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

Zhenzhen Bao[2], Xiaoyang Dong[3], Jian Guo[2], Zheng Li[4],
Danping Shi[1,5], Siwei Sun[1,5], and Xiaoyun Wang[3]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, China. `{shidanping,sunsiwei}@iie.ac.cn`
[2] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore. `{zzbao,guojian}@ntu.edu.sg`
[3] Institute for Advanced Study, Tsinghua University, China. `{xiaoyangdong,xiaoyunwang}@tsinghua.edu.cn`
[4] Faculty of Information Technology, Beijing University of Technology, China. `lizhengcn@bjut.edu.cn`
[5] 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, Rijndael, Haraka v2, Hash functions, MITM, Preimage attacks, 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. For that, the MMO-mode instantiated with AES is taken as an example for standardized way of building hash function based on block ciphers suggested by ISO [22]. 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 [26], ECHO [11], Grøstl [17], and Haraka v2 [27].

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 [7], Aumasson *et al.* devised preimage attacks on step-reduced MD5 and full 3-pass HAVAL [51], 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 [5] 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, 23, 24], and further evolved to differential views [15, 25]. 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, 47], HAVAL [19, 40] and round-reduced variants of many other hash functions such as SHA-0 and SHA-1 [4, 15, 25], SHA-2 [3], BLAKE [15], HAS-160 [20], RIPEMD and RIPEMD-160 [48], Stribog [1], Whilwind [2], and AES hashing modes [9, 38, 49]. Interestingly, the idea of MITM preimage attack also leads to the progress of collision attacks against reduced SHA-2 [30].

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 [49] improved the complexities in multi-target setting. Different from early MITM attacks on the MD-SHA family, their attacks select the neutral words (strictly, 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 bruteforce, 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 better differential/linear paths etc., 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 [16,28,33,36,45], cube(-like) attacks [31,32,44], integral attacks based on division properties [50], three-subset meet-in-the-middle attacks [39], and Demirci-Selçuk meet-in-the-middle attacks [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 [8,10,27] 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 [9,38,49]. 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 AddRoundKey and MixColumns, we treat them as a whole (a composition transformation) and formalize constraints that can result in all possible impacts

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

Target	#Round	Time-1	Time-2	(DoF^+, DoF^-, 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)	[49]
	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]
	8/12	$2^{112-\min(t,16)}$	2^{116}	(16, 32, 32)	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. 10
Rijndael-256	9/14	2^{248}	2^{253}	(16, 16, 8)	Fig. 12
Haraka-256 v2	7/10	2^{248}	2^{248}	(8, 8, 96)	[27]
	9/10	2^{224}	2^{224}	(32, 32, 64)	Fig. 13
Haraka-512 v2	8/10	2^{248}	2^{248}	(8, 8, 64)	[27]
	10/10	$2^{224-\min(t,32)}$	2^{224}	(128, 32, 64)	Fig. 14
	11/10	2^{240}	2^{240}	(128, 128, 16)	Fig. 15

– 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, such as DM-mode, 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^+, DoF^-, 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.

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, when we apply the automatic search to AES-like hashing, improved attacks than the previous best ones were obtained. That includes the first preimage attacks against 8-round AES-128, 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 [27], as well as a re-confirmation of the best attack on 8-round AES-192 hashing modes from [9]. The details of our results, together with a comparison to the previous related works, are summarized in Table 1.

2 AES-like Hashing and Meet-in-the-Middle 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 compression functions, i.e., given one or multiple targets T, find input chaining value h and message block M, such that $\mathsf{CF}(h, M) = T$. Such attacks are called pseudo-preimage attacks, in which the chaining value is free of choice, which is different from that in preimage attacks. Pseudo-preimages can be converted to (second-)preimages of hash functions using generic methods (details can be found in Appendix C).

2.1 AES-like Hashing

We focus on hash functions whose compression functions are built on AES-like block ciphers with a feed-forward mechanism. Typically, the compression functions can be constructed from block ciphers according to the twelve secure PGV-modes [37], out of which three modes, known as DM-mode, MMO-mode, and MP-mode, are commonly used in practice. We call such hash functions as AES-like hashing.

More concretely, AES-like hashing refers to the hash function whose underlying compression function is based on AES-like round functions as depicted in Fig. 2, where the state being manipulated is organized into an $\mathrm{N}_{\mathtt{row}} \times \mathrm{N}_{\mathtt{col}}$ 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 $S : \mathbb{F}_{2^c} \to \mathbb{F}_{2^c}$.
- **ShiftRows$_{\pi_t}$.** Permute the cell positions according to the permutation π_t.
- **MixColumns.** Update each column by left-multiplying an $\mathrm{N}_{\mathtt{row}} \times \mathrm{N}_{\mathtt{row}}$ MDS matrix (maximal distance separable matrix, with branch number $\mathtt{B_n} = \mathrm{N}_{\mathtt{row}}+1$, *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 $\mathrm{N}_{\mathtt{row}} + 1$).
- **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 Meet-in-the-Middle Preimage Attacks

The MITM approach was initially proposed by Diffie and Hellman in [14] as a generic time-memory trade-off technique to attack against encryption schemes with clear separations such as double DES. The basic requirements for applying the MITM approach in attacking symmetric-key ciphers are as follows.

– Invertibility: the round function is invertible. To apply the MITM attack to hash functions with iterative round-based CF, removing the feed-forward by the input to the CF should make the CF invertible. That is, given an intermediate state, it should be easy to compute all other states.

– Detachability: the whole cipher can be separated into two chunks of computation (*e.g.*, $E = E_2 \circ E_1$), each of which can be executed using part of the material to be recovered independently to the other.

From some starting states, one chunk is computed forward, and the other is computed backward (the starting states of the two chunks can be the same or naturally compatible ones). Each of them covers shorter computational flow than that of the whole cipher and exhaustively trialing on part of the materials, and provides a list of candidate values for an internal state at the ending point (at where the two chunks meet) of the two chunks. Between the two lists, complete pairwise matching can be applied (*i.e.*, each element in one list can be used to find a match with any element in the other list). A match provides a valid solution for the material to be recovered. Suppose the two lists are of 2^{d_1} and 2^{d_2} entries of n-bit values, respectively. To find a match with high probability, it is required $2^{(d_1+d_2)} \geq 2^n$, *i.e.*, $d_1 + d_2 \geq n$. Thus, suppose the two lists can be obtained with 2^{d_1} and 2^{d_2} computations, respectively, the minimum time complexity of a simple MITM attack is in the setting of $d_1 = d_2 = n/2$.

In [7], Aumasson *et al.* inverted the round-reduced compression function of MD5 and 3-pass HAVAL, in which the critical technique can be viewed as the application of local-collision combined with MITM. Sasaki and Aoki in [40] formally proposed to combine the MITM and local-collision approaches and successfully devised preimage attacks on full 3, 4, and 5-pass HAVAL. Since the pioneering works on preimage attacks on MD4, MD5, and HAVAL [7,29,40,41], the approach of MITM combined with local-collision has been applied and further developed for preimage attacks on many other hash functions. This method develops into *splice-and-cut* [5] MITM preimage attacks with support from *initial structure* [42] and (indirect) *partial matching* techniques.

Initial Structures [42]. From the idea of local-collision (informally, it is a collision within a few steps, created by inducing difference at one step and absorbing the differences at another step [29,34]), Sasaki and Aoki proposed a novel concept – *initial structure*. The initial structure is a few consecutive steps, where the two chunks overlapped and thus includes two sets of neutral words (also applies to bytes or bits). Although the neutral words, denoted by N^+ and N^-, appear simultaneously at these steps, they are involved in the computation of the proper chunk only (*e.g.*, N^+ involved in the forward chunk only). Besides, it is allowed that among the neutral words for one chunk, the values of some neutral words are selected according to the values of some other neutral words, such that changing them together does not impact the computation of the other chunk (the changes brought by them are mutually canceled). 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^+.

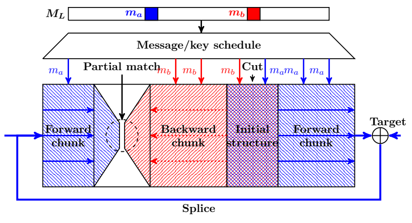
The function of the initial structure is to skip several steps at the beginning of chunks in a MITM attack so that the attack covers more rounds. To construct initial structures, it does not necessarily rely on the existence of *absorption properties* of the Boolean functions (BFs) as exploited in [7, 29]. The involved BF can be simply the XOR operation. Besides, in [42], with the so-called *cross absorption properties*, which is an extended version of the absorption properties, more complex initial structures were constructed. A key point for the construction of initial structures is how to define exploitable (cross) absorption properties and how to choose the neutral words, and pick out the best among many possible initial structures. Note that in [42], possible patterns of initial structures were searched manually.

Remark 1 (Comparison with 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. It was proposed with an informal definition and the ways to construct it were complex [24], which leaves more area to exploit. In [24], authors replaced the idea of initial structure with a more formal and general concept, which is named biclique (a complete bipartite graph with nodes being two sets of internal state and edges being candidate values of the material fed into the key/message schedule). 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 (e.g., message-modification and local-collision). This concept of biclique has been applied to both preimage attacks on hash functions (*e.g.*, SHA-1, SHA-2 and Skein-512 [24, 25]) and key-recovery attacks on block ciphers (*e.g.*, AES and IDEA [12, 23]).

Independent of the formalism of initial structure using concept of biclique, and instead of adhering to a formal definition, in this paper, we apply the essential idea behind the original concept of initial structure to an un-predefined number of rounds, formalize the essential idea using explicit rules, and resort to automatic tools to efficiently search for the best ones.

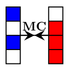
(Indirect-) Partial matching [5, 42]. In the two states for matching, as long as there remain one common position at which the value of the word can be computed independently between the forward and backward chunks, the matching can be performed. Partial matching requires the knowledge of the partial state only, which could possibly mean the two chunks can be computed for a few more steps, so for the whole attack. Further, apart from directly matching values of common words, any determined relations between values of words in the states for matching 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 $\#\text{MC}^1$ and $\#\text{AK}^1$, and each column provides $2 + 3 - 4 = 1$ byte filter, as exemplified in Fig. 1b.

Multi-targets [18, 49]. When multiple targets are available, it adds the degree of freedom to the chunk where targets are added to. For example, in the attack in



(a) The framework of the attack

Partial match between one column of $\#\mathrm{MC}^1$ and $\#\mathrm{AK}^1$:

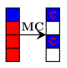


$$i.e., \begin{bmatrix} \#\mathrm{MC}^1[0] \\ - \\ \#\mathrm{MC}^1[2] \\ - \end{bmatrix} = \begin{bmatrix} \mathtt{e} & \mathtt{b} & \mathtt{d} & \mathtt{9} \\ \mathtt{9} & \mathtt{e} & \mathtt{b} & \mathtt{d} \\ \mathtt{d} & \mathtt{9} & \mathtt{e} & \mathtt{b} \\ \mathtt{b} & \mathtt{d} & \mathtt{9} & \mathtt{e} \end{bmatrix} \times \begin{bmatrix} - \\ \#\mathrm{AK}^1[1] \\ \#\mathrm{AK}^1[2] \\ \#\mathrm{AK}^1[3] \end{bmatrix} \Rightarrow$$

$$\begin{array}{ccc} \mathtt{d}\cdot\#\mathrm{MC}^1[0] & & \mathtt{d}\cdot(\mathtt{b}\cdot\#\mathrm{AK}^1[1]\oplus\mathtt{d}\cdot\#\mathrm{AK}^1[2]\oplus\mathtt{9}\cdot\#\mathrm{AK}^1[3]) \\ \oplus & = & \oplus \\ \mathtt{e}\cdot\#\mathrm{MC}^1[2] & & \mathtt{e}\cdot(\mathtt{9}\cdot\#\mathrm{AK}^1[1]\oplus\mathtt{e}\cdot\#\mathrm{AK}^1[2]\oplus\mathtt{b}\cdot\#\mathrm{AK}^1[3]) \end{array}$$

(b) Partial matching in (d)

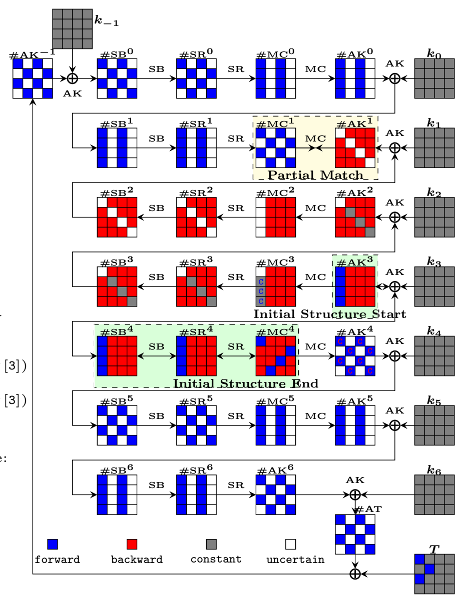
Constraints on $\#\mathrm{MC}^4[1,2,3]$ to build the initial structure:



$$i.e., \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} 0 \\ \#\mathrm{MC}^4[1] \\ \#\mathrm{MC}^4[2] \\ \#\mathrm{MC}^4[3] \end{bmatrix} = \begin{bmatrix} \mathtt{c}_0 \\ - \\ \mathtt{c}_1 \\ - \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} 3\cdot\#\mathrm{MC}^4[1] \oplus 1\cdot\#\mathrm{MC}^4[2] \oplus 1\cdot\#\mathrm{MC}^4[3] \\ 1\cdot\#\mathrm{MC}^4[1] \oplus 2\cdot\#\mathrm{MC}^4[2] \oplus 3\cdot\#\mathrm{MC}^4[3] \end{bmatrix} = \begin{bmatrix} \mathtt{c}_0 \\ \mathtt{c}_1 \end{bmatrix}$$

(c) Constraints on the initial structure in (d)



(d) An example attack on 7-round AES-hashing

Fig. 1: The MITM pseudo-preimage attack [38, 49]

Fig. 1d, considering only the internal states, the degree of freedom for backward is 24 bits, whereas that for the forward is 8. However, when there are multiple targets, the degree of freedom for forward can be directly increased.

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

1. Chunk separation – the location of initial structure and matching points for the forward/backward computations.
2. The neutral bytes – the selection of the neutral bytes and the constraints on them, which determines the degrees of freedom for each chunk.
3. The bytes for matching – the deterministic relation used for matching, which depends on the selection of neutral bytes and computation rules of the cipher.

After setting up the configuration, the 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).
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^+ (the values for matching can be similar to the left-hand side of the bottom equation in Fig. 1b).

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^- (the values for matching can be similar to the right-hand side of the bottom equation in Fig. 1b).
5. Check whether there is a match on indices between L^+ and L^-.[6]
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 directions 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 Hints for Optimizing MITM Attacks on AES-like Hashing

Degrees of Freedom. Shown by the complexity analysis, the MITM pseudo-preimage attack can benefit from larger degrees of freedom of the two computational chunks and matching. As mentioned above, 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 (AES hashing modes, Whirlpool etc. [38, 49]), the degree of freedom is imposed from the bytes in encryption states [7], and the attacks set the material fed into the key-schedule as arbitrary constant (take the attack in Fig. 1d for example, in which the key state are fixed to be constant). That is because, the key-schedule has a property that is any change on the secret key or a sub-key will quickly propagate to all other sub-keys. Introducing freedom from the key state, the differences lie in the active bytes will be diffused both to and among other key states and encryption states, which will make the analysis complex. In [9], the

[6] One can also directly execute the matching with L^+ once obtain one possible value of the matching point from the backward, instead of storing in L^- for later match. However, because the time complexity is generally the bottleneck, and the asymptotic complexity will be the same, we describe our attacks in this aligned form.

[7] 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 the message words. For different mode-of-operations, the correspondence might be different, *e.g.*, for DM-mode, the message is fed into the key-schedule thus the key-schedule in the block cipher can be viewed as message-schedule in the hash function; whereas for MMO-mode, the message is fed as plaintext into the encryption algorithm and the chaining value is fed into the key-schedule thus cannot be viewed as message-schedule. Therefore, we simply use encryption and key-schedule to be able to apply to both modes.

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 other chunk as little as possible to cover as many rounds as possible. To keep the analysis doable, the authors in [9] proposed that the neutral bytes in key states are all introduced for merely one chunk (the one that has relatively less degree of freedom), and found better ways to select neutral bytes manually.

For the ways to avoid impacts from neutral words for one chunk on the other (*i.e.*, canceling the changes brought by the neutral words), 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 to avoid the propagation of difference. In the attack on AES-like hashing, the ways to control the impacts of the neutral bytes can be adding constraints on those neutral bytes (where the degree of freedom will be consumed) when they are inputs to the following operations.

1. `AddRoundKey` and XOR: one can restrict that the XOR of two neutral bytes be constant. That will consume one-byte degree of freedom. 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).
2. `MixColumns` (MC): one can restrict that some output bytes of the `MC` be constant, even if the input contains neutral bytes. That can limit the impacts from neutral bytes for one chunk on the other chunk. Take the attack in Fig. 1d for example. The neutral bytes for backward, *i.e.*, the red cells in state $\#\mathrm{MC}^4$ are restricted such that if the blue cells are fixed to be constant, after `MixColumns`, the cells marked by `C` are constant. That is, changing the values in the red cells in $\#\mathrm{MC}^4$ has no impact on the blue cells marked (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 (from $\#\mathrm{AK}^4$ to $\#\mathrm{MC}^1$).
   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 $i + \mathrm{N}_{\mathtt{row}} - j \geq \mathrm{N}_{\mathtt{row}} + 1$, *i.e.*, $i \geq j + 1$.
3. `MixColumns`$\circ$`AddRoundKey` (XOR-MC): when there are forward neutral bytes in both the key and the encryption state, to control their impacts on the backward, one may first apply the above-mentioned way of restriction on `AddRoundKey` and then on `MixColumns`. Different from that, we apply restriction on the composition transformation of `AddRoundKey` and `MixColumns`. The reason is that, the XOR operation in `AddRoundKey` is byte-wise. Only when two bytes in the two states being *at the same position*, 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* (and in the same column), the difference in one byte can absorb the difference in the other byte.

However, when considering the composition transformation `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 lie in byte positions that are not completely overlapped, we can still use the difference of some neutral bytes to absorb the difference of other neutral bytes (but they should lie in columns with a common index.) Sect. 4.1 and the listed attacks will provide more formal descriptions and concrete examples. Explicitly, suppose that there are i forward neutral bytes lies in the key state, and j forward neutral bytes lies 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 $k + \mathrm{N}_{\texttt{row}} - t \geq \mathrm{N}_{\texttt{row}} + 1$, *i.e.*, $k \geq t + 1$.

Remark 2 (Comparison with previous MITM preimage 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, 49], 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 mentioned above, the ways to impose degree of freedom (to select the neutral bytes) were limited in previous works to make the analysis doable by manual (*e.g.*, fixing the material fed into the key-schedule and imposing neutral bytes in encryption state only in [38, 49], or allowing imposing neutral bytes from the key state but only for the forward chunk in [9]). In this paper, we release these restrictions by allowing the selection of neutral bytes in both encryption state and key state, and for both 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 (Comparison 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. However, the tool is semiautomatic for which, the 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 hinds for optimizing MITM attacks on AES-like hashing are not included, which is essentially the most challenging parts in our formalization.

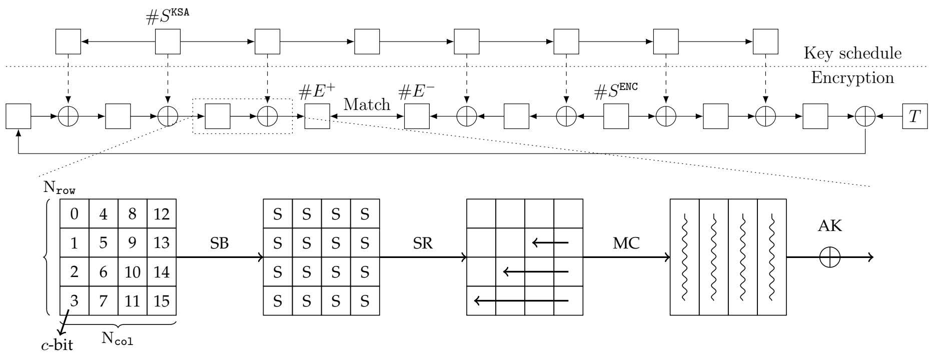


Fig. 2: A high-level overview of the MITM preimage attack

3 Formulate the MITM Preimage Attack on AES-like Hashing

Let $\mathcal{B}^{\mathtt{ENC}}$, $\mathcal{B}^{\mathtt{KSA}}$, $\mathcal{R}^{\mathtt{ENC}}$, $\mathcal{R}^{\mathtt{KSA}}$, $\mathcal{C}$, and $\mathcal{D}$ be subsets of $\mathcal{N} = \{0, 1, \cdots, n-1\}$ (with $n = \mathrm{N}_{\mathtt{row}} \cdot \mathrm{N}_{\mathtt{col}}$) whose elements are increasingly ordered. These ordered sets are used to reference the cells of certain states with n cells. 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 $\#S[0, 2, 7]$. Here, $\mathcal{B}^{\mathtt{ENC}}$ and $\mathcal{B}^{\mathtt{KSA}}$ refer to the neutral words from the internal state and message (or key of block cipher) for the forward chunk, and $\mathcal{R}^{\mathtt{ENC}}$ and $\mathcal{R}^{\mathtt{KSA}}$ for the backward chunk. We implicitly assume that the neutral words are whole cells, hence in what follows the degree of freedom refers to number of cells, rather than bits.

We now formulate the MITM preimage attack on a construction shown in Fig. 2. Without loss of generality, we assume that the states in the key schedule and encryption data paths both have n c-bit cells. Before we can mount a MITM preimage attack, four states: $\#S^{\mathtt{ENC}}$, $\#S^{\mathtt{KSA}}$, $\#E^{+}$, $\#E^{-}$, and six subsets of $\mathcal{N}$: $\mathcal{B}^{\mathtt{ENC}}$, $\mathcal{B}^{\mathtt{KSA}}$, $\mathcal{R}^{\mathtt{ENC}}$, $\mathcal{R}^{\mathtt{KSA}}$, $\mathcal{C}$, $\mathcal{D}$ (such that $\mathcal{B}^{\mathtt{ENC}} \cap \mathcal{R}^{\mathtt{ENC}} = \emptyset$ and $\mathcal{B}^{\mathtt{KSA}} \cap \mathcal{R}^{\mathtt{KSA}} = \emptyset$ for chunk independence) must be specified. $\#S^{\mathtt{ENC}}$ and $\#S^{\mathtt{KSA}}$ are the *starting states* in the encryption data path and key schedule data path respectively (corresponding to the location of an initial structure previously), and $\#E^{+}$ and $\#E^{-}$ are the *ending states* for the forward computation and backward computation respectively (corresponding to matching point previously). In the attack, partial knowledge of $\#E^{+}$ and $\#E^{-}$ used for matching is computed from the starting states $\#S^{\mathtt{ENC}}$ and $\#S^{\mathtt{KSA}}$. Note that to visualize these subsets and the attack, we will introduce a coloring system in Sect. 4, where cells referenced by $\mathcal{B}^{\mathtt{ENC}}$ and $\mathcal{B}^{\mathtt{KSA}}$ are `Blue`, and cells referenced by $\mathcal{R}^{\mathtt{ENC}}$ and $\mathcal{R}^{\mathtt{KSA}}$ are `Red`. The remaining cells in the starting states referenced by $\mathcal{G}^{\mathtt{ENC}}$ and $\mathcal{G}^{\mathtt{KSA}}$ are `Gray`, where $\mathcal{G}^{\mathtt{ENC}} = \mathcal{N} - \mathcal{B}^{\mathtt{ENC}} \cup \mathcal{R}^{\mathtt{ENC}}$ and $\mathcal{G}^{\mathtt{KSA}} = \mathcal{N} - \mathcal{B}^{\mathtt{KSA}} \cup \mathcal{R}^{\mathtt{KSA}}$. Moreover, $\mathcal{C}$ references the `Blue` and `Gray` cells in the ending state $\#E^{+}$, and $\mathcal{D}$ references the `Red` and `Gray` cells in the ending state $\#E^{-}$.

The six subsets of $\mathcal{N}$ must be chosen such that the following is fulfilled for some $\mathbb{X} \subseteq \mathbb{F}_{2^c}^{|\mathcal{B}^{\texttt{ENC}}|+|\mathcal{B}^{\texttt{KSA}}|}$ and $\mathbb{Y} \subseteq \mathbb{F}_{2^c}^{|\mathcal{R}^{\texttt{ENC}}|+|\mathcal{R}^{\texttt{KSA}}|}$, where $\mathbb{F}_{2^c}^{|\mathcal{B}^{\texttt{ENC}}|+|\mathcal{B}^{\texttt{KSA}}|}$ is the set of all possible values the `Blue` cells of the starting states can take, and $\mathbb{F}_{2^c}^{|\mathcal{R}^{\texttt{ENC}}|+|\mathcal{R}^{\texttt{KSA}}|}$ is the set of all possible values the `Red` cells of the starting states can take. We call $\lambda^+ = |\mathcal{B}^{\texttt{ENC}}| + |\mathcal{B}^{\texttt{KSA}}|$ the initial degrees of freedom (DoF) for the forward computation, and $\lambda^- = |\mathcal{R}^{\texttt{ENC}}| + |\mathcal{R}^{\texttt{KSA}}|$ the initial degrees of freedom (DoF) for the backward computation.

If the `Blue` cells $(\#S^{\texttt{ENC}}[\mathcal{B}^{\texttt{ENC}}], \#S^{\texttt{KSA}}[\mathcal{B}^{\texttt{KSA}}])$ in the starting states can only take values in $\mathbb{X} \subseteq \mathbb{F}_{2^c}^{|\mathcal{B}^{\texttt{ENC}}|+|\mathcal{B}^{\texttt{KSA}}|}$ with $|\mathbb{X}| = (2^c)^{d_1} \leq (2^c)^{|\mathcal{B}^{\texttt{ENC}}|+|\mathcal{B}^{\texttt{KSA}}|}$, and the `Red` cells $(\#S^{\texttt{ENC}}[\mathcal{R}^{\texttt{ENC}}], \#S^{\texttt{KSA}}[\mathcal{R}^{\texttt{KSA}}])$ in the starting states can only take values in $\mathbb{Y} \subseteq \mathbb{F}_{2^c}^{|\mathcal{R}^{\texttt{ENC}}|+|\mathcal{R}^{\texttt{KSA}}|}$ with $|\mathbb{Y}| = (2^c)^{d_2} \leq (2^c)^{|\mathcal{R}^{\texttt{ENC}}|+|\mathcal{R}^{\texttt{KSA}}|}$, then after we fixed the `Gray` cells $(\#S^{\texttt{ENC}}[\mathcal{G}^{\texttt{ENC}}], \#S^{\texttt{KSA}}[\mathcal{G}^{\texttt{KSA}}])$ in the starting states to some constant in $\mathbb{F}_{2^c}^{(n-|\mathcal{B}^{\texttt{ENC}}|-|\mathcal{R}^{\texttt{ENC}}|)+(n-|\mathcal{B}^{\texttt{KSA}}|-|\mathcal{R}^{\texttt{KSA}}|)}$, we can compute $(2^c)^{d_1}$ different values of $\#E^+[\mathcal{C}]$ in the forward direction which only depend on $(\#S^{\texttt{ENC}}[\mathcal{B}^{\texttt{ENC}}], \#S^{\texttt{KSA}}[\mathcal{B}^{\texttt{KSA}}])$. We store these $(2^c)^{d_1}$ different values in an imaginary list L^+. Similarly, we can compute $(2^c)^{d_2}$ different values of $\#E^-[\mathcal{D}]$ in the backward direction which only depend on $(\#S^{\texttt{ENC}}[\mathcal{R}^{\texttt{ENC}}], \#S^{\texttt{KSA}}[\mathcal{R}^{\texttt{KSA}}])$. We store these $(2^c)^{d_2}$ different values in an imaginary list L^-. For the two lists L^+ and L^- we can perform an m-cell matching test, that is, we expect $|L^+ \times L^-|/(2^c)^m$ pairs from $L^+ \times L^-$ to pass the test. We call m the degrees of matching (denoted by DoM). Note that $\mathcal{B}^{\texttt{ENC}}$ and $\mathcal{B}^{\texttt{KSA}}$ indicate the sources of the degrees of freedom for the forward computation, and $\mathcal{R}^{\texttt{ENC}}$ and $\mathcal{R}^{\texttt{KSA}}$ indicate the sources of the degrees of freedom for the backward computation. Since in the forward computation and backward computation $(\#S^{\texttt{ENC}}[\mathcal{B}^{\texttt{ENC}}], \#S^{\texttt{KSA}}[\mathcal{B}^{\texttt{KSA}}])$ and $(\#S^{\texttt{ENC}}[\mathcal{R}^{\texttt{ENC}}], \#S^{\texttt{KSA}}[\mathcal{R}^{\texttt{KSA}}])$ 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^+) and d_2 the degrees of freedom for the backward computation (denoted by DoF^-).

With this configuration, it is shown that the time complexity to find a full n-cell match for the two ending state is $(2^c)^{n-\min\{d_1,d_2,m\}}$. Therefore, for a valid MITM preimage attack, we must have $\text{DoF}^+ \geq 1$, $\text{DoF}^- \geq 1$ and $\text{DoM} \geq 1$. In the following section, we will show how to automatically determine $\mathcal{B}^{\texttt{ENC}}$, $\mathcal{B}^{\texttt{KSA}}$, $\mathcal{R}^{\texttt{ENC}}$, $\mathcal{R}^{\texttt{KSA}}$, $\mathcal{C}$, 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. While in our description of the attack we have no notion of the initial structure (starting states are defined instead), we claim that this description is more general and covers all attacks proposed in the open literature that employ initial structures. Secondly, as shown in Fig. 2, we emphasize that the starting states for the encryption data path and key schedule data path are not necessarily in the same "round". The sets referencing the cells satisfying the stated properties are all we need.

Besides, our program enumerated 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 algorithm. That is, for an N-round targeted cipher, our program generates MILP-models for each of the possible combinations $\{(\texttt{init}_r^{\mathrm{E}}, \texttt{init}_r^{\mathrm{K}}, \texttt{match}_r) \mid 0 \le \texttt{init}_r^{\mathrm{E}} < N,\ -1 \le \texttt{init}_r^{\mathrm{K}} < N,\ 0 \le \texttt{init}_r^{\mathrm{E}} < N,\ \texttt{init}_r^{\mathrm{E}} \ne \texttt{match}_r\}$, where $\texttt{init}_r^{\mathrm{E}}$ is the location of starting point in encryption, $\texttt{init}_r^{\mathrm{K}}$ is that in key-schedule, and $\texttt{match}_r$ 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 [3, 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. 16a. Therefore, to perform matching, one can decompose state as $\#K = \#K^+ + \#K^-$, and translate the computation in Fig. 16a into its equivalent form shown in Fig. 16b, since $\mathrm{MC}(\#E^+) \oplus \#K = \mathrm{MC}(\#E^+ \oplus \mathrm{MC}^{-1}(\#K^+)) \oplus \#K^-$. Full explanation can be found in Appendix C.

In the following description of our modeling method, for simplicity, we let the number of rows of the state $\mathrm{N}_{\texttt{row}}$ be 4, and thus, the branch number of the `MixColumns` $\mathtt{B_n} = \mathrm{N}_{\texttt{row}} + 1$ 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 in the backward chunk, and unknown in the forward chunk.
- `Blue` (`B`): known in the forward chunk, and unknown in the backward chunk.
- `White` (`W`): unknown in both the forward and backward chunk.

For the ith 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 $(x^{\#S}, y^{\#S}) = (1,1)$ 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

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

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 $(\#S^{\mathtt{ENC}}[\mathcal{B}^{\mathtt{ENC}}], \#S^{\mathtt{KSA}}[\mathcal{B}^{\mathtt{KSA}}])$, and the `Red` cells in the starting states are used to capture $(\#S^{\mathtt{ENC}}[\mathcal{R}^{\mathtt{ENC}}], \#S^{\mathtt{KSA}}[\mathcal{R}^{\mathtt{KSA}}])$.

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

$$\begin{cases} \lambda^+ = \sum_i x_i^{\#S} - \sum_i \beta_i^{\#S} \\ \lambda^- = \sum_i y_i^{\#S} - \sum_i \beta_i^{\#S} \end{cases}. \tag{3}$$

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 jth columns of the ending states $\#E^-$ and $\#E^+$ which are linked by the MixColumns operation. When t $(t \geq 5)$ out of the 8 bytes of the two columns are known, there is a filter of $t-4$ bytes.

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

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

Constraints for the States in the Computation Paths. This is the most essential part of this work, where we extend the construction of attacks on the basis of previous works, refine and apply the key idea behind the initial structures to a greater extent, and explicitly describe more possible ways of propagation of the properties (expressed in the four colors) of the cells involved in the computation paths of both the encryption and key schedule algorithm. Therefore, we would like to devote one separate section (Sect. 4.1) for the details of this part. Here we only give some high-level descriptions.

Let f be a function that transforms a state $\#S_{\texttt{IN}}$ into the state $\#S_{\texttt{OUT}}$. Then the coloring scheme of $(\#S_{\texttt{IN}}, \#S_{\texttt{OUT}})$ 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 $(\#S^{\texttt{ENC}}[\mathcal{R}^{\texttt{ENC}}], \#S^{\texttt{KSA}}[\mathcal{R}^{\texttt{KSA}}])$ in the starting states to some carefully constructed set $\mathbb{Y}$ defined in Sect. 3, we may able to change certain `Red` cells to `Blue` cells (or even `Gray` cells) along the forward computation path starting from the starting states to the ending state $\#E^{+}$. By doing so, the initial degrees of freedom λ^{-} of the `Red` cells in the starting states is reduced, and 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 (DoF^{+}) and for the backward computation (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, $\min\{\mathrm{DoF}^{+}, \mathrm{DoF}^{-}, \mathrm{DoM}\}$ should be maximized. To this end, we can introduce an auxiliary variable $v_{\texttt{Obj}}$, impose the constraints

$$\begin{cases} v_{\texttt{Obj}} \le \mathrm{DoF}^{+} \\ v_{\texttt{Obj}} \le \mathrm{DoF}^{-} \\ v_{\texttt{Obj}} \le \mathrm{DoM} \end{cases}$$

and set the objective function to maximize $v_{\texttt{Obj}}$.

In the multi-target setting, it is supposed 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), DoF^{-} (resp. DoF^{+}) can be directly increased, the objective is to maximize $\min\{\mathrm{DoF}^{+}, \mathrm{DoM}\}$ ($\min\{\mathrm{DoF}^{-}, \mathrm{DoM}\}$).

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

Recalling our description of the MITM preimage attack in Sect. 3, before we perform the attack on a given target with predefined starting states and ending states, we have to determine $\mathcal{B}^{\texttt{ENC}}$, $\mathcal{B}^{\texttt{KSA}}$, $\mathcal{R}^{\texttt{ENC}}$, and $\mathcal{R}^{\texttt{KSA}}$ for the starting states

$\#S^{\texttt{ENC}}$ and $\#S^{\texttt{KSA}}$. In our visualizations of the attacks, the `Blue` cells in the starting states $\#S^{\texttt{ENC}}$ and $\#S^{\texttt{KSA}}$ are meant to capture $\mathcal{B}^{\texttt{ENC}}$ and $\mathcal{B}^{\texttt{KSA}}$ respectively. Similarly, the `Red` cells in the starting states are used to capture $\mathcal{R}^{\texttt{ENC}}$ and $\mathcal{R}^{\texttt{KSA}}$, and the `Gray` cells in the starting states are used to capture $\mathcal{G}^{\texttt{ENC}}$, and $\mathcal{G}^{\texttt{KSA}}$.

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 λ^+ and λ^- respectively. Along the computation paths leading to the ending states, the initial degrees of freedom are consumed according to the coloring schemes along the computation paths.

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

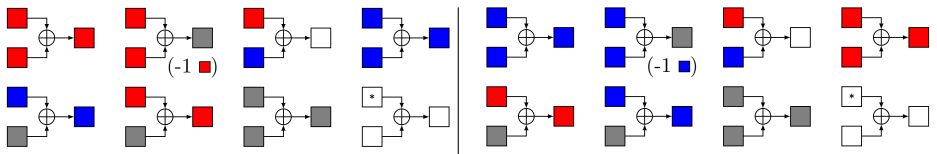
$$\begin{cases} \mathrm{DoF}^+ = \lambda^+ - \sigma^+ \\ \mathrm{DoF}^- = \lambda^- - \sigma^- \end{cases} . \tag{5}$$

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 back 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 $(2^c)^{n-\min\{\mathrm{DoF}^+,\mathrm{DoF}^-,\mathrm{DoM}\}}$, we always require $\mathrm{DoF}^+ \geq 1$ and $\mathrm{DoF}^- \geq 1$. Moreover, σ^+ is computed as $\sum \sigma^+(\#S_{\texttt{IN}} \to \#S_{\texttt{out}})$ along the computation path that consumes DoF for the forward computation, where $\sigma^+(\#S_{\texttt{IN}} \to \#S_{\texttt{out}})$ is the DoF for the forward computation consumed by the transition from state $\#S_{\texttt{IN}}$ to $\#S_{\texttt{out}}$, and σ^- is computed as $\sum \sigma^-(\#S_{\texttt{IN}} \to \#S_{\texttt{out}})$ along the computation path that consumes the DoF for the backward computation. To show how to compute σ^+ 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 of the two 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



(a) For the forward computation ($\texttt{XOR}^{+}\texttt{-RULE}$) **(b)** For the backward computation ($\texttt{XOR}^{-}\texttt{-RULE}$)

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

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 $\texttt{XOR}^{+}\texttt{-RULE}$, where a "+" sign signifies the forward computation) shown in Fig. 3a. Similarly, if the 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 $\texttt{XOR}^{-}\texttt{-RULE}$, which is visualized in Fig. 3b. Note that $\texttt{XOR}^{-}\texttt{-RULE}$ (Fig. 3b) can be obtained from $\texttt{XOR}^{+}\texttt{-RULE}$ (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 $\texttt{XOR}^{+}\texttt{-RULE}$ 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 [46], and the set of rules $\texttt{XOR}^{-}\texttt{-RULE}$ 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 🟥 $\oplus$ 🟥 $\rightarrow$ ▨ 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 $\texttt{MC}^{+}\texttt{-RULE}$) for the coloring schemes of the input and output columns. Examples of valid coloring schemes are shown in Fig. 4.

▶ $\texttt{MC}^{+}\texttt{-RULE-1}$. 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);

▶ $\texttt{MC}^{+}\texttt{-RULE-2}$. 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);

▶ $\texttt{MC}^{+}\texttt{-RULE-3}$. If all the input cells are `Gray`, then all the output cells are `Gray` (can perform bi-direction computations on fixed constants);

▶ $\texttt{MC}^{+}\texttt{-RULE-4}$. If there are `Red` and `Blue` cells but no `White` cells in the input column, each output cell must be `Blue` or `White`. Moreover, the number of `Red` and `White` cells from the input and output columns must be greater or equal to 5 (can partially cancel the impacts from 🟥 on 🟦 within an input

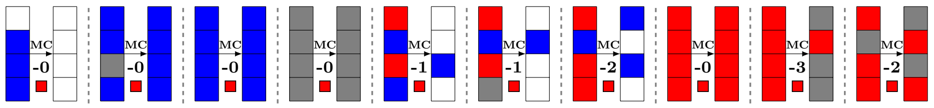


Fig. 4: Some valid coloring schemes for the MixColumns in the forward computation

column by consuming λ^-, and perform partial forward computations. Because of the MDS property of `MixColumns`, this is possible only when the condition is fulfilled);

- ▶ `MC`$^+$`-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, the number of `Red` cells from the input and output columns must be greater or equal to 5 (can partially cancel the difference within an input column by consuming λ^-. 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, and here we only show how to convert `MC`$^+$`-RULE-4` into a system of linear inequalities as an example. Other rules can be put into linear inequalities in a similar way.

First, we introduce a 0-1 indicator variable δ for the input column and necessary constraints into the model such that $\delta = 1$ if and only if the condition specified in the if-clause of `MC`$^+$`-RULE-4` 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 columns since $(x_0^{\#A}, y_0^{\#A}, \cdots, x_3^{\#A}, y_3^{\#A}) \in \mathbb{F}_2^8$. We define $\mathbb{V} \subseteq \mathbb{F}_2^8$ as the set of vectors

$$\{(x_0^{\#A}, y_0^{\#A}, \cdots, x_3^{\#A}, y_3^{\#A}, \delta) : (x_0^{\#A}, y_0^{\#A}, \cdots, x_3^{\#A}, y_3^{\#A}) \in \mathbb{F}_2^8\}, \tag{6}$$

where $\delta = 1$ if and only if the precondition of `MC`$^+$`-RULE-4` is satisfied. This subset can be described by linear inequalities with the convex hull computation method [46]. Now, with the help of the δ variable, we can convert `MC-RULE-4` into a system of inequalities.

The statement that the output cells must be `Blue` or `White` and the number of `Red` cells and `White` cells must be greater or equal to 5 is equivalent to

$$\begin{cases} \sum_{i=0}^{3} y_i^{\#B} \leq 0 \\ (4 - \sum_{i=0}^{3} x_i^{\#A}) + (4 - \sum_{i=0}^{3} x_i^{\#B}) \geq 5 \end{cases} . \tag{7}$$

Therefore, the overall constraint that (Eq. 7) must hold when $\delta = 1$ can be described by the following system of linear inequalities:

$$\begin{cases} 100(1-\delta) \geq \sum_{i=0}^{3} y_i^{\#B} \\ 100(1-\delta) \geq 5 - (4 - \sum_{i=0}^{3} x_i^{\#A}) - (4 - \sum_{i=0}^{3} x_i^{\#B}) \end{cases} , \tag{8}$$

where the number '100' can be replaced by any big enough positive integer, and this modeling is known as "big-M method". Also note that the above method is quite generic, in practice more *ad-hoc* approach can be employed to convert the rules into linear inequalities.

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 $\#C[0,1,2,3] = \mathrm{MC}^{-1}(\#A[0,1,2,3] + \#B[0,1,2,3])$, 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 there are no `Red` cells, no `White` cells and no `Blue` cells in the input columns, 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 4×2 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 🟦 on 🟥 within two input columns by consuming λ^+, 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 4×2 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 λ^+. Because of the MDS property of `MixColumns`, this is possible only when the condition is fulfilled).

Remark 5. One may be tempted to model the XOR-MC operation by applying `XOR`$^-$`-RULE` and `MC`$^-$`-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`$^-$`-RULE` results in `White` cells after the XOR operation. Subsequently, applying `MC`$^-$`-RULE`, we will end up with a

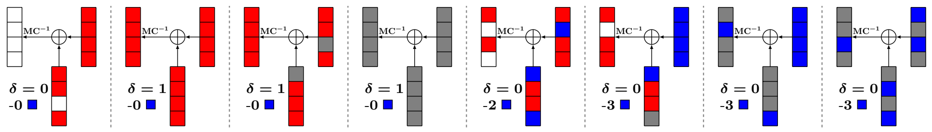
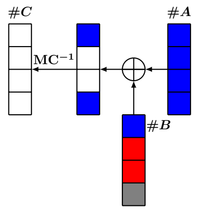


Fig. 5: Some valid coloring schemes for the XOR-MC in the backward computation



$$\begin{bmatrix} \#C[0] \\ \#C[1] \\ \#C[2] \\ \#C[3] \end{bmatrix} = \begin{bmatrix} \texttt{e} & \texttt{b} & \texttt{d} & \texttt{9} \\ \texttt{9} & \texttt{e} & \texttt{b} & \texttt{d} \\ \texttt{d} & \texttt{9} & \texttt{e} & \texttt{b} \\ \texttt{b} & \texttt{d} & \texttt{9} & \texttt{e} \end{bmatrix} \times \begin{bmatrix} \#A[0] \oplus \#B[0] \\ \#A[1] \\ \#A[2] \\ \#A[3] \end{bmatrix} \oplus \boxed{\begin{bmatrix} \texttt{e} & \texttt{b} & \texttt{d} & \texttt{9} \\ \texttt{9} & \texttt{e} & \texttt{b} & \texttt{d} \\ \texttt{d} & \texttt{9} & \texttt{e} & \texttt{b} \\ \texttt{b} & \texttt{d} & \texttt{9} & \texttt{e} \end{bmatrix} \times \begin{bmatrix} 0 \\ \#B[1] \\ \#B[2] \\ \#B[3] \end{bmatrix}}$$

Fig. 6: The inaccuracy of modeling XOR-MC in the backward computation by applying `XOR`$^-$`-RULE` and `MC`$^-$`-RULE` separately.

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

$$\begin{array}{l} \texttt{e} \cdot (\#A[0] \oplus \#B[0]) \oplus \texttt{b} \cdot \#A[1] \oplus \texttt{d} \cdot \#A[2] \oplus \texttt{9} \cdot \#A[3] = \texttt{C}_0 \\ \texttt{d} \cdot (\#A[0] \oplus \#B[0]) \oplus \texttt{9} \cdot \#A[1] \oplus \texttt{e} \cdot \#A[2] \oplus \texttt{b} \cdot \#A[3] = \texttt{C}_2 \\ \texttt{b} \cdot (\#A[0] \oplus \#B[0]) \oplus \texttt{d} \cdot \#A[1] \oplus \texttt{9} \cdot \#A[2] \oplus \texttt{e} \cdot \#A[3] = \texttt{C}_3 \end{array} \tag{9}$$

where $\texttt{C}_0$, $\texttt{C}_2$, and $\texttt{C}_3$ 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.

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 $(\#A[0], \cdots, \#A[3])$ and $(\#B[0], \cdots, \#B[3])$ be the two input columns and $(\#C[0], \cdots, \#C[3])$ be the output column. Given a valid coloring scheme of $\#A$, $\#B$, and $\#C$, the consumed DoF (measured in cells)

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

equals to 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 δ 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 δ:

$$\begin{cases} -\delta + \sum_{i=0}^{3} y_i^{\#A} + \sum_{i=0}^{3} y_i^{\#B} \leq 7 \\ y_i^{\#A} \geq \delta, i \in \{0,1,2,3\} \\ y_i^{\#B} \geq \delta, i \in \{0,1,2,3\} \end{cases} . \tag{10}$$

Then we have $\sigma^{+}((\#A[0,\cdots,3],\#B[0,\cdots,3]) \rightarrow \#C[0,\cdots,3]) = -4\delta + \sum_{i=0}^{3} y_i^{\#C}$. 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 tool presented in the previous section, 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 [27].

For most of the 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 (an attack on 8-round AES-128 hashing mode and an attack on full Haraka-512 v2) in detail. The list of optimal attacks we found is presented in Table 1. With the help of the visualizations of these attacks, the reader can reconstruct the actual 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

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, 11, and 12. Notably, apart from new attacks with better complexities, an 8-round attack on AES-128 and 9-round attacks on AES-256 hashing mode were found, which extend one more round compared with previous attacks [9,38,49].

To be clear, in the figures, some information are presented, such as which states are the starting states, 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 (λ^+, λ^-), and the final configuration $(\mathrm{DoF}^+, \mathrm{DoF}^-, \mathrm{DoM})$ 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 $\#\mathrm{SB}^4$ and $\#k_4$, and the ending states are $\#\mathrm{MC}^1$ and $\#\mathrm{SB}^2$. Also, we have $\mathcal{B}^{\texttt{ENC}} = [0, 5, 10, 15]$, $\mathcal{B}^{\texttt{KSA}} = [0, 1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14]$, $\mathcal{R}^{\texttt{ENC}} = [1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14]$, $\mathcal{R}^{\texttt{KSA}} = \emptyset$, $\mathcal{C} = [0, 2, 5, 7, 8, 10, 13, 15]$, 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 $\#\mathrm{SB}^4$, $\#k_3$, and $\#\mathrm{MC}^3$ 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 $\#\mathrm{SB}^3$, $\#k_2$, and $\#\mathrm{MC}^2$, and that consumes 3 cells of degrees of freedom for the forward computation. The states $\#\mathrm{MC}^4$ and $\#\mathrm{AK}^4$ are enclosed by a dashed light-purple frame ⬚, which means $\texttt{MC}^+\texttt{-RULE}$ is applied to them, and that consumes 9 cells of degrees of freedom for the backward computation. Similarly, the $\texttt{MC}^+\texttt{-RULE}$ is applied to states $\#\mathrm{MC}^5$ and $\#\mathrm{AK}^5$, and that consumes 2 cells of degrees of freedom for the backward computation. Accordingly, in the solution of our model, $\mathrm{DoF}^+ = 17 - 12 - 3 = 2$ and $\mathrm{DoF}^- = 12 - 9 - 2 = 1$, which indicates that the values of $(\#\mathrm{SB}^4[\mathcal{B}^{\texttt{ENC}}], \#k_4[\mathcal{B}^{\texttt{KSA}}])$ are restricted to a subset $\mathbb{X}$ of $\mathbb{F}_{2^8}^{17}$ with $(2^8)^2$ elements, and the values of $(\#\mathrm{SB}^4[\mathcal{R}^{\texttt{ENC}}], \#k_4[\mathcal{R}^{\texttt{KSA}}])$ 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 $(\#\mathrm{SB}^4[\mathcal{R}^{\texttt{ENC}}], \#k_4[\mathcal{R}^{\texttt{KSA}}])$, and the backward computation is irrelevant of $(\#\mathrm{SB}^4[\mathcal{B}^{\texttt{ENC}}], \#k_4[\mathcal{B}^{\texttt{KSA}}])$. Since the degrees of freedom for the forward and backward computations (DoF^+ and 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 Appendix B, we explicitly show in this example, how to obtain $\mathbb{X}$ and $\mathbb{Y}$ such that the required properties are fulfilled with negligible complexity. Here, we only briefly present the concrete procedure of the pseudo-preimage attack on 8-round AES-128 hashing corresponding to Fig. 7.

The procedure of the attack on 8-round AES-128 hashing

1. Random sample without replacement for values of the following variables:
   - $\mathtt{C}_{1,0}, \mathtt{C}_{1,1}, \ldots, \mathtt{C}_{1,11}$: 12 byte-values that will be the pre-determined impacts from blue cells in $\#\mathrm{SB}^4$ and k_3 on red cells in $\#\mathrm{MC}^3$ (marked by $\mathtt{C}$).
   - $\mathtt{C}_{2,0}, \mathtt{C}_{2,1}, \mathtt{C}_{2,2}$: 3 byte-values that will be the pre-determined impacts from blue cells in k_2 on red cells in $\#\mathrm{MC}^2$ (marked by $\mathtt{C}$).
2. Random sample without replacement for values of the following variables:
   - $\mathtt{C}_{3,0}, \mathtt{C}_{3,1}, \mathtt{C}_{3,2}$: 3 byte-values of the gray cells in k_4.
   - $\mathtt{C}_{4,0}, \mathtt{C}_{4,1}$: 2 byte-values that will be the impacts from red cells in $\#\mathrm{MC}^5$ on blue cells in $\#\mathrm{AK}^5$ (those marked by $\mathtt{C}$).

   – $\mathtt{C}_{5,0}, \mathtt{C}_{5,1}, \ldots, \mathtt{C}_{5,8}$: 9 byte-values of the gray cells in $\#\mathrm{AK}^4$.
3. Obtain 2^8 values of neutral bytes for the forward chunk and 2^8 values of neutral bytes for the backward chunk fulfilling the pre-determined impacts (as presented in Fig. 1c and Appendix B.)
4. Forward computation: for each of the 2^8 values of the neutral bytes:
   – Compute all required sub-key bytes.
   – Compute forward from $\#\mathrm{SB}^4$ to $\#\mathrm{MC}^1$ (blue cells in Fig. 7), obtain a candidate value for matching (as the left-hand side of the bottom equation in Fig. 1b) and store in a table L^+.
5. Backward computation: for each of the 2^8 values of the neutral bytes:
   – Compute backward from $\#\mathrm{MC}^5$ to $\#\mathrm{AK}^1$ (red cells in Fig. 7), obtain a candidate value for matching (as the right-hand side of the bottom equation in Fig. 1b) and store in a table L^-.
6. Matching: find matches between L^+ and L^-, for each such match of partial state, tested for full-state matching. Output (H_{N-1}, M_N) if a full match is found, otherwise, go back to Step 2 to repeat.

Complexity. From Eq. 1, the time complexity of this pseudo-preimage attack on 8-round AES-128 hashing is $2^{128-8\times\min\{\mathrm{DoF}^+,\mathrm{DoF}^-,\mathrm{DoM}\}} = 2^{120}$.

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 are 2 unique features observed from Fig. 7, which made the extra round possible. Firstly, the backward chunk covers one more round (specifically round 3) compared with that in [9, 38]. This is only possible after the consumption of 12 blue bytes of freedom degrees. Without the introduction of DoF from key bytes in [9] this year, this would not be possible. Secondly, 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 in round 1 at the matching point. This was not the case in [38] and [9]. This behaviour imposed lesser constraints for red bytes and saved some DoF for the forward chunk.

As depicted in Fig. 8, 9, 10, 11, 12 and summarized in Table 1, when our attack model is applied to hashing modes based on other AES variants, we are able to re-discover *all* previous best attacks against AES-192 and improve by one round from 8 to 9 rounds against AES-256 as in [9]. This is yet another piece of evidence that the solution space covered in our attack model is a superset of those in the previous works [9, 38].

5.2 Improved Attacks on Haraka v2

Haraka v2 [27] 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 π_s) in the DM-mode as follows:

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

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 π_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 $\mathbf{mix}_s$) 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$^+$ (which replaces the DM-mode with Sponge-based construction) [21] and Gravity-SPHINCS (which extends one round on top of the 5-round version) [6]. Gravity-SPHINCS is one of the round-1 and SPHINCS$^+$ is one of the round-2 candidates to the NIST post-quantum cryptography competition.

In [27], among other types of cryptanalysis, the designers provide MITM preimage attacks on 3.5-round Haraka-256 v2 and on 4-round Haraka-512 v2, both of which have complexity 2^{248}.

For both the two 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. 13, of which the complexity is $2^{256-8\times\min\{\mathrm{DoF}^+,\ \mathrm{DoF}^-,\ \mathrm{DoM}\}} = 2^{256-8\times\min\{4,\ 4,\ 8\}} = 2^{224}$. 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 extended 5.5-round (11 AES-rounds) is presented in App. A.2 and visualized in Fig. 15. In the following, we present one of the attacks on full Haraka-512 v2, which is visualized in Fig. 14.

A MITM Preimage Attack on Full Haraka-512 v2. From Fig. 14, it can be seen that in the searching of our model, the starting state is $\#\mathrm{SB}^5$, and the ending states are $\#\mathrm{MC}^8$ and $\#\mathrm{AC}^8$. Also, we have $\mathcal{B}^{\mathrm{ENC}} = [16, 17, \dots, 63]$, $\mathcal{R}^{\mathrm{ENC}} = [1, 6, 11, 12]$, $\mathcal{C} = [16\cdot i + j \mid i \in \{0, 3\},\ j \in \{1, 2, 3, 4, 5, 6, 8, 9, 11, 12, 14, 15\}] \cup [16\cdot i + j \mid i \in \{1, 2\},\ j \in \{0, 2, 3, 5, 6, 7, 8, 9, 10, 12, 13, 15\}]$, and $\mathcal{D} = [16\cdot i + j \mid i \in \{1, 3\},\ j \in \{0, 1, 5, 6, 10, 11, 12, 15\}]$. Therefore, the initial degrees of freedom for the forward computation and backward computation are 48 and 4 respectively, *i.e.*, $(\lambda^+, \lambda^-) = (48, 4)$, and the degree of matching is $\mathrm{DoM} = (3 + 2 - 4) \times 8 = 8$. The $\mathtt{MC^{-}\text{-}RULE}$ applied to states $\#\mathrm{MC}^2$ and $\#\mathrm{AC}^2$ consumes 32 cells of degrees of freedom for the forward computation. Accordingly, in the solution of our model, $\mathrm{DoF}^+ = 48 - 32 = 16$ and $\mathrm{DoF}^- = 4$. This indicates that the values of $\#\mathrm{SB}^5[\mathcal{B}^{\mathrm{ENC}}]$ are restricted to a subset $\mathbb{X}$ of $\mathbb{F}_{2^8}^{48}$ with $2^{8\times 16}$ elements, and the values of $\#\mathrm{SB}^5[\mathcal{R}^{\mathrm{ENC}}]$ are restricted to a subset $\mathbb{Y}$ of $\mathbb{F}_{2^8}^{4}$ with $2^{8\times 4}$ elements. To be more concrete, $\mathbb{X}$ and $\mathbb{Y}$ should be chosen such that the forward computation is irrelevant of $\#\mathrm{SB}^5[\mathcal{R}^{\mathrm{ENC}}]$ and the backward computation is irrelevant of $\#\mathrm{SB}^5[\mathcal{B}^{\mathrm{ENC}}]$. In summary, the decisive parameters for the obtained attack is $(\mathrm{DoF}^+, \mathrm{DoF}^-, \mathrm{DoM}) = (16, 4, 8)$. The procedure of the preimage attack on full Haraka-512 v2 is as follows (let $\mathrm{mD} = \min\{\mathrm{DoF}^+, \mathrm{DoF}^-, \mathrm{DoM}\}$).

1. Random sample without replacement for values of 32 bytes that will be the pre-determined impacts from blue cells in $\#\mathrm{AC}^2$ on red cells in $\#\mathrm{MC}^2$ (marked by `C`). Obtain $(2^8)^{\mathrm{mD}}$ values for neutral bytes in state $\#\mathrm{AC}^2$ for forward chunk fulfilling the pre-determined impacts (similar to Fig. 1c).
2. Random sample without replacement for values of the 12 gray cells in $\#\mathrm{SB}^5$.
3. Forward computation: for each of the $(2^8)^{\mathrm{mD}}$ values of neutral bytes for forward chunk in state $\#\mathrm{AC}^2$, compute forward to $\#\mathrm{MC}^8$, obtain a candidate value of 8 bytes (one byte for each of the column 4, 5, 6, 7, 12, 13, 14, 15) for matching (similar to Fig. 1b for each column), and store in table L^+.
4. Backward computation: for each of the $(2^8)^{\mathrm{mD}}$ values of neutral bytes for backward chunk in state $\#\mathrm{SB}^5$, compute backward to $\#\mathrm{AC}^8$, obtain a candidate value of 8 bytes for matching (similar to Fig. 1b for each column), and store in table L^-.
5. Find matches between L^+ and L^-. For each match of partial state, tested for full-state matching (except those 32 bytes illustrated in hatched pattern which are free of choice). If no match left, repeat from Step 2 (or Step 1 if candidate values for gray cells are exhaustively sampled).

Complexity. From Eq. 1, the time complexity of this preimage attack on full Haraka-512 v2 is $2^{256-8\times\min\{\mathrm{DoF}^+,\ \mathrm{DoF}^-,\ \mathrm{DoM}\}} = 2^{224}$. In the multi-target setting, given 2^t targets, it is $2^{256-8\times\min\{\mathrm{DoF}^+,\ \mathrm{DoF}^-+t/8,\ \mathrm{DoM}\}} = 2^{224-\min\{t,\ 32\}}$.

Note that our attacks on Haraka v2 do not directly break the security of SPHINCS$^+$-Haraka and Gravity-SPHINCS-Haraka. For SPHINCS$^+$-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 full version (5-round) and the extended version (5.5-round) of Haraka-512 v2, 8-round AES-128, 9-round AES-256, and 9-round Rijndael-256 hashing modes, and 4.5-round Haraka-256 v2. These results are possible due to two important factors: Firstly, the direct modeling of the attack into MILP already covers a larger set of attack configurations than before; Secondly, we generalize the model especially for the components of `AddRoundKey` and `MixColumns` operations, these further enlarge the configuration space covered by the new model.

References

1. 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.

2. R. AlTawy and A. M. Youssef. Second Preimage Analysis of Whirlwind. In D. Lin, M. Yung, and J. Zhou, editors, *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*, volume 8957 of *LNCS*, pages 311–328. Springer, 2014.
3. 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.
4. 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.
5. 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.
6. 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`.
7. 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.
8. S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo. GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption. In W. Fischer and N. Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, Heidelberg, Sept. 2017.
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. W. Diffie and M. E. Hellman. Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *IEEE Computer*, 10(6):74–84, 1977.
15. 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.
16. K. Fu, M. Wang, Y. Guo, S. Sun, and L. Hu. MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck. In T. Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 268–288. Springer, Heidelberg, Mar. 2016.
17. P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. Grøstl – 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. 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.
21. 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`.
22. 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, Geneve, Switzerland, October, 2010.
23. 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.
24. 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.
25. 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.
26. 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.
27. S. Kölbl, 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`.
28. S. Kölbl, 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.
29. G. Leurent. MD4 is Not One-Way. In K. Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 412–428. Springer, Heidelberg, Feb. 2008.
30. 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.
31. Z. Li, W. Bi, X. Dong, and X. Wang. Improved Conditional Cube Attacks on Keccak Keyed Modes with MILP Method. In T. Takagi and T. Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 99–127. Springer, Heidelberg, Dec. 2017.
32. Z. Li, X. Dong, W. Bi, K. Jia, X. Wang, and W. Meier. New Conditional Cube Attack on Keccak Keyed Modes. *IACR Trans. Symm. Cryptol.*, 2019(2):94–124, 2019.
33. G. Liu, M. Ghosh, and L. Song. Security Analysis of SKINNY under Related-Tweakey Settings (Long Paper). *IACR Trans. Symm. Cryptol.*, 2017(3):37–72, 2017.

34. F. Mendel and V. Rijmen. Weaknesses in the HAS-V Compression Function. In K.-H. Nam and G. Rhee, editors, *ICISC 07*, volume 4817 of *LNCS*, pages 335–345. Springer, Heidelberg, Nov. 2007.
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, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, 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, D. Gerault, P. Lafourcade, Q. Yang, Y. Todo, K. Qiao, and L. Hu. Analysis of AES, SKINNY, and Others with Constraint Programming. *IACR Trans. Symm. Cryptol.*, 2017(1):281–306, 2017.
46. 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.
47. 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.
48. 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.

49. 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.
50. 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.
51. 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.

A Visualization of Attacks

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

A.2 MITM Preimage Attacks on Haraka v2

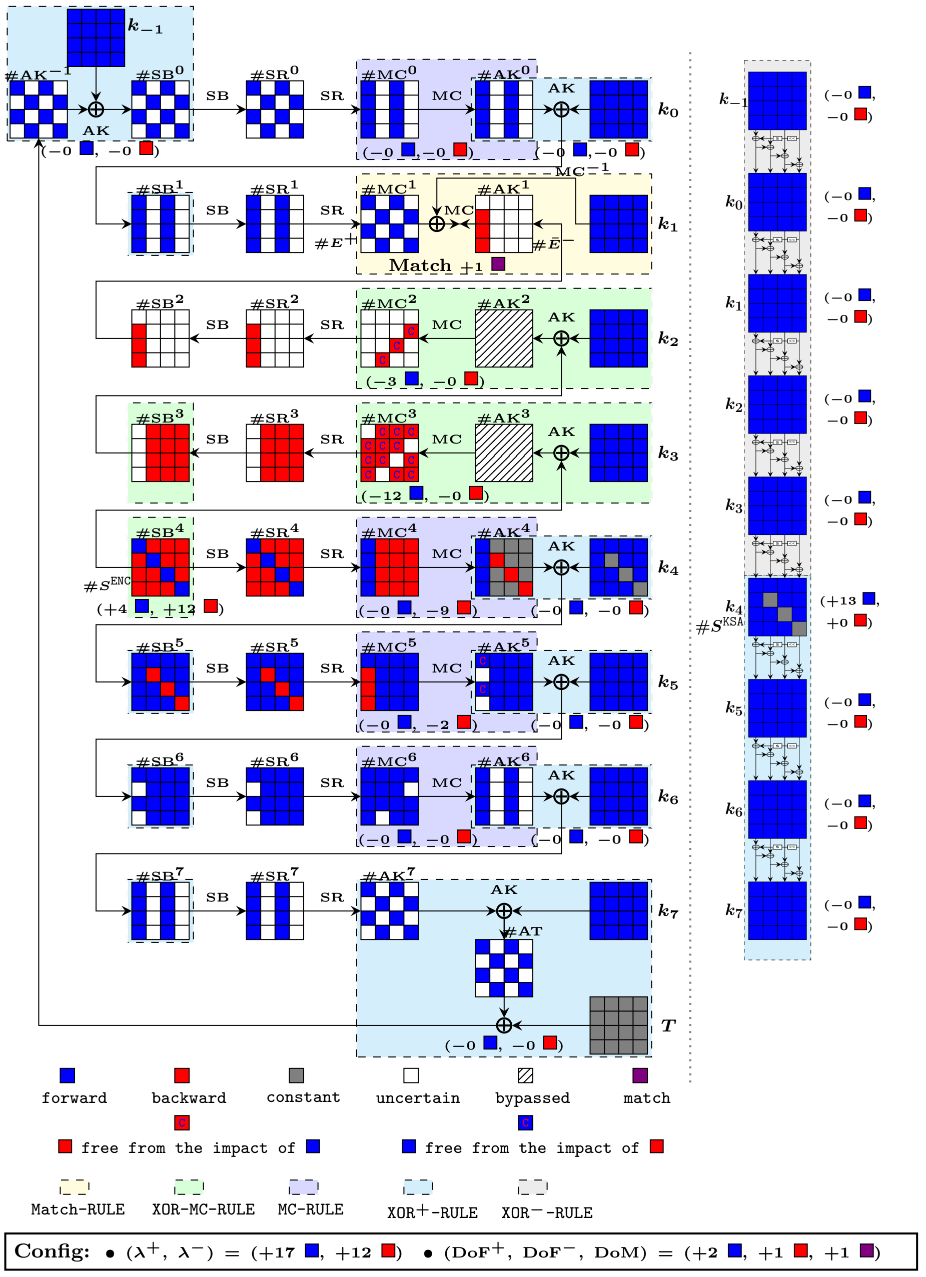


Fig. 7: An MITM pseudo-preimage attack on 8-round AES-128 hashing. Note that, because the use of `XOR-MC-RULE`, we do not introduce any variable in our MILP model for states $\#AK^2$ and $\#AK^3$, and thus we bypass them.

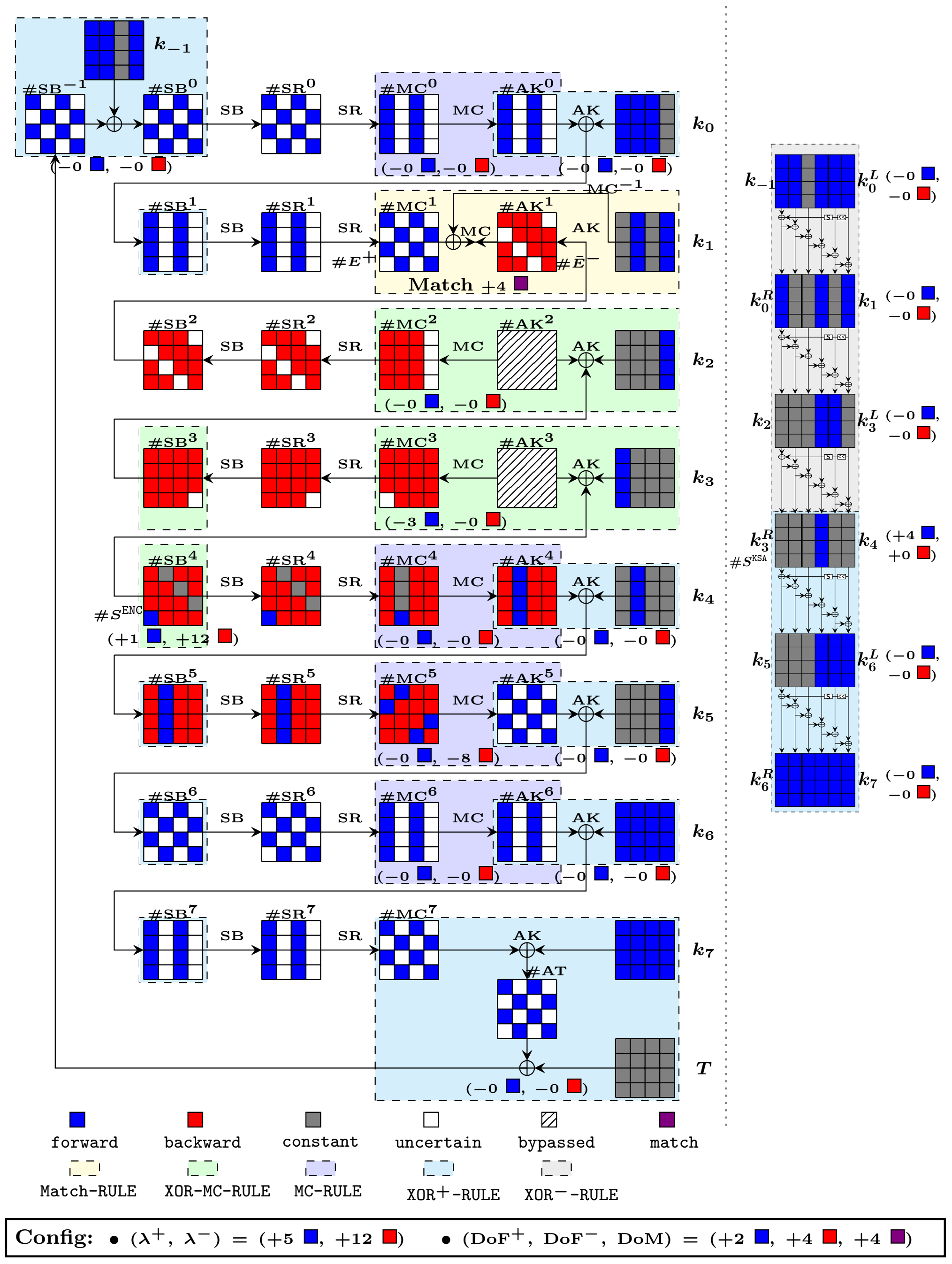
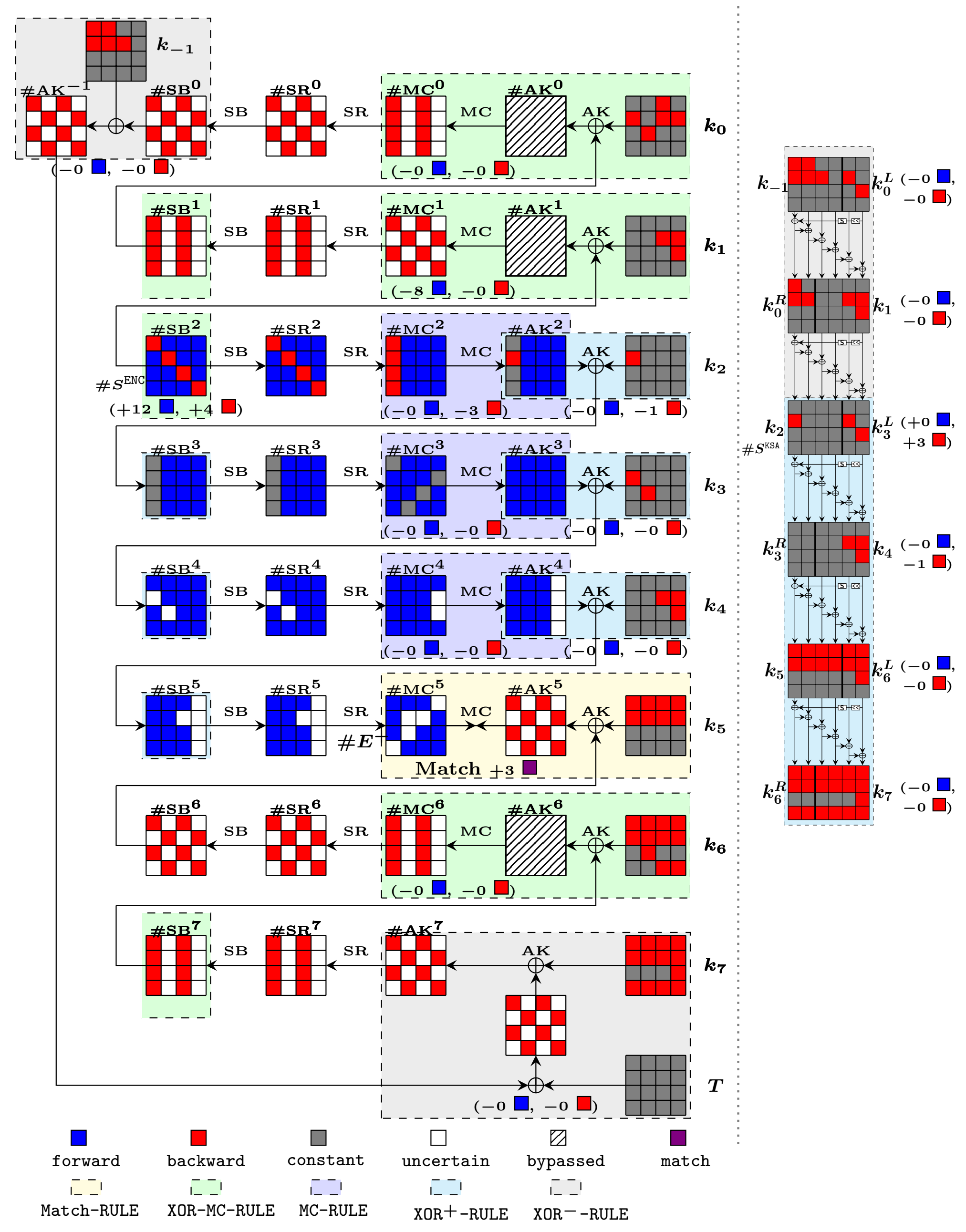


Fig. 8: Example I of the new 8-round preimage attack on AES-192 hashing mode



Config: • $(\lambda^+, \lambda^-) = (+12$ 🟦, $+7$ 🟥) • $(\text{DoF}^+, \text{DoF}^-, \text{DoM}) = (+4$ 🟦, $+2$ 🟥, $+3$ 🟪)

Fig. 9: Example II of the new 8-round preimage attack on AES-192 hashing mode

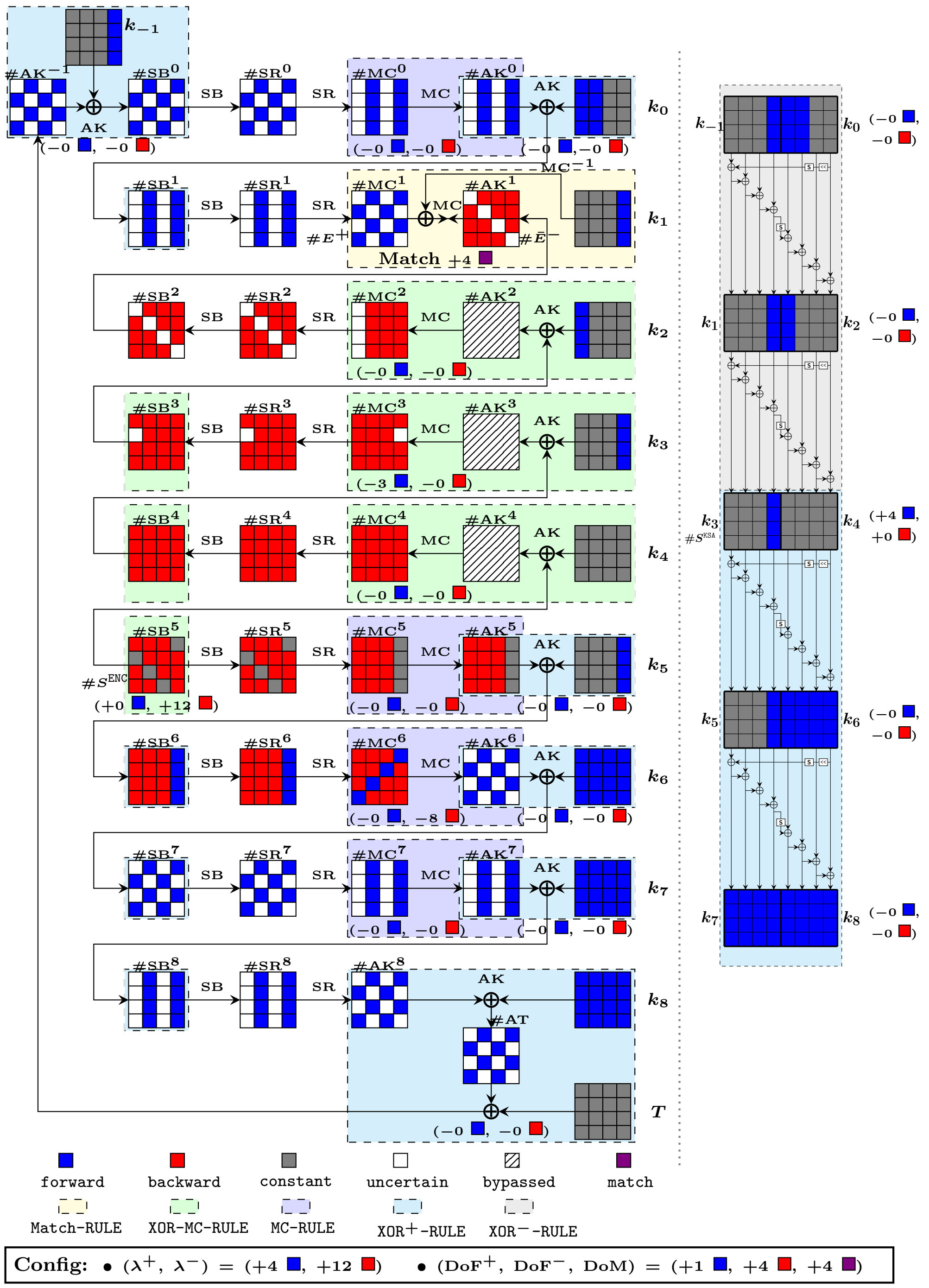
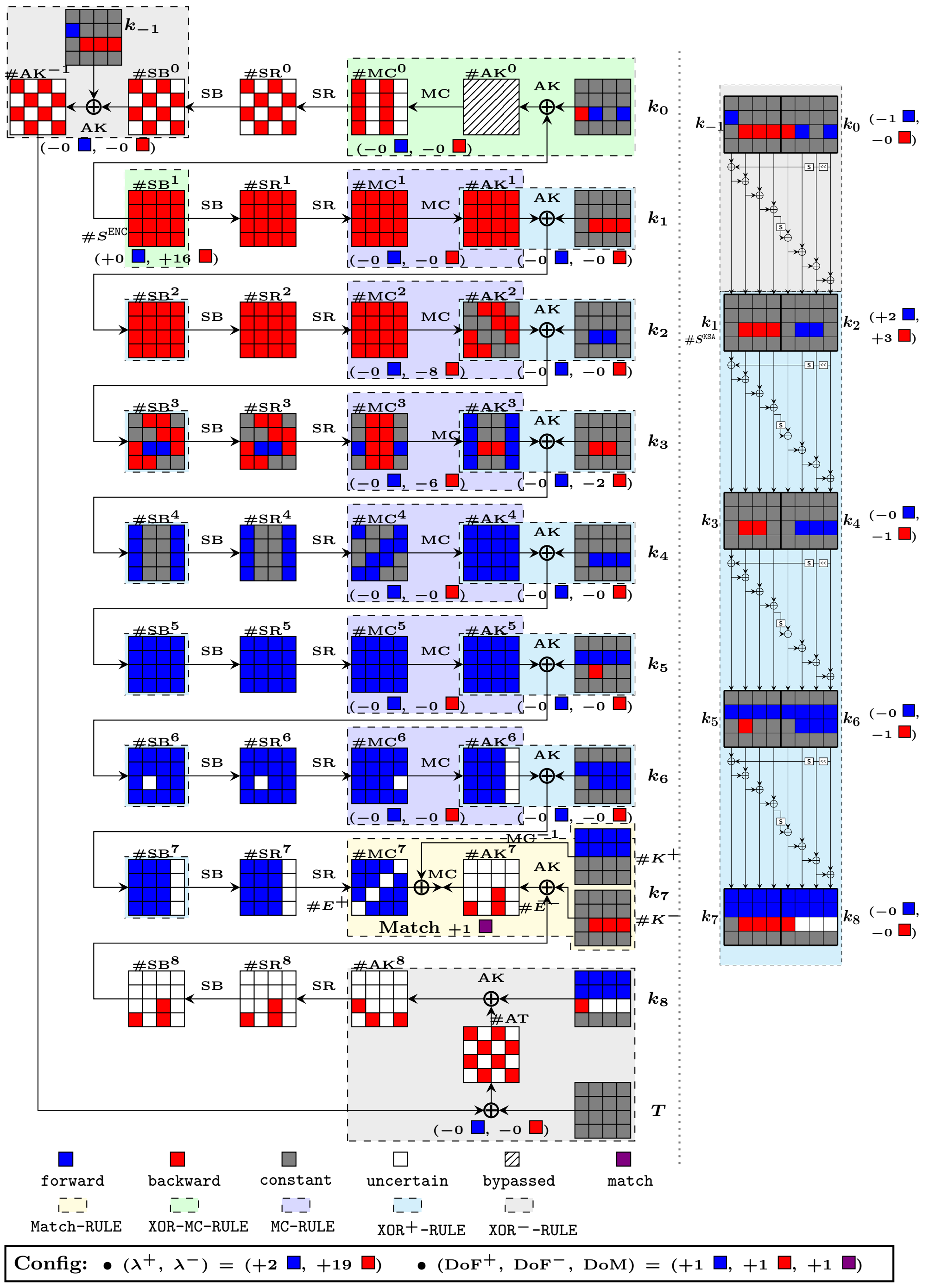


Fig. 10: Example I of the 9-round preimage attack on AES-256 hashing mode



Config: • $(\lambda^+, \lambda^-) = (+2$ 🟦$, +19$ 🟥$)$ • $(\text{DoF}^+, \text{DoF}^-, \text{DoM}) = (+1$ 🟦$, +1$ 🟥$, +1$ 🟪$)$

Fig. 11: Example II of the 9-round preimage attack on AES-256 hashing mode

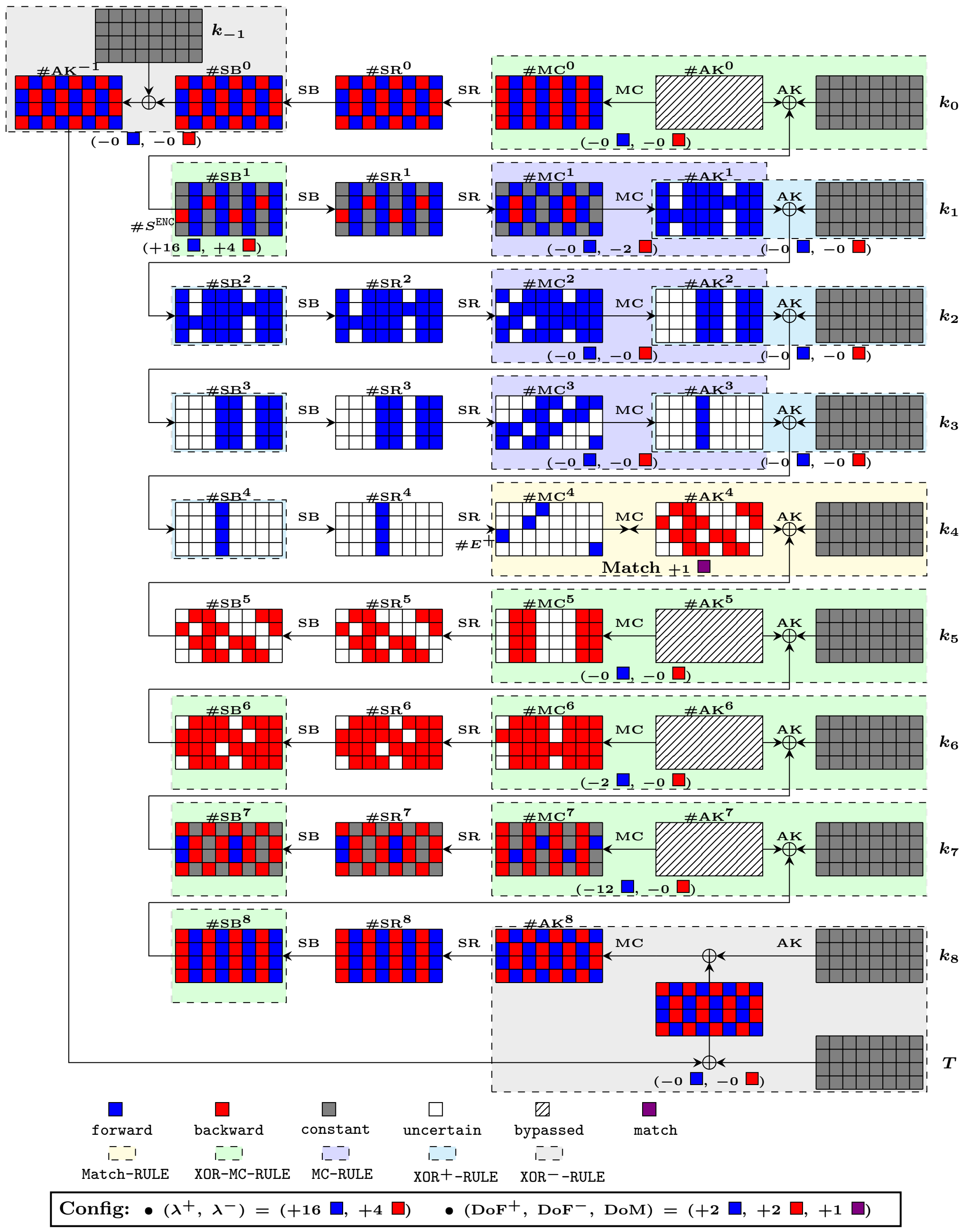
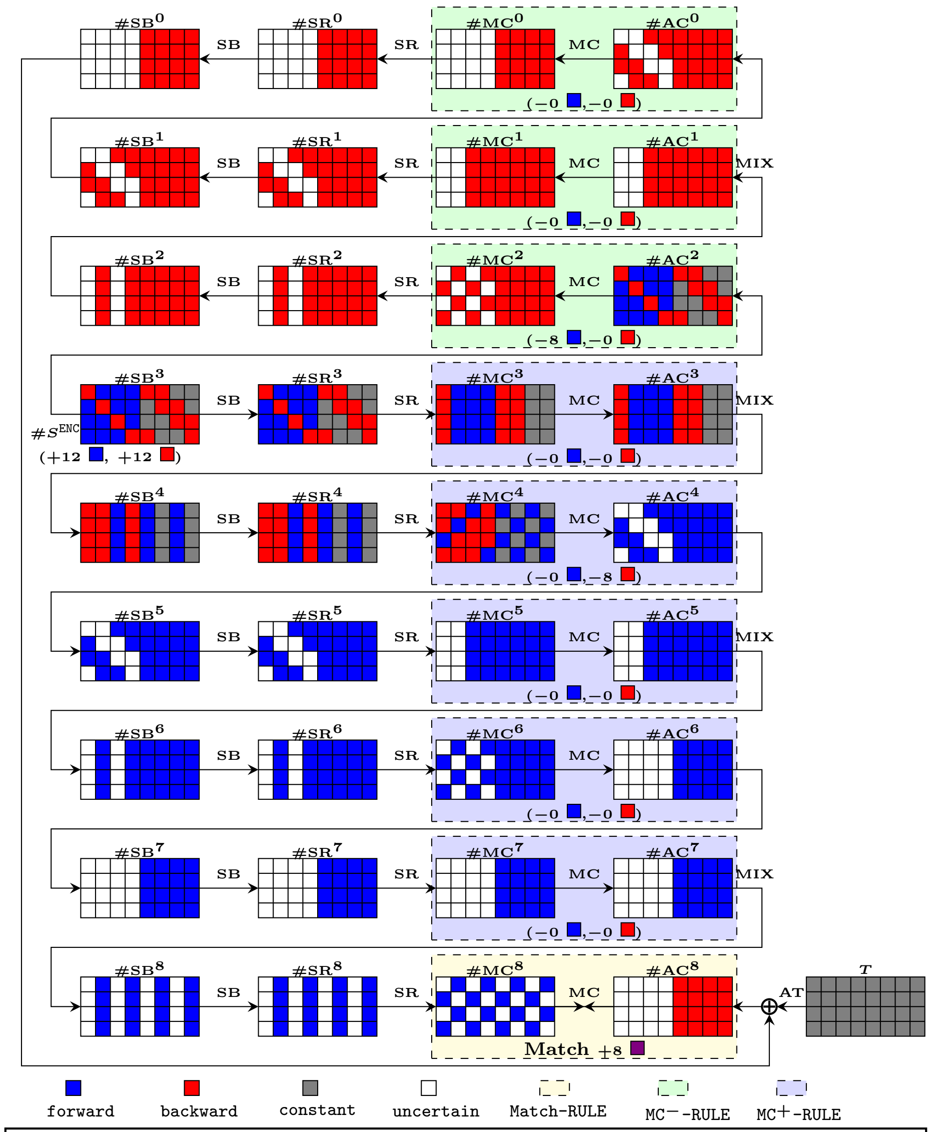


Fig. 12: Example of the 9-round preimage attack on Rijndael-256-128/192/256 hashing mode



#SB⁰ SB #SR⁰ SR #MC⁰ MC #AC⁰
(−0, −0)
#SB¹ SB #SR¹ SR #MC¹ MC #AC¹ MIX
(−0, −0)
#SB² SB #SR² SR #MC² MC #AC²
(−8, −0)
#SB³ SB #SR³ SR #MC³ MC #AC³ MIX
#S^ENC
(+12, +12)
(−0, −0)
#SB⁴ SB #SR⁴ SR #MC⁴ MC #AC⁴
(−0, −8)
#SB⁵ SB #SR⁵ SR #MC⁵ MC #AC⁵ MIX
(−0, −0)
#SB⁶ SB #SR⁶ SR #MC⁶ MC #AC⁶
(−0, −0)
#SB⁷ SB #SR⁷ SR #MC⁷ MC #AC⁷ MIX
(−0, −0)
#SB⁸ SB #SR⁸ SR #MC⁸ MC #AC⁸ AT T
Match +8
forward
backward
constant
uncertain
Match-RULE
MC⁻-RULE
MC⁺-RULE

Config: • $(\lambda^+, \lambda^-) = (+12$ 🟦, $+12$ 🟥$)$ • $(\text{DoF}^+, \text{DoF}^-, \text{DoM}) = (+4$ 🟦, $+4$ 🟥, $+8$ 🟪$)$

Fig. 13: Meet-in-the-middle attack on 4.5-round (or 9-AES-round) `Haraka-256 v2` (matching at the last round).

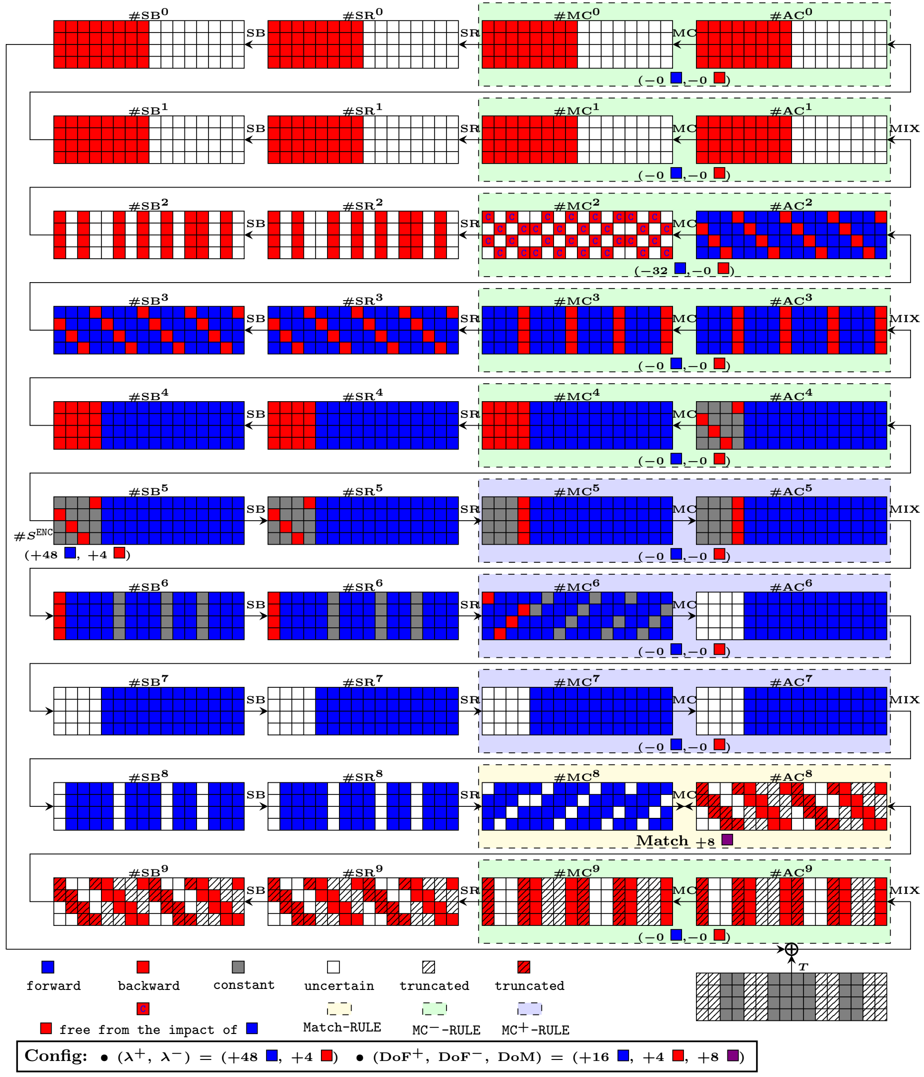


Fig. 14: An MITM preimage attack on full Haraka-512 v2. Note that in our MILP-models, the position of the used hash bits are treated and used as constant in gray cell of the target T, and the bits discarded are treated as 'uncertain' although we distinct them using hatched pattern. However, in the attack procedure, the discarded bits are free of choice such that the state cells in hatched pattern are free of matching.

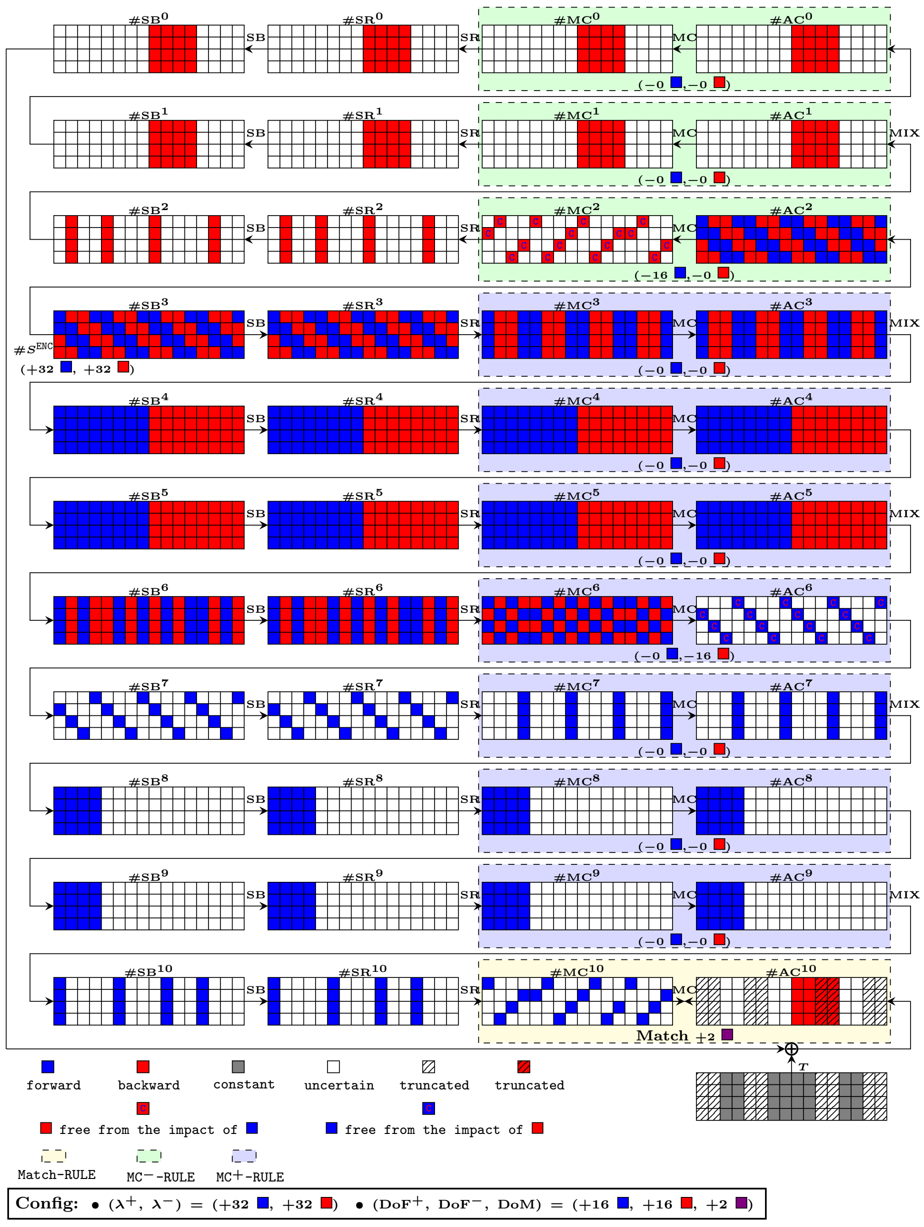


Fig. 15: An MITM preimage attack on the extended 5.5-round (11 AES-rounds) Haraka-512 v2. Note that in our MILP-models, the position of the used hash bits are treated and used as constant in gray cell of the target T, and the bits discarded are treated as 'uncertain' although we distinct them using hatched 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.

An MITM Preimage Attack on the Extended 5.5-round Haraka-512 v2 From Fig. 15, it can be seen that in the searching of our model, the starting state is $\#\mathrm{SB}^3$, and the ending states are $\#\mathrm{MC}^{10}$ and $\#\mathrm{AC}^{10}$. Also, we have $\mathcal{B}^{\mathrm{ENC}} = [16 \cdot i + j \mid i \in \{0,1,2,3\},\ j \in \{0,1,5,6,10,11,12,15\}]$, $\mathcal{R}^{\mathrm{ENC}} = [16 \cdot i + j \mid i \in \{0,1,2,3\},\ j \in \{2,3,4,7,8,9,13,14\}]$, $\mathcal{C} = [16 \cdot i + j \mid i \in \{0,3\},\ j \in \{0,7,10,13\}] \cup [16 \cdot i + j \mid i \in \{1,2\},\ j \in \{1,4,11,14\}]$, and $\mathcal{D} = [32, 33, \ldots, 39]$. 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 $\mathrm{DoM} = (1+4-4) \times 2 = 2$. The `MC⁻-RULE` applied to states $\#\mathrm{MC}^2$ and $\#\mathrm{AC}^2$ consumes 16 cells of degrees of freedom for the forward computation. And the `MC⁺-RULE` applied to states $\#\mathrm{MC}^6$ and $\#\mathrm{AC}^6$ consumes 16 cells of degrees of freedom for the backward computation. Accordingly, in the solution of our model, $\mathrm{DoF}^+ = 32 - 16 = 16$ and $\mathrm{DoF}^- = 32 - 16 = 16$. This indicates that the values of $\#\mathrm{SB}^3[\mathcal{B}^{\mathrm{ENC}}]$ are restricted to a subset $\mathbb{X}$ of $\mathbb{F}_{2^8}^{32}$ with $2^{8\times 16}$ elements, and the values of $\#\mathrm{SB}^3[\mathcal{R}^{\mathrm{ENC}}]$ 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 $\#\mathrm{SB}^3[\mathcal{R}^{\mathrm{ENC}}]$ and the backward computation is irrelevant of $\#\mathrm{SB}^3[\mathcal{B}^{\mathrm{ENC}}]$. In summary, the decisive parameters for the obtained attack is $(\mathrm{DoF}^+, \mathrm{DoF}^-, \mathrm{DoM}) = (16, 16, 2)$.

The procedure of the preimage attack on the extended 5.5-round Haraka-512 v2 is given in the following (let $\mathrm{mDoF} = \min\{\mathrm{DoF}^+, \mathrm{DoF}^-, \mathrm{DoM}\}$):

1. Random sample without replacement for values of 16 bytes that will be the pre-determined impacts from blue cells in $\#\mathrm{AC}^2$ on red cells in $\#\mathrm{MC}^2$ (marked by `C`). Obtain $(2^8)^{\mathrm{mDoF}}$ values for neutral bytes in state $\#\mathrm{AC}^2$ for forward chunk fulfilling the pre-determined impacts (similar to Fig. 1c).
2. Random sample without replacement for values of 16 bytes that will be the pre-determined impacts from red cells in $\#\mathrm{MC}^6$ on blue cells in $\#\mathrm{AC}^6$ (marked by `C`). Obtain $(2^8)^{\mathrm{mDoF}}$ values for neutral bytes in state $\#\mathrm{MC}^6$ for backward chunk fulfilling the pre-determined impacts (similar to Fig. 1c).
3. Forward computation: for each of the $(2^8)^{\mathrm{mDoF}}$ values of neutral bytes for forward chunk in state $\#\mathrm{AC}^2$, compute forward to $\#\mathrm{MC}^{10}$, obtain a candidate value of 2 bytes (byte 33 and 36) for matching, and store in table L^+.
4. Backward computation: for each of the $(2^8)^{\mathrm{mDoF}}$ values of neutral bytes for backward chunk in state $\#\mathrm{MC}^6$, compute backward to $\#\mathrm{AC}^{10}$, obtain a candidate value of 2 bytes for matching (corresponding to byte 33 and 36 after the inverse of `MixColumns`), and store in table L^-.
5. Find matches between L^+ and L^-. For each match of partial state, tested for full-state matching (except those 32 bytes illustrated in hatched pattern which are free of choice). If no match left, repeat from Step 2 (or Step 1 if candidate values of 16-byte impacts `C` are exhaustively sampled).

Complexity. Applying Eq. 1, one have that the time complexity of this attack on extended 5.5-round (11 AES-rounds) Haraka-512 v2 is $2^{256-8\times\min\{\mathrm{DoF}^+,\ \mathrm{DoF}^-,\ \mathrm{DoM}\}} = 2^{256-8\times\min\{16,\ 16,\ 2\}} = 2^{240}$.

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

B.1 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 $\mathtt{C}_{1,0}, \mathtt{C}_{1,1}, \ldots, \mathtt{C}_{1,11}$, $\mathtt{C}_{2,0}, \mathtt{C}_{2,1}, \mathtt{C}_{2,2}$, and $\mathtt{C}_{3,0}, \mathtt{C}_{3,1}, \mathtt{C}_{3,2}$ are pre-determined constants:

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

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

$$\begin{bmatrix} k_4[\ 5] \\ k_4[10] \\ k_4[15] \end{bmatrix} = \begin{bmatrix} k_3[\ 1] \oplus\ \mathrm{S}_{\mathrm{RD}}(k_3[14]) \oplus k_3[\ 5] \\ k_3[\ 2] \oplus\ \mathrm{S}_{\mathrm{RD}}(k_3[15]) \oplus k_3[\ 6] \oplus k_3[10] \\ k_3[\ 3] \oplus\ \mathrm{S}_{\mathrm{RD}}(k_3[\ 0]) \oplus k_3[\ 7] \oplus k_3[11] \oplus k_3[15] \end{bmatrix} = \begin{bmatrix} \mathtt{C}_{3,0} \\ \mathtt{C}_{3,1} \\ \mathtt{C}_{3,2} \end{bmatrix}$$

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

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

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

$$\left[\begin{array}{cccccccccccccccccccc}
\mathtt{9}&\mathtt{e}&\mathtt{b}&\mathtt{d}&0&0&0&0&0&0&0&0&0&0&0&0&\mathtt{9}&0&0&0\\
\mathtt{d}&\mathtt{9}&\mathtt{e}&\mathtt{b}&0&0&0&0&0&0&0&0&0&0&0&0&\mathtt{d}&0&0&0\\
\mathtt{b}&\mathtt{d}&\mathtt{9}&\mathtt{e}&0&0&0&0&0&0&0&0&0&0&0&0&\mathtt{b}&0&0&0\\
0&0&0&0&\mathtt{e}&\mathtt{b}&\mathtt{d}&\mathtt{9}&0&0&0&0&0&0&0&0&0&\mathtt{b}&0&0\\
0&0&0&0&\mathtt{9}&\mathtt{e}&\mathtt{b}&\mathtt{d}&0&0&0&0&0&0&0&0&0&\mathtt{e}&0&0\\
0&0&0&0&\mathtt{d}&\mathtt{9}&\mathtt{e}&\mathtt{b}&0&0&0&0&0&0&0&0&0&\mathtt{9}&0&0\\
0&0&0&0&0&0&0&0&\mathtt{e}&\mathtt{b}&\mathtt{d}&\mathtt{9}&0&0&0&0&0&0&\mathtt{d}&0\\
0&0&0&0&0&0&0&0&\mathtt{9}&\mathtt{e}&\mathtt{b}&\mathtt{d}&0&0&0&0&0&0&\mathtt{b}&0\\
0&0&0&0&0&0&0&0&\mathtt{b}&\mathtt{d}&\mathtt{9}&\mathtt{e}&0&0&0&0&0&0&\mathtt{9}&0\\
0&0&0&0&0&0&0&0&0&0&0&0&\mathtt{e}&\mathtt{b}&\mathtt{d}&\mathtt{9}&0&0&0&\mathtt{9}\\
0&0&0&0&0&0&0&0&0&0&0&0&\mathtt{d}&\mathtt{9}&\mathtt{e}&\mathtt{b}&0&0&0&\mathtt{b}\\
0&0&0&0&0&0&0&0&0&0&0&0&\mathtt{b}&\mathtt{d}&\mathtt{9}&\mathtt{e}&0&0&0&\mathtt{e}\\
\mathtt{b}&\mathtt{d}&\mathtt{9}&\mathtt{e}&\mathtt{b}&\mathtt{d}&\mathtt{9}&\mathtt{e}&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&\mathtt{d}&\mathtt{9}&\mathtt{e}&\mathtt{b}&\mathtt{d}&\mathtt{9}&\mathtt{e}&\mathtt{b}&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&\mathtt{9}&\mathtt{e}&\mathtt{b}&\mathtt{d}&\mathtt{9}&\mathtt{e}&\mathtt{b}&\mathtt{d}&0&0&0&0
\end{array}\right] \times \begin{bmatrix} 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]\\ \#\mathrm{SB}^4[\ 0]\\ \#\mathrm{SB}^4[\ 5]\\ \#\mathrm{SB}^4[10]\\ \#\mathrm{SB}^4[15] \end{bmatrix} = \begin{bmatrix} \mathtt{C}_{1,0}\\ \mathtt{C}_{1,1}\\ \mathtt{C}_{1,2}\\ \mathtt{C}_{1,3}\\ \mathtt{C}_{1,4}\\ \mathtt{C}_{1,5}\\ \mathtt{C}_{1,6}\\ \mathtt{C}_{1,7}\\ \mathtt{C}_{1,8}\\ \mathtt{C}_{1,9}\\ \mathtt{C}_{1,10}\\ \mathtt{C}_{1,11}\\ \mathtt{C}_{2,0}\\ \mathtt{C}_{2,1}\\ \mathtt{C}_{2,2} \end{bmatrix} \qquad (12)$$

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

Solving the equations. Straightforwardly, it seems that the system of equations (Eq. 11) must be solved 2^{112} times (for 2^{112} trials on different constants). However, we show one can keep the amortized complexity of solving these equations be less than 2^8 per trial by a precomputation with negligible complexity.

In our attack, we fix the values of $\mathtt{C}_{1,0}, \mathtt{C}_{1,1}, \ldots, \mathtt{C}_{1,11}$, and $\mathtt{C}_{2,0}, \mathtt{C}_{2,1}, \mathtt{C}_{2,2}$ during the 2^{112} times of iterations (note that the attack is expected to require $2^{14\times 8}$ distinct values of constants, which can be obtained by enumerating $\mathtt{C}_{3,0}, \cdots, \mathtt{C}_{3,2}, \mathtt{C}_{4,0}, \mathtt{C}_{4,1}, \mathtt{C}_{5,0}, \cdots, \mathtt{C}_{5,8}$). In this way, we can build a lookup table (denote by $\mathcal{T}$) for the solutions indexed by the values of $(\mathtt{C}_{3,0}, \mathtt{C}_{3,1}, \mathtt{C}_{3,2})$ by precomputation, and reuse this table among different loops on different constants.

Concretely, we can first obtain the 2^{40} solutions determined by the first 15 linear equations in Eq. 12. That can be done by either first considering each column separately and then merging them together, or using Gaussian elimination to solve the linear equations. Then, for each solution, we compute the left-hand side of the last three non-linear equations in Eq. 11, and obtain a three-byte value. We store the solution to the entry of $\mathcal{T}$ indexed by the three-byte value. These three-byte values are the possible values of $(\mathtt{C}_{3,0}, \mathtt{C}_{3,1}, \mathtt{C}_{3,2})$. As a consequence, after the precomputation, given an arbitrary value of $(\mathtt{C}_{3,0}, \mathtt{C}_{3,1}, \mathtt{C}_{3,2})$, by looking up the table $\mathcal{T}$, we can immediately obtain the solutions of Eq. 11. In this way, the complexity of solving the equations is negligible.

Note that, in the attack, we only need to use 2^8 out of the 2^{16} solutions for the forward computation. That is because the degree of freedom for backward is

8-bit, which is the limitation of the maximum degree of freedom we use. Thus, we can fix one of the free byte-variables $\#\mathrm{SB}^4[0], \#\mathrm{SB}^4[5], \#\mathrm{SB}^4[10], \#\mathrm{SB}^4[15]$ and solve the first 15 linear equations to obtain 2^{32} solutions only. And for each value of $(\mathtt{C}_{3,0}, \mathtt{C}_{3,1}, \mathtt{C}_{3,2})$, we can obtain 2^8 solutions.

B.2 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 $\#\mathrm{MC}^5$ to make the neutral bytes for backward chunk has two-byte constant influence on state $\#\mathrm{AK}^5$. Such constraint has been used in many previous attacks)

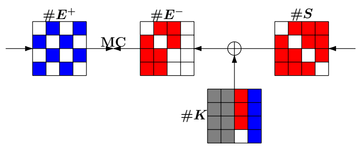
$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} 0 \\ \#\mathrm{MC}^5[1] \\ \#\mathrm{MC}^5[2] \\ \#\mathrm{MC}^5[3] \end{bmatrix} = \begin{bmatrix} \mathtt{C}_{4,0} \\ - \\ \mathtt{C}_{4,1} \\ - \end{bmatrix}, \text{ i.e., } \begin{bmatrix} 3 \cdot \#\mathrm{MC}^5[1] \oplus 1 \cdot \#\mathrm{MC}^5[2] \oplus 1 \cdot \#\mathrm{MC}^5[3] \\ 1 \cdot \#\mathrm{MC}^5[1] \oplus 2 \cdot \#\mathrm{MC}^5[2] \oplus 3 \cdot \#\mathrm{MC}^5[3] \end{bmatrix} = \begin{bmatrix} \mathtt{C}_{4,0} \\ \mathtt{C}_{4,1} \end{bmatrix}$$

Because of the property of `MixColumns`, this equation has $2^{(3-2)\times 8}$ solutions for each possible values of $\mathtt{C}_{4,0}$ and $\mathtt{C}_{4,1}$. Thus, given $\mathtt{C}_{4,0}$, $\mathtt{C}_{4,1}$ 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 for all possible values $\mathtt{C}_{4,0}$ and $\mathtt{C}_{4,1}$ and store in a table to reuse in the attack.

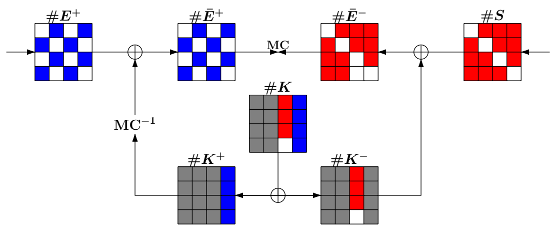
C Additional Techniques for MITM Preimage Attacks

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^{ℓ} ($\ell < n - 2$) 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 [29], 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^{ℓ} 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 [7, 9, 29].

Tricks for matching the ending states as indirect matching and matching through MixColumns used in [3, 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. 16a.



(a) The sub-key addition at the boundary of the matching states



(b) Equivalent transformation of the sub-key addition at the boundary of the matching states

Fig. 16: Tricks for matching the ending states

Therefore, to perform matching, one can decompose state as $\#K = \#K^{+} + \#K^{-}$, and translate the computation in Fig. 16a into its equivalent form shown in Fig. 16b, since $\mathrm{MC}(\#E^{+}) \oplus \#K = \mathrm{MC}(\#E^{+} \oplus \mathrm{MC}^{-1}(\#K^{+})) \oplus \#K^{-}$.

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 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. 16a are preserved by decomposing $\#K$ as presented in Fig. 16b.