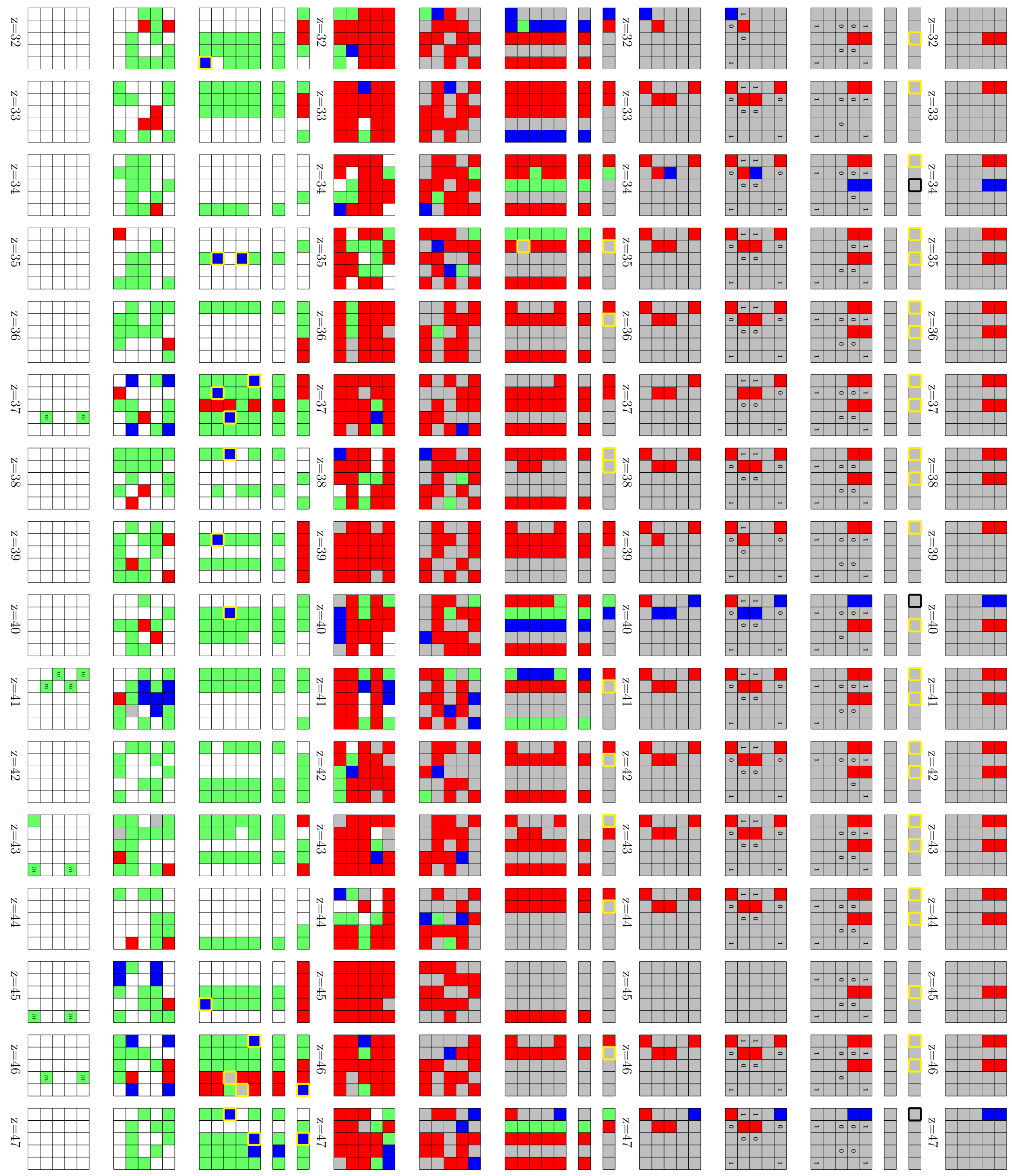

Fig. 17: The MitM preimage attack on 4-round Keccak-512 (part III)

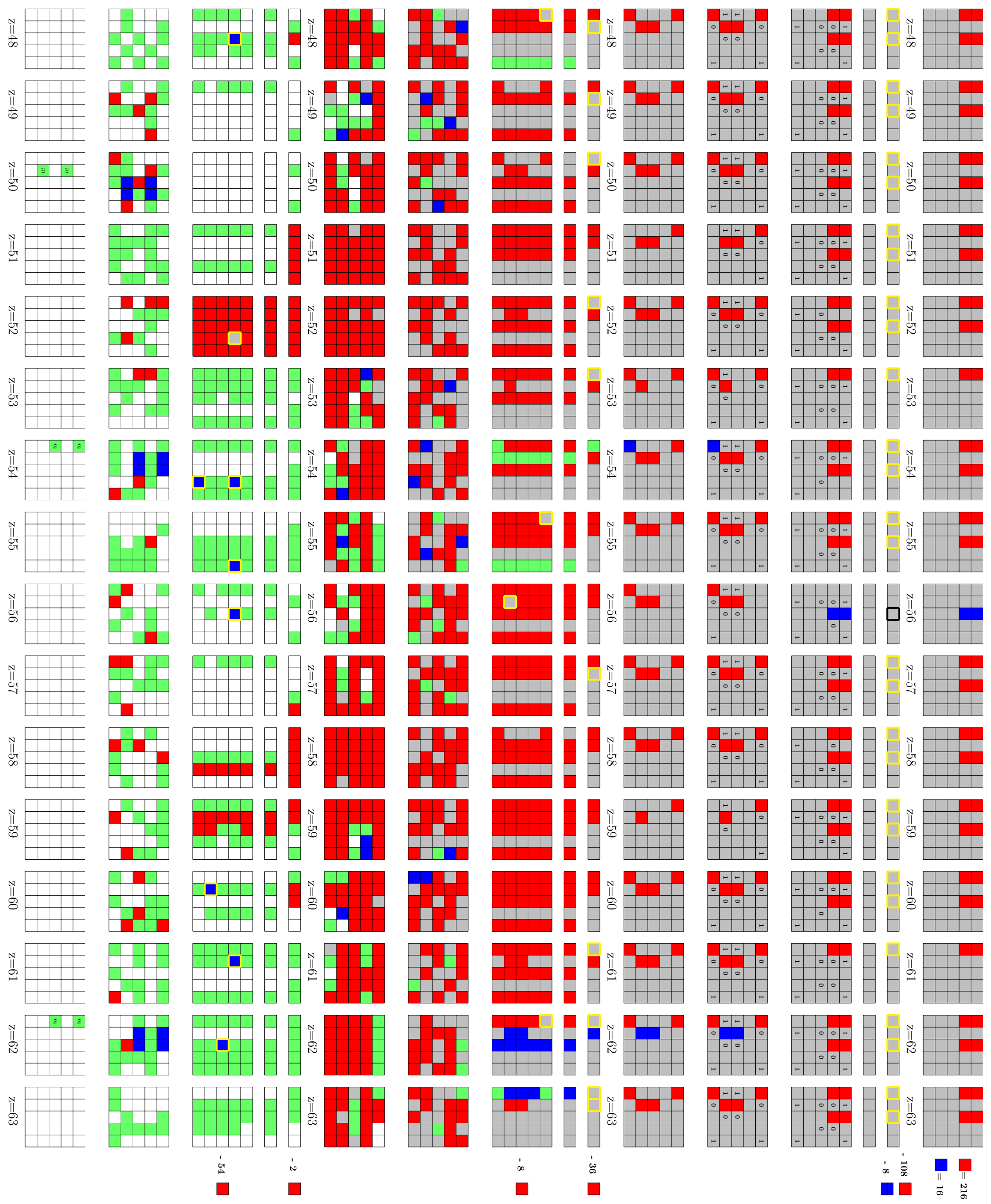

Fig. 18: The MitM preimage attack on 4-round Keccak-512 (part IV)

$\theta^{(0)}_{\{1,1,4\}}=0,\theta^{(0)}_{\{1,1,19\}}=0,\theta^{(0)}_{\{1,1,22\}}=0,\theta^{(0)}_{\{1,1,28\}}=0,\theta^{(0)}_{\{1,1,29\}}=0,\theta^{(0)}_{\{1,1,33\}}=0,$ $\theta^{(0)}_{\{1,1,37\}}=0,\theta^{(0)}_{\{1,1,61\}}=0,\theta^{(0)}_{\{1,2,3\}}=0,\theta^{(0)}_{\{1,2,10\}}=0,\theta^{(0)}_{\{1,2,25\}}=0,\theta^{(0)}_{\{1,2,28\}}=0,$ $\theta^{(0)}_{\{1,2,34\}}=0,\theta^{(0)}_{\{1,2,35\}}=0,\theta^{(0)}_{\{1,2,39\}}=0,\theta^{(0)}_{\{1,2,43\}}=0,\theta^{(0)}_{\{3,1,5\}}=0,\theta^{(0)}_{\{3,1,8\}}=0,$ $\theta^{(0)}_{\{3,1,20\}}=0,\theta^{(0)}_{\{3,1,39\}}=0,\theta^{(0)}_{\{3,1,44\}}=0,\theta^{(0)}_{\{3,1,47\}}=0,\theta^{(0)}_{\{3,1,52\}}=0,\theta^{(0)}_{\{3,1,56\}}=0,$ $\theta^{(0)}_{\{3,2,13\}}=0,\theta^{(0)}_{\{3,2,18\}}=0,\theta^{(0)}_{\{3,2,21\}}=0,\theta^{(0)}_{\{3,2,26\}}=0,\theta^{(0)}_{\{3,2,30\}}=0,\theta^{(0)}_{\{3,2,43\}}=0,$ $\theta^{(0)}_{\{3,2,46\}}=0,\theta^{(0)}_{\{3,2,58\}}=0,$
$\theta^{(0)}_{\{1,0,3\}}=1,\theta^{(0)}_{\{1,0,6\}}=1,\theta^{(0)}_{\{1,0,18\}}=1,\theta^{(0)}_{\{1,0,37\}}=1,\theta^{(0)}_{\{1,0,42\}}=1,\theta^{(0)}_{\{1,0,45\}}=1,$ $\theta^{(0)}_{\{1,0,50\}}=1,\theta^{(0)}_{\{1,0,54\}}=1,\theta^{(0)}_{\{1,4,9\}}=1,\theta^{(0)}_{\{1,4,28\}}=1,\theta^{(0)}_{\{1,4,33\}}=1,\theta^{(0)}_{\{1,4,36\}}=1,$ $\theta^{(0)}_{\{1,4,41\}}=1,\theta^{(0)}_{\{1,4,45\}}=1,\theta^{(0)}_{\{1,4,58\}}=1,\theta^{(0)}_{\{1,4,61\}}=1,\theta^{(0)}_{\{4,0,8\}}=1,\theta^{(0)}_{\{4,0,11\}}=1,$ $\theta^{(0)}_{\{4,0,17\}}=1,\theta^{(0)}_{\{4,0,18\}}=1,\theta^{(0)}_{\{4,0,22\}}=1,\theta^{(0)}_{\{4,0,26\}}=1,\theta^{(0)}_{\{4,0,50\}}=1,\theta^{(0)}_{\{4,0,57\}}=1,$ $\theta^{(0)}_{\{4,4,3\}}=1,\theta^{(0)}_{\{4,4,27\}}=1,\theta^{(0)}_{\{4,4,34\}}=1,\theta^{(0)}_{\{4,4,49\}}=1,\theta^{(0)}_{\{4,4,52\}}=1,\theta^{(0)}_{\{4,4,58\}}=1,$ $\theta^{(0)}_{\{4,4,59\}}=1,\theta^{(0)}_{\{4,4,63\}}=1.$

Table 6: Bit conditions in $\theta^{(0)}$ of 3-round experiment on `Keccak`-512

$$
\begin{aligned}
&A^{(2)}_{\{3,0,36\}} \oplus A^{(2)}_{\{3,3,36\}} \oplus (A^{(3)}_{\{1,1,0\}} \oplus 1)\cdot(A^{(2)}_{\{0,2,61\}} \oplus A^{(2)}_{\{0,0,61\}}) \oplus A^{(3)}_{\{0,1,0\}} \oplus \theta^{(3)}_{\{3,3,36\}} \oplus (A^{(3)}_{\{1,1,0\}} \oplus 1)\cdot\theta^{(2)}_{\{0,0,61\}} = 0,\\
&A^{(2)}_{\{3,0,58\}} \oplus A^{(2)}_{\{3,3,58\}} \oplus (A^{(3)}_{\{1,1,22\}} \oplus 1)\cdot(A^{(2)}_{\{0,2,19\}} \oplus A^{(2)}_{\{0,0,19\}}) \oplus A^{(3)}_{\{0,1,22\}} \oplus \theta^{(3)}_{\{3,3,58\}} \oplus (A^{(3)}_{\{1,1,22\}} \oplus 1)\cdot\theta^{(2)}_{\{0,0,19\}} = 0,\\
&A^{(2)}_{\{3,0,2\}} \oplus A^{(2)}_{\{3,3,2\}} \oplus (A^{(3)}_{\{1,1,30\}} \oplus 1)\cdot(A^{(2)}_{\{0,2,27\}} \oplus A^{(2)}_{\{0,0,27\}}) \oplus A^{(3)}_{\{0,1,30\}} \oplus \theta^{(3)}_{\{3,3,2\}} \oplus (A^{(3)}_{\{1,1,30\}} \oplus 1)\cdot\theta^{(2)}_{\{0,0,27\}} = 0,\\
&A^{(2)}_{\{3,0,27\}} \oplus A^{(2)}_{\{3,3,27\}} \oplus (A^{(3)}_{\{1,1,55\}} \oplus 1)\cdot(A^{(2)}_{\{0,2,52\}} \oplus A^{(2)}_{\{0,0,52\}}) \oplus A^{(3)}_{\{0,1,55\}} \oplus \theta^{(3)}_{\{3,3,27\}} \oplus (A^{(3)}_{\{1,1,55\}} \oplus 1)\cdot\theta^{(2)}_{\{0,0,52\}} = 0,\\
&A^{(2)}_{\{4,1,1\}} \oplus A^{(2)}_{\{4,4,1\}} \oplus (A^{(3)}_{\{2,1,21\}} \oplus 1)\cdot(A^{(2)}_{\{1,3,40\}} \oplus A^{(2)}_{\{1,1,40\}}) \oplus A^{(3)}_{\{1,1,21\}} \oplus \theta^{(3)}_{\{4,4,1\}} \oplus (A^{(3)}_{\{2,1,21\}} \oplus 1)\cdot\theta^{(2)}_{\{1,1,40\}} = 0,\\
&A^{(2)}_{\{4,1,4\}} \oplus A^{(2)}_{\{4,4,4\}} \oplus (A^{(3)}_{\{2,1,24\}} \oplus 1)\cdot(A^{(2)}_{\{1,3,43\}} \oplus A^{(2)}_{\{1,1,43\}}) \oplus A^{(3)}_{\{1,1,24\}} \oplus \theta^{(3)}_{\{4,4,4\}} \oplus (A^{(3)}_{\{2,1,24\}} \oplus 1)\cdot\theta^{(2)}_{\{1,1,43\}} = 0,\\
&A^{(2)}_{\{4,1,5\}} \oplus A^{(2)}_{\{4,4,5\}} \oplus (A^{(3)}_{\{2,1,25\}} \oplus 1)\cdot(A^{(2)}_{\{1,3,44\}} \oplus A^{(2)}_{\{1,1,44\}}) \oplus A^{(3)}_{\{1,1,25\}} \oplus \theta^{(3)}_{\{4,4,5\}} \oplus (A^{(3)}_{\{2,1,25\}} \oplus 1)\cdot\theta^{(2)}_{\{1,1,44\}} = 0,\\
&A^{(2)}_{\{4,1,17\}} \oplus A^{(2)}_{\{4,4,17\}} \oplus (A^{(3)}_{\{2,1,37\}} \oplus 1)\cdot(A^{(2)}_{\{1,3,56\}} \oplus A^{(2)}_{\{1,1,56\}}) \oplus A^{(3)}_{\{1,1,37\}} \oplus \theta^{(3)}_{\{4,4,17\}} \oplus (A^{(3)}_{\{2,1,37\}} \oplus 1)\cdot\theta^{(2)}_{\{1,1,56\}} = 0,\\
&A^{(2)}_{\{4,1,20\}} \oplus A^{(2)}_{\{4,4,20\}} \oplus (A^{(3)}_{\{2,1,40\}} \oplus 1)\cdot(A^{(2)}_{\{1,3,59\}} \oplus A^{(2)}_{\{1,1,59\}}) \oplus A^{(3)}_{\{1,1,40\}} \oplus \theta^{(3)}_{\{4,4,20\}} \oplus (A^{(3)}_{\{2,1,40\}} \oplus 1)\cdot\theta^{(2)}_{\{1,1,59\}} = 0,\\
&A^{(2)}_{\{4,1,41\}} \oplus A^{(2)}_{\{4,4,41\}} \oplus (A^{(3)}_{\{2,1,61\}} \oplus 1)\cdot(A^{(2)}_{\{1,3,16\}} \oplus A^{(2)}_{\{1,1,16\}}) \oplus A^{(3)}_{\{1,1,61\}} \oplus \theta^{(3)}_{\{4,4,41\}} \oplus (A^{(3)}_{\{2,1,61\}} \oplus 1)\cdot\theta^{(2)}_{\{1,1,16\}} = 0,\\
&A^{(2)}_{\{4,1,43\}} \oplus A^{(2)}_{\{4,4,43\}} \oplus (A^{(3)}_{\{2,1,63\}} \oplus 1)\cdot(A^{(2)}_{\{1,3,18\}} \oplus A^{(2)}_{\{1,1,18\}}) \oplus A^{(3)}_{\{1,1,63\}} \oplus \theta^{(3)}_{\{4,4,43\}} \oplus (A^{(3)}_{\{2,1,63\}} \oplus 1)\cdot\theta^{(2)}_{\{1,1,18\}} = 0.
\end{aligned}
\tag{26}
$$

Experiments. We use two message blocks (M_1, M_2) to conduct the MitM experiment on 3-round `Keccak-512`. For message block M_2, we randomly choose its values, and set $A_{\{0,0,z\}} = A_{\{0,1,z\}}$ and $A_{\{2,0,z\}} = A_{\{2,1,z\}}$ for the 🟥 and 🟦 bits listed in Table 5 due to the linear structure. To make the experiment simple, we carefully set the inner part to make the 64 bit conditions in Table 6 satisfied directly, and only test the MitM episodes. We pre-compute the hash value as the target. We only test one MitM episode, where the other bits exclude 🟥 and 🟦 bits in $A^{(0)}$ keep unchanged with the above settings. Similar with Algorithm 1, $2^8 \times 2^8$ values of 🟥 and 🟦 bits are exhausted and filtered with Equ. (26) in the episode. We make the experiment for 10 times with different M_2 and precomputed targets, and the right values for 🟥 and 🟦 bits will remain among the $2^8 \times 2^8 \times 2^{-11} = 2^5$ matched states. The details of the code are provided at ....

D The Attack Algorithm of 3-round `Xoodyak-XOF`

By our MILP model, we derive the MitM characteristic on 3-round `Xoodyak-XOF` in Figure 23. The 3-round attack on `Xoodyak-XOF` is given in Algorithm 3.

E MitM Preimage Attack on Round-Reduced `Ascon-XOF`

E.1 Figures of 4-round Attack on `Ascon-XOF`

Solving our MILP model for `Ascon`, we get a 4-round MITM preimage attack as shown in Figure 24.

E.2 3-round Attack on `Ascon-XOF`

We follow similar framework in Figure 6 and perform the attack with three message blocks (M_1, M_2, M_3), where M_3 has two padding bits '10'. The MITM attack is placed in the 3rd block. Solving with our MILP model for `Ascon`, we get a 3-round MITM preimage attack as shown in Figure 25. The starting state $A^{(0)}$ contains 10 🟦 bits and 39 🟥 bits. There are totally 54 conditions on ⬛ bits of $A^{(0)}$, which are listed in Table 7. In the computation from $A^{(0)}$ to $A^{(2)}$, the accumulated consumed degree of freedom of 🟥 is 29 and there is no DoF of 🟦 consumed. Therefore, $\mathrm{DoF}_{\mathcal{B}} = 10$, $\mathrm{DoF}_{\mathcal{R}} = 39-29 = 10$. The degree of matching is counted by the deterministic relations of $A^{(2)}$ according to Observation 3, we get $\mathrm{DoM} = 10$.

Before the MITM attack, we need to find a right (M_1, M_2) to make the 54 conditions in Table 7 hold with probability of 2^{-54}. Once we find a right (M_1, M_2), we search the space of M_3 with MITM method to find the preimage, whose size is 2^{62} (there are 2 padding bits fixed). Therefore, to find a 128-bit target preimage, we need $2^{128-62} = 2^{66}$ right (M_1, M_2), which need $2^{66+54} = 2^{120}$ time complexity to find. Then we call Algorithm 4 to perform the MITM preimage attack on the 3-round `Ascon`. In Line 7, the time complexity is $2^{120} \times 2^{-54} \times 2^{13} \times 2^{39} = 2^{118}$. The time of one MITM episode between Line 9 to 17 is about 2^{10} 3-round `Ascon`. The total time is $2^{120} + 2^{118} + 2^{120-54+13+29} \times 2^{10} = 2^{120.58}$. The memory to store U is 2^{39}.

F An Experiment on 3-round `Ascon-XOF`

We give an experimental test of the MITM episode on 3-round `Ascon-XOF` in Figure 26 to verify our method. In the attack, the starting state $A^{(0)}$ have 7 🟥 and 5 🟦 bits. The consumes of degree of freedom happen at the second linear layer, which consuming 2 🟥. Therefore, $\mathrm{DoF}_{\mathcal{B}} = 5$, $\mathrm{DoF}_{\mathcal{R}} = 7-2 = 5$. In Table 8,

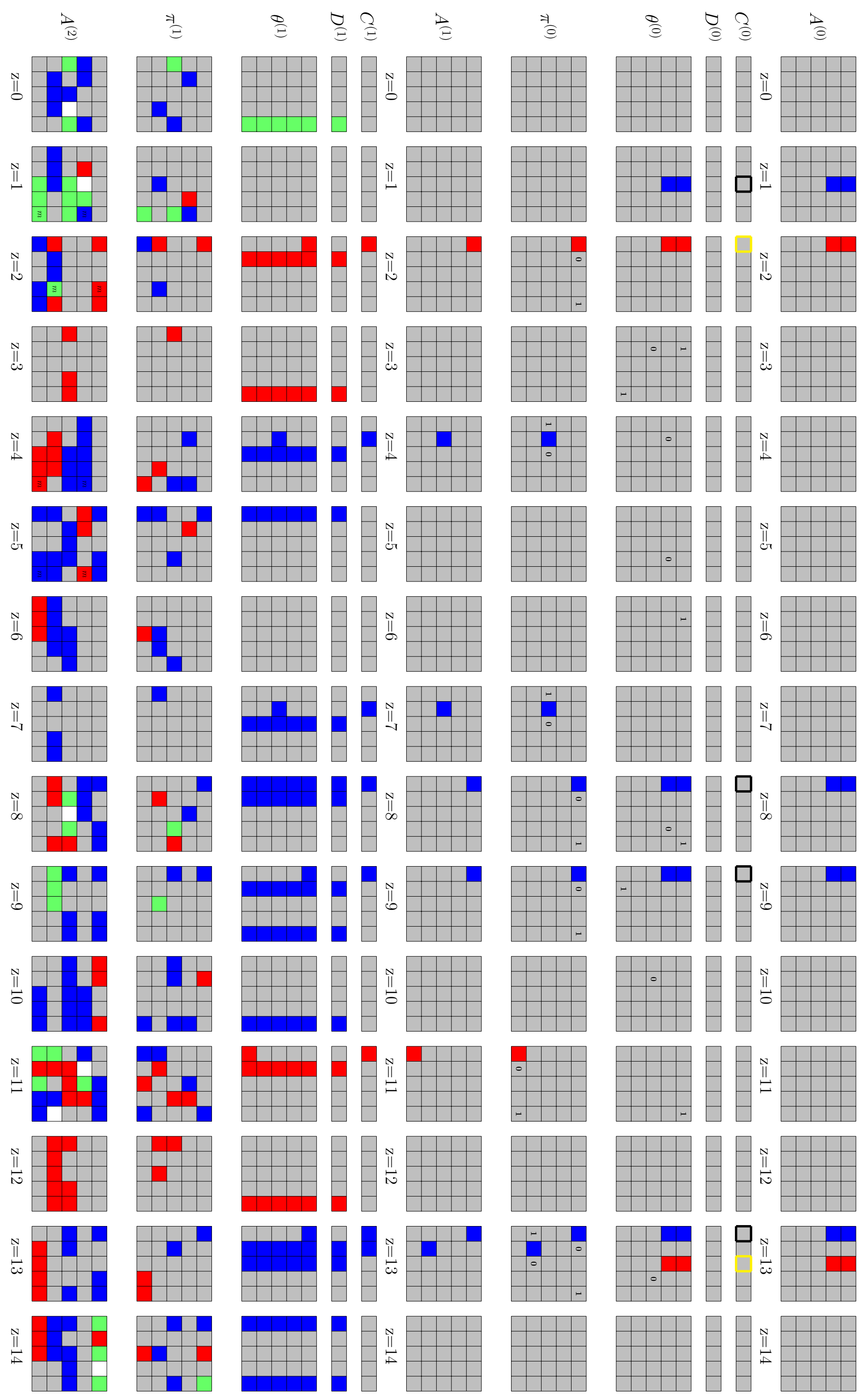

Fig. 19: The 3-round experiment on Keccak-512 (Part I)

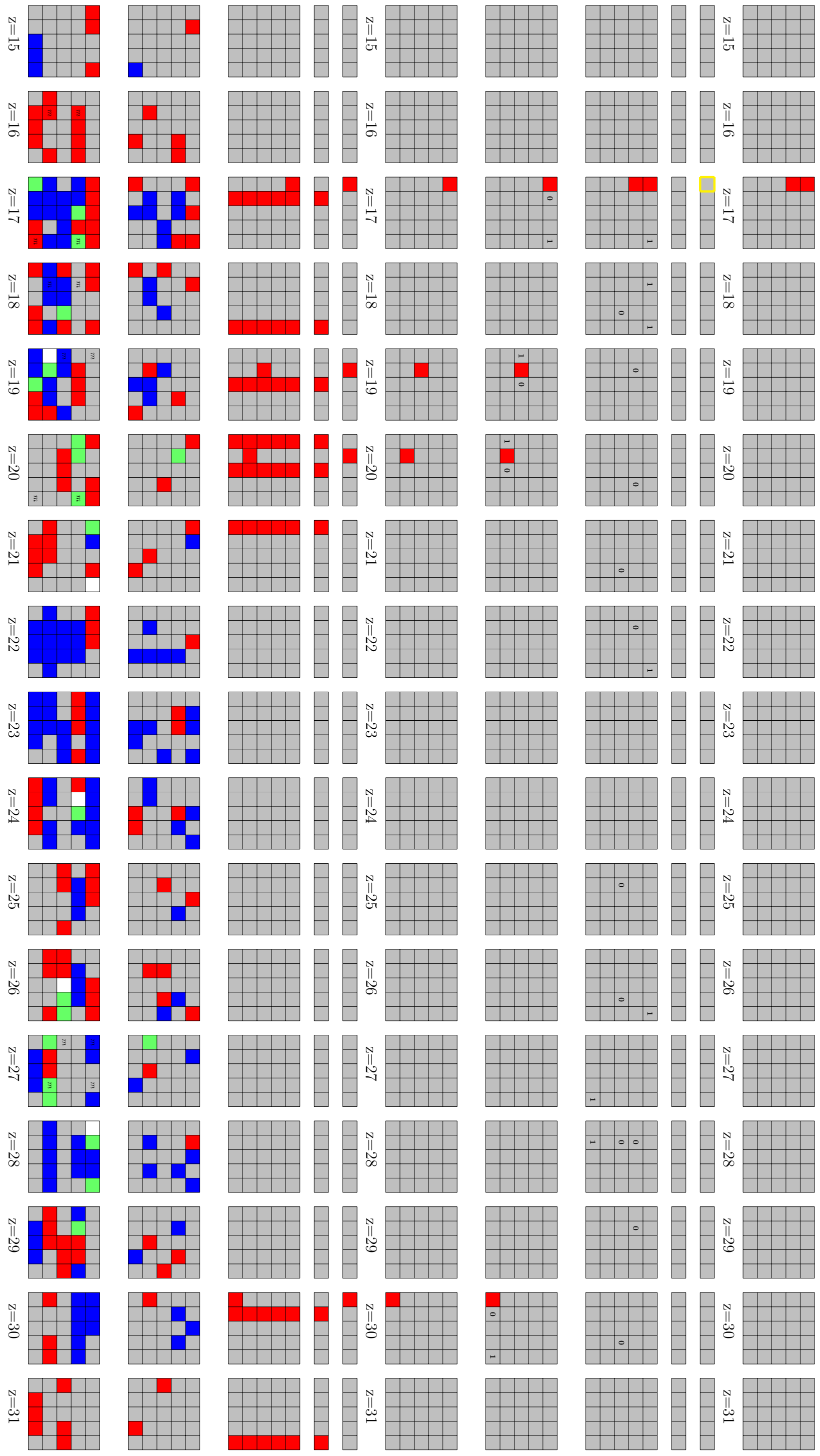

Fig. 20: The 3-round experiment on Keccak-512 (Part II)

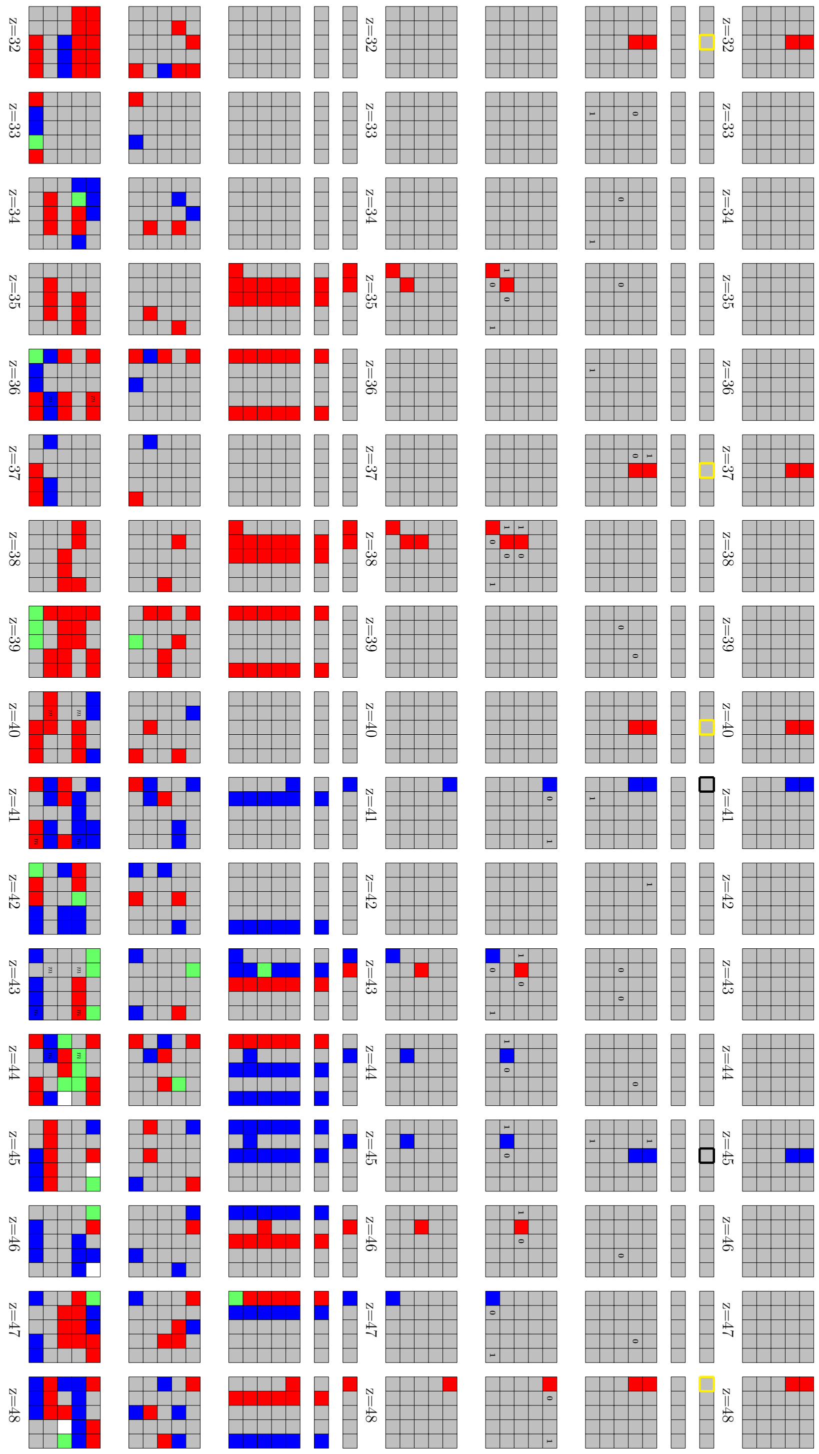

Fig. 21: The 3-round experiment on Keccak-512 (Part III)

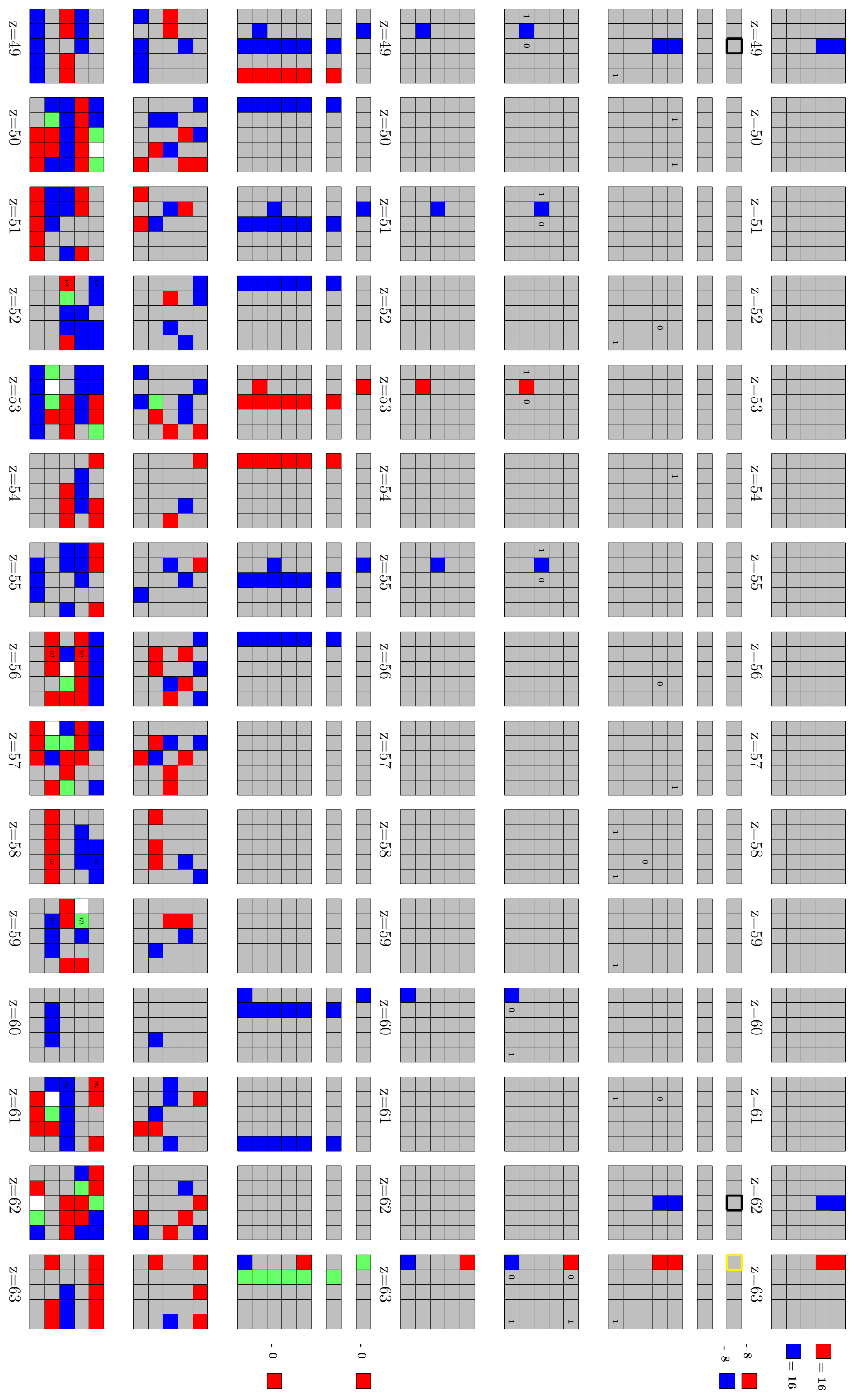

Fig. 22: The 3-round experiment on Keccak-512 (Part IV)

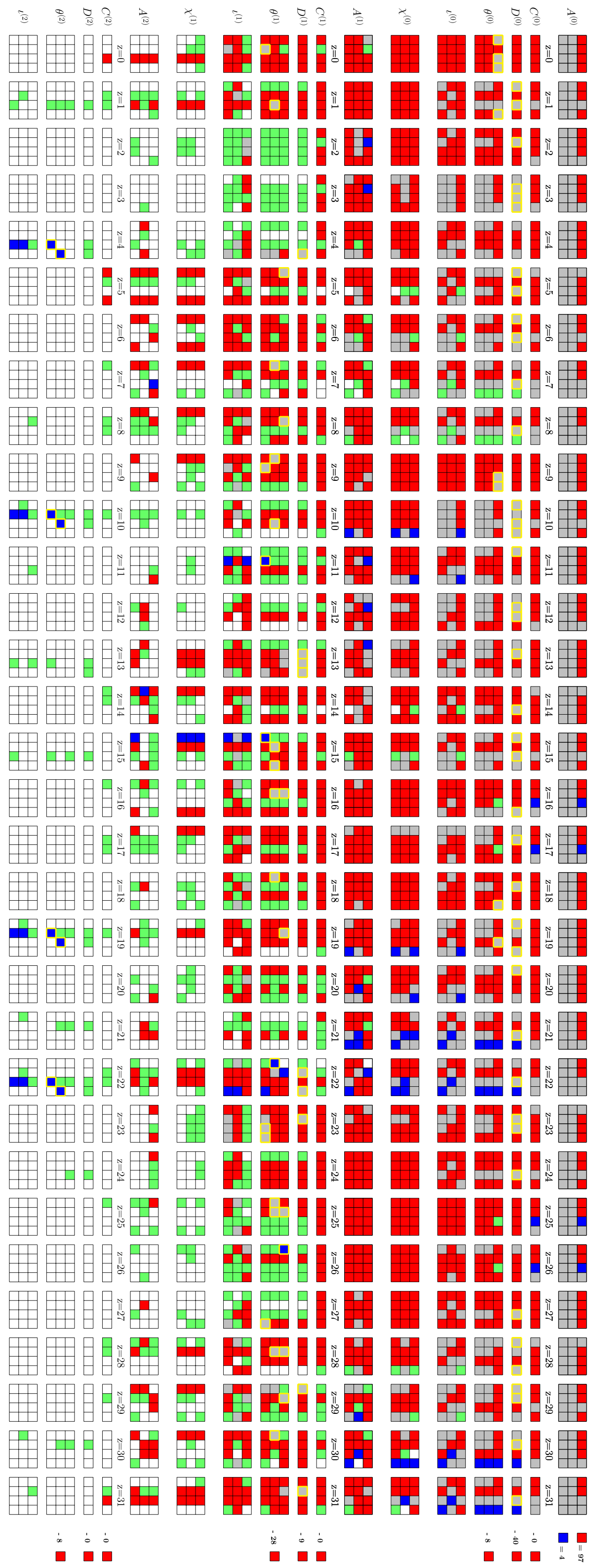

Fig. 23: The mitm preimage attack on 3-round Xoodyak-XOF

Algorithm 3: Preimage Attack on 3-round `Xoodyak-XOF`

```
for $2^x$ values of $M_1$ do
    Compute the inner part of the 2nd block and solve the system of 53 linear equations
    if the equations have solutions   /* with probability of $2^{-40}$ */
    then
        for each of the $2^{12}$ solutions of $M_2$ do
            /* With $x = 55$, there are $2^{55-40+12} = 2^{27}$ iterations */
            Traversing the $2^{97}$ values of 🟥 in $A^{(0)}$ while fixing 🟦 as 0, compute forward to determine 93-bit ▨/🟦 bits (denoted as $c_{\mathcal{R}} \in \mathbb{F}_2^{93}$), and the 4-bit matching point in Equ. (22), i.e., compute four bits $f'_{\mathcal{M}} = f_{\mathcal{R}} \oplus f_{\mathcal{G}}$. Build the table $U$ and store the 97-bit 🟥 bits of $A^{(0)}$ as well as the 4-bit matching point in $U[c_{\mathcal{R}}]$.
            for $c_{\mathcal{R}} \in \mathbb{F}_2^{93}$ do
                Randomly pick a 97-bit 🟥 $e \in U[c_{\mathcal{R}}]$, and set 🟦 in $A^{(0)}$ as 0, compute to the matching point to get 4 bits $f'''_{\mathcal{M}} = f_{\mathcal{G}} + Const(e)$
                for $2^4$ values in $U[c_{\mathcal{R}}]$ do
                    Restore the values of 🟥 of $A^{(0)}$ and the corresponding matching point (i.e., 4 $f_{\mathcal{R}} \oplus f_{\mathcal{G}} = f'_{\mathcal{M}}$) in a list $L_1$ (indexed by matching point)
                end
                for $2^4$ values of 🟦 do
                    Set the 97-bit 🟥 in $A^{(1)}$ as $e$. Compute to the matching point to get eight $f''_{\mathcal{M}} = f_{\mathcal{B}} + f_{\mathcal{G}} + Const(e)$. Together with $f'''_{\mathcal{M}}$, compute $f_{\mathcal{B}} = f''_{\mathcal{M}} + f'''_{\mathcal{M}}$ and store 🟦 in $L_2$ indexed by matching point.
                end
                for values matched between $L_1$ and $L_2$ do
                    Compute $\iota^{(2)}$ from the matched 🟥 and 🟦 cells
                    if $\iota^{(2)}$ satisfy the first 5 Sbox   /* Probability of $2^{-5}$. This step is to avoid computing all Sboxes of $\iota^{(2)}$ and only use partial Sboxes to filter first. */
                    then
                        Check $\iota^{(2)}$ against the remaining 119 Sboxes
                        if it leads to the given hash value then
                            Output the preimage
                        end
                    end
                end
            end
        end
    end
end
```

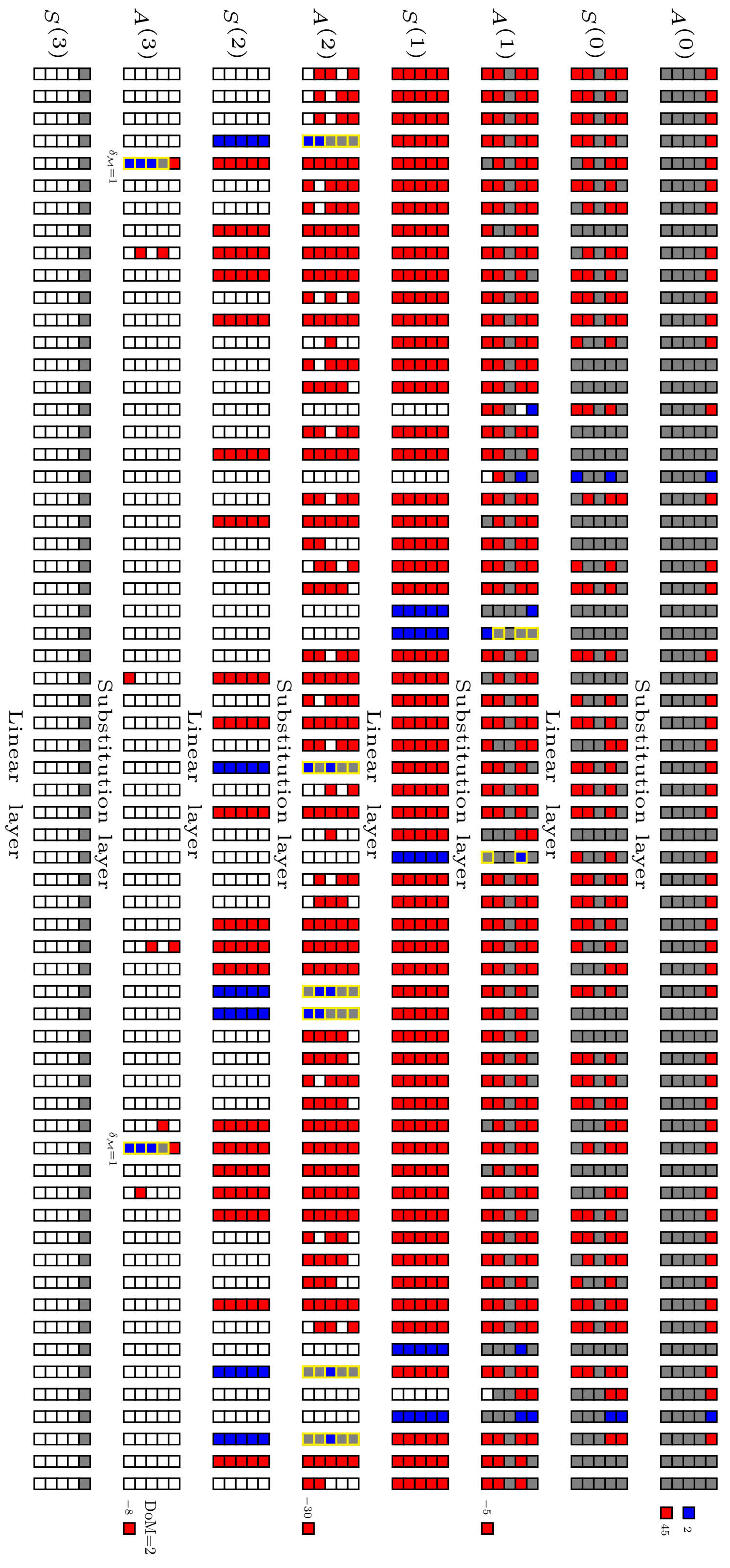

Fig. 24: The MitM preimage attack on 4-round `Ascon-XOF`

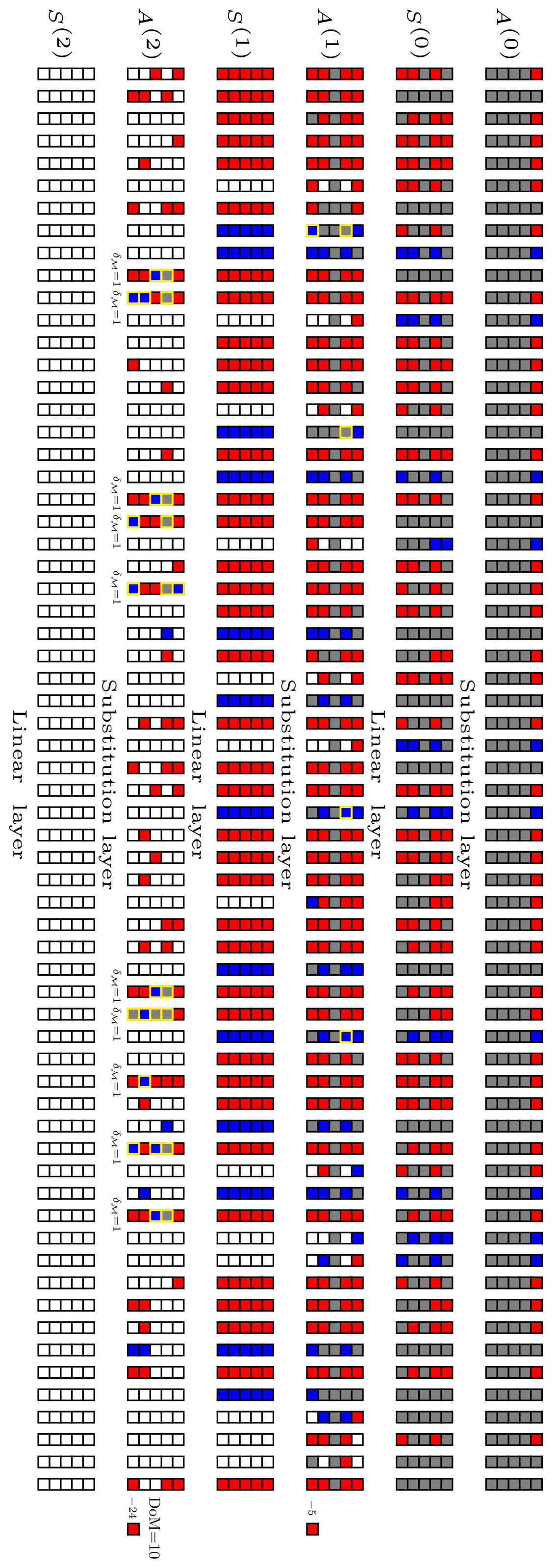

Fig. 25: The MitM preimage attack on 3-round `Ascon-XOF`

$A^{(0)}_{\{0,1\}} = 1; A^{(0)}_{\{2,1\}} = 0; A^{(0)}_{\{5,1\}} = 1; A^{(0)}_{\{7,1\}} = 1, A^{(0)}_{\{7,3\}} \oplus A^{(0)}_{\{7,4\}} = 1; A^{(0)}_{\{8,1\}} = 1;$
$A^{(0)}_{\{11,1\}} = 1; A^{(0)}_{\{12,1\}} = 1; A^{(0)}_{\{14,1\}} = 1; A^{(0)}_{\{15,1\}} = 1,\ A^{(0)}_{\{15,3\}} \oplus A^{(0)}_{\{15,4\}} = 1;$
$A^{(0)}_{\{18,1\}} = 1,\ A^{(0)}_{\{18,3\}} \oplus A^{(0)}_{\{18,4\}} = 1; A^{(0)}_{\{19,1\}} = 1; A^{(0)}_{\{21,1\}} = 0,\ A^{(0)}_{\{21,3\}} \oplus A^{(0)}_{\{21,4\}} = 1;$
$A^{(0)}_{\{22,1\}} = 1; A^{(0)}_{\{23,1\}} = 1,\ A^{(0)}_{\{23,3\}} \oplus A^{(0)}_{\{23,4\}} = 1; A^{(0)}_{\{24,1\}} = 1;$
$A^{(0)}_{\{26,1\}} = 0,\ A^{(0)}_{\{26,3\}} \oplus A^{(0)}_{\{26,4\}} = 1; A^{(0)}_{\{29,1\}} = 1,\ A^{(0)}_{\{29,3\}} \oplus A^{(0)}_{\{29,4\}} = 1; A^{(0)}_{\{30,1\}} = 1;$
$A^{(0)}_{\{33,1\}} = 0; A^{(0)}_{\{36,1\}} = 0,\ A^{(0)}_{\{36,3\}} \oplus A^{(0)}_{\{36,4\}} = 1; A^{(0)}_{\{37,1\}} = 0,\ A^{(0)}_{\{37,3\}} \oplus A^{(0)}_{\{37,4\}} = 1;$
$A^{(0)}_{\{38,1\}} = 1; A^{(0)}_{\{39,1\}} = 0; A^{(0)}_{\{41,1\}} = 0; A^{(0)}_{\{42,1\}} = 0,\ A^{(0)}_{\{42,3\}} \oplus A^{(0)}_{\{42,4\}} = 1; A^{(0)}_{\{43,1\}} = 0;$
$A^{(0)}_{\{44,1\}} = 1; A^{(0)}_{\{48,1\}} = 0; A^{(0)}_{\{49,1\}} = 1,\ A^{(0)}_{\{49,3\}} \oplus A^{(0)}_{\{49,4\}} = 1;$
$A^{(0)}_{\{50,1\}} = 1,\ A^{(0)}_{\{50,3\}} \oplus A^{(0)}_{\{50,4\}} = 1; A^{(0)}_{\{51,1\}} = 0; A^{(0)}_{\{52,1\}} = 0;$
$A^{(0)}_{\{53,1\}} = 1,\ A^{(0)}_{\{53,3\}} \oplus A^{(0)}_{\{53,4\}} = 1; A^{(0)}_{\{54,1\}} = 1,\ A^{(0)}_{\{54,3\}} \oplus A^{(0)}_{\{54,4\}} = 1;$
$A^{(0)}_{\{55,1\}} = 0,\ A^{(0)}_{\{55,3\}} \oplus A^{(0)}_{\{55,4\}} = 1; A^{(0)}_{\{56,1\}} = 0; A^{(0)}_{\{58,1\}} = 0;$
$A^{(0)}_{\{61,1\}} = 1,\ A^{(0)}_{\{61,3\}} \oplus A^{(0)}_{\{61,4\}} = 1$

Table 7: Bit Conditions in 3-round Attack on `Ascon-XOF`

Algorithm 4: Preimage Attack on 3-round `Ascon-XOF`

```
for $2^{120}$ values of $(M_1, M_2)$ do
    Compute the inner part of the 3rd block
    if the conditions of Table 7 are satisfied /* probability of $2^{-54}$ */
     then
        for $2^{13}$ values of ■ in $A^{(0)}$ /* 64-2-10-39=13 */
         do
            Traversing the $2^{39}$ values for 🟥 in $A^{(0)}$ while fixing 🟦 as a random
              constant, build the table $U$ with the index of $c_{\mathcal{R}} \in \mathbb{F}_2^{29}$.
            for $c_{\mathcal{R}} \in \mathbb{F}_2^{29}$ do
                for $2^{10}$ values in $U[c_{\mathcal{R}}]$ do
                    Compute to the matching point, store them in a list $L_1$
                end
                for $2^{10}$ values of 🟦 do
                    Compute to the matching point $L_2$
                end
                for values matched between $L_1$ and $L_2$ do
                    if it leads to the given hash value then
                        Output the preimage
                    end
                end
            end
        end
    end
end
```

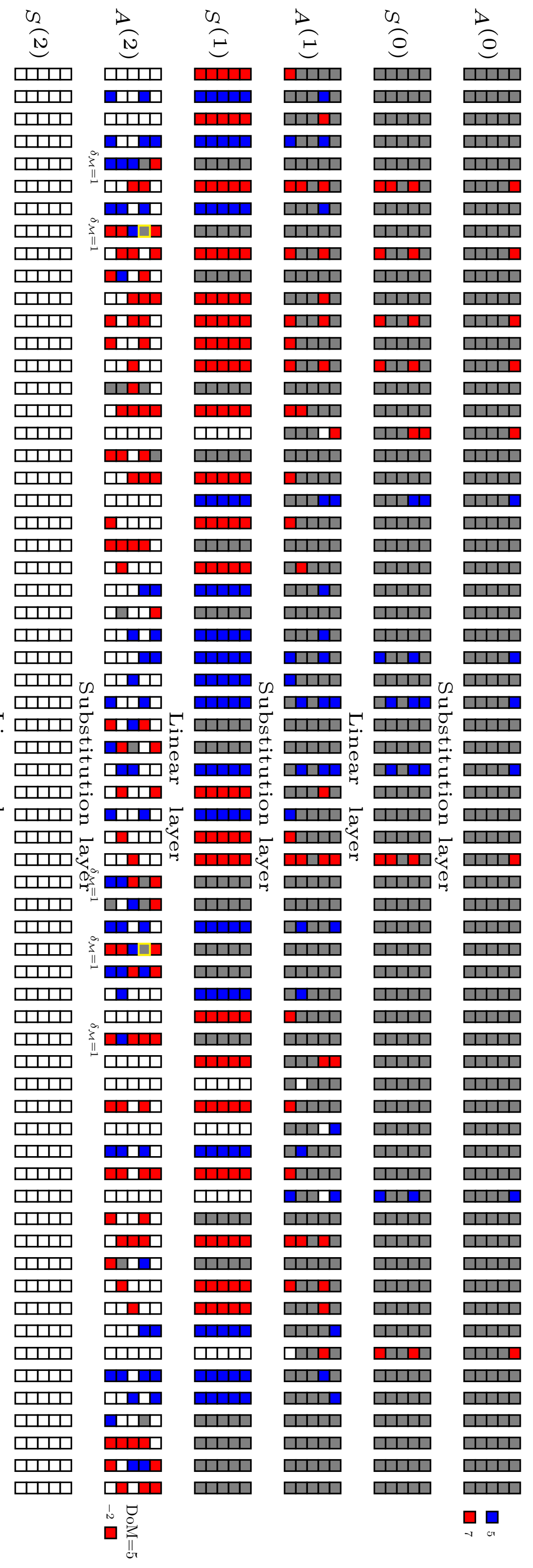

Fig. 26: The experiment on 3-round Ascon-XOF

$A^{(0)}_{\{5,1\}} = 1; A^{(0)}_{\{8,1\}} = 1, A^{(0)}_{\{8,3\}} \oplus A^{(0)}_{\{8,4\}} = 1; A^{(0)}_{\{11,1\}} = 1, A^{(0)}_{\{11,3\}} \oplus A^{(0)}_{\{11,4\}} = 1;$
$A^{(0)}_{\{13,1\}} = 1, A^{(0)}_{\{13,3\}} \oplus A^{(0)}_{\{13,4\}} = 1; A^{(0)}_{\{16,1\}} = 0,\ A^{(0)}_{\{16,3\}} \oplus A^{(0)}_{\{16,4\}} = 1;$
$A^{(0)}_{\{19,1\}} = 0, A^{(0)}_{\{19,3\}} \oplus A^{(0)}_{\{19,4\}} = 1; A^{(0)}_{\{26,1\}} = 1,\ A^{(0)}_{\{16,3\}} \oplus A^{(0)}_{\{26,4\}} = 1; A^{(0)}_{\{28,1\}} = 0;$
$A^{(0)}_{\{31,1\}} = 0; A^{(0)}_{\{35,1\}} = 1; A^{(0)}_{\{50,1\}} = 1,\ A^{(0)}_{\{50,3\}} \oplus A^{(0)}_{\{50,4\}} = 1; A^{(0)}_{\{57,1\}} = 1,\ A^{(0)}_{\{57,3\}} \oplus A^{(0)}_{\{57,4\}} = 1;$

Table 8: Bit Conditions in 3-round Experiment on `Ascon-XOF`

there are totally 20 conditions on $A^{(0)}$. The degree of matching is counted by the deterministic relations of $A^{(2)}$ according to Observation 3, we get DoM = 5 and list the equations for filtering as Equ. (27).

$$\begin{cases}
A^{(3)}_{\{4,4\}} \cdot A^{(3)}_{\{4,1\}} \oplus A^{(3)}_{\{4,3\}} \oplus A^{(3)}_{\{4,2\}} \cdot A^{(3)}_{\{4,1\}} \oplus A^{(3)}_{\{4,2\}} \oplus A^{(3)}_{\{4,1\}} \cdot A^{(3)}_{\{4,0\}} \oplus A^{(3)}_{\{4,1\}} \oplus A^{(3)}_{\{4,0\}} = S^{(3)}_{\{4,0\}}, \\
A^{(3)}_{\{7,4\}} \cdot A^{(3)}_{\{7,1\}} \oplus A^{(3)}_{\{7,3\}} \oplus A^{(3)}_{\{7,2\}} \cdot A^{(3)}_{\{7,1\}} \oplus A^{(3)}_{\{7,2\}} \oplus A^{(3)}_{\{7,1\}} \cdot A^{(3)}_{\{7,0\}} \oplus A^{(3)}_{\{7,1\}} \oplus A^{(3)}_{\{7,0\}} = S^{(3)}_{\{7,0\}}, \\
A^{(3)}_{\{36,4\}} \cdot A^{(3)}_{\{36,1\}} \oplus A^{(3)}_{\{36,3\}} \oplus A^{(3)}_{\{36,2\}} \cdot A^{(3)}_{\{36,1\}} \oplus A^{(3)}_{\{36,2\}} \oplus A^{(3)}_{\{36,1\}} \cdot A^{(3)}_{\{36,0\}} \oplus A^{(3)}_{\{36,1\}} \oplus A^{(3)}_{\{36,0\}} = S^{(3)}_{\{36,0\}}, \\
A^{(3)}_{\{39,4\}} \cdot A^{(3)}_{\{39,1\}} \oplus A^{(3)}_{\{39,3\}} \oplus A^{(3)}_{\{39,2\}} \cdot A^{(3)}_{\{39,1\}} \oplus A^{(3)}_{\{39,2\}} \oplus A^{(3)}_{\{39,1\}} \cdot A^{(3)}_{\{39,0\}} \oplus A^{(3)}_{\{39,1\}} \oplus A^{(3)}_{\{39,0\}} = S^{(3)}_{\{39,0\}}, \\
A^{(3)}_{\{43,4\}} \cdot A^{(3)}_{\{43,1\}} \oplus A^{(3)}_{\{43,3\}} \oplus A^{(3)}_{\{43,2\}} \cdot A^{(3)}_{\{43,1\}} \oplus A^{(3)}_{\{43,2\}} \oplus A^{(3)}_{\{43,1\}} \cdot A^{(3)}_{\{43,0\}} \oplus A^{(3)}_{\{43,1\}} \oplus A^{(3)}_{\{43,0\}} = S^{(3)}_{\{43,0\}},
\end{cases} \tag{27}$$

Experiments. To make the experiment simple and only test the MITM episode, we carefully set the inner part to make the 20 bit conditions in Table 8 satisfied directly, and only test the MITM episodes. We pre-compute the hash value as the target. We only test one MITM episode, where the other bits exclude 🟥 and 🟦 bits in $A^{(0)}$ keep unchanged with the above settings. Similar with Algorithm 2, we need to compute a table of 🟥 solutions indexed by 2-bit ▨ of $A^{(2)}$, $2^5 \times 2^5$ values of 🟥 and 🟦 bits are exhausted and filtered with Equ. (27) in the episode. We make the experiment for 32 times with different message blocks and pre-computed targets, and the right values for 🟥 and 🟦 bits always remain among the $2^5 \times 2^5 \times 2^{-5} = 2^5$ matched states. The details of the code are also provided at ....