

# Automatic Search of Meet-in-the-Middle Preimage Attacks on AES-like Hashing

Zhenzhen Bao<sup>2</sup>, Xiaoyang Dong<sup>3</sup>, Jian Guo<sup>2</sup>, Zheng Li<sup>4</sup>,  
Danping Shi<sup>1,5</sup>, Siwei Sun<sup>1,5</sup>, and Xiaoyun Wang<sup>3</sup>

- <sup>1</sup> State Key Laboratory of Information Security, Institute of Information Engineering,  
Chinese Academy of Sciences, China. {shidanping,sunsiwei}@iie.ac.cn
- <sup>2</sup> Division of Mathematical Sciences, School of Physical and Mathematical Sciences,  
Nanyang Technological University, Singapore. {zzbao,guojian}@ntu.edu.sg
- <sup>3</sup> Institute for Advanced Study, Tsinghua University, China.  
{xiaoyangdong,xiaoyunwang}@tsinghua.edu.cn
- <sup>4</sup> Faculty of Information Technology, Beijing University of Technology, China.  
lizhengcn@bjut.edu.cn
- <sup>5</sup> School of Cyber Security, University of Chinese Academy of Sciences, China.

**Abstract.** The Meet-in-the-Middle (MITM) preimage attack is highly effective in breaking the preimage resistance of many hash functions, including but not limited to the full MD5, HAVAL, and Tiger, and reduced SHA-0/1/2. It was also shown to be a threat to hash functions built on block ciphers like AES by Sasaki in 2011. Recently, such attacks on AES hashing modes evolved from merely using the freedom of choosing the internal state to also exploiting the freedom of choosing the message state. However, detecting such attacks especially those evolved variants is difficult. In previous works, the search space of the configurations of such attacks is limited, such that manual analysis is practical, which results in sub-optimal solutions. In this paper, we remove artificial limitations in previous works, formulate the essential ideas of the construction of the attack in well-defined ways, and translate the problem of searching for the best attacks into optimization problems under constraints in Mixed-Integer-Linear-Programming (MILP) models. The MILP models capture a large solution space of valid attacks and the objectives are for the optimal. With such MILP models and using the off-the-shelf solver, it is efficient to search for the best attacks exhaustively. As a result, we obtain the first attacks against the full (5-round) and an extended (5.5-round) version of Haraka-512 v2, and 8-round AES-128 hashing modes, as well as improved attacks covering more rounds of Haraka-256 v2 and other members of AES and Rijndael hashing modes.

**Keywords:** AES, Haraka v2, MITM, Preimage, Automatic search, MILP

## 1 Introduction

The hash function is one of the most important cryptographic primitives, due to its wide and crucial applications such as digital signatures, verification of message integrity and passwords etc. To support these applications, collision

resistance, preimage resistance, and second-preimage resistance form the three basic security requirements for cryptographic hash functions. Unlike many public-key cryptographic systems, whose security can be usually reduced to some hard mathematical problems, most of the hash function standards in use could not enjoy such a security reduction. The confidence of the security strength of many symmetric-key primitives mainly relies on intensive and persistent cryptanalysis from the research community. Hence, such effort is of utmost importance, especially against the basic security properties of the standards and the ones used in practice. In this paper, we mainly focus on preimage resistance of hash functions built on the block cipher Advanced Encryption Standard (AES) [13] and the like (we call them AES-like hashing for short). Typical examples are the three PGV-modes [37] – Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO), and Miyaguchi-Preneel (MP), instantiated with AES. Both PGV-modes and AES have long-standing security supported by rigorous and massive cryptanalysis, including the recent quantum collision attacks [15, 22]. The MMO-mode instantiated with AES is standardized by Zigbee [1] and also suggested by ISO [24] as a standard way of building hash function based on block ciphers. Furthermore, many feature-rich cryptographic protocols, *e.g.*, multi-party computation protocols, use hash functions as building blocks and their instances adopt AES-MMO due to its high efficiency when implemented with AES-NI. Besides, since the standardization of AES, many new ciphers follow a similar design strategy or using AES round function directly as building blocks to share the security proof and implementation benefits, *e.g.*, hash functions Grindahl [29], ECHO [11], Grøstl [17], and Haraka v2 [30].

**THE MITM PREIMAGE ATTACKS.** Informally, preimage resistance refers to the property that, for a hash function  $H$  and a target  $T$  given at random, it is *computationally* difficult to find a message  $x$ , such that  $H(x) = T$ . Theoretically, for a secure hash function with a digest of  $n$  bits in size, the expected number of  $H$  evaluations required to find such an  $x$  is  $2^n$ . Any such algorithm with a time complexity lower than  $2^n$  is considered as a *preimage attack*.

In [8], Aumasson *et al.* devised preimage attacks on step-reduced MD5 and full 3-pass HAVAL [50], in which the key technique can be viewed as the application of local-collision combined with the Meet-in-the-Middle (MITM) approach. Sasaki and Aoki in [40] formally proposed to combine the MITM and local-collision approaches and successfully devised preimage attacks on full versions of 3, 4, and 5-pass HAVAL. Further, they in [6] proposed the *splice-and-cut* technique and in [42] invented the concept of the *initial structure*, which add more strength to the MITM attack, and successfully broke the preimage resistance of the full MD5. These techniques were then formalized as *bicliques* [12, 26, 27], and further evolved to differential views [16, 28]. Since these pioneering works, the MITM preimage attack turned out to be very powerful and found many applications in the last decade. It broke the theoretical preimage security claims of MD4 [18], MD5 [42], Tiger [18, 46], HAVAL [19, 40] and round-reduced variants of many other hash functions such as SHA-0 and SHA-1 [5, 16, 28], SHA-2 [4], BLAKE [16],

HAS-160 [21], RIPEMD and RIPEMD-160 [47], Stribog [2], Whilwind [3], and AES hashing modes [9, 38, 48]. Interestingly, the idea of MITM preimage attack also leads to the progress of collision attacks against reduced SHA-2 [33].

The core of a MITM preimage attack on the hash function is generally a MITM pseudo-preimage attack on its compression function (denoted by CF). The basic idea of the attack on the CF is as follows (take the DM-mode as an example). First, the iterative round-based computation of the CF is divided at an intermediate round (starting point) into two chunks. One chunk is computed forward (named as *forward chunk*), the other is computed backward (named as *backward chunk*), and 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 (matching point). 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*). When an initial structure is used, it covers few consecutive rounds at the starting point, within which 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.

In [38], Sasaki applied such MITM preimage attack to AES-hashing modes. Together with the partial matching technique, the attack successfully penetrated 7 out of the 10, 12, 14 rounds respectively for AES-128, AES-192, and AES-256. Later, Wu *et al.* in [48] improved the complexities in multi-target setting. Different from early MITM attacks on the MD-SHA family, their attacks select the neutral bytes from the internal state and fix the material fed into the key/message-schedule to an arbitrary constant. Recently, such attacks on AES hashing modes evolved to not only using the freedom of selecting the internal state but also exploiting the freedom of selecting the message state (key materials of the block ciphers), and improved results are achieved in [9]. Due to the fact that there are too many possible configurations (selection of neutral words, position of initial structure and matching rounds, extra conditions imposed to limit propagation of neutral words, etc.) to test out by brute-force, all existing attacks cover only a small portion of configurations, which were believed to potentially give better cryptanalysis results according to the attackers' intuition and experiences.

**AUTOMATIC TOOLS.** In the last decade, cryptanalysis has also made significant progress from manual methods to those aided by dedicated computer programs searching for best differential/linear paths etc. [34] and best attacks [14], then to automatic tools such as Mixed Integer Linear Programming (MILP), Constrained Programming (CP), Satisfiability Solvers (SAT), and Simple Theorem Prover (STP). These automatic tools convert the problem of finding better cryptanalytic attacks to optimization problems solvable by the tools, under certain constraints, which ensure the validity of the attacks. They not only enlarge the possible solution space covered by previous manual methods and dedicated search programs, but also helped generalize and even re-define the attack models which in turn further enlarge solution spaces. As a result, these tools have made significant advances

in cryptanalysis, such as differential/linear path search [31, 36, 45], cube(-like) attacks [20, 44], integral attacks based on division properties [49], three-subset and Demirci-Selçuk meet-in-the-middle attacks [39, 43]. These usually lead to attacks for more rounds and/or lower time/memory complexities. With these available capacities, a more accurate security assessment is possible, and many recent primitive designs [10, 30] benefited from these tools in determining the round number and the security margin with better confidence.

It is important to note that, literally every problem in cryptanalysis, complex or simple, can be converted into one under automatic tools. However, when the problem is complex, tools may not be able to output solutions in real time. Hence, different from the traditional manual cryptanalysis, the difficulty of tool-aided cryptanalysis is to find a proper model, which balances the problem solving time and size of solution space the model covers (number of attack configurations in case of AES-like hashing). Obviously, a model covering larger solution space comes with lesser constraints, which is harder to solve by the tools, but has bigger chances to offer better cryptanalysis results. All our effort in this paper is to convert the preimage finding problem into one under the MILP language, by a model covering largest possible solution space, while keeping the model solvable in practical time within our computation capacity in hands.

**OUR CONTRIBUTIONS.** In this paper, we manage to automatize the search for the best MITM preimage attacks with MILP models. We focus ourselves on hash functions built on AES and AES-like ciphers.

We extend the construction of attacks by removing the limitations taken by previous works [9, 38, 48]. That includes releasing the boundaries of the initial structure by applying the essential idea to every possible round; considering the possibility of imposing degree of freedom both from the internal state and from the message, which is done by allowing selecting neutral bytes from both of the encryption state and key state, and for both directions of computation; considering a desynchronized selection of neutral bytes in the encryption computation flow and the key-schedule flow (meaning that we allow the key state, from which the neutral bytes be selected, be at any possible round, instead of adhering to the round at where neutral bytes are selected in the encryption state) as appeared already *e.g.*, in [9, 18].

We formalize the essential idea behind the advanced techniques used in the MITM preimage attack, including the above mentioned extended form of initial structure and the partial matching, using explicit-defined rules. In our formulation of the MITM preimage attacks, we do not pre-set any hard boundaries for the initial structures (*i.e.*, the number of rounds and which rounds are covered), but allow it to evolve automatically according to certain rules from well-defined and potentially desynchronized starting states towards a clear objective. Thanks to this formulation, the MITM preimage attack is ready to be transformed into MILP models covering a larger solution space than previous works.

We refine the MILP model for the operations involved in AES-like round functions to accurately capture all possible effects of them on the forward and backward computation paths. For example, instead of separately treating the

**Table 1:** Results of applications of our tool compared with previous best results

Target	#Round	Time-1	Time-2	(DoF <sup>+</sup> , DoF <sup>-</sup> , DoM) in bits	Ref.
AES-128	7/10	$2^{120}$	$2^{125}$	( 8, 8, 32 )	[38]
	7/10	$2^{120-\min(t,24)}$	$2^{123}$	( 8, 32, 32 )	[48]
	7/10	$2^{104}$	$2^{117}$	( 24, 32, 24 )	[9]
	8/10	$2^{120}$	$2^{125}$	( 16, 8, 8 )	Fig. 7
AES-192	7/12	$2^{120}$	$2^{125}$	( 8, 8, 32 )	[38]
	7/12	$2^{96}$	$2^{113}$	( 32, 32, 32 )	[9]
	8/12	$2^{112-\min(t,16)}$	$2^{116}$	( 16, 32, 32 )	[9]
	9/12	$2^{120}$	$2^{125}$	( 8, 8, 8 )	Fig. 8
AES-256	7/14	$2^{120}$	$2^{125}$	( 8, 8, 32 )	[38]
	8/14	$2^{96}$	$2^{113}$	( 32, 32, 32 )	[9]
	9/14	$2^{120-\min(t,24)}$	$2^{123}$	( 8, 32, 32 )	Fig. 9
Rijndael-256	9/14	$2^{248}$	$2^{253}$	( 16, 16, 8 )	Fig. 11
Haraka-256 v2	7/10	$2^{248}$	$2^{248}$	( 8, 8, 96 )	[30]
	9/10	$2^{224}$	$2^{224}$	( 32, 32, 64 )	Fig. 12
Haraka-512 v2	8/10	$2^{248}$	$2^{248}$	( 8, 8, 64 )	[30]
	10/10	$2^{224-\min(t,32)}$	$2^{224}$	( 128, 32, 64 )	Fig. 13
	11/10	$2^{240}$	$2^{240}$	( 128, 128, 16 )	Fig. 14

– Following [9], we use Time-1 to represent the time complexity of pseudo-preimage. Here,  $2^t$  is the number of available targets for multi-target pseudo-preimage attacks; use Time-2 to represent the complexity of using the (multi-target) pseudo-preimage attacks to do (second-)preimage attacks when requiring an upper layer of meet-in-the-middle procedure of conversion for some PGV-modes, and here a single target is given. For Haraka-512 v2, the conversion is not needed and Time-2 should be the same with Time-1.

– The unit of complexity is one computation of the compression function.

– #Round is the number of AES-like round (one Haraka v2 round consists of two AES-like rounds).

– (DoF<sup>+</sup>, DoF<sup>-</sup>, DoM) is (the degree of freedom for forward computation, the degree of freedom for backward computation, the degree of matching), please refer to Sect. 3.

AddRoundKey and MixColumns, we treat them as a whole (a composition transformation) and formalize constraints that can result in all possible impacts from the input states to the output state. In doing that, the models can capture the solutions where the difference in the active cells in the key state and that in encryption state be mutually (partially) canceled, which is impossible when treat the two operations separately. Such treatment further enlarges the search space to capture more potentially better attacks.

With such MILP models and using off-the-shelf solver, we apply the automatic search to AES-like hashing. Improved attacks than the previous ones were obtained. That includes the first preimage attacks on 8-round AES-128, 9-round AES-192, 9-round AES-256, 9-round Rijndael hashing modes, 4.5-round (9 AES-rounds) Haraka-256 v2 and the full 5-round (10 AES-rounds) version and extended 5.5-round (11 AES-rounds) version of Haraka-512 v2. The detailed results, together with a comparison to the previous related works, are summarized in Table 1.

## 2 AES-like Hashing and MITM Preimage Attacks

Most current hash functions are based on compression functions (CF) with fixed length input and output; and the support for variable-length messages can be achieved through domain extenders. Here, we focus on the challenge of inverting the CF, i.e., given one or multiple targets  $T$ , find input chaining value  $h$  and message block  $M$ , such that  $\text{CF}(h, M) = T$ . Such attacks are called pseudo-preimage attacks, in which the chaining value is free of choice. Pseudo-preimages can be converted to (second-)preimages of hash functions using generic methods (details can be found in Supplementary Material C).

### 2.1 AES-like Hashing

Typically, the compression function of hash functions can be constructed from block ciphers applying the secure PGV-modes [37]. When the underlying block ciphers are AES-like, we call the hash functions as AES-like hashing. Concretely, in AES-like hashing, the underlying compression function is based on AES-like round functions as depicted in Fig. 2, where the state being manipulated is organized into an  $N_{\text{row}} \times N_{\text{col}}$  two-dimensional array of  $c$ -bit cells. One AES-like round function typically consists of the following operations:

- **SubBytes.** Substitute each cell according to an S-boxes  $S : \mathbb{F}_{2^c} \rightarrow \mathbb{F}_{2^c}$ .
- **ShiftRows $_{\pi_t}$ .** Permute the cell positions according to the permutation  $\pi_t$ .
- **MixColumns.** Update each column by left-multiplying an  $N_{\text{row}} \times N_{\text{row}}$  MDS matrix (maximal distance separable matrix, with branch number  $B_n = N_{\text{row}} + 1$ , i.e., as long as the input/output of the MDS matrix is non-zero, the sum of non-zero elements in the input and output is at least  $N_{\text{row}} + 1$ ).
- **AddRoundKey.** XOR a round key or a round-dependent constant into the state depending on whether the intended construction is keyed or not.

### 2.2 Advanced Techniques in Meet-in-the-Middle Preimage Attacks

Since the pioneering works on preimage attacks on MD4, MD5, and HAVAL [8, 32, 40, 41], the MITM approach has been applied and further developed for preimage attacks on many other hash functions. This method develops into *splice-and-cut* [6] MITM preimage attacks with support from *initial structure* [42] and (indirect) *partial matching* techniques.

*Initial Structures* [42]. From the idea of local-collision, Sasaki and Aoki proposed a novel concept – *initial structure*. The purpose of the initial structure is to skip several steps/rounds at the beginning of chunks in a MITM attack so that the attack covers more steps/rounds. It is a few consecutive starting steps, where the two chunks overlapped. Although the two sets of neutral words, denoted by  $N^+$  and  $N^-$ , appear simultaneously at these steps, they are only involved in the computation of one chunk each. Besides, one can add constraints to the values of neutral words of one chunk, such that different values lead to constant impact on the computation of the opposite chunk. Thus, a proper initial structure should

satisfy that, steps after the initial structure (forward chunk) can be computed independently of  $N^-$  and steps before the initial structure (backward chunk) can be computed independently of  $N^+$ .

*Remark 1 (Related work – the formalism of Biclique).* Notably, the initial structure was viewed as the most promising and underutilized technique for MITM preimage attack in the subsequent years since its invention. In [27], authors replaced the idea of initial structure with a more formal and general concept, which is named biclique. With this formalism, one can view the structure in a differential view, and built it by applying various tools available for collision search and differential attacks. This concept of biclique has been applied to both preimage attacks on hash functions (*e.g.*, SHA-1, SHA-2 and Skein-512 [27, 28]) and key-recovery attacks on block ciphers (*e.g.*, AES and IDEA [12, 26]).

In this paper, independent of the formalism using concept of biclique, and instead of adhering to a formal definition, we apply the essential idea behind the original concept of initial structure. We formalize the basic idea using explicit rules and extend the initial structure to be less structured.

*(Indirect-) Partial matching [6, 42].* In the two ending states for matching, as long as there remain one common word of which the value can be computed independently between the forward and backward chunks, the matching can be performed. Further, apart from directly matching values of common words, any determined relations between words in the states at the matching point can be exploited to filter out miss-matched computations. For example, Sasaki in [38] exploited the following property of the AES MixColumns to do indirect matching: knowing any  $b$  bytes ( $b > 4$ ) among the input and output of MixColumns on one column, one can built a filter of  $b - 4$  bytes. For example, in Fig. 1d, it is possible to do partial matching between states #MC<sup>1</sup> and #AK<sup>1</sup>, and each column provides  $2 + 3 - 4 = 1$  byte filter, as exemplified in Fig. 1b.

*Multi-targets [18, 48].* When multiple targets are available, it adds the degree of freedom to the chunk where the targets are added to.

**The Attack Framework.** The procedure (Fig. 1) and complexities of the MITM pseudo-preimage attack depend on the following configurations:

1. Chunk separation – the position of initial structure and matching points.
2. The neutral bytes – the selection and the constraints on the neutral bytes, which determine the degrees of freedom for each chunk.
3. The bytes for matching – the deterministic relation used for matching, which determines the filtering ability (degree of matching).

After setting up the configuration, the basic attack procedure goes as follows. Denote the neutral bytes for the forward and backward chunk by  $N^+$  and  $N^-$ :

1. Assign arbitrary compatible values to all bytes except those that depend on the neutral bytes (*e.g.*, the Gray cells in Fig. 1d).

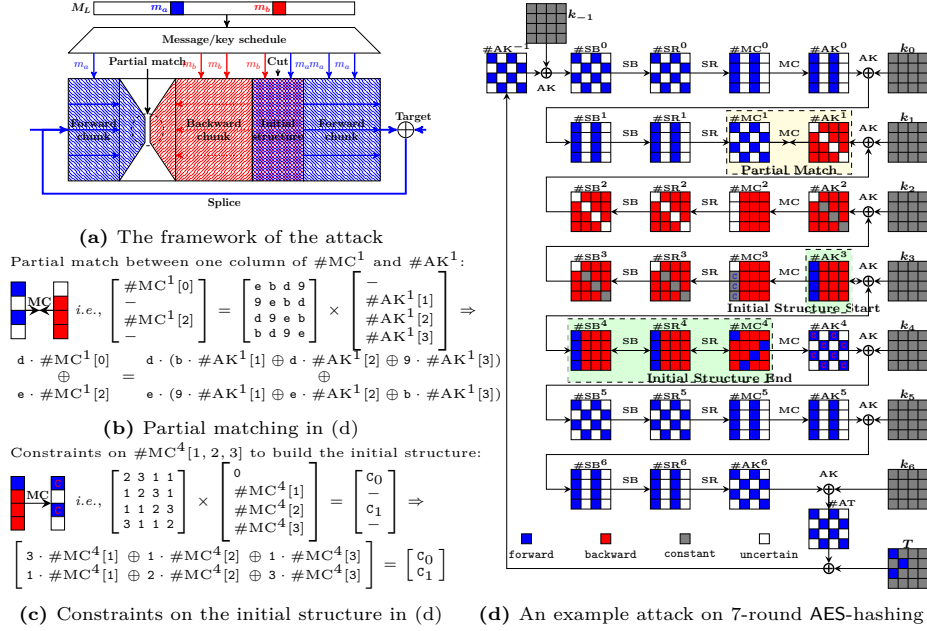


Fig. 1: The MITM pseudo-preimage attack [38, 48]

2. Obtain possible values of neutral bytes  $N^+$  and  $N^-$  under the constraints on them (e.g., in Fig. 1c). Suppose there are  $2^{d_1}$  values for  $N^+$ , and  $2^{d_2}$  for  $N^-$ .
3. For all  $2^{d_1}$  values of  $N^+$ , compute forward from the initial structure to the matching point to get a table  $L^+$ , whose indices are the values for matching, and the elements are the values of  $N^+$ .
4. For all  $2^{d_2}$  values of  $N^-$ , compute backward from the initial structure to the matching point to get a table  $L^-$ , whose indices are the values for matching, and the elements are the values of  $N^-$ .
5. Check whether there is a match on indices between  $L^+$  and  $L^-$ .
6. In case of partial-matching exist in the above step, for the surviving pairs, check for a full-state match. In case none of them are fully matched, repeat the procedure by changing values of fixed bytes till find a full match.

**The Attack Complexity.** Denote the size of the internal state by  $n$ , the degree of freedom in the forward and backward chunks by  $d_1$  and  $d_2$ , and the number of bits for the match by  $m$ , the time complexity of the attack is [9]:

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

### 2.3 Basic Rules Applied to MITM Attacks on AES-like Hashing

**Sources of Degrees of Freedom.** Shown by the complexity analysis, the MITM attack benefits from larger degrees of freedom in both chunks and matching.



In early MITM preimage attacks on the MD-SHA family, the degree of freedom comes from the message words. Whereas, in early MITM preimage attacks on AES-like hashing [38, 48], the degree of freedom comes from the bytes in encryption states<sup>6</sup>, and the attacks set the material fed into the key-schedule as arbitrary constant. In [9], the authors proposed to introduce neutral bytes not only from the encryption state but also from the key state. The principle is that, for one chunk, one adds as much degree of freedom as possible to improve the computational complexity, and at the same time, keeps their impacts on the opposite chunk as little as possible to cover as many rounds as possible. To keep the analysis manually doable, the authors in [9] proposed that the neutral bytes in key states are all introduced for merely one chunk.

**Ways to Control Impacts on the Opposite Chunk.** For the ways to cancel impacts from neutral words for one chunk on the opposite chunk, recall that early preimage attacks on MD-SHA used the (cross) absorption properties of Boolean functions by setting an input variable to a special value to absorb the difference in another input variable. In the attack on AES-like hashing, the ways to control the impacts of the neutral bytes is to add constraints on those neutral bytes when they are inputs to the following operations. Note that adding constraints means consuming the degree of freedom.

- **AddRoundKey** and **XOR**: one can restrict that the **XOR** of two neutral bytes be constant. The rationale is to use the difference in one neutral byte (*e.g.*, in the key state) to absorb the difference in another neutral byte (*e.g.*, in the encryption state). That will consume one-byte degree of freedom.
- **MixColumns** (**MC**): Even if the input contains neutral bytes (active) for one chunk, one can add restriction on their values, such that their impacts on some output bytes of the **MC** be constant. Therefore, the opposite chunk can be computed independently as long as the constant impacts are known. Take the attack in Fig. 1d for example. In the computation from  $\#MC^4$  to  $\#AK^4$ , the values of **Red** cells in state  $\#MC^4$  are restricted such that changing them does not change impact on the **Blue** cells marked by **C** in  $\#AK^4$  (exemplified in Fig. 1c). This restriction consumes the degree of freedom that lies in neutral bytes for backward chunk, but enables the independent forward computation. Explicitly, if there are  $i$  neutral bytes for one chunk involved in the input of **MC**, then we can control their impacts on  $j$  bytes of the output be constant by consuming  $j$  bytes degree of freedom. For AES-like hashing, because the matrix **MC** in **MixColumns** is **MDS**, there is a limitation for applying this control, that is  $i + N_{\text{row}} - j \geq N_{\text{row}} + 1$ , *i.e.*,  $i \geq j + 1$ .
- **MixColumns**  $\circ$  **AddRoundKey** (**XOR-MC**): in backward chunk, when there are forward neutral bytes in both the key and the encryption state, to control their impacts, one may first apply the above-mentioned way of restriction

---

<sup>6</sup> In a hash function, there is no encryption and key-schedule. Here, focusing on hash functions built on block ciphers, we use them to represent the two algorithms that updating the chaining values and that updating the message words. For different mode-of-operations, the correspondence might be different.

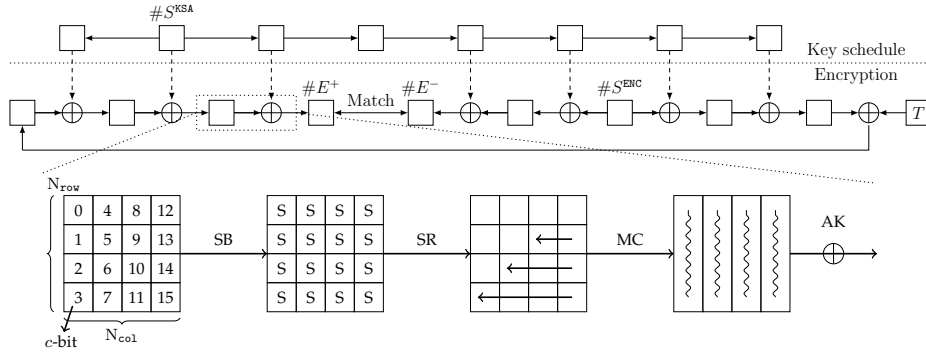
on `AddRoundKey` and then on `MixColumns`. Besides that, we apply restriction on the composition transformation of `AddRoundKey` and `MixColumns`. The rationale is that, the XOR operation in `AddRoundKey` is byte-wise. Only when two bytes being *at the same position* in two states, the difference in one byte can absorb the difference in the other byte. As for `MixColumns`, only when two bytes being *in the same state*, the difference in one byte can absorb the difference in the other byte. However, when considering the composition `MixColumns`  $\circ$  `AddRoundKey`, even when the neutral bytes for the forward chunk lie in different states (some in the key state and some in the encryption state) and in different byte positions, we can still use the difference of some neutral bytes to absorb the difference of others. Sect. 4.1 and the listed attacks will provide formal descriptions and concrete examples.

Explicitly, suppose that there are  $i$  forward neutral bytes in the key state, and  $j$  forward neutral bytes in the encryption state, and they lie in columns with a common index. Let  $k$  be the number of different byte positions considering these neutral bytes together (*i.e.*,  $k$  equals the Hamming weight of the ‘OR’ between the indicator vector of whether a position has a neutral byte in the key state and that in the encryption state). Then, considering the MDS property of MC in `MixColumns`, we can control the impacts of neutral bytes on  $t$  bytes of the output by consuming  $t$  bytes degree of freedom as long as  $k + N_{\text{row}} - t \geq N_{\text{row}} + 1$ , *i.e.*,  $k \geq t + 1$ .

*Remark 2 (Relation with previous MITM attacks on AES hashing modes).* Note that the ways to control the impacts have already been used in previous MITM preimage attacks on AES-like hashing [9, 38, 48], which is an essential element for constructing the initial structure. In this paper, we consider the possibility to impose such constraints to any round, and in this sense, the boundaries of the initial structure disappear. Besides, as has been mentioned above, the ways to select the neutral bytes were limited in previous works to make the analysis doable by manual. In this paper, we remove these restrictions by allowing the selection of neutral bytes in both encryption state and key state, and for both forward and backward chunks.

In the subsequent sections, we will base on these ideas to get explicit rules for selecting neutral bytes, consuming degree of freedom on neutral bytes to control their impacts. Incorporating with other optimization techniques (*e.g.*, partial matching and multi-targets), we convert the problem of searching for the best configurations into optimization problems under constraints in MILP-models. With the obtained MILP-models and the off-the-shelf solver, we can search for the best MITM attacks on AES-like hashing exhaustively.

*Remark 3 (Relation with another work on using MILP to searching MITM attack).* In [39], Sasaki already applied the MILP formalization to search the three-subset MITM attack on GIFT-64. In the tool, which rounds covered by an initial structure are predefined. Neutral bits are all from the key state because the goal is a key-recovery attack. Besides, because it is dedicated to GIFT-64 (with a bit-permutation linear layer), the previously mentioned rules for optimizing



**Fig. 2:** A high-level overview of the MITM preimage attack

MITM attacks on AES-like hashing are not included, which is essentially the most challenging parts in our formalization.

### 3 Formulate the MITM Attack on AES-like Hashing

To search for MITM attacks on AES-like hashing, we now formulate the attack with the general construction shown in Fig. 2.

Denote the *starting states* in the encryption data path and key-schedule data path by  $\#S^{\text{ENC}}$  and  $\#S^{\text{KSA}}$ , respectively (corresponding to the location of an initial structure previously); and denote the *ending states* for the forward computation and backward computation by  $\#E^+$  and  $\#E^-$ , respectively (corresponding to the previous matching). In the formalized attack, partial knowledge of  $\#E^+$  and  $\#E^-$  that is used for matching is supposed to be obtained by computing from  $\#S^{\text{ENC}}$  and  $\#S^{\text{KSA}}$  forward and backward, respectively <sup>7</sup>.

Without loss of generality, we assume that the states in the encryption data paths and the key-schedule both have  $n$   $c$ -bit cells (with  $n = N_{\text{row}} \cdot N_{\text{col}}$ ). To reference the cells of certain  $n$ -cell states, denote by  $\mathcal{B}^{\text{ENC}}, \mathcal{B}^{\text{KSA}}, \mathcal{R}^{\text{ENC}}, \mathcal{R}^{\text{KSA}}, \mathcal{C}$ , and  $\mathcal{D}$  the ordered subsets of  $\mathcal{N} = \{0, 1, \dots, n-1\}$  whose elements are increasingly ordered. Here, the  $\mathcal{B}^{\text{ENC}}$  and  $\mathcal{B}^{\text{KSA}}$  refer to the neutral cells from the internal state and message (or key state of the underlying block cipher) for the forward chunk, and  $\mathcal{R}^{\text{ENC}}$  and  $\mathcal{R}^{\text{KSA}}$  for the backward chunk. The  $\mathcal{C}$  and  $\mathcal{D}$  refer to the known and active cells in the ending states  $\#E^+$  and  $\#E^-$  of the forward and backward chunks, respectively. For example, we may have  $\mathcal{C} = \{0, 2, 7\}$ , and for a 16-cell state  $\#S$ ,  $\#S[\mathcal{C}]$  is defined to be  $(\#S[0], \#S[2], \#S[7])$  or  $\#S[0, 2, 7]$ .

Before one can mount a MITM preimage attack, these four states:  $\#S^{\text{ENC}}, \#S^{\text{KSA}}, \#E^+, \#E^-$ , and six subsets  $\mathcal{B}^{\text{ENC}}, \mathcal{B}^{\text{KSA}}, \mathcal{R}^{\text{ENC}}, \mathcal{R}^{\text{KSA}}, \mathcal{C}, \mathcal{D}$  ( $\mathcal{B}^{\text{ENC}} \cap \mathcal{R}^{\text{ENC}} = \emptyset$  and  $\mathcal{B}^{\text{KSA}} \cap \mathcal{R}^{\text{KSA}} = \emptyset$  for independence between chunks) must be specified.

<sup>7</sup> Note that after finding out a formalized attack, adaptation will be made manually to launch a concrete attack; the forward and backward computations may start from the most decisive states instead of  $\#S^{\text{ENC}}$  and  $\#S^{\text{KSA}}$  while keeping the complexity.

Note that to visualize these subsets and the attack, we will introduce a coloring system in Sect. 4, where cells referenced by  $\mathcal{B}^{\text{ENC}}$  and  $\mathcal{B}^{\text{KSA}}$  are **Blue**, and cells referenced by  $\mathcal{R}^{\text{ENC}}$  and  $\mathcal{R}^{\text{KSA}}$  are **Red**. The remaining cells in the starting states referenced by  $\mathcal{G}^{\text{ENC}}$  and  $\mathcal{G}^{\text{KSA}}$  are **Gray**, where  $\mathcal{G}^{\text{ENC}} = \mathcal{N} - \mathcal{B}^{\text{ENC}} \cup \mathcal{R}^{\text{ENC}}$  and  $\mathcal{G}^{\text{KSA}} = \mathcal{N} - \mathcal{B}^{\text{KSA}} \cup \mathcal{R}^{\text{KSA}}$ . Moreover,  $\mathcal{C}$  references the **Blue** cells in the ending state  $\#E^+$ , and  $\mathcal{D}$  the **Red** cells in  $\#E^-$ .

In what follows, the degree of freedom (DoF) refers to number of cells, rather than bits. We call  $\lambda^+ = |\mathcal{B}^{\text{ENC}}| + |\mathcal{B}^{\text{KSA}}|$  the initial DoF for the forward chunk, and  $\lambda^- = |\mathcal{R}^{\text{ENC}}| + |\mathcal{R}^{\text{KSA}}|$  the initial DoF for the backward chunk. For forward and backward chunks being computed independently, these initial DoFs might be consumed by adding constraints on neutral cells in  $\#S^{\text{ENC}}$  and  $\#S^{\text{KSA}}$ . Thus, neutral cells in the starting states may not take all  $2^{c \cdot \lambda^+}$  and  $2^{c \cdot \lambda^-}$  values.

If the forward neutral cells ( $\#S^{\text{ENC}}[\mathcal{B}^{\text{ENC}}], \#S^{\text{KSA}}[\mathcal{B}^{\text{KSA}}]$ ) (in **Blue**) in the starting states can only take values in  $\mathbb{X} \subseteq \mathbb{F}_{2^c}^{|\mathcal{B}^{\text{ENC}}| + |\mathcal{B}^{\text{KSA}}|}$  with  $|\mathbb{X}| = (2^c)^{d_1} \leq (2^c)^{|\mathcal{B}^{\text{ENC}}| + |\mathcal{B}^{\text{KSA}}|}$ , and the backward neutral cells ( $\#S^{\text{ENC}}[\mathcal{R}^{\text{ENC}}], \#S^{\text{KSA}}[\mathcal{R}^{\text{KSA}}]$ ) (in **Red**) in the starting states can only take values in  $\mathbb{Y} \subseteq \mathbb{F}_{2^c}^{|\mathcal{R}^{\text{ENC}}| + |\mathcal{R}^{\text{KSA}}|}$  with  $|\mathbb{Y}| = (2^c)^{d_2} \leq (2^c)^{|\mathcal{R}^{\text{ENC}}| + |\mathcal{R}^{\text{KSA}}|}$ , then after fixing the **Gray** cells ( $\#S^{\text{ENC}}[\mathcal{G}^{\text{ENC}}], \#S^{\text{KSA}}[\mathcal{G}^{\text{KSA}}]$ ) in the starting states to some constant in  $\mathbb{F}_{2^c}^{(n - |\mathcal{B}^{\text{ENC}}| - |\mathcal{R}^{\text{ENC}}|) + (n - |\mathcal{B}^{\text{KSA}}| - |\mathcal{R}^{\text{KSA}}|)}$ , the attacker can compute  $(2^c)^{d_1}$  different values of  $\#E^+[\mathcal{C}]$  in the forward direction which only depend on ( $\#S^{\text{ENC}}[\mathcal{B}^{\text{ENC}}], \#S^{\text{KSA}}[\mathcal{B}^{\text{KSA}}]$ ). The attacker stores these  $(2^c)^{d_1}$  values in a list  $L^+$ . Similarly, the attacker can compute  $(2^c)^{d_2}$  different values of  $\#E^-[\mathcal{D}]$  in the backward direction which only depend on ( $\#S^{\text{ENC}}[\mathcal{R}^{\text{ENC}}], \#S^{\text{KSA}}[\mathcal{R}^{\text{KSA}}]$ ). The attacker stores these  $(2^c)^{d_2}$  values in a list  $L^-$ . For the two lists  $L^+$  and  $L^-$ , the attacker can perform an  $m$ -cell matching. Then,  $|L^+ \times L^-| / (2^c)^m$  pairs from  $L^+ \times L^-$  are expected to pass the test.

We call  $m$  the degrees of matching (denoted by DoM). Note that  $\mathcal{B}^{\text{ENC}}$  and  $\mathcal{B}^{\text{KSA}}$  indicate the sources of the degrees of freedom for the forward computation, and  $\mathcal{R}^{\text{ENC}}$  and  $\mathcal{R}^{\text{KSA}}$  indicate the sources of the degrees of freedom for the backward computation. Since in the forward computation and backward computation, ( $\#S^{\text{ENC}}[\mathcal{B}^{\text{ENC}}], \#S^{\text{KSA}}[\mathcal{B}^{\text{KSA}}]$ ) and ( $\#S^{\text{ENC}}[\mathcal{R}^{\text{ENC}}], \#S^{\text{KSA}}[\mathcal{R}^{\text{KSA}}]$ ) are restricted to  $\mathbb{X}$  and  $\mathbb{Y}$  respectively, with  $|\mathbb{X}| = (2^c)^{d_1}$  and  $|\mathbb{Y}| = (2^c)^{d_2}$ , we call  $d_1$  the degrees of freedom for the forward computation (denoted by DoF<sup>+</sup>) and  $d_2$  the degrees of freedom for the backward computation (denoted by DoF<sup>-</sup>).

With this configuration, it is shown that the time complexity to find a full  $n$ -cell match between the two ending states is  $(2^c)^{n - \min\{d_1, d_2, m\}}$ . Therefore, for a valid MITM preimage attack, we must have DoF<sup>+</sup>  $\geq 1$ , DoF<sup>-</sup>  $\geq 1$  and DoM  $\geq 1$ . In the following section, we will show how to automatically determine  $\mathcal{B}^{\text{ENC}}, \mathcal{B}^{\text{KSA}}, \mathcal{R}^{\text{ENC}}, \mathcal{R}^{\text{KSA}}, \mathcal{C}$ , and  $\mathcal{D}$  with MILP such that the complexity  $(2^c)^{n - \min\{\text{DoF}^+, \text{DoF}^-, \text{DoM}\}}$  of the corresponding attack is minimized *when the starting states and ending states are given*. Note that the choices of the starting states and ending states are quite limited and thus can be enumerated automatically.

*Remark 4.* Our program enumerates all combinations of the locations of starting and ending points in encryption, and all combinations of the locations of starting points in the encryption and key-schedule algorithm. That is, for an  $N$ -round

targeted cipher, our program generates MILP-models for each of the possible combinations  $\{(\text{init}_r^E, \text{init}_r^K, \text{match}_r) \mid 0 \leq \text{init}_r^E < N, -1 \leq \text{init}_r^K < N, 0 \leq \text{match}_r < N, \text{init}_r^E \neq \text{match}_r\}$ , where  $\text{init}_r^E$  is the location of starting point in encryption,  $\text{init}_r^K$  is that in key-schedule, and  $\text{match}_r$  is the location of the matching point. To find the optimal attacks, the MILP solver solves them all.

*Note 1 (Tricks for matching the ending states as indirect matching and matching through MixColumns used in [4, 9, 18]).* Note that in the MITM preimage attack on AES-like hash functions, the last sub-key addition leading to  $\#E^-$  is close to the boundary of the forward and backward computation as illustrated in Fig. 15a. Therefore, to perform matching, one can decompose state as  $\#K = \#K^+ + \#K^-$ , and translate the computation in Fig. 15a into its equivalent form shown in Fig. 15b, since  $\text{MC}(\#E^+) \oplus \#K = \text{MC}(\#E^+ \oplus \text{MC}^{-1}(\#K^+)) \oplus \#K^-$ . Full explanation can be found in Supplementary Material C.

In the following description of our modeling method, for simplicity, we let the number of rows of the state  $N_{\text{row}}$  be 4, and thus, the branch number of the MixColumns  $B_n = N_{\text{row}} + 1$  be 5. However, the modeling method can be directly applied to other AES-like hashing that formalized in Sect. 2.1.

## 4 Programming the MITM Preimage Attacks with MILP

To facilitate the visualization of our analysis, each cell can take one of the four colors (**Gray**, **Red**, **Blue**, and **White**) according to certain rules, and a valid coloring scheme in our model corresponds to a MITM pseudo-preimage attack. The semantics of the colors of cells are listed as follows.

- **Gray (G)**: known constant in both forward and backward chunk.
- **Red (R)**: known and active in the backward chunk but unknown in the forward.
- **Blue (B)**: known and active in the forward chunk but unknown in the backward.
- **White (W)**: unknown in both the forward and backward chunk.

For the  $i$ th cell of a state  $\#S$ , we introduce two 0-1 variables  $x_i^{\#S}$  and  $y_i^{\#S}$  to encode its color, where  $(x_i^{\#S}, y_i^{\#S}) = (0, 0)$  represents W,  $(x_i^{\#S}, y_i^{\#S}) = (0, 1)$  represents R,  $(x_i^{\#S}, y_i^{\#S}) = (1, 0)$  represents B, and  $(x_i^{\#S}, y_i^{\#S}) = (1, 1)$  represents G. The encoding scheme is chosen such that  $x_i^{\#S} = 1$  if and only if  $\#S[i]$  is a known cell for the forward computation, and  $y_i^{\#S} = 1$  if and only if  $\#S[i]$  is a known cell for the backward computation. Under this encoding scheme, the number of **Blue** cells and **Gray** cells (known cells for the forward computation) in  $\#S$  can be computed as  $\sum_i x_i^{\#S}$ . Similarly, the number of **Red** cells and **Gray** cells (known cells in the backward computation) in  $\#S$  can be computed as  $\sum_i y_i^{\#S}$ . We also introduce an indicator 0-1 variable  $\beta_i^{\#S}$  for each cell such that  $\beta_i^{\#S} = 1$  if and only if the cell  $\#S[i]$  is **Gray**, which can be described by the following constraints

$$\begin{cases} x_i^{\#S} - \beta_i^{\#S} \geq 0 \\ y_i^{\#S} - \beta_i^{\#S} \geq 0 \\ x_i^{\#S} + y_i^{\#S} - 2\beta_i^{\#S} \leq 1 \end{cases} . \quad (2)$$

Under these constraints, the number of **Blue** cells in  $\#S$  can be computed as  $\sum_i x_i^{\#S} - \sum_i \beta_i^{\#S}$ , and the number of **Red** cells in  $\#S$  can be computed as  $\sum_i y_i^{\#S} - \sum_i \beta_i^{\#S}$ . Moreover, the **Blue** cells in the starting states are used to capture  $(\#S^{\text{ENC}}[\mathcal{B}^{\text{ENC}}], \#S^{\text{KSA}}[\mathcal{B}^{\text{KSA}}])$ , and the **Red** cells in the starting states are used to capture  $(\#S^{\text{ENC}}[\mathcal{R}^{\text{ENC}}], \#S^{\text{KSA}}[\mathcal{R}^{\text{KSA}}])$ .

**Constraints for the Starting States.** For the starting states, we introduce two additional variables  $\lambda^+$  and  $\lambda^-$  that compute the so-called *initial degrees of freedom*, where  $\lambda^+$  (the initial DoF for the forward computation) is defined as the number of **Blue** cells in  $\#S^{\text{ENC}}$  and  $\#S^{\text{KSA}}$ , and  $\lambda^-$  (the initial DoF for the backward computation) is defined as the number of **Red** cells in  $\#S^{\text{ENC}}$  and  $\#S^{\text{KSA}}$ . Putting the definitions into equations, we have

$$\begin{cases} \lambda^+ = \sum_i x_i^{\#S^{\text{ENC}}} - \sum_i \beta_i^{\#S^{\text{ENC}}} + \sum_i x_i^{\#S^{\text{KSA}}} - \sum_i \beta_i^{\#S^{\text{KSA}}} \\ \lambda^- = \sum_i y_i^{\#S^{\text{ENC}}} - \sum_i \beta_i^{\#S^{\text{ENC}}} + \sum_i y_i^{\#S^{\text{KSA}}} - \sum_i \beta_i^{\#S^{\text{KSA}}} \end{cases} \quad (3)$$

**Constraints for the Ending States.** To be concrete, we describe the constraints for matching through the `MixColumns` operation of AES.

*Property 1.* Let  $(\#E^-[4j], \#E^-[4j+1], \#E^-[4j+2], \#E^-[4j+3])^T$  and  $(\#E^+[4j], \#E^+[4j+1], \#E^+[4j+2], \#E^+[4j+3])^T$  be the  $j$ th columns of the ending states  $\#E^-$  and  $\#E^+$  that are linked by the `MixColumns` operation. When  $t$  ( $t \geq 5$ ) out of the 8 bytes of the two columns are active, there is a filter of  $t - 4$  bytes.

Since the time complexity of the attack is  $(2^c)^{n - \min\{\text{DoF}^+, \text{DoF}^-, \text{DoM}\}}$ , we must impose the constraint  $\text{DoM} \geq 1$  to ensure a valid attack. The known and active bytes of the  $j$ th column of the ending state  $E^+$  for the forward computation path from the starting states to  $E^+$  is the number of **Blue** cells, which can be computed in our model as  $\sum_{i=0}^3 (x_{4j+i}^{\#E^+} - \beta_{4j+i}^{\#E^+})$ . Similarly, the known bytes of the  $j$ th column of the ending state  $E^-$  for the backward computation path from the starting states to  $E^-$  is the number of **Red** cells, which can be computed as  $\sum_{i=0}^3 (y_{4j+i}^{\#E^-} - \beta_{4j+i}^{\#E^-})$ . Therefore, according to Property 1, we have the following constraints (suppose each state has four columns):

$$\begin{cases} \text{DoM} = \sum_{j=0}^3 \left( \sum_{i=0}^3 (x_{4j+i}^{\#E^+} - \beta_{4j+i}^{\#E^+}) + \sum_{i=0}^3 (y_{4j+i}^{\#E^-} - \beta_{4j+i}^{\#E^-}) - 4 \right) \\ \text{DoM} \geq 1 \end{cases} \quad (4)$$

**Constraints for the States in the Computation Paths.** This is an essential part of this work. In this part, we extend the construction of attacks on the basis of previous works. We refine and apply the critical idea behind the initial structure to a greater extent, and explicitly describe more possible ways to propagate the attributes (expressed in the four colors) of the cells that are involved in computation paths in both the encryption and the key-schedule. Therefore, we

would like to devote one separate whole section (Sect. 4.1) for the details of this part. Here we only give some high-level descriptions.

Let  $f$  be an operation that transforms a state  $\#S_{\text{IN}}$  into a state  $\#S_{\text{OUT}}$ . Then the coloring scheme of  $(\#S_{\text{IN}}, \#S_{\text{OUT}})$  must obey certain rules associated with  $f$  and the direction of the computation in which  $f$  is involved, such that the semantics of the colors are respected.

If we restrict the **Red** cells  $(\#S^{\text{ENC}}[\mathcal{R}^{\text{ENC}}], \#S^{\text{KSA}}[\mathcal{R}^{\text{KSA}}])$  in the starting states to some carefully constructed set  $\mathbb{Y}$  defined in Sect. 3, it may be valid to transform certain **Red** cells in  $\#S_{\text{IN}}$  to **Gray** cells (or even **Blue** cells) in  $\#S_{\text{OUT}}$  by some operations along the forward computation path (starting from the starting states to the ending state  $\#E^+$ ). By doing so, impacts from the **Red** cells on the forward computation are limited, meanwhile, the degrees of freedom of the **Red** cells in the starting states should be reduced from  $\lambda^-$ ; similar situations happen along the backward computation path (starting from the starting states to the ending state  $\#E^-$ ). In our MILP model, we must keep track of how much degrees of freedom are consumed to ensure the remaining degrees of freedom for the forward computation ( $\text{DoF}^+$ ) and for the backward computation ( $\text{DoF}^-$ ) always greater or equal to one. The variables and constraints introduced for the above purpose are detailed in Sect. 4.1.

**The Objective Function.** To minimize the time complexity of the attack,  $\min\{\text{DoF}^+, \text{DoF}^-, \text{DoM}\}$  should be maximized. To this end, we can introduce an auxiliary variable  $v_{\text{obj}}$ , impose the constraints

$$\begin{cases} v_{\text{obj}} \leq \text{DoF}^+ \\ v_{\text{obj}} \leq \text{DoF}^- \\ v_{\text{obj}} \leq \text{DoM} \end{cases}$$

and set the objective function to maximize  $v_{\text{obj}}$ .

In the multi-target setting, we suppose that the degree of freedom for the chunk to which the targets are added can be directly increased. Thus, for models where the starting point (resp. matching point) is at the upper round than the matching point (resp. starting point),  $\text{DoF}^-$  (resp.  $\text{DoF}^+$ ) can be directly increased, the objective is to maximize  $\min\{\text{DoF}^+, \text{DoM}\}$  (resp.  $\min\{\text{DoF}^-, \text{DoM}\}$ ).

#### 4.1 MILP Constraints for the States in the Computation Paths and the Consumption of Degrees of Freedom

Recalling the formalized framework of MITM attack in Sect. 3, before we perform the attack on a given target with predefined positions of starting states and ending states, we have to determine  $\mathcal{B}^{\text{ENC}}$ ,  $\mathcal{B}^{\text{KSA}}$ ,  $\mathcal{R}^{\text{ENC}}$ , and  $\mathcal{R}^{\text{KSA}}$  for the starting states  $\#S^{\text{ENC}}$  and  $\#S^{\text{KSA}}$ . In our visualizations of the attacks, the **Blue** cells in the starting states  $\#S^{\text{ENC}}$  and  $\#S^{\text{KSA}}$  are meant to capture  $\mathcal{B}^{\text{ENC}}$  and  $\mathcal{B}^{\text{KSA}}$  respectively. Similarly, the **Red** cells in the starting states are used to capture  $\mathcal{R}^{\text{ENC}}$  and  $\mathcal{R}^{\text{KSA}}$ , and the **Gray** cells in the starting states are used to capture  $\mathcal{G}^{\text{ENC}}$ , and  $\mathcal{G}^{\text{KSA}}$ .



Therefore, according to Eq. (3), the number of **Blue** cells and the number of **Red** cells in the starting states correspond to the initial degrees of freedom  $\lambda^+$  and  $\lambda^-$ , respectively. To control the impacts from neutral cells in one direction on the opposite direction, along the computation paths leading to the ending states, the initial degrees of freedom are consumed according to the coloring schemes.

Basically, forward computation consumes  $\lambda^-$ , and backward computation consumes  $\lambda^+$ . The consumption of degrees of freedom is counted in cells. Let  $\sigma^+$  and  $\sigma^-$  be the accumulated degrees of freedom that have been consumed in the backward and forward computation paths, respectively. We have

$$\begin{cases} \text{DoF}^+ = \lambda^+ - \sigma^+ \\ \text{DoF}^- = \lambda^- - \sigma^- \end{cases} \quad (5)$$

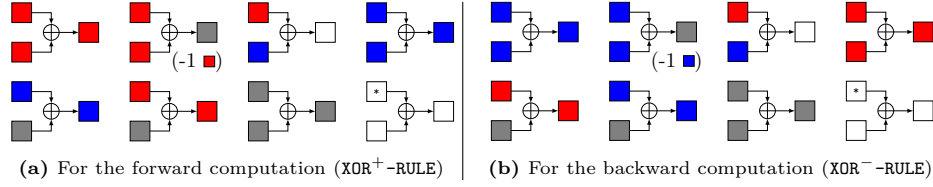
That is, the remaining DoF for the forward computation is computed as the initial DoF of the forward computation minus the DoF consumed by the backward computation (from the starting state to the ending state  $\#E^-$ ), and the remaining DoF of the backward computation is computed as the initial DoF of the backward computation minus the DoF consumed by the forward computation (from the starting state to the ending state  $\#E^+$ ). Since the complexity of the attack is  $(2^c)^{n-\min\{\text{DoF}^+, \text{DoF}^-, \text{DoM}\}}$ , we always require  $\text{DoF}^+ \geq 1$  and  $\text{DoF}^- \geq 1$ . Moreover,  $\sigma^+$  is computed as  $\sum \sigma^+(\#S_{\text{IN}} \rightarrow \#S_{\text{out}})$  along the computation path that consumes DoF for the forward computation, where  $\sigma^+(\#S_{\text{IN}} \rightarrow \#S_{\text{OUT}})$  is the DoF for the forward computation consumed by the transition from state  $\#S_{\text{IN}}$  to  $\#S_{\text{OUT}}$ , and  $\sigma^-$  is computed as  $\sum \sigma^-(\#S_{\text{IN}} \rightarrow \#S_{\text{OUT}})$  along the computation path that consumes the DoF for the backward computation. To show how to compute  $\sigma^+$  in our model, we will take the most complicated XOR-MC operation as an example. For other operations, one can obtain the constraints similarly.

According to the semantics of the colors, the rules for coloring the input and output states of an operation, and how they consume the degree of freedom to limit the impacts should be different for the forward and the backward computation paths. Therefore, for each type of operations, we will give two sets of rules for different directions of the computation.

First of all, an invertible S-box preserves the color of the input cell, and the ShiftRows permutes the coloring scheme of the input state according to the permutations associated with the ShiftRows in both forward and backward computations. Both S-box and ShiftRows operations can not be used to reduce the impacts via consuming the degree of freedom. In the sequel, we will focus on more nontrivial operations.

**XOR.** The XOR operations exist in the `AddRoundKey` and the key/message-schedule (if any). Here we need to distinguish two different directions. If the XOR to be modeled is involved in the forward computation path from the starting states to the ending state  $\#E^+$ , the coloring scheme of the input and output cells of the XOR operation obeys the set of rules (denoted by `XOR+-RULE`, where a “+” sign signifies the forward computation) shown in Fig. 3a. Similarly, if the





**Fig. 3:** Rules for XOR operations, where a “\*” means that the cell can be any color

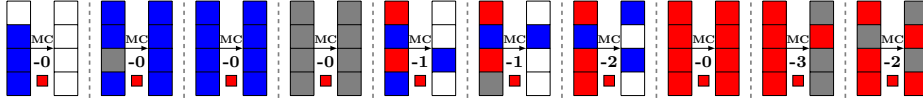
XOR to be modeled is involved in the backward computation path from the starting states to the ending state  $\#E^-$ , the coloring scheme of the input and output cells of the XOR operation obeys the set of rules named as XOR<sup>-</sup>-RULE, which is visualized in Fig. 3b. Note that XOR<sup>-</sup>-RULE (Fig. 3b) can be obtained from XOR<sup>+</sup>-RULE (Fig. 3a) by exchanging the Red cells and Blue cells, since the meanings of Red and Blue are dual for the forward and backward computations.

Let  $\#A[0]$ ,  $\#B[0]$  be the input cells and  $\#C[0]$  be the output cell. The set of rules XOR<sup>+</sup>-RULE restricts  $(x_0^{\#A}, y_0^{\#A}, x_0^{\#B}, y_0^{\#B}, x_0^{\#C}, y_0^{\#C})$  to a subset of  $\mathbb{F}_2^6$ , which can be described by a system of linear inequalities by using the convex hull computation method [45], and the set of rules XOR<sup>-</sup>-RULE can be described similarly.

Within each of the two sets of rules for XOR operations, only one coloring scheme consumes the degree of freedom, *e.g.*, the  $\blacksquare \oplus \blacksquare \rightarrow \blacksquare$  in Fig. 3a, which describes the possibility that the difference in one cell cancels that in another.

**MixColumns.** For the MixColumns operation in the forward computation, we have the following set of rules (denoted by MC<sup>+</sup>-RULE) for the coloring schemes of the input and output columns. Examples of valid coloring schemes are shown in Fig. 4.

- ▶ MC<sup>+</sup>-RULE-1. If there is at least one White cell in the input column, all the output cells are White (one unknown cell in the input causes all cells in the output be unknown);
- ▶ MC<sup>+</sup>-RULE-2. If there are Blue cells but no White cells and no Red cell in the input column, then all the output cells are Blue (can perform full forward computations);
- ▶ MC<sup>+</sup>-RULE-3. If all the input cells are Gray, then all the output cells are Gray (can perform bi-direction computations on fixed constants);
- ▶ MC<sup>+</sup>-RULE-4. If there are Red and Blue cells but no White cells in the input column, each output cell must be Blue or White. Moreover, a condition should be fulfilled, that is, the sum of the numbers of Blue and Gray cells in the input and output columns must be no more than 3 (*i.e.*, 8 – 5) (can partially cancel the impacts from  $\blacksquare$  on  $\blacksquare$  within an input column by consuming  $\lambda^-$ , and perform partial forward computations. Because of the MDS property of MixColumns, this is possible only when the condition is fulfilled);
- ▶ MC<sup>+</sup>-RULE-5. If there are Red cells but no White cells and no Blue cells in the input column, then each output cell must be Red or Gray. Moreover, a



**Fig. 4:** Some valid coloring schemes for the MixColumns in the forward computation

condition should be fulfilled, that is, the number of **Gray** cells in the input and output columns must be no more than 3 (*i.e.*,  $8 - 5$ ) (can partially cancel the difference within an input column by consuming  $\lambda^-$ . Because of the MDS property of **MixColumns**, this is possible only when the condition is fulfilled).

All the above rules can be described by linear inequalities.

First, we introduce three 0-1 indicator variables  $\mu, v, \omega$  for the input column and necessary constraints into the model to satisfy the following cases.

- ▶  $\mu = 1, v = 0, \omega = 0$  if and only if **MC<sup>+</sup>-RULE-1** is fulfilled;
- ▶  $\mu = 0, v = 1, \omega = 0$  if and only if **MC<sup>+</sup>-RULE-2** is fulfilled;
- ▶  $\mu = 0, v = 1, \omega = 1$  if and only if **MC<sup>+</sup>-RULE-3** is fulfilled;
- ▶  $\mu = 0, v = 0, \omega = 0$  if and only if **MC<sup>+</sup>-RULE-4** is fulfilled;
- ▶  $\mu = 0, v = 0, \omega = 1$  if and only if **MC<sup>+</sup>-RULE-5** is fulfilled.

This can be done as follows.

Let  $(\#A[0], \#A[1], \#A[2], \#A[3])^T$  and  $(\#B[0], \#B[1], \#B[2], \#B[3])^T$  be the input and output columns. Without any restriction, there are  $2^8$  possible coloring schemes for the input column since  $(x_0^{\#A}, y_0^{\#A}, \dots, x_3^{\#A}, y_3^{\#A}) \in \mathbb{F}_2^8$ . We define the set of vectors

$$\{(x_0^{\#A}, y_0^{\#A}, \dots, x_3^{\#A}, y_3^{\#A}, \mu) : (x_0^{\#A}, y_0^{\#A}, \dots, x_3^{\#A}, y_3^{\#A}) \in \mathbb{F}_2^8\}, \quad (6)$$

where  $\mu = 1$  if and only if there exists  $i \in \{0, 1, 2, 3\}$  such that  $(x_i^{\#A}, y_i^{\#A}) = (0, 0)$ . This subset can be described by linear inequalities with the convex hull computation method [45].

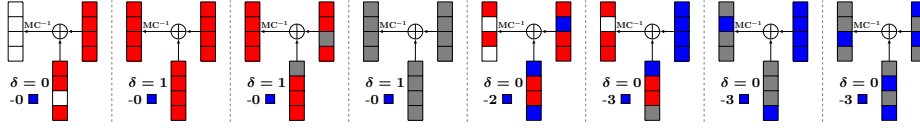
$v = 1$  if and only if  $x_i^{\#A} = 1$  for each  $i \in \{0, 1, 2, 3\}$ . This can be done by linear inequalities

$$\begin{cases} \sum_{i=0}^3 x_i^{\#A} - 4v \geq 0 \\ \sum_{i=0}^3 x_i^{\#A} - v \leq 3 \end{cases} . \quad (7)$$

$\omega = 1$  if and only if  $y_i^{\#A} = 1$  for each  $i \in \{0, 1, 2, 3\}$ . This can be done by similar inequalities as Eq. (7).

Now, with the help of these variables  $\mu, \epsilon, \omega$ , we can convert **MC<sup>+</sup>-RULE** into a system of inequalities:

$$\begin{cases} \sum_{i=0}^3 x_i^{\#B} + 4\mu \leq 4 \\ \sum_{i=0}^3 y_i^{\#B} + 4\mu \leq 4 \\ \sum_{i=0}^3 (x_i^{\#A} + x_i^{\#B}) - 5v \leq 3 \\ \sum_{i=0}^3 (x_i^{\#A} + x_i^{\#B}) - 8v \geq 0 \\ \sum_{i=0}^3 y_i^{\#B} - 4\omega = 0 \end{cases} . \quad (8)$$



**Fig. 5:** Some valid coloring schemes for the XOR-MC in the backward computation

Since the semantics of the **Red** cells and **Blue** cells are dual in the forward and backward computation, the set of rules for backward computation (denoted by  $MC^-$ -RULE) can be obtained from  $MC^+$ -RULE by exchanging the words **Blue** and **Red**. We omit the details to save spaces.

**XOR then MixColumns (XOR-MC).** For the operation which maps the two input columns  $(\#A[0], \#A[1], \#A[2], \#A[3])^T$  and  $(\#B[0], \#B[1], \#B[2], \#B[3])^T$  to  $\#C[0, 1, 2, 3] = MC^{-1}(\#A[0, 1, 2, 3] + \#B[0, 1, 2, 3])$ , we have the following rules for the coloring schemes of the input and output columns. Note that this operation only appears in the backward computation for all the targets in this paper. Therefore, we only specify the set of rules for XOR-MC for the backward computation.

- ▶ **XOR-MC-RULE-1.** If there is at least one **White** cell in the input columns, all the output cells are **White** (one unknown cell in the input causes all cells in the output be unknown);
- ▶ **XOR-MC-RULE-2.** If there are **Red** cells but no **White** cells and no **Blue** cells in the input columns, all output cells are **Red** (can perform full backward computations);
- ▶ **XOR-MC-RULE-3.** If all input cells are **Gray**, then all output cells are **Gray** (can perform bi-direction computations on fixed constants);
- ▶ **XOR-MC-RULE-4.** If there are **Blue** cells and **Red** cells but no **White** cells in the input columns, each output cell must be **Red** or **White**. Moreover, when combining the two input columns as a  $4 \times 2$  matrix, the number of rows with one or two **Blue** cells plus the number of **White** cells in the output column must be greater or equal to 5 (can partially cancel the impacts from  $\blacksquare$  on  $\blacksquare$  within two input columns by consuming  $\lambda^+$ , and perform partial backward computations. Because of the MDS property of inverse **MixColumns**, this is possible only when the condition is fulfilled);
- ▶ **XOR-MC-RULE-5.** If there are **Blue** cells but no **Red** cells and no **White** cells in the input columns, each output cell must be **Blue** or **Gray**. Moreover, when combining the two input columns as a  $4 \times 2$  matrix, the number of rows with one or two **Blue** cells plus the number of **Blue** cells in the output column must be greater or equal to 5 (can partially cancel the difference within two input columns by consuming  $\lambda^+$ . Because of the MDS property of **MixColumns**, this is possible only when the condition is fulfilled).

All the above rules can be described by similar linear inequalities for  $MC^-$ -RULE. Three 0-1 indicator variables  $\mu, v, \omega$  also be introduced for the input columns.

$\mu = 1$  if and only if there exists  $i \in \{0, 1, 2, 3\}$  such that  $(x_i^{\#A}, y_i^{\#A}) = (0, 0)$  or  $(x_i^{\#B}, y_i^{\#B}) = (0, 0)$ .  $v = 1$  if and only if  $x_i^{\#A} = 1$  and  $x_i^{\#B} = 1$  for each  $i \in \{0, 1, 2, 3\}$ .  $\omega = 1$  if and only if  $y_i^{\#A} = 1$  and  $y_i^{\#B} = 1$  for each  $i \in \{0, 1, 2, 3\}$ . These constraints can be generated from that of MC-RULE. For example, introduce  $\mu^{\#A}$  (resp  $\mu^{\#B}$ ) for input column  $(\#A[0], \#A[1], \#A[2], \#A[3])^T$  (resp  $(\#B[0], \#B[1], \#B[2], \#B[3])^T$ ) and necessary constraints as Eq. (6). Then  $\mu = 1$  if and only if  $\mu^{\#A} = 1$  or  $\mu^{\#B} = 1$ . Then

- ▶  $\mu = 1, v = 0, \omega = 0$  if and only if XOR-MC-RULE-1 is fulfilled;
- ▶  $\mu = 0, v = 0, \omega = 1$  if and only if XOR-MC-RULE-2 is fulfilled;
- ▶  $\mu = 0, v = 1, \omega = 1$  if and only if XOR-MC-RULE-3 is fulfilled;
- ▶  $\mu = 0, v = 0, \omega = 0$  if and only if XOR-MC-RULE-4 is fulfilled;
- ▶  $\mu = 0, v = 1, \omega = 0$  if and only if XOR-MC-RULE-5 is fulfilled.

Another four 0-1 variables  $\tau_0, \tau_1, \tau_2, \tau_3$  are introduced for each row,  $\tau_i = 1$  if and only if  $\#A[i]$  or  $\#B[i]$  is Blue cell.

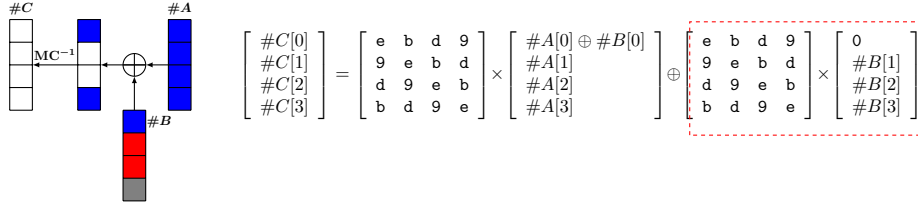
Now, with the help of these variables  $\mu, \epsilon, \omega, \tau_i$  for  $i \in \{0, 1, 2, 3\}$ , we can convert XOR-MC-RULE into a system of inequalities:

$$\begin{cases} \sum_{i=0}^3 x_i^{\#C} + 4\mu \leq 4 \\ \sum_{i=0}^3 y_i^{\#C} + 4\mu \leq 4 \\ \sum_{i=0}^3 (y_i^{\#C} - \tau_i) - 5\omega - \mu \leq -1 \\ \sum_{i=0}^3 (y_i^{\#C} - \tau_i) - 8\omega \geq -4 \\ \sum_{i=0}^3 x_i^{\#C} - 4v = 0 \end{cases} \quad (9)$$

*Remark 5.* One may attempt to model the XOR-MC operation by applying XOR<sup>-</sup>-RULE and MC<sup>-</sup>-RULE separately. This approach is valid but misses important coloring schemes that may lead to better attacks. For example, considering the input columns shown in Fig. 6, applying XOR<sup>-</sup>-RULE results in White cells after the XOR operation. Subsequently, applying MC<sup>-</sup>-RULE, we will end up with a full column of White cells. However, if we model the XOR-MC operation as a whole, we can still preserve some Red cells from impact according to the sixth sub-figure in Fig. 5. This coloring scheme can be explained by the equation shown in Fig. 6, where the second term of the right-hand side of the equation is known for the backward computation. Therefore, we can restrict the values of  $(\#B[0], \#A[0], \#A[1], \#A[2], \#A[3])$  such that

$$\begin{aligned} \mathbf{e} \cdot (\#A[0] \oplus \#B[0]) \oplus \mathbf{b} \cdot \#A[1] \oplus \mathbf{d} \cdot \#A[2] \oplus \mathbf{9} \cdot \#A[3] &= \mathbf{C}_0 \\ \mathbf{d} \cdot (\#A[0] \oplus \#B[0]) \oplus \mathbf{9} \cdot \#A[1] \oplus \mathbf{e} \cdot \#A[2] \oplus \mathbf{b} \cdot \#A[3] &= \mathbf{C}_2 \\ \mathbf{b} \cdot (\#A[0] \oplus \#B[0]) \oplus \mathbf{d} \cdot \#A[1] \oplus \mathbf{9} \cdot \#A[2] \oplus \mathbf{e} \cdot \#A[3] &= \mathbf{C}_3 \end{aligned} \quad (10)$$

where  $\mathbf{C}_0$ ,  $\mathbf{C}_2$ , and  $\mathbf{C}_3$  are constants, which implies that only  $\#C[1]$  is unknown for the backward computation (see the sixth sub-figure in Fig. 5). The principle is to let the differences of multiple cells in two input columns mutually canceled at particular output cells.



**Fig. 6:** The inaccuracy of modeling XOR-MC in the backward computation by applying XOR<sup>-</sup>-RULE and MC<sup>-</sup>-RULE separately.

*Compute consumed DoF in XOR-MC-RULES.* In all of our applications, the XOR-MC operation only appears in the backward computation and thus only consumes the DoF for the forward computation. Let  $(\#A[0], \dots, \#A[3])$  and  $(\#B[0], \dots, \#B[3])$  be the two input columns and  $(\#C[0], \dots, \#C[3])$  be the output column. Given a valid coloring scheme of  $\#A$ ,  $\#B$ , and  $\#C$ , the consumed DoF (measured in cells)

$$\sigma^+(\#A[0, \dots, 3], \#B[0, \dots, 3]) \rightarrow \#C[0, \dots, 3]$$

equals the number of Red and Gray cells (known cells of the output column in the backward computation) when there is at least one Blue cell in the input columns. Otherwise, the consumed DoF is zero.

Let  $\delta$  be a 0-1 indicator variable such that  $\delta = 1$  if and only if there are no Blue cells and no White cells in the input columns, which can be achieved by imposing the following constraints on  $\delta$ :

$$\begin{cases} -\delta + \sum_{i=0}^3 y_i^{\#A} + \sum_{i=0}^3 y_i^{\#B} \leq 7 \\ y_i^{\#A} \geq \delta, i \in \{0, 1, 2, 3\} \\ y_i^{\#B} \geq \delta, i \in \{0, 1, 2, 3\} \end{cases} \quad (11)$$

Then we have  $\sigma^+(\#A[0, \dots, 3], \#B[0, \dots, 3]) \rightarrow \#C[0, \dots, 3] = -4\delta + \sum_{i=0}^3 y_i^{\#C}$ . In Fig. 5 we give some example coloring schemes of the XOR-MC operation together with their consumed DoF. Similarly, the constraints describing how the XOR and MC operations consume DoF can be deduced.

## 5 Applications

Equipped with the presented tool, we evaluated the security of hash functions built on AES and AES-like ciphers, including all members of AES and the members of Rijndael with 256-bit block-size [13] in PGV-modes (note the equivalence among PGV-modes for the attacks as shown in [9]) and Haraka v2 [30].

For all targets, improved attacks are identified. In particular, our tool found the first preimage attacks on 8-round AES-128 hashing modes, and on the full 5-round and the extended 5.5-round (10 and 11 AES-rounds) Haraka-512 v2. Due to the page limit, we only describe two attacks in detail. The list of optimal



attacks we found is presented in Table 1. With the help of the visualizations of these attacks, one can reconstruct concrete attacks and confirm the complexities.

The time for finding each of the optimal attacks is within hours, including enumerating all possible combinations of the locations of starting and ending points in encryption, and all possible combinations of the locations of starting points in the encryption and key-schedule. For example, to get the presented attack on 8-round AES-128 hashing modes, our program generated all possible MILP-models and the MILP solver Gurobi solved them all, which took about two hours on a PC with an Intel Core i7-7500U CPU and 8 GB memory.

### 5.1 Improved Attacks on AES and Rijndael Hashing Modes

**Searching the attacks.** We apply our method to AES hashing modes. With our tool, many new attacks are found automatically. We list some examples for each member of AES and also the members of Rijndael with 256-bit block-size [13] (denoted by Rijndael-256) in Fig. 7, 8, 9, 10, and 11. Notably, apart from new attacks with better complexities, an 8-round attack on AES-128 and 9-round attacks on AES-192 and AES-256 hashing mode were found, which extend one more round compared with previous attacks [9, 38, 48].

To be clear, in the figures, some information are presented, such as which states are the starting states (in the searching for the attacks, not necessarily in the concrete attacks), how independent computation flows propagated in the states, and where the two chunks meet. Besides, which rules are applied to the states and how the degrees of freedom are consumed by the specific coloring scheme in our MILP models are also exhibited. Furthermore, the initial degrees of freedom ( $\lambda^+$ ,  $\lambda^-$ ), and the final configuration (DoF<sup>+</sup>, DoF<sup>-</sup>, DoM) which determines the attack complexity are summarized at the bottom.

For example, from Fig. 7, it can be seen that, in the searching of our model, the starting states are  $\#SB^4$  and  $\#k_4$ , and the ending states are  $\#MC^1$  and  $\#SB^2$ . Also, we have  $\mathcal{B}^{\text{ENC}} = [0, 5, 10, 15]$ ,  $\mathcal{B}^{\text{KSA}} = [0, 1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14]$ ,  $\mathcal{R}^{\text{ENC}} = [1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14]$ ,  $\mathcal{R}^{\text{KSA}} = \emptyset$ ,  $\mathcal{C} = [0, 2, 5, 7, 8, 10, 13, 15]$ , and  $\mathcal{D} = [1, 2, 3]$ . Accordingly, the initial degrees of freedom for the forward computation and backward computation are 17 and 12 respectively, and the degree of matching is  $2+3-4 = 1$ . The states  $\#SB^4$ ,  $\#k_3$ , and  $\#MC^3$  are enclosed by a dashed light-green frame , which means that XOR-MC-RULE is applied to them, and the specific coloring scheme consumes 12 cells of degrees of freedom for the forward computation. Similarly, the XOR-MC-RULE is applied to states  $\#SB^3$ ,  $\#k_2$ , and  $\#MC^2$ , and that consumes 3 cells of degrees of freedom for the forward computation. The states  $\#MC^4$  and  $\#AK^4$  are enclosed by a dashed light-purple frame , which means MC<sup>+</sup>-RULE is applied to them, and that consumes 9 cells of degrees of freedom for the backward computation. Similarly, the MC<sup>+</sup>-RULE is applied to states  $\#MC^5$  and  $\#AK^5$ , and that consumes 2 cells of degrees of freedom for the backward computation. Accordingly, in the solution of our model, DoF<sup>+</sup> = 17 - 12 - 3 = 2 and DoF<sup>-</sup> = 12 - 9 - 2 = 1, which indicates that the values of ( $\#SB^4[\mathcal{B}^{\text{ENC}}]$ ,  $\#k_4[\mathcal{B}^{\text{KSA}}]$ ) are restricted to a subset  $\mathbb{X}$  of  $\mathbb{F}_{2^8}^{17}$  with  $(2^8)^2$  elements, and the values of ( $\#SB^4[\mathcal{R}^{\text{ENC}}]$ ,  $\#k_4[\mathcal{R}^{\text{KSA}}]$ ) are restricted to a subset  $\mathbb{Y}$

of  $\mathbb{F}_{2^8}^{12}$  with  $2^8$  elements. To be more concrete,  $\mathbb{X}$  and  $\mathbb{Y}$  should be chosen such that the forward computation is irrelevant of  $(\#SB^4[\mathcal{R}^{\text{ENC}}], \#k_4[\mathcal{R}^{\text{KSA}}])$ , and the backward computation is irrelevant of  $(\#SB^4[\mathcal{B}^{\text{ENC}}], \#k_4[\mathcal{B}^{\text{KSA}}])$ . Since the degrees of freedom for the forward and backward computations ( $\text{DoF}^+$  and  $\text{DoF}^-$ ) are derived rather formally without giving the actual contents of  $\mathbb{X}$  and  $\mathbb{Y}$ , some readers may doubt whether such  $\mathbb{X}$  and  $\mathbb{Y}$  really exist. In the following (in the precomputation phase and more details in Supplementary Material B.1), we explicitly show in this example, how to obtain  $\mathbb{X}$  and  $\mathbb{Y}$  such that the required properties are fulfilled, and under the configuration obtained by the MILP model, how to launch the concrete attack.

### The attack on 8-round AES-128 hashing (refer to Fig. 7)

*The Precomputation Phase (precompute possible initial values of neutral bytes)*

1. To be able to compute backward chunk independently of forward neutral bytes, the forward neutral bytes should have constant impacts on the 12 **C**-marked **Red** bytes in  $\#MC^3$  and on the 3 **C**-marked **Red** bytes in  $\#MC^2$ . Therefore, denote the 12 constant impacts on 12 bytes in  $\#MC^3$  by  $C_{1,0}, C_{1,1}, C_{1,2}, C_{1,3}, C_{1,4}, C_{1,5}, C_{1,6}, C_{1,7}, C_{1,8}, C_{1,9}, C_{1,10}, C_{1,11}$ , we derive constraints on forward neutral bytes, which is a system linear equation Eq. (12). Similarly, denote the 3 constant impacts on 3 bytes in  $\#MC^2$  by  $C_{2,0}, C_{2,1}, C_{2,2}$ , we derive constraints on forward neutral bytes, which is a system of linear equation Eq. (13). In total, requiring impacts to be constant will impose 15 bytes constraints on forward neutral bytes (20 bytes) as shown in the system of linear equation Eq. (16). Solving Eq. (16), one gets  $2^{40}$  solutions (where  $40 = (20 - 15) \times 8$ ).

In the following main procedure, the values of  $C_{1,0}, C_{1,1}, \dots, C_{1,11}$ , and  $C_{2,0}, C_{2,1}, C_{2,2}$  are fixed such that we only need to solve Eq. (16) once. However, the main procedure will need to trail on many values of **Gray** bytes in  $k_4$  (*i.e.*,  $k_4[5, 10, 15]$ ) to find full match. So here, we precompute values of forward neutral bytes that correspond to each value of  $k_4[5, 10, 15]$ . That can be done as follows. For each of the  $2^{40}$  solution,  $k_3$  and  $\#SB^4[0, 5, 10, 15]$  are determined. Compute  $k_4$  using  $k_3$ , and store  $k_4$  and the values of  $\#SB^4[0, 5, 10, 15]$  in table  $T_1$  indexed by the values of 3 **Gray** bytes  $k_4[5, 10, 15]$ .

- Note that there are  $2^{24}$  entries in  $T_1$ , and the total size of  $T_1$  is about  $2^{40}$ . Under each index, there are about  $2^{16}$  elements. We can either use  $2^{16}$  or  $2^8$  of them. The total complexity of the full attack will be the same (because  $\text{DoF}^+$  and  $\text{DoM}$  are all one byte). Thus, we use  $2^8$ . Therefore, the complexity of this procedure is  $2^{32}$ , and the memory requirement is  $2^{32}$ .
2. To be able to compute forward chunk independently of backward neutral bytes, the backward neutral bytes should have constant impacts on the 2 **C**-marked **Blue** bytes in  $\#AK^5$ . Therefore, denote the 2 constant impacts on 2 bytes in  $\#AK^5$  by  $C_{4,0}$  and  $C_{4,1}$ , we derive constraints on backward neutral bytes, which is a linear equation system Eq. (17). For each possible  $C_{4,0}$  and  $C_{4,1}$ , when solve Eq. (17), one gets  $2^8$  solutions.

In the following main procedure, we need to trail on many values of  $(\mathbf{C}_{4,0}, \mathbf{C}_{4,1})$  to find a full match. So here, we precompute values of backward neutral bytes that correspond to each value of  $(\mathbf{C}_{4,0}, \mathbf{C}_{4,1})$ , store values of  $\#\text{MC}^5[1, 2, 3]$  fulfilling Eq. (17) in table  $T_2$  indexed by the values of  $(\mathbf{C}_{4,0}, \mathbf{C}_{4,1})$ .

- There are  $2^{16}$  entries in  $T_2$ , and the total size of  $T_2$  is  $2^{24}$ . Under each index, there are  $2^8$  elements.

*The Main Procedure.* During the following procedure, the values of  $\mathbf{C}_{1,0}, \mathbf{C}_{1,1}, \dots, \mathbf{C}_{1,11}$ , and  $\mathbf{C}_{2,0}, \mathbf{C}_{2,1}, \mathbf{C}_{2,2}$  are fixed.

1. For each of the  $2^x$  values of 9 Gray bytes in  $\#\text{AK}^4$ , for each index  $i$  of the  $2^{24}$  indexes of  $T_1$  (each  $i$  corresponds to each candidate value of the 3 Gray bytes in  $k_4$ ), for each index  $j$  of the  $2^{16}$  indexes of  $T_2$  (each  $j$  corresponds to each candidate value of the 2-byte impact on  $\mathbf{C}$ -marked cells by in  $\#\text{AK}^5$ ), do: Initialize an empty table  $L_1$ .
  - (a) For each of the  $2^8$  elements in  $T_1[i]$ , start from state  $\#\text{SB}^4$  and  $k_4$ , compute forward (cells in Blue) with the knowledge of the fixed impact  $j$  on  $\#\text{AK}^5$  to the matching point  $\#\text{MC}^1$ . Compute the one-byte value  $m_1$  for matching (defined in left-hand side of the equation in Fig. 1b), and use  $m_1$  as the index to store the values of  $(\#\text{SB}^4[0, 5, 10, 15], k_4)$  into  $L_1[m_1]$  (there is about  $2^{8-8} = 1$  element in each  $L_1[m_1]$ ).
  - (b) For each of the  $2^8$  elements in  $T_2[j]$ , start from state  $\#\text{MC}^5$ , compute backward (cells in Red) with the knowledge of fixed value  $i$  (i.e., 3 Gray bytes) in state  $k_4$  and the fixed impacts on  $\#\text{MC}^3$  and  $\#\text{MC}^2$  (i.e.,  $\mathbf{C}_{1,0}, \mathbf{C}_{1,1}, \mathbf{C}_{1,2}, \mathbf{C}_{1,3}, \mathbf{C}_{1,4}, \mathbf{C}_{1,5}, \mathbf{C}_{1,6}, \mathbf{C}_{1,7}, \mathbf{C}_{1,8}, \mathbf{C}_{1,9}, \mathbf{C}_{1,10}, \mathbf{C}_{1,11}, \mathbf{C}_{2,0}, \mathbf{C}_{2,1}, \mathbf{C}_{2,2}$ ) to the matching point  $\#\text{AK}^1$ . Compute the one-byte value  $m_2$  for matching (defined in right-hand side of the equation in Fig. 1b), and use it to lookup the list  $L_1$ :
    - i. For each of the elements in  $L_1[m_2]$  (expected to exist  $2^{8-8} = 1$ ): restart the forward and backward computations combining the knowledge of values in both directions (the values of  $\#\text{SB}^4[0, 5, 10, 15]$ ,  $k_4$ , and  $\#\text{MC}^5$ ) to the matching point  $(\#\text{MC}^1, \#\text{AK}^1)$ , test for full match on 128-bit state.

*Complexity.* The computational and memory complexity of the precomputation phase is about  $2^{32}$ . For the main procedure, in the inner loop, there will be  $2^{(8+8-8)} = 2^8$  solutions left after the one-byte (8-bit) matching ( $m_1$  and  $m_2$ ) in Step 1 (b) i. In order to find a 128-bit full match, one has to match the other 120 bits. Hence, for the outer loop, it requires  $x + 24 + 16 = 120 - 8$ , i.e.,  $x = 72$ . Therefore, the time complexity for the main procedure is about  $2^{(x+24+16)+8} = 2^{x+40} = 2^{120}$ .

We implemented the full attack on this 8-round AES-128-hashing (with partial matchings), which verified the complexity. The codes and results are available via [https://github.com/MITM-AES-like-Hashing/AES128\\_8R](https://github.com/MITM-AES-like-Hashing/AES128_8R).



Apart from the biclique attacks in [12]<sup>8</sup>, the best previous pseudo-preimage attacks against AES-128 hashing modes remain as 7 rounds since 2011, with a time complexity of  $2^{120}$  by Sasaki [38] and improved to  $2^{112}$  by Bao *et al.* in 2019 [9]. Our attack presented here penetrates one more round. There is a unique features observed from Fig. 7, which made the extra round possible. The backward chunk covers one more round compared with that in [9, 38]. This is only possible after the consumption of 12 and 3 Blue bytes of freedom degrees (forward neutral bytes) in consecutive two rounds. Without the introduction of DoF from key bytes in [9], this would not be possible. Note that the backward chunk only outputs 3 bytes, which are just sufficient to form a filter of one byte together with the 2 Blue bytes before the MixColumns at the matching point.

As depicted in Fig. 8, 9, 10, 11 and summarized in Table 1, when our search models are applied to hashing modes based on other AES variants, they are also able to improve by one round against AES-192 and AES-256 hashing as in [9]. Some configurations (*e.g.*, Fig. 8, 10) are more involved, in which the key states have neutral bytes for both forward and backward chunks. That might be hard to be found by manual.

## 5.2 Improved Attacks on Haraka v2

Haraka v2 [30] is a family of hash functions designed to be efficient for short-input and for post-quantum applications. It includes two versions, denoted by Haraka-256 v2 and Haraka-512 v2, both output 256-bit hash digests and claim 256-bit security against (second)-preimage attacks. They only process short-input ( $s$ -bit string, denoted by  $x$ ) and thus employ  $s$ -bit permutation (denoted by  $\pi_s$ ) in the DM-mode as follows:

$$\text{Haraka-256 v2}(x) = \pi_s(x) \oplus x \quad \text{and} \quad \text{Haraka-512 v2}(x) = \text{trunc}(\pi_s(x) \oplus x)$$

where `trunc` truncates 512-bit state to 256-bit output. To achieve high performance on platforms supporting AES-NI and share security analysis of AES, the round function of the permutation  $\pi_s$  first applies two layers of  $b$  AES-round-functions in parallel on a state that can be evenly divided into  $b$  sub-states (each of which is identical to the state of AES), then it applies a shuffle (denoted by `mixs`) among the columns of the state. For Haraka-256 v2,  $s = 256, b = 2$ , and for Haraka-512 v2,  $s = 512, b = 4$ . For both of them, the number of rounds is 5 that involves 10 AES-rounds in sequential.

The modified versions of Haraka v2 are used in instantiations of SPHINCS<sup>+</sup> (which replaces the DM-mode with Sponge-based construction) [23] and Gravity-SPHINCS (which extends one round on top of the 5-round version) [7]. Gravity-SPHINCS is one of the first round, and SPHINCS<sup>+</sup> is one of the third round alternate candidates of digital signatures in the NIST Post-Quantum Cryptography Standardization Process.

<sup>8</sup> Known for having broken full AES and AES hashing modes with slight advantage over brute force.

The former version of **Haraka** (named as **Haraka v1**) was broken by Jean [25] due to its weak round constants. Then an updated version **Haraka v2** [30] was published. The designers provide MITM preimage attacks on 3.5-round **Haraka-256 v2** and on 4-round **Haraka-512 v2**.

**Searching the attacks.** For both versions of **Haraka v2**, our tool produced improved MITM preimage attacks. In particular, for **Haraka-256 v2**, our tool found attacks that cover up to 4.5-round (9 AES-rounds). An example that has the optimal complexity is visualized in Fig. 12, of which the complexity is  $2^{256-8 \times \min\{\text{DoF}^+, \text{DoF}^-, \text{DoM}\}} = 2^{256-8 \times \min\{4, 4, 8\}} = 2^{224}$ . Note that this attack directly implies an attack cover 4-round (8 AES-rounds) with the same complexity. For **Haraka-512 v2**, our tool finds attacks that penetrate the full 5-round (10 AES-rounds) and the extended 5.5-round (11 AES-rounds) version. The detailed configuration of one of the attacks on the full 5-round (10 AES-rounds) is visualized in Fig. 13. In the following, we present one of the searching results on the extended 5.5-round (11 AES-rounds) and the concrete attack corresponding to the configuration visualized in Fig. 14

From Fig. 14, it can be seen that in the searching of our model, the starting state is  $\#\text{SB}^3$ , and the ending states are  $\#\text{MC}^{10}$  and  $\#\text{AC}^{10}$ . Also, we have  $\mathcal{B}^{\text{ENC}} = [16 \cdot i + j \mid i \in \{0, 1, 2, 3\}, j \in \{0, 1, 5, 6, 10, 11, 12, 15\}]$ ,  $\mathcal{R}^{\text{ENC}} = [16 \cdot i + j \mid i \in \{0, 1, 2, 3\}, j \in \{2, 3, 4, 7, 8, 9, 13, 14\}]$ ,  $\mathcal{C} = [16 \cdot i + j \mid i \in \{0, 3\}, j \in \{0, 7, 10, 13\}] \cup [16 \cdot i + j \mid i \in \{1, 2\}, j \in \{1, 4, 11, 14\}]$ , and  $\mathcal{D} = [16 \cdot i + j \mid i \in \{2\}, j \in \{0, 1, \dots, 7\}]$ . Therefore, both of the initial degrees of freedom for the forward computation and backward computation are 32, *i.e.*,  $\lambda^+ = \lambda^- = 32$ , and the degree of matching is  $\text{DoM} = (1 + 4 - 4) \times 2 = 2$ . The  $\text{MC}^-$ -RULE applied to states  $\#\text{MC}^2$  and  $\#\text{AC}^2$  consumes 16 cells of degrees of freedom for the forward computation. And the  $\text{MC}^+$ -RULE applied to states  $\#\text{MC}^6$  and  $\#\text{AC}^6$  consumes 16 cells of degrees of freedom for the backward computation. Accordingly, in the solution of our model,  $\text{DoF}^+ = 32 - 16 = 16$  and  $\text{DoF}^- = 32 - 16 = 16$ . This indicates that the values of  $\#\text{SB}^3[\mathcal{B}^{\text{ENC}}]$  are restricted to a subset  $\mathbb{X}$  of  $\mathbb{F}_{2^8}^{32}$  with  $2^{8 \times 16}$  elements, and the values of  $\#\text{SB}^3[\mathcal{R}^{\text{ENC}}]$  are restricted to a subset  $\mathbb{Y}$  of  $\mathbb{F}_{2^8}^{32}$  with  $2^{8 \times 16}$  elements. To be more concrete,  $\mathbb{X}$  and  $\mathbb{Y}$  should be chosen such that the forward computation is irrelevant of  $\#\text{SB}^3[\mathcal{R}^{\text{ENC}}]$  and the backward computation is irrelevant of  $\#\text{SB}^3[\mathcal{B}^{\text{ENC}}]$ . In summary, the decisive parameters for the obtained attack is  $(\text{DoF}^+, \text{DoF}^-, \text{DoM}) = (16, 16, 2)$ . From these parameters, one can directly obtain that the time complexity of the corresponding pseudo-preimage attack is  $(2^8)^{32 - \min\{\text{DoF}^+, \text{DoF}^-, \text{DoM}\}} = 2^{240}$ .

The concrete procedure of the preimage attack on the extended 5.5-round **Haraka-512 v2** is given in the following.

#### The concrete attack on 11-AES-round **Haraka-512 v2** (refer to Fig. 14)

1. For each of the  $2^x$  values of impacts (16-byte impacts on the **C**-marked Red cells in  $\#\text{MC}^2$  and 16-byte impacts on the **C**-marked Blue cells in  $\#\text{AC}^6$ ), do: Initialize two empty tables  $L_1$  and  $L_2$

- (a) With the knowledge of the value of 16-byte impacts on the **C**-marked **Red** cells in  $\#MC^2$ , we can collect  $2^{16 \times 8} = 2^{128}$  possible values of **Blue** bytes (neutral bytes for the forward) in  $\#AC^2$  by solving sets of linear equations column-by-column. For example, in the first column of  $\#MC^2$  and  $\#AC^2$ , the two **Blue** bytes and 1-byte impact (denoted by  $C_0$ ) on the **C**-marked cell have to meet:  $9 \cdot \#AC^2[0] \oplus e \cdot \#AC^2[1] = C_0$ . There are 16 sets of such linear equations, one set per column. For each column, we obtain  $2^8$  solutions. Hence, it is expected to get  $2^{128}$  solutions by solving 16 sets of linear equations with 32 variables in total. The number  $2^{128}$  is also the degrees of freedom for forward chunk.
- (b) For each of the  $2^{128}$  solutions for **Blue** bytes (neutral bytes for the forward) in  $\#AC^2$ , compute forward with the knowledge of the 16-byte impacts on the **C**-marked cells in  $\#AC^6$  to the matching point  $\#MC^{10}$ , extract the two-byte value for matching (denoted by  $m_1$ ), store the values of **Blue** bytes in  $\#AC^2$  in  $L_1[m_1]$ .
- (c) Similarly, collect  $2^{128}$  possible values for **Red** bytes (neutral bytes for the backward) in state  $\#MC^6$  and compute backward to the matching point  $\#AC^{10}$ , extract the two-byte value for matching (denoted by  $m_2$ ), store the values of **Red** bytes in  $\#MC^6$  in  $L_2[m_2]$ .
- (d) For entries with common index (*i.e.*, 16-bit partial match) between  $L_1$  and  $L_2$ , form pairs of values of **Blue** bytes in  $\#AC^2$  and **Red** bytes in  $\#MC^6$ ; for each pair, restart the forward and backward computations combining the knowledge of values in both direction, test for full match. on 256 bits.

*Complexity.* In Step 1 (d), it is expected to find  $2^{128+128-16} = 2^{240}$  matches on 16 bits. Among them, it is expected to left 1 solution that also match on the other 240 bits, that implies a full match on 256 bits. Hence, to find a full match, it is expected to need  $2^x$  outer loops where  $x = 0$ . The memory requirement is  $2 \cdot 2^{128}$  to store  $L_1$  and  $L_2$ . The time complexity of Step 1 (a) is no more than  $2^{128}$ . The same complexity also applies to Step 1 (b) and Step 1 (c). The time complexity of Step 1 (d) is approximately  $2^{16} \times 2^{2 \times 112} = 2^{240}$  ( $L_1$  and  $L_2$  contains  $2^{16}$  entries each; each entry is expected to contain  $2^{112}$  values. Under a common 16-bit index, there are  $2^{2 \times 112}$  pairs to check for full match.) Therefore, the total time complexity is  $2^{240}$ .

Note that our attacks on **Haraka v2** do not directly break the security of **SPHINCS<sup>+</sup>-Haraka** and **Gravity-SPHINCS-Haraka**. For **SPHINCS<sup>+</sup>-Haraka**, the security relies on a preimage resistance of 128-bit rather than 256-bit. For **Gravity-SPHINCS-Haraka**, the security relies on a collision resistance of 128-bit rather than preimage resistance, besides, the underlying **Haraka v2** variants have increased the AES-like rounds from 10 to 12, while our attacks cover at most 11 rounds.

## 6 Conclusions

In conclusion, we modeled the MITM preimage attack into the language of MILP, generalized the attack model, and obtained better results in terms of number of

attacked rounds against AES-like hashing including the 8-round AES-128, 9-round AES-192, 9-round AES-256, and 9-round Rijndael-256 hashing modes, 4.5-round Haraka-256 v2, the full version (5-round) and the extended version (5.5-round) of Haraka-512 v2.

## References

1. Z. Alliance. ZigBee 2007 specification. *Online: <http://www.zigbee.org/>*, 2007.
2. R. AlTawy and A. M. Youssef. Preimage Attacks on Reduced-Round Stribog. In D. Pointcheval and D. Vergnaud, editors, *AFRICACRYPT 14*, volume 8469 of *LNCS*, pages 109–125. Springer, Heidelberg, May 2014.
3. R. AlTawy and A. M. Youssef. Second Preimage Analysis of Whirlwind. In D. Lin, M. Yung, and J. Zhou, editors, *Inscrypt 2014*, volume 8957 of *LNCS*, pages 311–328. Springer, 2014.
4. K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang. Preimages for Step-Reduced SHA-2. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 578–597. Springer, Heidelberg, Dec. 2009.
5. K. Aoki and Y. Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 70–89. Springer, Heidelberg, Aug. 2009.
6. K. Aoki and Y. Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 103–119. Springer, Heidelberg, Aug. 2009.
7. J.-P. Aumasson and G. Endignoux. Gravity-SPHINCS. Technical report, National Institute of Standards and Technology, 2017. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions>.
8. J.-P. Aumasson, W. Meier, and F. Mendel. Preimage Attacks on 3-Pass HAVAL and Step-Reduced MD5. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 120–135. Springer, Heidelberg, Aug. 2009.
9. Z. Bao, L. Ding, J. Guo, H. Wang, and W. Zhang. Improved Meet-in-the-Middle Preimage Attacks against AES Hashing Modes. *IACR Transactions on Symmetric Cryptology*, 2019(4):318–347, Jan. 2020.
10. C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 123–153. Springer, Heidelberg, Aug. 2016.
11. R. Benadjila, O. Billet, H. Gilbert, G. Macario-Rat, T. Peyrin, M. Robshaw, and Y. Seurin. SHA-3 proposal: ECHO. *Submission to NIST (updated)*, page 113, 2009.
12. A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique Cryptanalysis of the Full AES. In D. H. Lee and X. Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 344–371. Springer, Heidelberg, Dec. 2011.
13. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
14. P. Derbez and P.-A. Fouque. Automatic Search of Meet-in-the-Middle and Impossible Differential Attacks. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 157–184. Springer, Heidelberg, Aug. 2016.
15. X. Dong, S. Sun, D. Shi, F. Gao, X. Wang, and L. Hu. Quantum Collision Attacks on AES-like Hashing with Low Quantum Random Access Memories. *IACR Cryptol. ePrint Arch.*, 2020:1030, 2020.

16. T. Espitau, P.-A. Fouque, and P. Karpman. Higher-Order Differential Meet-in-the-middle Preimage Attacks on SHA-1 and BLAKE. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 683–701. Springer, Heidelberg, Aug. 2015.
17. P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schl affer, and S. S. Thomsen. Gr ostl – a SHA-3 candidate. <http://www.groestl.info/Groestl.pdf>, March 2011.
18. J. Guo, S. Ling, C. Rechberger, and H. Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 56–75. Springer, Heidelberg, Dec. 2010.
19. J. Guo, C. Su, and W. Yap. An Improved Preimage Attack against HAVAL-3. *Inf. Process. Lett.*, 115(2):386–393, 2015.
20. Y. Hao, G. Leander, W. Meier, Y. Todo, and Q. Wang. Modeling for Three-Subset Division Property Without Unknown Subset - Improved Cube Attacks Against Trivium and Grain-128AEAD. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020*, volume 12105 of *LNCS*, pages 466–495. Springer, 2020.
21. D. Hong, B. Koo, and Y. Sasaki. Improved Preimage Attack for 68-Step HAS-160. In D. Lee and S. Hong, editors, *ICISC 09*, volume 5984 of *LNCS*, pages 332–348. Springer, Heidelberg, Dec. 2010.
22. A. Hosoyamada and Y. Sasaki. Finding Hash Collisions with Quantum Computers by Using Differential Trails with Smaller Probability than Birthday Bound. In *EUROCRYPT 2020*, pages 249–279, 2020.
23. A. Hulsing, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, P. Kampanakis, S. Kolbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, and J.-P. Aumasson. SPHINCS+. Technical report, National Institute of Standards and Technology, 2019. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.
24. ISO/IEC. 10118-2:2010 Information technology — Security techniques – Hash-functions – Part 2: Hash-functions using an  $n$ -bit block cipher. 3rd ed., International Organization for Standardization, Geneva, Switzerland, October, 2010.
25. J. Jean. Cryptanalysis of Haraka. *IACR Trans. Symmetric Cryptol.*, 2016(1):1–12, 2016.
26. D. Khovratovich, G. Leurent, and C. Rechberger. Narrow-Bicliques: Cryptanalysis of Full IDEA. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 392–410. Springer, Heidelberg, Apr. 2012.
27. D. Khovratovich, C. Rechberger, and A. Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In A. Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 244–263. Springer, Heidelberg, Mar. 2012.
28. S. Knellwolf and D. Khovratovich. New Preimage Attacks against Reduced SHA-1. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 367–383. Springer, Heidelberg, Aug. 2012.
29. L. R. Knudsen, C. Rechberger, and S. S. Thomsen. The Grindahl Hash Functions. In A. Biryukov, editor, *FSE 2007*, volume 4593 of *LNCS*, pages 39–57. Springer, Heidelberg, Mar. 2007.
30. S. K obl, M. M. Lauridsen, F. Mendel, and C. Rechberger. Haraka v2 - Efficient Short-Input Hashing for Post-Quantum Applications. *IACR Trans. Symm. Cryptol.*, 2016(2):1–29, 2016. <http://tosc.iacr.org/index.php/ToSC/article/view/563>.
31. S. K obl, G. Leander, and T. Tiessen. Observations on the SIMON Block Cipher Family. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 161–185. Springer, Heidelberg, Aug. 2015.

32. G. Leurent. MD4 is Not One-Way. In K. Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 412–428. Springer, Heidelberg, Feb. 2008.
33. J. Li, T. Isobe, and K. Shibutani. Converting Meet-In-The-Middle Preimage Attack into Pseudo Collision Attack: Application to SHA-2. In A. Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 264–286. Springer, Heidelberg, Mar. 2012.
34. M. Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In A. D. Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 366–375. Springer, Heidelberg, May 1995.
35. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997.
36. N. Mouha, Q. Wang, D. Gu, and B. Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In C. Wu, M. Yung, and D. Lin, editors, *Inscrypt 2011*, volume 7537 of *LNCS*, pages 57–76. Springer, 2011.
37. B. Preneel, R. Govaerts, and J. Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 368–378. Springer, Heidelberg, Aug. 1994.
38. Y. Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In A. Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 378–396. Springer, Heidelberg, Feb. 2011.
39. Y. Sasaki. Integer Linear Programming for Three-Subset Meet-in-the-Middle Attacks: Application to GIFT. In A. Inomata and K. Yasuda, editors, *IWSEC 18*, volume 11049 of *LNCS*, pages 227–243. Springer, Heidelberg, Sept. 2018.
40. Y. Sasaki and K. Aoki. Preimage Attacks on 3, 4, and 5-Pass HAVAL. In J. Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 253–271. Springer, Heidelberg, Dec. 2008.
41. Y. Sasaki and K. Aoki. Preimage Attacks on Step-Reduced MD5. In Y. Mu, W. Susilo, and J. Seberry, editors, *ACISP 08*, volume 5107 of *LNCS*, pages 282–296. Springer, Heidelberg, July 2008.
42. Y. Sasaki and K. Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 134–152. Springer, Heidelberg, Apr. 2009.
43. D. Shi, S. Sun, P. Derbez, Y. Todo, B. Sun, and L. Hu. Programming the Demirci-Selçuk Meet-in-the-Middle Attack with Constraints. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 3–34. Springer, Heidelberg, Dec. 2018.
44. L. Song, J. Guo, D. Shi, and S. Ling. New MILP Modeling: Improved Conditional Cube Attacks on Keccak-Based Constructions. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 65–95. Springer, Heidelberg, Dec. 2018.
45. S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song. Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, Heidelberg, Dec. 2014.
46. L. Wang and Y. Sasaki. Finding Preimages of Tiger Up to 23 Steps. In S. Hong and T. Iwata, editors, *FSE 2010*, volume 6147 of *LNCS*, pages 116–133. Springer, Heidelberg, Feb. 2010.
47. L. Wang, Y. Sasaki, W. Komatsubara, K. Ohta, and K. Sakiyama. (Second) Preimage Attacks on Step-Reduced RIPEMD/RIPEMD-128 with a New Local-

- Collision Approach. In A. Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 197–212. Springer, Heidelberg, Feb. 2011.
48. S. Wu, D. Feng, W. Wu, J. Guo, L. Dong, and J. Zou. (Pseudo) Preimage Attack on Round-Reduced Grøstl Hash Function and Others. In A. Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 127–145. Springer, Heidelberg, Mar. 2012.
  49. Z. Xiang, W. Zhang, Z. Bao, and D. Lin. Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 648–678. Springer, Heidelberg, Dec. 2016.
  50. Y. Zheng, J. Pieprzyk, and J. Seberry. HAVAL - A One-Way Hashing Algorithm with Variable Length of Output. In J. Seberry and Y. Zheng, editors, *AUSCRYPT'92*, volume 718 of *LNCS*, pages 83–104. Springer, Heidelberg, Dec. 1993.

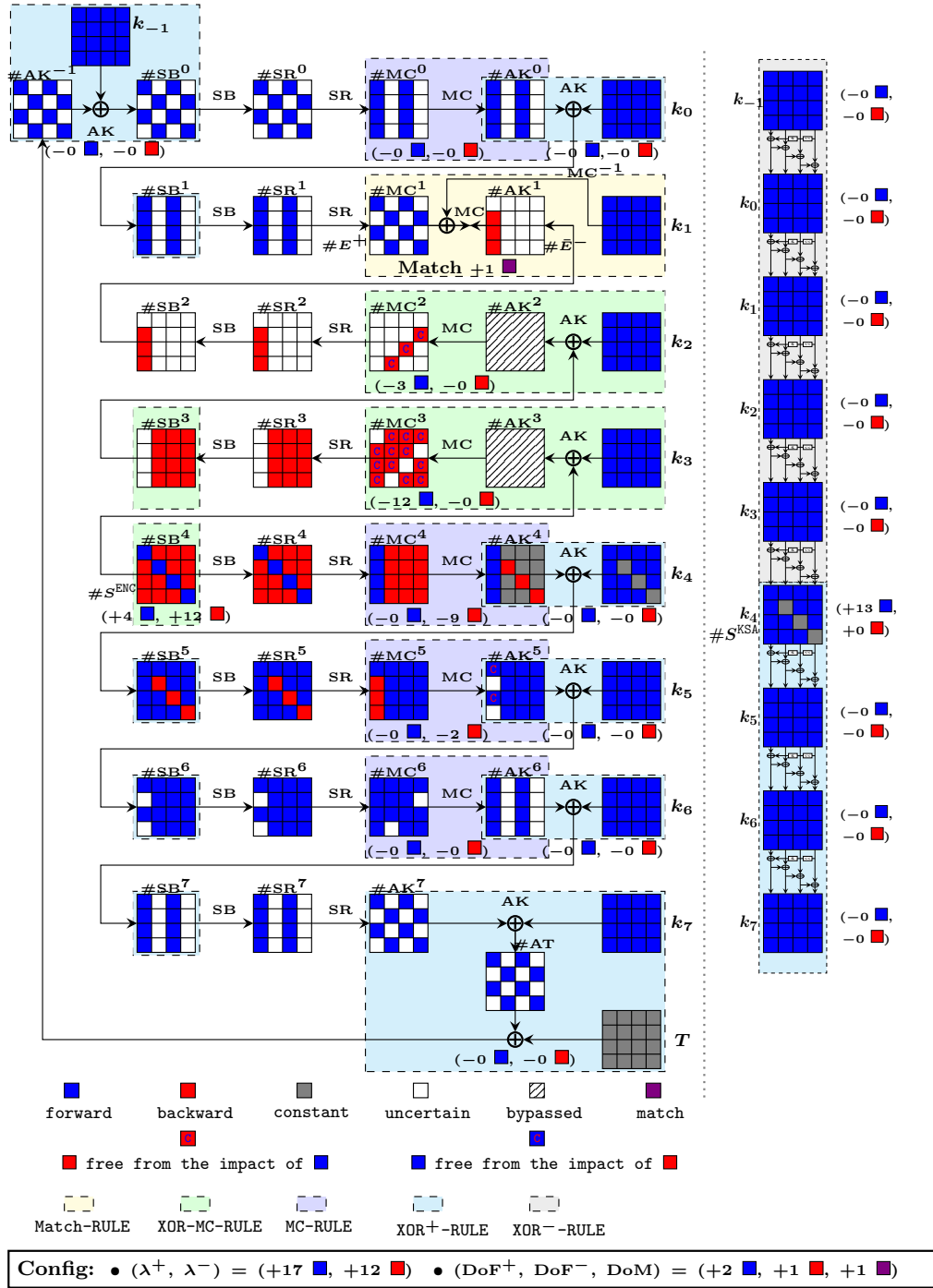
# Supplementary Material

## A Visualization of Attacks

A.1 MITM Preimage Attacks on **AES** and **Rijndael** Hashing Modes

A.2 MITM Preimage Attacks on **Haraka v2**





**Fig. 7:** An MITM pseudo-preimage attack on 8-round AES-128 hashing. Note that, because the use of XOR-MC-RULE, we do not introduce any variable in our MILP model for states  $\#AK^2$  and  $\#AK^3$ , and thus we bypass them.

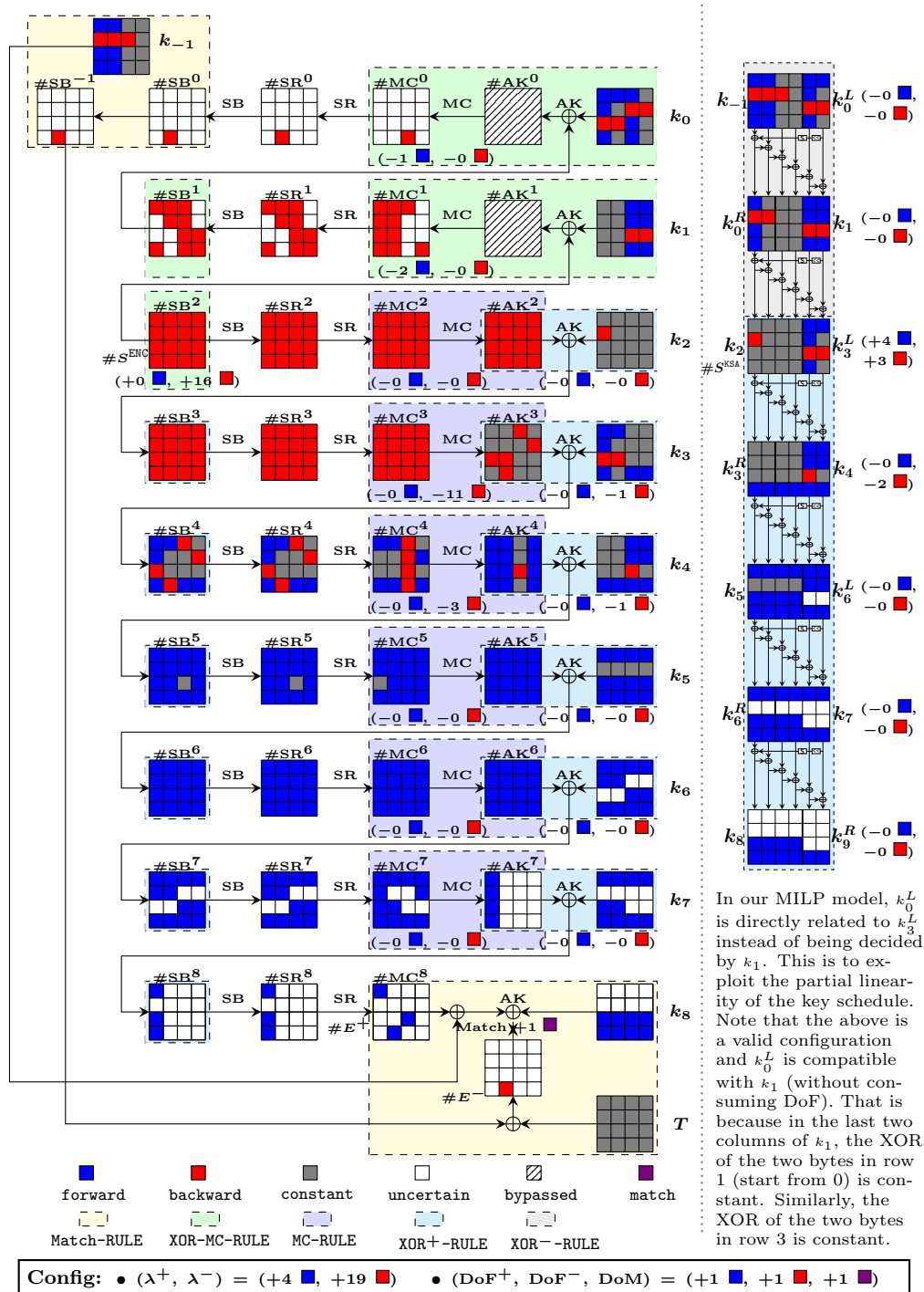


Fig. 8: An MITM pseudo-preimage attack on 9-round AES-192 hashing mode

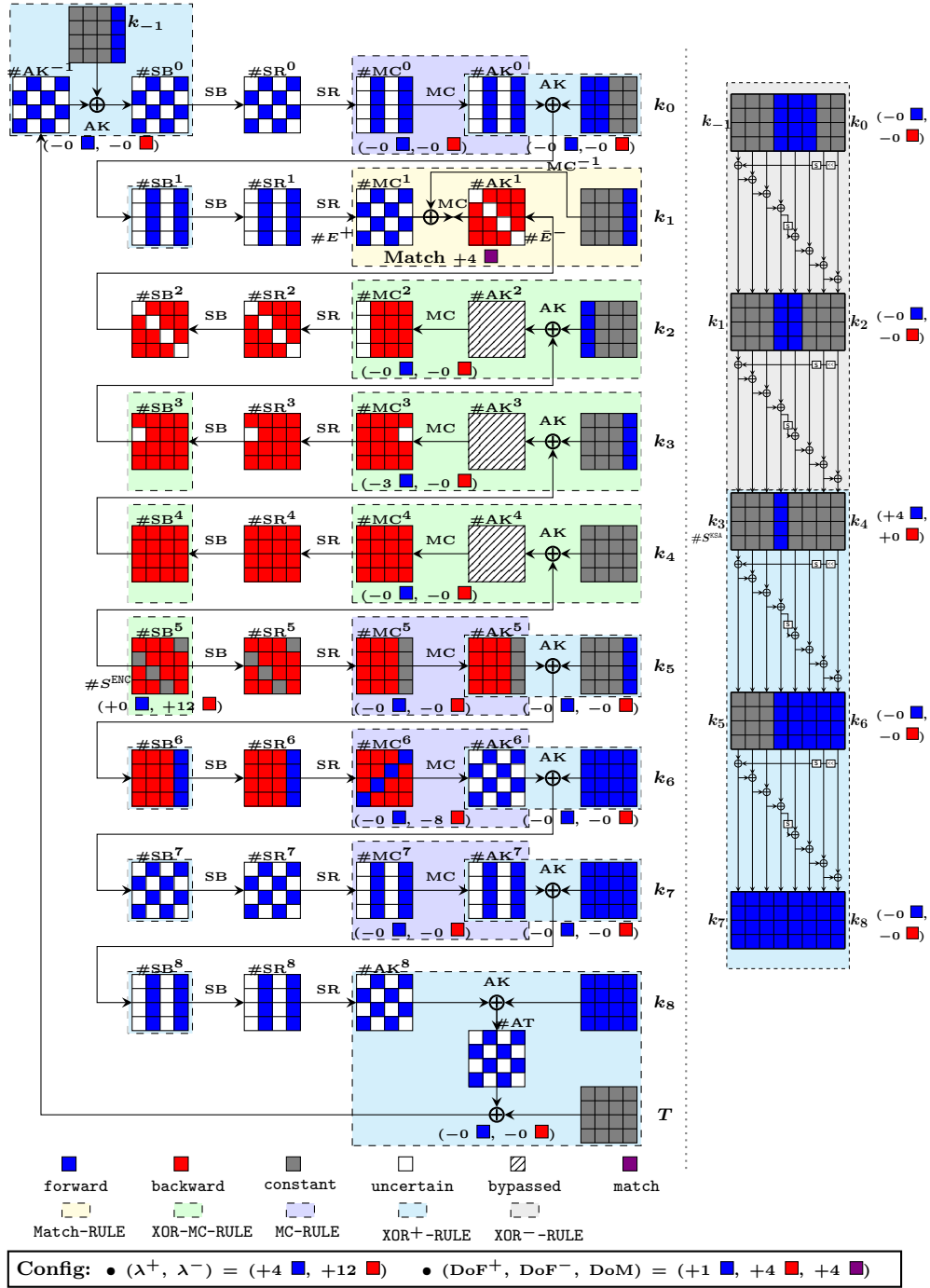


Fig. 9: Example I of the 9-round preimage attack on AES-256 hashing mode

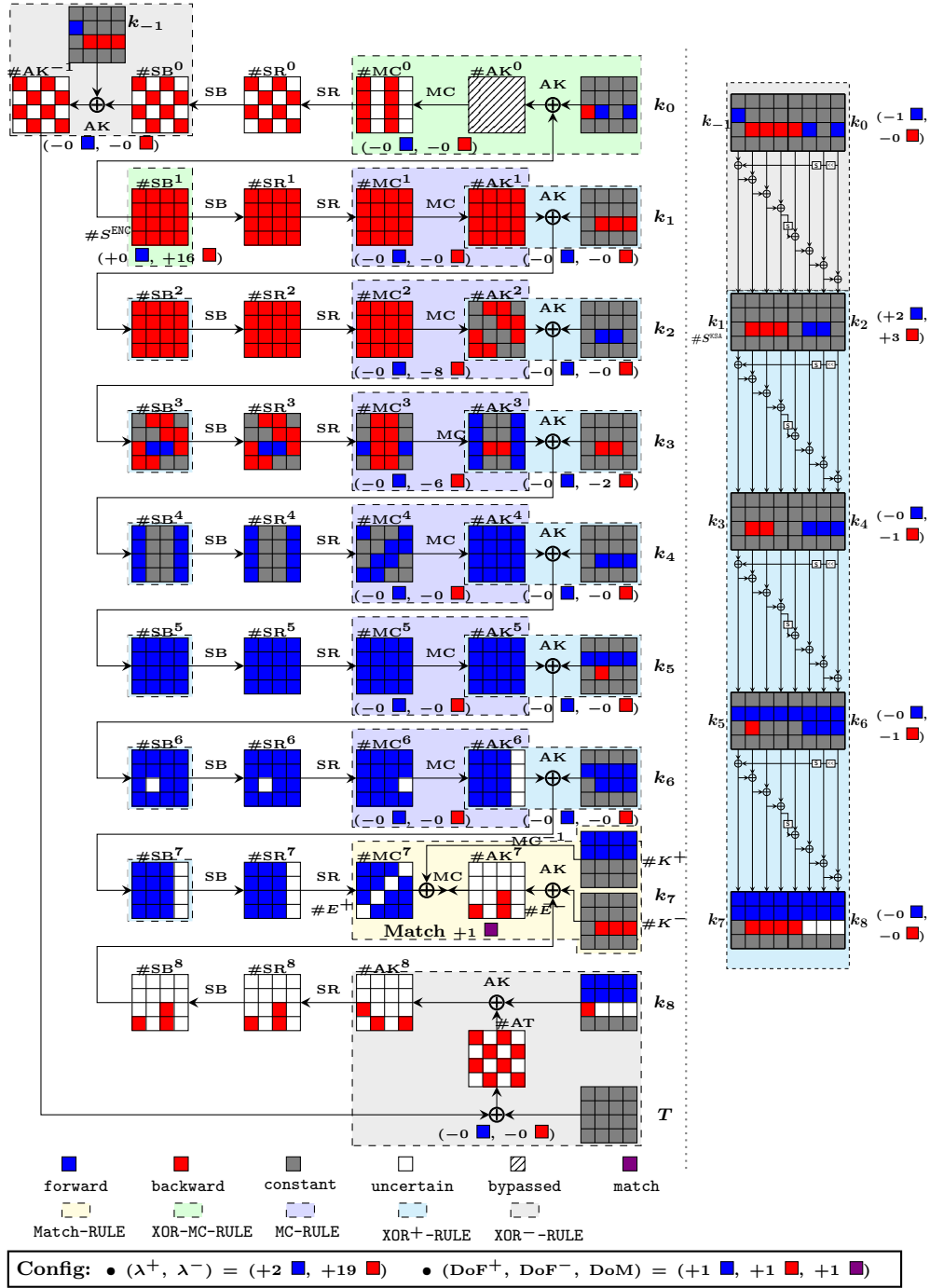


Fig. 10: Example II of the 9-round preimage attack on AES-256 hashing mode

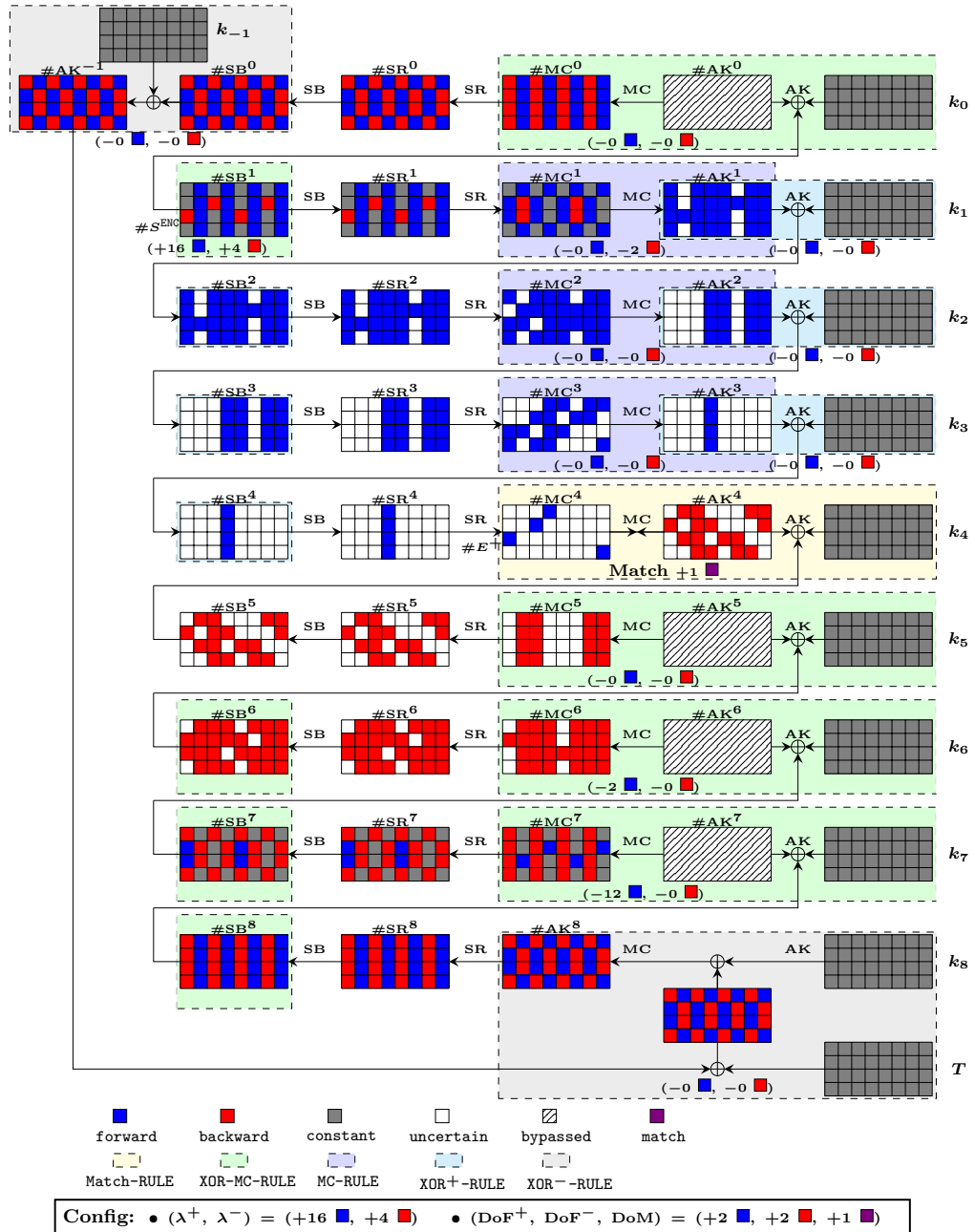


Fig. 11: Example of the 9-round preimage attack on Rijndael-256-128/192/256 hashing mode

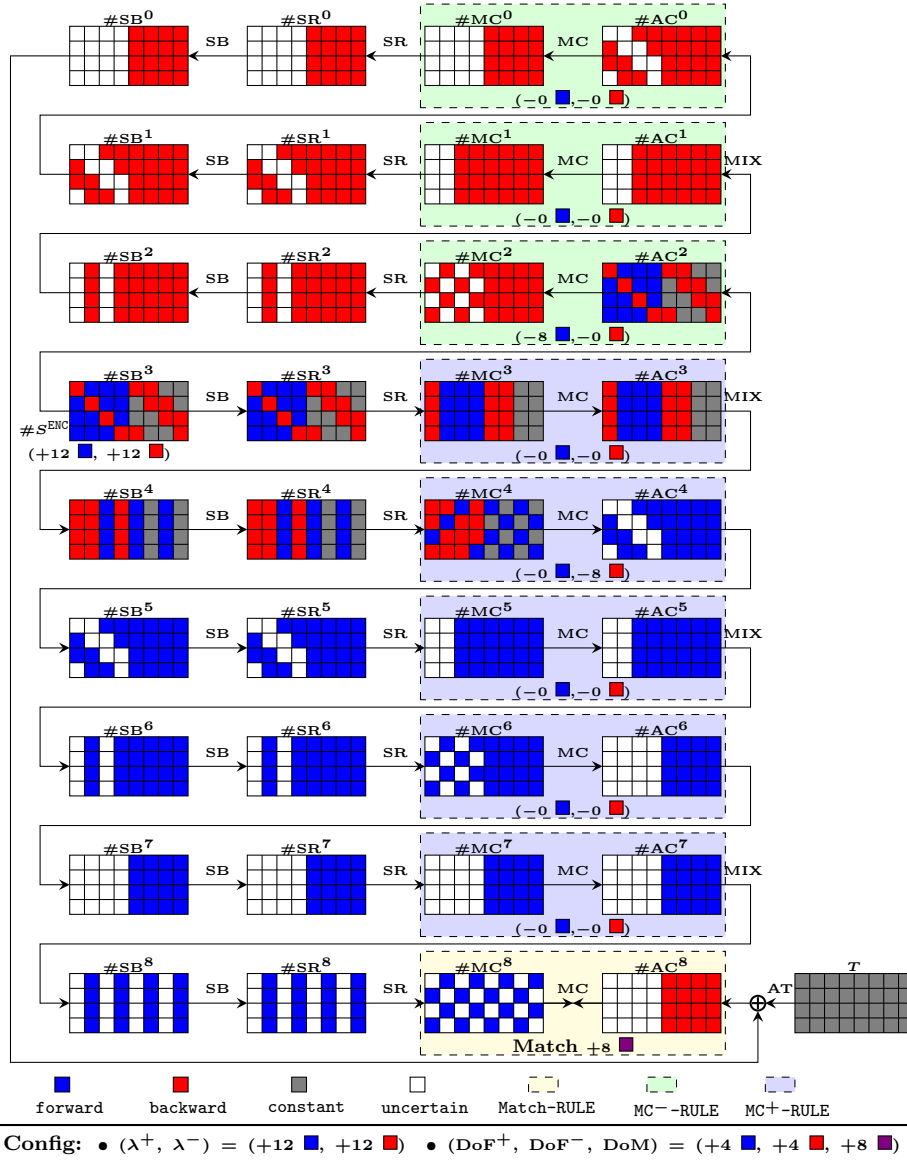
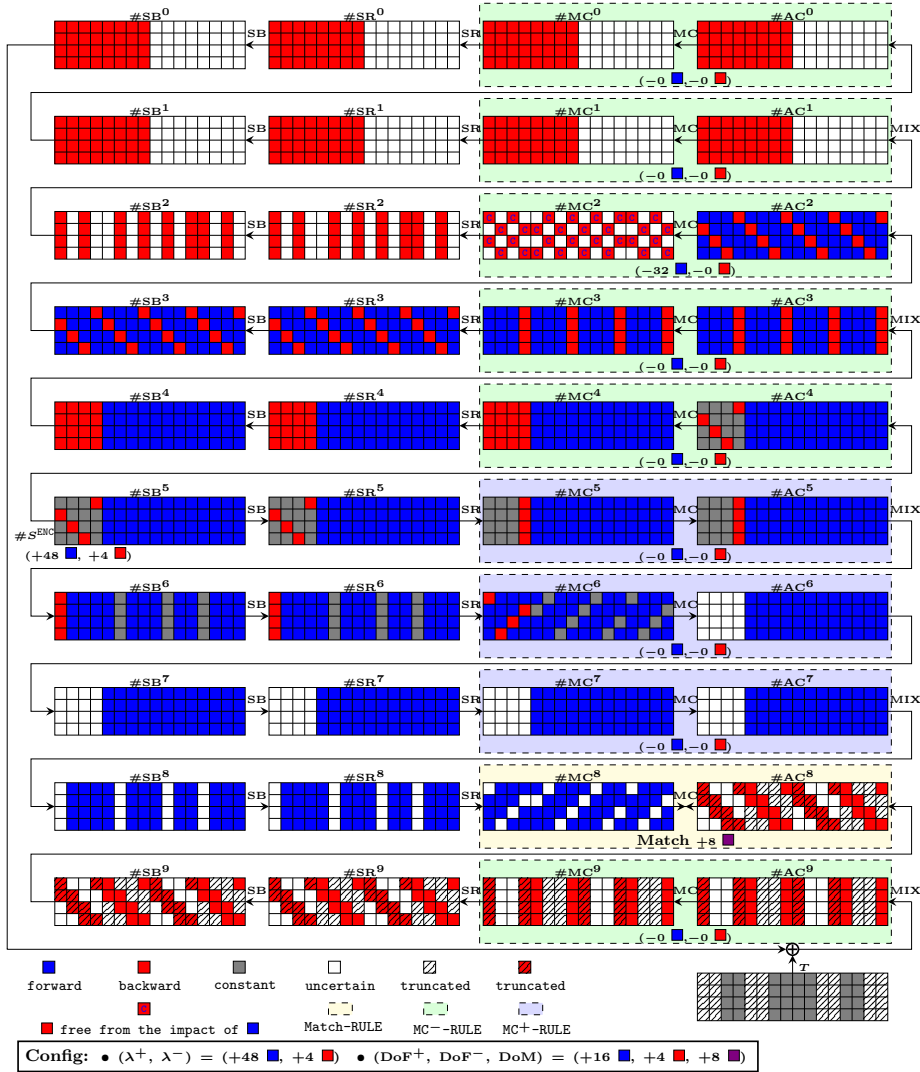
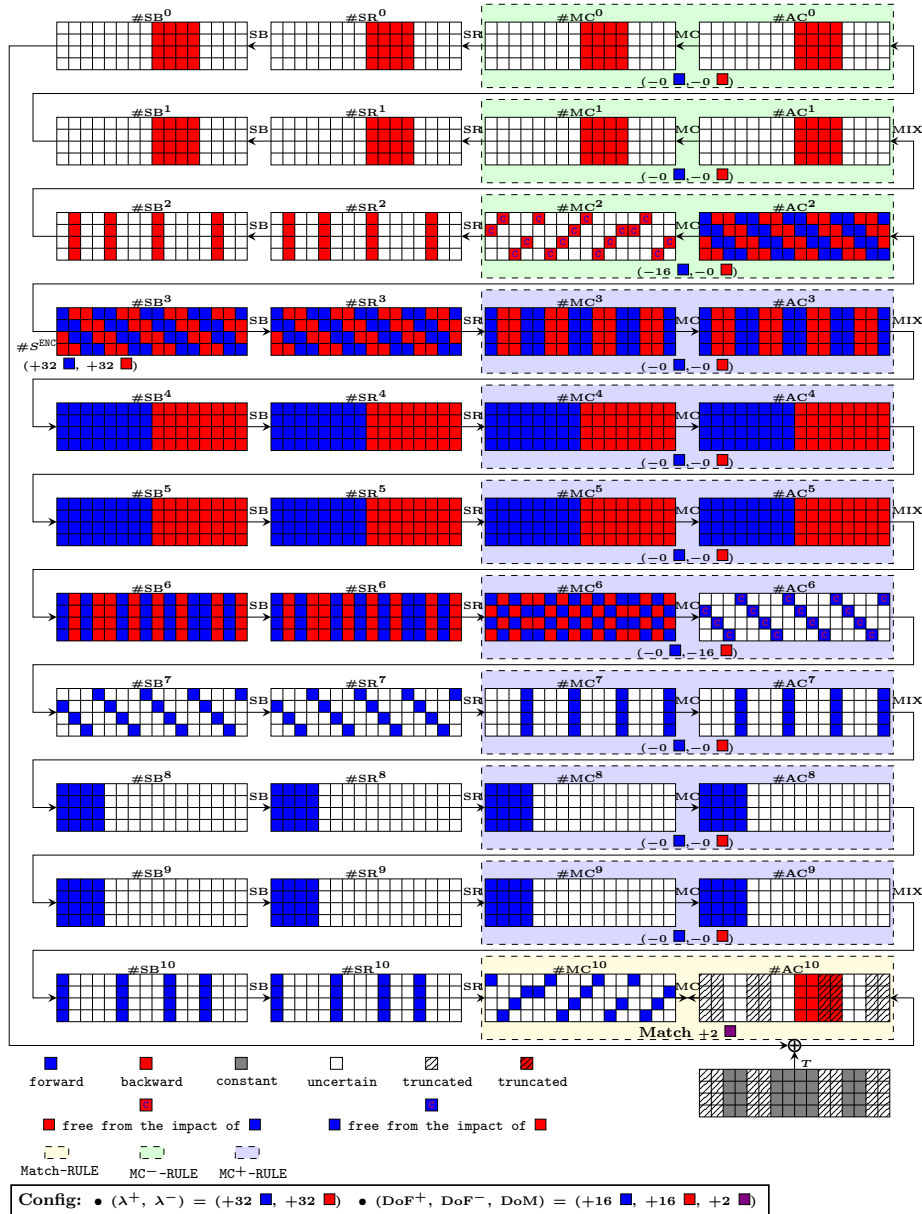


Fig. 12: Meet-in-the-middle attack on 4.5-round (or 9-AES-round) Haraka-256 v2 (matching at the last round).



**Fig. 13:** An MITM preimage attack on full Haraka-512 v2. Note that in our MILP-models, the position of the used hash bits are treated and used as constant in gray cell of the target  $T$ , and the bits discarded are treated as ‘uncertain’ although we distinct them using hatched pattern. However, in the attack procedure, the discarded bits are free of choice such that the state cells in hatched pattern are free of matching.



**Fig. 14:** An MITM preimage attack on the extended 5.5-round (11 AES-rounds) Haraka-512 v2. Note that in our MILP-models, the position of the used hash bits are treated and used as constant in gray cell of the target  $T$ , and the bits discarded are treated as ‘uncertain’ although we distinct them using hatched pattern. However, in the attack procedure, the discarded bits are free of choice such that the state cells in hatched pattern are free of matching.



## B Details in the Attack on 8-round AES-128 Hashing

*Remark.* The generations of initial values of neutral bytes for the forward (Blue) and the backward (Red) are two separate procedures. Between Red and Blue bytes, there is no dependency. To generate initial values of neutral bytes, there are two sets of equations – one is on the fixed impacts and the Blue bytes, the other is on other fixed impacts and the Red bytes.

Note that by ‘fixed impacts’, we mean the constant impacts from one direction on the other. E.g., the forward neutral bytes (Blue) have constant impacts on the backward active bytes (Red). The values of impacts are not the values of ‘C’-marked cells, but a common value that will be XORed to the ‘C’-marked cells in the MITM procedure. For the MITM procedure, the impacts on the ‘C’-marked cells are fixed before computations of both forward and backward chunks. Knowing the fixed impacts, the forward computations and the backward computations are also done independently before matching through MixColumns (the fixed impacts are directly XORed to the corresponding states).

In Sect. B.1, we explain how to generate the initial values of forward neutral bytes and backward neutral bytes by establishing equations and solving in precomputation procedures. Note that all precomputations are done once, and for all.

### B.1 Solving Equations to Obtain Values of Neutral Bytes in the 8-Round Attack on AES-128 Hashing Mode

**Obtain the values of the neutral bytes for forward chunk.** In the 8-round attack on AES-128 hashing mode (see Fig. 7), the values of the neutral bytes for the forward computation should be the solutions of the following equations, where  $C_{1,0}, C_{1,1}, \dots, C_{1,11}, C_{2,0}, C_{2,1}, C_{2,2}$ , and  $C_{3,0}, C_{3,1}, C_{3,2}$  are pre-determined constants:

$$\begin{bmatrix} \mathbf{e} & \mathbf{b} & \mathbf{d} & \mathbf{9} \\ \mathbf{9} & \mathbf{e} & \mathbf{b} & \mathbf{d} \\ \mathbf{d} & \mathbf{9} & \mathbf{e} & \mathbf{b} \\ \mathbf{b} & \mathbf{d} & \mathbf{9} & \mathbf{e} \end{bmatrix} \times \begin{bmatrix} k_3[0] \oplus \#SB^4[0] & k_3[4] & k_3[8] & k_3[12] \\ k_3[1] & k_3[5] \oplus \#SB^4[5] & k_3[9] & k_3[13] \\ k_3[2] & k_3[6] & k_3[10] \oplus \#SB^4[10] & k_3[14] \\ k_3[3] & k_3[7] & k_3[11] & k_3[15] \oplus \#SB^4[15] \end{bmatrix} = \begin{bmatrix} - & C_{1,3} & C_{1,6} & C_{1,9} \\ C_{1,0} & C_{1,4} & C_{1,7} & - \\ C_{1,1} & C_{1,5} & - & C_{1,10} \\ C_{1,2} & - & C_{1,8} & C_{1,11} \end{bmatrix} \quad (12)$$

$$\begin{bmatrix} \mathbf{e} & \mathbf{b} & \mathbf{d} & \mathbf{9} \\ \mathbf{9} & \mathbf{e} & \mathbf{b} & \mathbf{d} \\ \mathbf{d} & \mathbf{9} & \mathbf{e} & \mathbf{b} \\ \mathbf{b} & \mathbf{d} & \mathbf{9} & \mathbf{e} \end{bmatrix} \times \begin{bmatrix} k_2[4] & k_2[8] & k_2[12] \\ k_2[5] & k_2[9] & k_2[13] \\ k_2[6] & k_2[10] & k_2[14] \\ k_2[7] & k_2[11] & k_2[15] \end{bmatrix} = \begin{bmatrix} \mathbf{e} & \mathbf{b} & \mathbf{d} & \mathbf{9} \\ \mathbf{9} & \mathbf{e} & \mathbf{b} & \mathbf{d} \\ \mathbf{d} & \mathbf{9} & \mathbf{e} & \mathbf{b} \\ \mathbf{b} & \mathbf{d} & \mathbf{9} & \mathbf{e} \end{bmatrix} \times \begin{bmatrix} k_3[0] \oplus k_3[4] & k_3[4] \oplus k_3[8] & k_3[8] \oplus k_3[12] \\ k_3[1] \oplus k_3[5] & k_3[5] \oplus k_3[9] & k_3[9] \oplus k_3[13] \\ k_3[2] \oplus k_3[6] & k_3[6] \oplus k_3[10] & k_3[10] \oplus k_3[14] \\ k_3[3] \oplus k_3[7] & k_3[7] \oplus k_3[11] & k_3[11] \oplus k_3[15] \end{bmatrix} = \begin{bmatrix} - & - & - \\ - & - & C_{2,2} \\ - & C_{2,1} & - \\ C_{2,0} & - & - \end{bmatrix} \quad (13)$$

$$\begin{bmatrix} k_4[5] \\ k_4[10] \\ k_4[15] \end{bmatrix} = \begin{bmatrix} k_3[1] \oplus \text{SRD}(k_3[14]) \oplus k_3[5] \\ k_3[2] \oplus \text{SRD}(k_3[15]) \oplus k_3[6] \oplus k_3[10] \\ k_3[3] \oplus \text{SRD}(k_3[0]) \oplus k_3[7] \oplus k_3[11] \oplus k_3[15] \end{bmatrix} = \begin{bmatrix} C_{3,0} \\ C_{3,1} \\ C_{3,2} \end{bmatrix} \quad (14)$$

Combining the above constraints, we have the following system of equations:

$$\begin{array}{c}
 \left[ \begin{array}{ccc}
 9 \cdot (k_3[0] \oplus \#SB^4[0]) \oplus e \cdot k_3[1] & \oplus b \cdot k_3[2] & \oplus d \cdot k_3[3] \\
 d \cdot (k_3[0] \oplus \#SB^4[0]) \oplus 9 \cdot k_3[1] & \oplus e \cdot k_3[2] & \oplus b \cdot k_3[3] \\
 b \cdot (k_3[0] \oplus \#SB^4[0]) \oplus d \cdot k_3[1] & \oplus 9 \cdot k_3[2] & \oplus e \cdot k_3[3] \\
 \hline
 e \cdot k_3[4] & \oplus b \cdot (k_3[5] \oplus \#SB^4[5]) \oplus d \cdot k_3[6] & \oplus 9 \cdot k_3[7] \\
 9 \cdot k_3[4] & \oplus e \cdot (k_3[5] \oplus \#SB^4[5]) \oplus b \cdot k_3[6] & \oplus d \cdot k_3[7] \\
 d \cdot k_3[4] & \oplus 9 \cdot (k_3[5] \oplus \#SB^4[5]) \oplus e \cdot k_3[6] & \oplus b \cdot k_3[7] \\
 \hline
 e \cdot k_3[8] & \oplus b \cdot k_3[9] & \oplus d \cdot (k_3[10] \oplus \#SB^4[10]) \oplus 9 \cdot k_3[11] \\
 9 \cdot k_3[8] & \oplus e \cdot k_3[9] & \oplus b \cdot (k_3[10] \oplus \#SB^4[10]) \oplus d \cdot k_3[11] \\
 b \cdot k_3[8] & \oplus d \cdot k_3[9] & \oplus 9 \cdot (k_3[10] \oplus \#SB^4[10]) \oplus e \cdot k_3[11] \\
 \hline
 e \cdot k_3[12] & \oplus b \cdot k_3[13] & \oplus d \cdot k_3[14] & \oplus 9 \cdot (k_3[15] \oplus \#SB^4[15]) \\
 d \cdot k_3[12] & \oplus 9 \cdot k_3[13] & \oplus e \cdot k_3[14] & \oplus b \cdot (k_3[15] \oplus \#SB^4[15]) \\
 b \cdot k_3[12] & \oplus d \cdot k_3[13] & \oplus 9 \cdot k_3[14] & \oplus e \cdot (k_3[15] \oplus \#SB^4[15]) \\
 \hline
 b \cdot (k_3[0] \oplus k_3[4]) \oplus d \cdot (k_3[1] \oplus k_3[5]) \oplus 9 \cdot (k_3[2] \oplus k_3[6]) & \oplus e \cdot (k_3[3] \oplus k_3[7]) \\
 d \cdot (k_3[4] \oplus k_3[8]) \oplus 9 \cdot (k_3[5] \oplus k_3[9]) \oplus e \cdot (k_3[6] \oplus k_3[10]) & \oplus b \cdot (k_3[7] \oplus k_3[11]) \\
 9 \cdot (k_3[8] \oplus k_3[12]) \oplus e \cdot (k_3[9] \oplus k_3[13]) \oplus b \cdot (k_3[10] \oplus k_3[14]) & \oplus d \cdot (k_3[11] \oplus k_3[15]) \\
 \hline
 k_3[1] \oplus S_{RD}(k_3[14]) \oplus k_3[5] & & & \\
 k_3[2] \oplus S_{RD}(k_3[15]) \oplus k_3[6] & \oplus k_3[10] & & \\
 k_3[3] \oplus S_{RD}(k_3[0]) \oplus k_3[7] & \oplus k_3[11] & \oplus k_3[15] & 
 \end{array} \right] = \left[ \begin{array}{c}
 C_{1,0} \\
 C_{1,1} \\
 C_{1,2} \\
 \hline
 C_{1,3} \\
 C_{1,4} \\
 C_{1,5} \\
 \hline
 C_{1,6} \\
 C_{1,7} \\
 C_{1,8} \\
 \hline
 C_{1,9} \\
 C_{1,10} \\
 C_{1,11} \\
 \hline
 C_{2,0} \\
 C_{2,1} \\
 C_{2,2} \\
 \hline
 C_{3,0} \\
 C_{3,1} \\
 C_{3,2}
 \end{array} \right] \quad (15)
 \end{array}$$

Essentially, the coefficients in the first 15 rows of equation form a  $15 \times 20$  matrix. That is:

$$\begin{array}{c}
 \left[ \begin{array}{cccccccccccccccccccc}
 9 & e & b & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\
 d & 9 & e & b & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d & 0 & 0 & 0 & 0 \\
 b & d & 9 & e & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & e & b & d & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 9 & e & b & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & e & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & d & 9 & e & b & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & e & b & d & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & e & b & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b & d & 9 & e & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & e & b & d & 9 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d & 9 & e & b & 0 & 0 & 0 & b & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b & d & 9 & e & 0 & 0 & 0 & e & 0 & 0 & 0 & 0 & 0 \\
 b & d & 9 & e & b & d & 9 & e & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & d & 9 & e & b & d & 9 & e & b & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & e & b & d & 9 & e & b & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
 \end{array} \right] \times \left[ \begin{array}{c}
 k_3[0] \\
 k_3[1] \\
 k_3[2] \\
 k_3[3] \\
 k_3[4] \\
 k_3[5] \\
 k_3[6] \\
 k_3[7] \\
 k_3[8] \\
 k_3[9] \\
 k_3[10] \\
 k_3[11] \\
 k_3[12] \\
 k_3[13] \\
 k_3[14] \\
 k_3[15] \\
 \#SB^4[0] \\
 \#SB^4[5] \\
 \#SB^4[10] \\
 \#SB^4[15]
 \end{array} \right] = \left[ \begin{array}{c}
 C_{1,0} \\
 C_{1,1} \\
 C_{1,2} \\
 C_{1,3} \\
 C_{1,4} \\
 C_{1,5} \\
 C_{1,6} \\
 C_{1,7} \\
 C_{1,8} \\
 C_{1,9} \\
 C_{1,10} \\
 C_{1,11} \\
 C_{2,0} \\
 C_{2,1} \\
 C_{2,2}
 \end{array} \right] \quad (16)
 \end{array}$$

Because the rank of this matrix is full (i.e., 15), the number of solutions for an arbitrary vector of constants is  $2^{(20-15) \times 8} = 40$ .

The last three rows in Eq. (15) impose 3 byte-constraints, which are non-linear (through the AES SBox  $S_{RD}$ ) on  $k_3[0]$ ,  $k_3[14]$ , and  $k_3[15]$ . By experiment, we verified that for each possible value of  $(C_{3,0}, C_{3,1}, C_{3,1})$ , there are exactly  $2^{40-24} = 2^{16}$  out of the  $2^{40}$  solutions made the three equation hold.

**Obtain the values of the neutral bytes for backward chunk.** Essentially, the values of the neutral bytes for backward chunk can be obtained by solving the following solutions (see Fig. 7, which is imposed on state  $\#MC^5$  to make the neutral bytes for backward chunk has two-byte constant influence on state

$\#AK^5$ . Such constraint has been used in many previous attacks)

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} 0 \\ \#MC^5[1] \\ \#MC^5[2] \\ \#MC^5[3] \end{bmatrix} = \begin{bmatrix} C_{4,0} \\ - \\ C_{4,1} \\ - \end{bmatrix}, \text{ i.e., } \begin{bmatrix} 3 \cdot \#MC^5[1] \oplus 1 \cdot \#MC^5[2] \oplus 1 \cdot \#MC^5[3] \\ 1 \cdot \#MC^5[1] \oplus 2 \cdot \#MC^5[2] \oplus 3 \cdot \#MC^5[3] \end{bmatrix} = \begin{bmatrix} C_{4,0} \\ C_{4,1} \end{bmatrix} \quad (17)$$

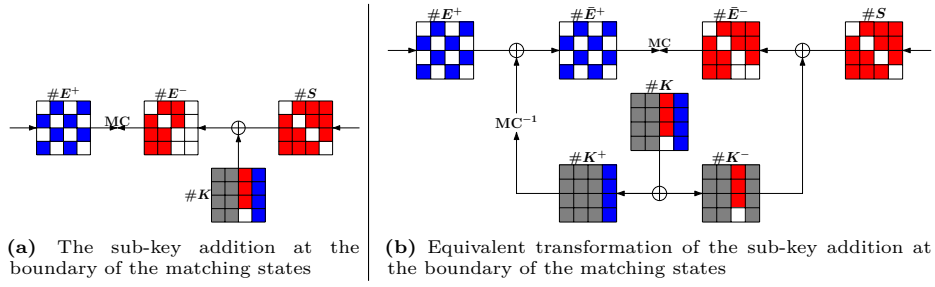
Because of the property of `MixColumns`, this equation has  $2^{(3-2) \times 8}$  solutions for each possible values of  $C_{4,0}$  and  $C_{4,1}$ . Thus, given  $C_{4,0}$ ,  $C_{4,1}$  we can obtain  $2^8$  values for the neutral bytes for backward computations. The way to compute them can either be using the Gaussian elimination to solve the linear equations, or precompute them (as described in the above attack procedure) for all possible values  $C_{4,0}$  and  $C_{4,1}$  and store in a table ( $T_2$  as described in the above attack procedure) to reuse in the attack.

## C Additional Techniques for MITM Preimage Attacks

**Convert Pseudo-preimage Attacks to Preimage Attacks.** For  $n$ -bit narrow-pipe iterated hash function, by an unbalanced meet-in-the-middle approach, a *pseudo-preimage* attack with a computational complexity of  $2^\ell$  ( $\ell < n - 2$ ) can be converted into a *preimage* attack with computational complexity of  $2^{(n+\ell)/2+1}$  [35, Fact9.99]. Note that, here, the unbalanced meet-in-the-middle approach is a more general procedure which is different with our focused meet-in-the-middle technique used in the pseudo-preimage attack on the compression function. It is a higher level of meet-in-the-middle procedure which calls our meet-in-the-middle pseudo-preimage attack as sub-procedures. In [32], Leurent improved this general unbalanced meet-in-the-middle method in the case where given  $k$  targets, the complexity of a pseudo-preimage attack can be reduced from  $2^\ell$  to  $2^\ell/k$ . The improved method uses these multi-target pseudo-preimage attacks to form an unbalanced-tree, and uses the expandable message technique to overcome the length padding. The overall time complexity of this improved method can be  $((n - \ell) \cdot \ln 2 + 1) \cdot 2^\ell$ . For more details, please refer to [8, 9, 32].

**Tricks for matching the ending states as indirect matching and matching through MixColumns used in [4, 9, 18].** In the MITM preimage attack on AES-like hash functions, the last sub-key addition leading to  $\#E^-$  is close to the boundary of the forward and backward computation as illustrated in Fig. 15a. Therefore, to perform matching, one can decompose state as  $\#K = \#K^+ + \#K^-$ , and translate the computation in Fig. 15a into its equivalent form shown in Fig. 15b, since  $MC(\#E^+) \oplus \#K = MC(\#E^+ \oplus MC^{-1}(\#K^+)) \oplus \#K^-$ .

The decomposition of state  $\#K$  moves all known cells of  $\#K$  for the forward computation (**Blue** and **Gray** cells) into  $\#K^+$ . Then  $\#K^-$  contains only **Red** cells (known cells for the backward computation) and **White** cells (unknown cells for both the forward and backward computation). The unknown cells for both the forward and backward computation are placed at  $\#K^-$  rather than  $\#K^+$  because they step into the encryption data path directly (unlike  $\#K^+$  for which



**Fig. 15:** Tricks for matching the ending states

the  $MC^{-1}$  operation has to be applied before  $\#K^+$  goes into the encryption data path), and thus keep the effect of unknown cells local. In contrast, one **White** cell in  $\#K^+$  would make one column of the state in the encryption data path **White**. With this approach, the coloring scheme of  $\#E^+$  is left intact, and some **Red** cells in  $\#E^-$  are protected from being destroyed by the **Blue** cells in the original  $\#K$ . For example, the three **Red** cells in the last column of  $\#S$  shown in Fig. 15a are preserved by decomposing  $\#K$  as presented in Fig. 15b.