MITM Meets Guess-and-Determine: Further Improved Preimage Attacks against AES-like Hashing

Zhenzhen Bao[1], Jian Guo[1], Danping Shi[2,3], and Yi Tu[1]

[1] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore.
{zzbao,guojian}@ntu.edu.sg, TUYI0002@e.ntu.edu.sg

[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. shidanping@iie.ac.cn

[3] University of Chinese Academy of Sciences, Beijing, China.

Abstract. Since the Meet-in-the-Middle preimage attack against 7-round AES hashing was found by Sasaki in 2011, the development of this research direction has never been stopped. In 2019, Bao *et al.* found the degree of freedom from the message (or the key of the underlying block cipher) were useful, before the Mixed-Integer-Linear-Programming (MILP) modeling was introduced to find the optimal attack configurations in 2020. In this paper, we move one step further in this research direction by introducing more techniques such as guess-and-determine, round independence, and symmetry etc. to the MILP search model. To demonstrate the power of the enhanced model, we apply it to the popular AES-like hash functions Whirlpool, Grøstl, and AES hashing modes, and obtain general improvements over the existing best (pseudo-)preimage attacks. In particular, the number of attacked rounds on Whirlpool and AES-256 hashing modes is extended from 6 to 7 and 9 to 10, respectively. Time complexity improvements are also obtained on variants of lesser rounds, as well as the 6-round Grøstl-256 and the 8-round Grøstl-512. Computer experiments on trial versions of the full attack procedure have confirmed the correctness of our results.

Keywords: Whirlpool, Grøstl, AES hashing modes, hash function, MITM, Preimage, Automatic search, Guess and Determine, MILP

1 Introduction

Hash function is a function mapping a document of arbitrary length into a short fixed-length digest. For a cryptographically secure hash function, it should fulfill three basic security requirements: collision resistance, preimage resistance, and second-preimage resistance. In this paper, we focus on the security notation of preimage resistance, i.e., it should be computationally difficult to invert the function. Specially, for an ideal hash function H with n-bit digest and a target T given at random, it should cost no less than 2^n compression function evaluations to find an input x such that $H(x) = T$. Preimage attack refers to an algorithm achieving this in lesser evaluations.

Traditionally, there are two common methods to construct cryptographic hash functions. One is to convert from block ciphers through mode of operations, and the other is to build from scratch. There are 12 secure PGV modes [20], which enjoy the proof of security reduction of the hash function to the underlying block cipher. This method is especially useful when a block cipher like AES [8] has long-standing security against intensive cryptanalysis. When it is already implemented for other purposes like encryption, the same implementation can be re-used to construct a hash function by implementing the additional mode only. In this way it also leads to performance merits. Particularly, hash function constructed from AES through PGV modes are called AES hashing, and they have been standardized by Zigbee [1] and also suggested by ISO [17]. Due to the well understood security and software and hardware efficiencies, many dedicated block ciphers and hash functions built from scratch follow similar design strategy by using an AES-like round function, such as Whirlpool [6], Grøstl [11], PHOTON [13], and LED [14] etc.

The MITM Preimage Attacks. The Meet-in-the-Middle attack in general has a long history application to cryptanalysis of ciphers. It was then developed into a MITM preimage attack against hash functions in 2008 by Sasaki *et al.* [3,23]. This attack was found to be powerful against many hash functions, and broke the full version of MD4 [12], MD5 [24], Tiger [12,27], HAVAL [15,23], as well as lightweight block cipher KTANTAN [7,28]. The basic idea of the attack is to split the cipher into two chunks, and find the so-called neutral bits from each side, which are independent from the computation of (part of) the state of the other side. Hence, the two chunks can be computed independently, then the (partial) preimage finding problem can be converted into a birthday-like problem.

In 2011, Sasaki [22] for the first time applied the MITM preimage attack to AES hashing reduced to 7 rounds. To avoid dealing with the key schedule, the key value of AES was pre-set to a constant, and hence the same number of rounds was attacked for AES hashing based on all three versions of AES (AES-128, AES-192, and AES-256). In 2019, Bao *et al.* [4] revisited the problem and found that the degree of freedoms from the key values can be utilized for at least one side of the computation. This observation led to the attack complexity improvements over 7-round AES-128 hashing, and increased the number of attacked rounds from 7 to 8 for the AES-hashing based on AES-192 and AES-256. In 2020, Bao *et al.* [5] further extended the attack to 8-round AES-128 hashing by introducing the Mixed-Integer-Linear-Programming (MILP) into the search of better attack configurations. It was noted that the choices of neutral bits and the number of possibilities of transition in each round through the linear layers such as XOR and MixColumn are too many to carry out a bruteforce search by the traditional search programs.

Based on Bao *et al.*'s work in [5], in this paper we work further towards improving upon the previous results by enhancing the MILP search model and find more applications. This is achieved by enlarging the search space the MILP program covers, and in the meanwhile optimizing the model so that the more complicated MILP program outputs better attack configuration in practical time.

1.1 Enlarging the Search Space

Since Bao *et al.*'s search was complete, it is not possible to find better results without introducing new techniques into the model. In order to enlarge the attack configuration search space, we introduce three different techniques: guess-and-determine, a further relaxed model, and independent handling of neutral bits in linear layers.

Guess-and-Determine. GAD has been a popular technique and has countless applications in cryptanalysis against symmetric-key primitives such as stream ciphers [9, 16, 30] and block ciphers [10, 21] etc. The basic idea is that, in the process of some attack, the gain of guessing some state or key bits is higher than the price, *i.e.*, the guess itself comes with a probability p and the attack needs to repeat at least $1/p$ times in order to have a correct guess. This technique has also been used in the MITM preimage attacks [2, 12, 25, 29], where the guess allows further computation of some more state/message bits which help either extend the attack to more rounds or lower the time complexities. It is noted that this powerful technique has not been incorporated into the MILP models for the MITM preimage attacks by Bao *et al.* [5].

Relaxed Model. We also notice that in the MILP model by Bao *et al.*, although the idea of cancellation of some neutral bits to constants has already appeared, this was only used in one of the two chunks for the connivence of modeling. While it is obvious that more possibilities will be reached by allowing cancellation in both directions, the immediate drawback is that this also contributes to further complicating the MILP model.

Independent Linear Layer. In previous models, once the neutral bits from two directions reach into the same byte position, this byte is considered being *white* as influenced by both directions, and no longer usable in either chunk computation. Note this white byte does not only affect the current round, but also the following rounds propagated through round functions. Trying to *rescue* these white bytes from being wasted, we note the subset of white bytes, which can be expressed as the linear combination of the neutral bits from both directions, can be useful in the final matching phase. Hence, in our enhanced model we pick out and keep track of these linear white bytes.

1.2 Model Optimizations

With the above new techniques introduced into the MILP model, the search space can be significantly enlarged. However, the side effect is, the entire model can become too complicated and as a result the program will not be able output anything in practical time in some applications. Unlike other problems which allow complexity projections, we must wait for the MILP program to output in order to obtain the corresponding new attack configuration. To overcome this issue, we propose the following model optimizations.

Round-Dependent Modeling. We note some techniques are not always necessary in all rounds of the attack. For example, from the analysis of MITM preimage attacks in the literature as well as those found by our enhanced model, we notice the guess-and-determine technique becomes more useful in rounds away from the splitting point. Hence, we manually set this technique to be used in some selected rounds according to our experience, and this elegant human intervene turns out to be surprisingly powerful in shortening the run time of the MILP programs.

Symmetry and Similarity. Thanks to the design nature of AES hashing and many AES-like hash functions, the truncated state and/or message can be viewed as repetitions of a smaller unit. Taking AES hashing for example, the 4x4 state can be viewed as 4 repetitions of a 2x2 state, which is similar to the design rationale behind Mini-AES [19]. Under this assumption, although we lose the configurations not fulfilling this symmetry, the MILP model size is reduced significantly, and our experiments show that such a reduction can also generate attack configurations very close to the optimal ones in practical time for some cases. The same reduction is possible when the round function and message expansion function are similar to each other, *e.g.*, in Whirlpool both follow a AES-like round for a 8x8 state. In such cases, assumption that the state and round message take the same truncated view in every round will have the same effect in reducing the model size as the *symmetry* property above.

1.3 Applications to Whirlpool and Grøstl

To demonstrate the usefulness of our enhanced model, we apply it to the popular AES-like hashing Whirlpool and Grøstl, and obtain broad improvements upon the previous best results. For Whirlpool, out of the total 10 rounds, the number of attacked rounds is improved from 6 to 7, and in the meanwhile complexities for 5- and 6-round attacks are also reduced. For Grøstl, there are two main instances Grøstl-256 and Grøstl-512 named after the size of the digest in bits. We improved the attack complexities for 6-round Grøstl-256 and 8-round Grøstl-512 — the longest attacked variant. Besides, time complexity improvements are also obtained on variants of lesser rounds. Our results together with a comparison against the literature are summarized in Table 1.

Organization. The rest of the paper is organized as follows. Section 2 gives a brief introduction to the AES-like hashing and MITM preimage attacks. Section 3 describes the details of our enhanced MILP model. The application to Whirlpool and Grøstl is given in Section 4 and 5, respectively. Section 6 concludes the paper.

Table 1: Updated results on (pseudo-) preimage attacks on Whirlpool, Grøstl-256, Grøstl-512, and AES-256 hashing modes (with recomputed complexities for previous works)

Cipher (Target)	#R	Time-1	Mem-1	Time-2	Mem-2	Ref.
Whirlpool (Hash)	5/10	2^{416}	2^{96}	2^{448}	2^{96}	[25]
	5/10	2^{416}	$O(1)$	2^{465}	$O(1)$	[25]
	5/10	2^{352}	2^{160}	2^{433}	2^{160}	Fig. 13
	6/10	2^{448}	2^{256}	2^{481}	2^{256}	[25]
	6/10	2^{448}	$O(1)$	2^{504}	$O(1)$	[25]
	6/10	2^{440}	2^{192}	2^{477}	2^{192}	Fig. 12
	7/10	2^{480}	2^{128}	2^{497}	2^{128}	Fig. 10
Grøstl-256 (CF+OT)	5/10	2^{192}	2^{64}	$2^{234.67}$	$2^{213.33}$	$\ast$ [18, 31]
	5/10	2^{184}	2^{72}	2^{232}	2^{208}	Fig. 15, 16
	6/10	2^{240}	2^{64}	2^{252}	2^{252}	$\ast$ [18, 31]
	6/10	2^{224}	2^{128}	$2^{245.33}$	$2^{242.67}$	Fig. 5, 6
Grøstl-512 (CF+OT)	6/14	2^{320}	2^{192}	2^{448}	2^{384}	Fig. 19, 20
	7/14	2^{416}	2^{152}	2^{480}	2^{440}	Fig. 17, 18
	8/14	2^{496}	2^{16}	2^{508}	2^{508}	† [29]
	8/14	2^{472}	2^{120}	2^{504}	2^{504}	† [31]
	8/14	2^{472}	2^{224}	2^{500}	2^{500}	Fig. 7, 8
AES-256 Hasing (Hash)	9/14	$2^{120-min(t,24)}$	2^{8}	2^{123}	2^{8}	[5]
	10/14	2^{120}	2^{8}	2^{125}	2^{8}	Fig. 21

CF: Compression Function; OT: Output Transformation;
Time-1 and Time-2 are complexities of pseudo-preimage and preimage attacks following the notions in [4] when the target is a hash function, and complexities of inverting OT and CF+OT (pseudo-preimage) following the notions in [29] for Grøstl, respectively.
† The presented complexities for the attacks in [29] and [31] are recomputed by removing constant factors (*e.g.*, the cost C_{TL} for lookup table is replaced by 1) and replacing $C_2(2n, b)$ that is lower bounded by $b/2$ in [29] with $\overline{C_2(2n, b)}$ that can be 2^0 considering the amortized complexity. Thus, all complexities are computed follow the same way.
$\ast$ The 5- and 6-round attacks in [18] are on the OT of Grøstl-256. To convert to pseudo-preimage, we used the results for the case with no truncation in [18] . However, for the 6-round attack, in which guessing are required, it cannot be directly used. Thus, we combined the attack on OT in [18] with the best previous attack on the CF in [31].

The attack on AES-256 hashing modes and Some extra details on other attacks and figures are postponed to Appendix.

2 AES-like Hashing and MITM Preimage Attacks

In this section, we give a brief introduction to AES-like hash function in a general way, and describe the Meet-in-the-Middle Preimage Attacks, before we can introduce applications to Whirlpool and Grøstl in the next Sections.

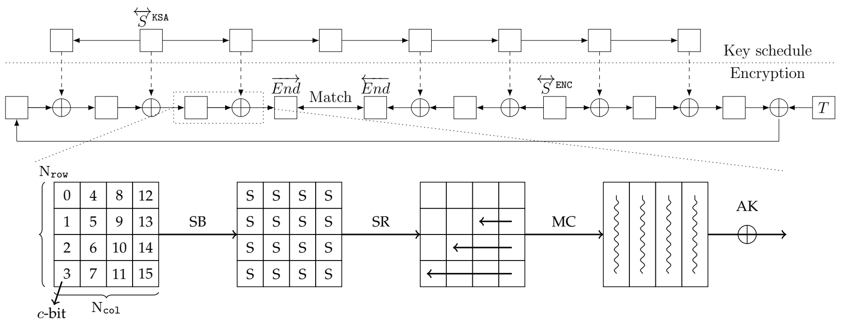


Fig. 1: Overview of AES-like hashing [5]

The AES-like hashing in our context refers to those using an AES-like round function as depicted in Figure 1, where the state can be viewed as a $N_{row} \times N_{col}$ matrix of c-bit cells. There are 4 general operations in order:

- SB applies a non-linear substitution-box operation to each cell.
- SR cyclically shifts each row by a pre-defined number of positions.
- MC mixes every column, *e.g.*, by multiplication of an (MDS) matrix.
- AK adds the round key (or round message).

For some designs, the SR and MC may work on the transpose of the matrix, *i.e.*, SR on columns and MC on rows. The main advantage of AES-like designs is the ease to count the number of active sboxes in the security resistance against differential and linear cryptanalysis.

The MITM Preimage Attacks generally cut the entire encryption process into two chunks and there are few neutral bits from each side, so that the computation (of partial state values) of each chunk can be done independently. The splice-and-cut technique [3] views the input and output of the compression function *connected* through the feedforward operation. There may or may not be any neutral bits from the message of the compression function (or key of the underlying block cipher). The number of attacked round depends on how many rounds the neutral bits can be maintained independent from each other in the computation of the state values, and hence the important part of the attack are: the places where the function is cut and where they meet again, the choices of the neutral bits, and how the neutral bits are propagated through the functions. All these parameters together form an attack configuration, and our work here is to find a (sub-)optimal attack configuration which result in longest number of attacked rounds or lowest possible complexities, through a better way of modeling the parameters and the behavior of the each components of the round function by the language of MILP.

3 MILP Model for the Configuration Search

3.1 Basic Model for MITM

The way how states are encoded and the rule how propagation is formalized corresponds to how to define variables and constraints to generate MILP models for searching attack configurations. We follow those in [5] for the description of the basic model, based on which our enhanced model will be introduced.

Notations and Encoding. Firstly, the attributes of state cells are encoded according to whether they are determined by the neutral bits from one or two directions or none, with two bit variables (x, y).

- `Gray` (▨): the value is predefined constant, thus is known and fixed in both forward and backward chunks; Indicating variables: $(x, y) = (1, 1)$
- `Blue` (🟦): the values are determined by forward neutral bytes and predefined constants, thus is known and active in the forward chunk but unknown in the backward; Indicating variables: $(x, y) = (1, 0)$
- `Red` (🟥): the values are determined by backward neutral bytes and predefined constants, thus is known and active in the backward chunk but unknown in the forward; Indicating variables: $(x, y) = (0, 1)$
- `White` (□): the values are determined by both forward and backward neutral bytes, thus cannot be computed independently in either forward or backward chunks; Indicating variables: $(x, y) = (0, 0)$

For convenience, `Black` (■) is used to represent any of the 4 cells (▨, 🟦, 🟥, □). x and y are defined with the following implications.

– $x = \begin{cases} 1 & \text{can be computed in the forward chunk; is } \texttt{Blue} \text{ or } \texttt{Gray} \\ 0 & \text{cannot be computed in the forward chunk; is } \texttt{Red} \text{ or } \texttt{White} \end{cases}$

– $y = \begin{cases} 1 & \text{can be computed in the backward chunk; is } \texttt{Red} \text{ or } \texttt{Gray} \\ 0 & \text{cannot be computed in the backward chunk; is } \texttt{Blue} \text{ or } \texttt{White} \end{cases}$

Additionally, β and ω are defined as follows.

– $\beta = \begin{cases} 1 & \text{is inactive, do not contain degree of freedom; is } \texttt{Gray} \\ 0 & \text{is active, is } \texttt{Blue}, \texttt{Red} \text{ or } \texttt{White} \end{cases}$

– $\omega = \begin{cases} 1 & \text{cannot be computed in both forward and backward chunks; is } \texttt{White} \\ 0 & \text{can be computed in at least one direction; is } \texttt{Blue}, \texttt{Red}, \text{ or } \texttt{Gray} \end{cases}$

Directly, $\beta = \mathrm{AND}(x, y)$, and $\omega = 1 - \mathrm{OR}(x, y)$.

Searching Framework. After the MITM preimage attack is modeled, the search of valid (optimal) attack configurations corresponds to the search of (optimal) solutions fulfilling all the constraints in the MILP models. The whole search includes many independent search models. Each search model includes four constant parameters, *i.e.*, the total number of rounds ($\texttt{total}_r$), the round index of an initial state in the encryption data-path ($\mathtt{init}_r^{\mathrm{E}}$), the round index of an initial state in the key-schedule data-path ($\mathtt{init}_r^{\mathrm{K}}$), and the round index of two ending states for matching ($\mathtt{match}_r$). For the complete search of $\mathtt{total}_r$-round attack, all possible combinations of values of $\mathtt{init}_r^{\mathrm{E}}$, $\mathtt{init}_r^{\mathrm{K}}$, and $\mathtt{match}_r$ should be tried; each combination corresponds to an independent model; The following description of the modeling method is for an individual model with a fixed ($\mathtt{total}_r$, $\mathtt{init}_r^{\mathrm{E}}$, $\mathtt{init}_r^{\mathrm{K}}$, $\mathtt{match}_r$).

Although the solving of MILP models does not necessarily start from solving constraints in a starting round, the constraints can be viewed as being imposed starting from some initial states (*i.e.*, $\overleftrightarrow{S}^{\mathtt{ENC}}$ and $\overleftrightarrow{S}^{\mathtt{KSA}}$ in round $\mathtt{init}_r^{\mathrm{E}}$ and $\mathtt{init}_r^{\mathrm{K}}$) and terminating at some ending states (*i.e.*, $\overrightarrow{End}$ and $\overleftarrow{End}$ in round $\mathtt{match}_r$).

In the initial state of the encryption data-path ($\overleftrightarrow{S}^{\mathtt{ENC}}$) and the initial state in the key-schedule data-path ($\overleftrightarrow{S}^{\mathtt{KSA}}$), the attribute of each cell is constrained to be one of `Blue` (🟦), `Red` (🟥), and `Gray` (⬛). Thus, the attribute of each cell in $\overleftrightarrow{S}^{\mathtt{ENC}}$ and $\overleftrightarrow{S}^{\mathtt{KSA}}$ has three possible assignments, *i.e.*, $(x_i, y_i) \in \{(1,0),(0,1),(1,1)\}$ for $\forall\ i \in \mathcal{N}$, where $\mathcal{N} = \{0, 1, \cdots, \mathrm{N}_{\mathtt{row}} \cdot \mathrm{N}_{\mathtt{col}} - 1\}$. Thus, the set of cells in state $\overleftrightarrow{S}^{\mathtt{ENC}}$ is a union of a set of `Blue` cells 🟦 (denoted by set of index $\mathcal{BL}^{\mathtt{ENC}}$), a set of `Red` cells 🟥 ($\mathcal{RD}^{\mathtt{ENC}}$), and a set of `Gray` cells ⬛ ($\mathcal{GY}^{\mathtt{ENC}}$). Similarly, the set of cells in state $\overleftrightarrow{S}^{\mathtt{KSA}}$ is a union of a set of `Blue` cells 🟦 ($\mathcal{BL}^{\mathtt{KSA}}$), a set of `Red` cells 🟥 ($\mathcal{RD}^{\mathtt{KSA}}$), and a set of `Gray` cells ⬛ ($\mathcal{GY}^{\mathtt{KSA}}$). Basic relations are that $\mathcal{BL}^{\mathtt{ENC}} \cap \mathcal{RD}^{\mathtt{ENC}} = \emptyset$, $\mathcal{BL}^{\mathtt{KSA}} \cap \mathcal{RD}^{\mathtt{KSA}} = \emptyset$, $\mathcal{GY}^{\mathtt{ENC}} = \mathcal{N} - \mathcal{BL}^{\mathtt{ENC}} \cup \mathcal{RD}^{\mathtt{ENC}}$, $\mathcal{GY}^{\mathtt{KSA}} = \mathcal{N} - \mathcal{BL}^{\mathtt{KSA}} \cup \mathcal{RD}^{\mathtt{KSA}}$; and $|\mathcal{BL}^{\mathtt{ENC}} \cup \mathcal{BL}^{\mathtt{KSA}}| \neq \emptyset$, $|\mathcal{RD}^{\mathtt{ENC}} \cup \mathcal{RD}^{\mathtt{KSA}}| \neq \emptyset$. The states in the starting round are called initial because initial degree of freedoms are all contained in these states. Denote the initial degree of freedom for the forward by $\overrightarrow{\iota}$, and that for backward by $\overleftarrow{\iota}$. Accordingly, one has the equations for $\overrightarrow{\iota}$ and $\overleftarrow{\iota}$ as in Eq. (1).

In the ending states ($\overrightarrow{End}$ and $\overleftarrow{End}$) in the round of matching, for each pair of input/output columns of the states before and after the MixColumns operation, the variable $\overrightarrow{\overleftarrow{m}}_i$ that indicates the degree of matching in column i can be constrained by the numbers of `Blue`, `Red`, and `Gray` cells (🟦, 🟥, ⬛). The total degree of matching of the attack, denoted as $\overrightarrow{\overleftarrow{m}}$, is the sum of the degrees of matching from all columns, as shown in Eq. (3).

In states of all rounds in both encryption and key-schedule data-path, constraints are imposed on attribute-indicating variables of states cells. The constraints indicate the relations of cell-attributes between consecutive states intra-round and inter-round. The change of cell-attributes from state to state is the attribute propagation.

For the attribute of `Blue` and `Red` propagating to the ending states and remaining to be able to make a match, special constraints for indicating whether to consume degree of freedom are imposed from the starting round to the matching round. Concrete ways of attribute propagation and how to preserve possible attribute propagations by consuming degree of freedom will be introduced shortly. Denote the accumulated consumed degree of freedom of forward by $\overrightarrow{\sigma}$ and that of backward by $\overleftarrow{\sigma}$. After the propagation, the essential degree of freedom of forward (denoted by $\overrightarrow{d_b}$) and that of backward (denoted by $\overleftarrow{d_r}$) in the attack are constrained as that in Eq. (2).

The search of a valid attack configuration corresponds to the search of a valid attribute propagation with $\min\{\overrightarrow{d_b}, \overleftarrow{d_r}, \overset{\rightarrow\leftarrow}{m}\} \geq 1$. The search of the optimal attack configuration corresponds to the search of a valid attribute propagation with maximized $\min\{\overrightarrow{d_b}, \overleftarrow{d_r}, \overset{\rightarrow\leftarrow}{m}\}$. Thus, the objective of the search model is to maximize a variable $\tau_{\texttt{Obj}}$, which is constrained by Eq. (4).

$$\begin{cases} \overrightarrow{\iota} = |\mathcal{BL}^{\texttt{ENC}}| + |\mathcal{BL}^{\texttt{KSA}}|, \\ \overleftarrow{\iota} = |\mathcal{RD}^{\texttt{ENC}}| + |\mathcal{RD}^{\texttt{KSA}}|. \end{cases} \tag{1}$$

$$\begin{cases} \overrightarrow{d_b} = \overrightarrow{\iota} - \overrightarrow{\sigma}, \\ \overleftarrow{d_r} = \overleftarrow{\iota} - \overleftarrow{\sigma}. \end{cases} \tag{2}$$

$$\overset{\rightarrow\leftarrow}{m} = \sum_{i=0}^{\mathrm{N}_{\texttt{col}}-1} \overset{\rightarrow\leftarrow}{m}_i \,. \tag{3}$$

$$\begin{cases} \tau_{\texttt{Obj}} \leq \overrightarrow{d_b}, \\ \tau_{\texttt{Obj}} \leq \overleftarrow{d_r}, \\ \tau_{\texttt{Obj}} \leq \overset{\rightarrow\leftarrow}{m} \end{cases}. \tag{4}$$

Basic Rules of Propagation and Matching. The attribute propagation and the matching are governed by two types of constraints. The first type is due to the specification of the hash function. The second type is due to the principle of the attack.

Technique improvements are reflected in improving the second type of constraints. For example, the essence of the initial structure [24] is reflected in adding constrained propagation through MixColumns for the purpose of reducing impacts. Importing freedom in key [4] is reflected by firstly allowing more attributes rather than only `Gray` for the key state cells, then combining AddRoundKey with MixColumns operations to build initial structures covering more rounds.

Remark 1. Previously, in [5], for the forward computation, the propagation of `Red`-attribute is designed to concede to the propagation of `Blue`, but `Blue` never gives in to `Red`, and vice versa for backward computation. Unlike in [5], the propagation of one attribute (`Blue` or `Red`) concedes to the propagation of the opposite attribute in both directions is considered in this work. In other words, the consuming degrees of freedom of one attribute is included in the constraints of both forward and backward computations. The considerations are as follows. Since the involving of freedom from key states, a concession of `Blue` to `Red` by consuming degrees of freedom of `Blue` and reserving a local `Red` may enable to cancel a remote `Red` in the key state in forward. Similarly, a concession of `Red` to a local `Blue` in backward may enable to cancel a remote `Blue`. Besides, a concession of `Blue` to reserve a local `Red` (or `Gray`) cell may enable this local `Red` (or `Gray`) cell to propagate and combine with other `Red` cells at a remote point to be mutually canceled through MixColumns and preserve remote `Blue` cells. In addition, an attribute of `Blue` or `Red` propagating to the ending states provides source of degree of matching no matter on which side. (*e.g.*, attack configurations in Fig. 3, Fig. 7, Fig. 5, Fig. 13). By allowing such self-cancellation constraints, the search will cover a larger space. The problem caused by this generalization is the efficiency of the search. When feasible, we allow attribute consuming its freedom in both directions for better solutions. When efficiency becomes a problem for large-version ciphers, we fall back to that one attribute only concedes in one direction to get a solution.

For convenience, in the following, basic rules of propagation are directly described in the framework that one attribute (of `Blue` or `Red`) may concede to the propagation of the opposite attribute by consuming its degree of freedom in both directions (thus, different from the models in [5]). The difference and how to get the modeling of the other framework will be indicated.

Modeling of the Attribute Propagation through SubBytes and ShiftRows. The SubBytes operation does not change the attribute of the cells, thus is not involved when building the model. The ShiftRows operation permutes the state cells, thus is modeled by a set of equations on attributes of corresponding cell permutation.

Modeling of the Attribute Propagation through AddRoundKey (`XOR-RULE`). The AddRoundKey operation is involved in the model when the cipher has KeySchedule (message schedule), and the attack exploits freedom from the key state. Basically, the attribute propagation through AddRoundKey is governed by a set of cell-wise constraints under the name `XOR-RULE`. The principle is that `White` is the dominant attribute, `Gray` is the recessive attribute, `Blue` and `Red` are mutually exclusive attributes. Meaning that

– a `White` cell `XOR`ed with a cell of any attribute results in a `White` cell, *i.e.*, (□ ⊕ ■) → □;
– a `Gray` cell `XOR`ed with a cell of any attribute results in the cell of the same attribute, *i.e.*,
(▨ ⊕ ■) → ■;
– a couple of `Blue` and `Red` cells results in a cell deteriorated to `White`, *i.e.*, (🟦 ⊕ 🟥) → □;
– a couple of `Blue` cells can keep the attributes without consuming or evolve to `Gray` by consuming a degree of freedom of `Blue`, *i.e.*,
(🟦 ⊕ 🟦) → 🟦 or (🟦 ⊕ 🟦) $\xrightarrow{-1\times}$🟦 ▨;
– a couple of `Red` cells can keep the attributes without consuming or evolve to `Gray` by consuming a degree of freedom of `Red`, *i.e.*,
(🟥 ⊕ 🟥) → 🟥 or (🟥 ⊕ 🟥) $\xrightarrow{-1\times}$🟥 ▨;

As for concrete formulas describing these propagation rules, the involved variables are small (six for the input and output, and two for indicating the consumption of degree of freedom for each direction), such that one can generate the set of inequalities using the convex hull computation method [26]. As for the solving efficiency related to `XOR-RULE`, the branches in the search are caused by allowing to consume degree of freedom to evolve to `Gray`.

Modeling of the Attribute Propagation through MixColumns (`MC-RULE`). The MixColumns operation is involved in models of all targeted ciphers in this work. Basically, the attribute propagation through MixColumns is governed by a set of column-wise constraints under the name `MC-RULE`. The constraints are mostly governed by the branch number ($\mathtt{Br_n}$) of the MixColumns. Again, the principle is that `White` is the constrainedant attribute, `Gray` is the recessive attribute, `Blue` and `Red` are mutually exclusive attributes. Concretely,

– any `White` cell in an input column results in all cells in the output column deteriorated to `White`, *i.e.*,
  $(i \times \square,\ j \times \blacksquare) \rightarrow (\mathrm{N}_{\mathtt{row}} \times \square)$, where $i \geq 1, i + j = \mathrm{N}_{\mathtt{row}}$;
– the `Gray` attribute inherits to the output without consuming degrees of freedom only if all cells in the input column are `Gray`, *i.e.*,
  $(\mathrm{N}_{\mathtt{row}} \times \text{▨}) \rightarrow (\mathrm{N}_{\mathtt{row}} \times \text{▨})$;
– existing no `White` cell, a column of i `Blue`, j `Red`, and k `Gray` cells propagate to a column of i' `Blue`, j' `Red`, k' `Gray`, and ℓ' `White` cells by consuming $j' + k'$ degree of freedom from `Blue`, and $i' + k'$ from `Red`, *i.e.*,
  $$(i \times \text{🟦},\ j \times \text{🟥},\ k \times \text{▨}) \xrightarrow{-(j'+k')\times\text{🟦}\ -\ (i'+k')\times\text{🟥}} (i' \times \text{🟦},\ j' \times \text{🟥},\ k' \times \text{▨},\ \ell' \times \square),$$
  where $i+j+k = i'+j'+k'+\ell' = \mathrm{N}_{\mathtt{row}}$ and $\begin{cases} j' + k' < i \leq \mathrm{N}_{\mathtt{row}} & \text{if } i \neq 0 \\ j' + k' = \mathrm{N}_{\mathtt{row}} & \text{otherwise} \end{cases}$,
  $\begin{cases} i' + k' < j \leq \mathrm{N}_{\mathtt{row}} & \text{if } j \neq 0 \\ i' + k' = \mathrm{N}_{\mathtt{row}} & \text{otherwise} \end{cases}$.
  Note that when $i \neq 0$, $j'+k' < i \Leftrightarrow \mathrm{N}_{\mathtt{row}} - i' - l' < i \Leftrightarrow i + i' + l' >= \mathrm{N}_{\mathtt{row}} + 1$, which is due to the branch number; similarly, $i' + k' < j$ when $j \neq 0$ is due to the branch number.

To formalize the concrete propagation rules into a system of inequalities, the involved number of variables is not small. Concretely, the involved variables include the binary variables that indicate the attribute of each cell in the input and output columns, *i.e.*, (x_i^I, y_i^I), (x_i^O, y_i^O), and ω_i^I for $i \in \{0, 1, \cdots, \mathrm{N}_{\mathtt{row}} - 1\}$. The general variables $c_{\boldsymbol{x}}$ and $c_{\boldsymbol{y}}$ for the consumed degree of freedom from `Blue` and `Red`, respectively. Apart from those variables, three auxiliary binary variables are introduced to indicating the following attributes of the input column:

$$\boldsymbol{\omega} = \begin{cases} 1 & \text{exists White cell,} \\ 0 & \text{otherwise.} \end{cases} \quad (5) \qquad \boldsymbol{x} = \begin{cases} 1 & \text{all are Blue/Gray,} \\ 0 & \text{exists Red/White.} \end{cases} \quad (6) \qquad \boldsymbol{y} = \begin{cases} 1 & \text{all are Red/Gray,} \\ 0 & \text{exists Blue/White.} \end{cases} \quad (7)$$

The constraints can then be formalized using inequalities listed in Eq. (11, 12, 13).

Remark 2. When only consider an attribute (`Blue` or `Red`) conceding to the opposite attribute (`Red` or `Blue`) in one direction instead of both, like that in previous work [5], the set of inequalities is similar but simpler. For example, for the propagation of the `Blue` attribute to the forward, it only consumes degrees of freedom from `Red` to preserve `Blue`. Thus, the last two inequalities in Eq. (12) can be $(\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} y_i^O) - \mathrm{N}_{\mathtt{iosum}} \cdot \boldsymbol{y} = 0$; In Eq. (13), inequalities on $c_{\boldsymbol{x}}$ can be removed.

As for the solving efficiency related to `MC-RULE`, the branches in the search are caused by allowing known active cells (🟦's or 🟥's) evolving to have a constant impact on the opposite attribute-propagation by consuming some degree of freedom, and the various ways to select which cells in the output column being evolved. This attribute propagation through MixColumns causes most search branches.

Modeling of the Matching through MixColumns and AddRoundKey: MATCH-RULE The modeling for matching also focuses on the MixColumns and AddRoundKey operations. The matching through MixColumns and AddRoundKey is governed by a set of column-wise constraints under the name MATCH-RULE.

The involved states are $\overrightarrow{End}$, $\overleftarrow{End}$, and the key state $\overrightarrow{End}^{\mathrm{KMC}}$ or $\overleftarrow{End}^{\mathrm{K}}$ in the matching round. The $\overrightarrow{End}$ and $\overleftarrow{End}$ are the states in the encryption data-path that have not been added with the key-state $\overrightarrow{End}^{\mathrm{KMC}}$ or $\overleftarrow{End}^{\mathrm{K}}$. The influence of the AddRoundKey for matching is that, because it is linear, only a White cell in key state has a bad impact on the matchable cells in encryption states. The Blue and Red cells in the key state may even provide degree of matching. Additionally, one can use $\overrightarrow{End} \oplus \overrightarrow{End}^{\mathrm{KMC}}$ as an equivalent key addition to $\overleftarrow{End} \oplus \overleftarrow{End}^{\mathrm{K}}$. The color pattern (most importantly, the distribution of White cells) of $\overrightarrow{End}^{\mathrm{KMC}}$ and $\overleftarrow{End}^{\mathrm{K}}$ are likely different. Thus, adding $\overrightarrow{End}^{\mathrm{KMC}}$ or $\overleftarrow{End}^{\mathrm{K}}$, these two ways may have different effects on the degree of matching. To find the optimal solution, we choose to use the key state, which has fewer White cells. Known the propagation direction of the key-schedule (*i.e.*, $\mathtt{init}_r^{\mathrm{K}}$), of the two states $\overrightarrow{End}^{\mathrm{KMC}}$ and $\overleftarrow{End}^{\mathrm{K}}$, the one that is near to the initial key state $\overleftrightarrow{S}^{\mathtt{KSA}}$ must have fewer White cells.

The conditions for the i-th column to have $\overrightarrow{\overleftarrow{m}}_i$ degree of matching are:

- exists both Blue and Red cell (🟦 and 🟥) in input/output columns;
- denote the number of known cells (*i.e.*, except White cells) by $\overline{m_k}_i$; when $\overline{m_k}_i > \mathrm{N}_{\mathtt{row}}$, $\overrightarrow{\overleftarrow{m}}_i = \overline{m_k}_i - \mathrm{N}_{\mathtt{row}}$. Denote the number of white cells by $\overline{m_w}_i$; Since $\overline{m_k}_i = 2 \cdot \mathrm{N}_{\mathtt{row}} - \overline{m_w}_i$, one have $\overrightarrow{\overleftarrow{m}}_i = \mathrm{N}_{\mathtt{row}} - \overline{m_w}_i$.

Accordingly, the concrete system of inequalities modeling MATCH-RULE can be obtained as listed in Eq. (14).

3.2 Enhanced Model with Guess-and-Determine

Guess-and-Determine Strategies. Because of the diffusion of MixColumns, especially the property of the MDS matrix, one unknown cell in the input column of MC makes all cells in the output column unknown (refer to point 1 of MC-RULE).

Sasaki *et al.* in [25] found the following interesting conclusion. Guessing the values of few unknown cells to continue the propagation of attribute to reach the meeting point, and check the consistency of the few guessing cells after (partial) matching, one can still achieve a better attack than a brute-force guessing. With the guess-and-determine approach, for Whirlpool, Sasaki *et al.* in [25] successfully increased one more attacked round than the 5-round attack in [29].

Concretely, denote 2^c by ς (where c is the number of bits in each cell of the state, in targeted ciphers of this work, it is 8), and let

- $\overrightarrow{d_{g_b}}$ be the number of cells only guessed to be Blue (forward computation);

– $\overleftarrow{d_{g_r}}$ be the number of cells only guessed to be `Red` (backward computation);
– $\overleftrightarrow{d_{g_{br}}}$ be the number of cells guessed to be both `Blue` and `Red`.

The framework of the MITM attack with guess-and-determine is as follows:

1. Randomly assign compatible values to all cells except for those depending on the neutral bytes (*i.e.*, `Gray` in the attack configuration), and values of impacts from one attribute to the opposite attribute;
2. Compute values $\{\overrightarrow{v}_i\}$ of forward neutral bytes and values $\{\overleftarrow{v}_i\}$ of backward neutral bytes under the values of `Gray` cells and values of impacts that they are supposed to fulfill.
3. For all $\varsigma^{\overrightarrow{d_b}}$ values $\{\overrightarrow{v}_i\}$ of forward neutral bytes, and $\varsigma^{(\overrightarrow{d_{g_b}}+\overleftrightarrow{d_{g_{br}}})}$ guessed values $\{\overrightarrow{v}_g\}$ for forward, compute forward to the matching point and store all $\varsigma^{\overrightarrow{d_b}+\overrightarrow{d_{g_b}}+\overleftrightarrow{d_{g_{br}}}}$ partial matching values $\{\overrightarrow{v}_m\}$ in a look up table $\overrightarrow{\mathcal{T}}$ (the values are $\overrightarrow{v}_i$ and $\overrightarrow{v}_g$, and the index is $\overrightarrow{v}_m$).
4. For all $\varsigma^{\overleftarrow{d_r}}$ values $\{\overleftarrow{v}_i\}$ of backward neutral bytes, and $\varsigma^{(\overleftarrow{d_{g_r}}+\overleftrightarrow{d_{g_{br}}})}$ guessed values $\{\overleftarrow{v}_g\}$ for backward, compute backward to the matching point, obtain the partial matching values $\overleftarrow{v}_m$.
5. For all values of $\overrightarrow{v}_i$ and $\overrightarrow{v}_g$ in entry $\overrightarrow{\mathcal{T}}[\overleftarrow{v}_m]$, use $\overrightarrow{v}_i$ and $\overleftarrow{v}_i$ to compute and check if the guessed values $\overrightarrow{v}_g$ and $\overleftarrow{v}_g$ are correct. For correct guesses, compute to the matching point to check if it is a full-state match. If so, use $\overrightarrow{v}_i$ and $\overleftarrow{v}_i$ to compute the preimage, output it, and return; otherwise, repeat from Step 1.

In the above framework of the MITM attack with guess-and-determine,

– the total degree of freedom for `Blue` with guessing is $\overrightarrow{d_b}+\overrightarrow{d_{g_b}}+\overleftrightarrow{d_{g_{br}}}$;
– the total degree of freedom for `Red` with guessing is $\overleftarrow{d_r}+\overleftarrow{d_{g_r}}+\overleftrightarrow{d_{g_{br}}}$;
– the expected number of matched pairs $\varsigma^{\overrightarrow{d_b}+\overrightarrow{d_{g_b}}+\overleftrightarrow{d_{g_{br}}}+\overleftarrow{d_r}+\overleftarrow{d_{g_r}}+\overleftrightarrow{d_{g_{br}}}-\overrightarrow{\overleftarrow{m}}}$;
– the required number of repetitions to get a full match at the guessing cells and a full-state match is $\varsigma^{-(\overrightarrow{d_b}+\overrightarrow{d_{g_b}}+\overleftrightarrow{d_{g_{br}}}+\overleftarrow{d_r}+\overleftarrow{d_{g_r}}+\overleftrightarrow{d_{g_{br}}}-\overrightarrow{\overleftarrow{m}})}\cdot\varsigma^{\overrightarrow{d_{g_b}}+\overleftrightarrow{d_{g_{br}}}+\overleftarrow{d_{g_r}}}\cdot\varsigma^{(n-\overrightarrow{\overleftarrow{m}})}$, which equals $\varsigma^{n-\overrightarrow{d_b}-\overleftarrow{d_r}-\overleftrightarrow{d_{g_{br}}}}$;

Thus, the complexity of the attack is $\varsigma^{n-\overrightarrow{d_b}-\overleftarrow{d_r}-\overleftrightarrow{d_{g_{br}}}}\cdot(\varsigma^{\overrightarrow{d_b}+\overrightarrow{d_{g_b}}+\overleftrightarrow{d_{g_{br}}}}+\varsigma^{\overleftarrow{d_r}+\overleftarrow{d_{g_r}}+\overleftrightarrow{d_{g_{br}}}}+\varsigma^{\overrightarrow{d_b}+\overrightarrow{d_{g_b}}+\overleftrightarrow{d_{g_{br}}}+\overleftarrow{d_r}+\overleftarrow{d_{g_r}}+\overleftrightarrow{d_{g_{br}}}-\overrightarrow{\overleftarrow{m}}})$, which equals

$$\begin{aligned}&\varsigma^n\cdot(\varsigma^{-(\overleftarrow{d_r}-\overrightarrow{d_{g_b}})}+\varsigma^{-(\overrightarrow{d_b}-\overleftarrow{d_{g_r}})}+\varsigma^{-(\overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})})\\ \simeq\ &\varsigma^n\cdot\max(\varsigma^{-(\overleftarrow{d_r}-\overrightarrow{d_{g_b}})},\varsigma^{-(\overrightarrow{d_b}-\overleftarrow{d_{g_r}})},\varsigma^{-(\overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})})\end{aligned}\tag{8}$$

Thus, the complexity is determined by

$$\min(\overleftarrow{d_r}-\overrightarrow{d_{g_b}},\overrightarrow{d_b}-\overleftarrow{d_{g_r}},\overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}}).$$

Building Model for Guessing. To model the mechanism of guess-and-determine, three binary variables, g_b, g_r, g_{br}, are introduced for each cell in the input state of MixColumns (invMixColumns for the backward computation).

The variables indicate whether the values of the cells should be guessed to be of one attribute. Concretely, $g_b = 1$ for guessing one `White` cell to be `Blue`. $g_r = 1$ for guessing one `White` cell to be `Red`. $g_{br} = 1$ for guessing one `White` cell to be `Blue` (for forward propagation) and also `Red` (for backward propagation).

With these guess-indicating variables, attribute-indicating variables for each cell in the input of MixColumns are thus constrained together with attribute-indicating variables for the cell in output state of last operations (*e.g.*, ShiftRows or AddRoundKey).

Besides, according to the complexity Eq. (8) of the attack with guess-and-determine, the objective should turn from $\min(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}})$ to $\min(\overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}})$. Thus, the variable that to be maximized is constrained as in Eq. (9) and (10).

$$\begin{cases} \overrightarrow{d_{g_b}} = \sum_{r=0,i=0}^{\mathtt{total}_r-1,n-1} {g_b}_i^r, \\ \overleftarrow{d_{g_r}} = \sum_{r=0,i=0}^{\mathtt{total}_r-1,n-1} {g_r}_i^r, \\ \overleftrightarrow{d_{g_{br}}} = \sum_{r=0,i=0}^{\mathtt{total}_r-1,n-1} {g_{br}}_i^r, \end{cases} \tag{9}$$

$$\begin{cases} \tau_{\mathtt{Obj}} \le \overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \\ \tau_{\mathtt{Obj}} \le \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \\ \tau_{\mathtt{Obj}} \le \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}. \end{cases} \tag{10}$$

Note that in the starting round, all cells are known and in the matching round, guessing brought no advantage. Thus, for these two rounds, such constraints on guessing can be omitted. Besides, heuristically, guessings are generally required around the matching points. Thus, for efficiency, these guessing constraints can be added only for those ending rounds, *i.e.*, round-dependent modeling.

When only consider the scenario where one attribute canceling its freedom for the propagation of the opposite attribute in only one direction, one variable g instead of three (g_b, g_r, g_{br}) is sufficient. Because in each direction, the guessing is for the propagation of only one attribute.

3.3 Separating Attributes in the Same States for Linear Operations

As mentioned above, the attribute-propagation of `Blue` and `Red` are mutually exclusive. But under some conditions, one attribute-propagation may concede to the propagation of the opposite attribute-propagation by consuming freedom of itself. The `XOR-RULE` and `MC-RULE` have different requirements for enabling such concession (cancellation). Combining these two operations, states of attribute that do not meet the individual requirements are possible to make such concession.

In [5], the propagation through the combination of AddRoundKey and Mix-Columns is characterized using the set of `XOR-MC-RULE`. In `XOR-MC-RULE`, the

OR of the attribute of encryption-state cell and key-state cell is regarded as the attribute of the input cell of MixColumns. In that way, the group of cells of the same attribute in both encryption and key states can jointly cancel impacts on the output cell of the opposite attribute. However, using only XOR-MC-RULE, the possibility of the following scenario is missed. That is, an attribute can be completely canceled via XOR-RULE before propagating through MixColumns, which is not possible using only the XOR-MC-RULE (because MC-RULE cannot turn active cells into inactive) (refer to Fig. 2a for an illustration). Thus, to find optimal attack configurations, applying XOR-RULE-then-MC-RULE and XOR-MC-RULE should be both considered in the models.

In this work, we model the combination of AddRoundKey and MixColumns by considering the separation of (Blue and Red) attribute propagations. Note that AddRoundKey and MixColumns are linear. Essentially, for linear operations, in the same state, the attributes of Blue and Red can separately propagate through them and then combine by cell-wise XOR upon the non-linear operation (*i.e.*, SubBytes).

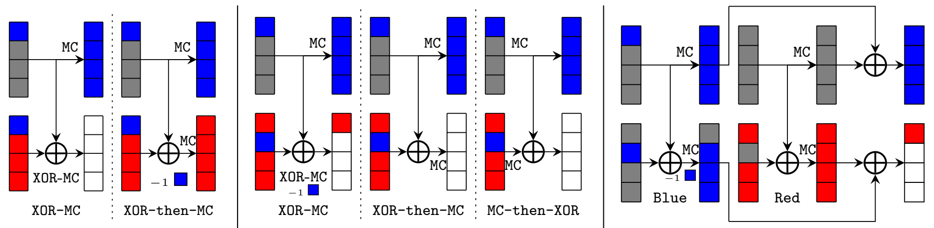
Additionally, the key-schedule also has linear operations. It is possible that before going through the non-linear operation in the key-schedule, the attribute of one cell in the round-key is a linear combination of Blue and Red. If not be separately considered, such a linear combination of Blue and Red becomes White. Separately, the Blue component in the linear combination can be used with the Blue component in the encryption state, and the Red component can be used with the Red component in the encryption state. Consequently, cancellation constraints to reduce impacts on the opposite attribute propagation that cannot be fulfilled previously become possible.

Moreover, at the matching point, if a key state cell is a linear combination of Blue and Red, it does not impact the matching ability of the corresponding state cell, and instead, the Blue component and Red component may provide degrees of matching.

Thus, we separately consider Blue-attribute propagation and Red-attribute propagation through linear operations in both encryption and key-schedule; Due to this separation, XOR-RULE and MC-RULE without XOR-MC-RULE is sufficient (refer to Fig. 2 for an illustration of the separation of attribute-propagations through AddRoundKey and MixColumns).

3.4 Using the Similarity between Encryption and Key-Schedule

In previous work [5], the target cipher is AES hashing mode. For AES, the operations in the key-schedule are different from those in the encryption. However, in Whirlpool, the key-schedule share the same operations except for AddRoundKey. This similarity between encryption and key-schedule enables an effect that might not occur in other ciphers. This effect is that the AddRoundKey can be directly moved around MixColumns in the encryption by changing to adding the key state before MixColumns (denoted by $\#\textsc{kmc}$) or the actual key state after MixColumns (denoted by $\boldsymbol{k}$).



(a) A propagation missed by XOR-MC-RULE

(b) The necessity of XOR-MC-RULE without separation

(c) The sufficiency without XOR-MC-RULE with separation

– In 2a, using XOR-MC-RULE, a result of full Red attribute can not be achieved, which can be obtained by XOR-then-MC-RULE.
– In 2b, without separation, (Blue and Red) attribute-propagation ruled by XOR-MC-RULE (the first) cannot be obtained directly by XOR-then-MC-RULE and MC-then-XOR-RULE (the last two).
– In 2c, with the separation of (Blue and Red) attribute-propagation, XOR-RULE and MC-RULE are sufficient (2c achieve the same results as the first in 2b).

Fig. 2: Combination of linear operations and separation of attribute-propagations

It is possible that moving AddRoundKey before MixColumns and using the key state before MixColumns bring more advantages for the attribute-propagation on the encryption data-path. Take the scenario of preserving Blue by consuming Red for example; considering one column, suppose there are a few Red cells in #**MC** and a few Red cells in #**KMC**; these few Red cells will be diffused into the whole column if there is no constraint. Adding #**KMC** with #**MC** before MixColumns in the encryption and letting the Red in #**MC** be canceled by the few Red in #**KMC** might consumes fewer degrees of Red than adding #**AK** with $\boldsymbol{k}$.

Similarly, it is also possible that adding #**AK** with $\boldsymbol{k}$ after the MixColumns has more advantages than adding #**MC** with #**KMC**. Thus, to find optimal attack configurations, both scenarios should be considered. The essential difference between the scenarios is to either firstly use freedom in the key state to directly cancel impacts before diffusion or to postpone the insertion of the key state in order to postpone impacts from the key. We name the choice of applying the first scenario by AK-MC-RULE and the second scenario by MC-AK-RULE. The integration of the two scenarios into one model is in the form of indicator constraints that is available in Gurobi. Note that, for forward computation, AK-MC-RULE corresponds to using #**MC** $\oplus$ #**KMC**, MC-AK-RULE corresponds to using #**AC** $\oplus$ $\boldsymbol{k}$; for backward computation, AK-MC-RULE corresponds to using #**SB** $\oplus$ $\boldsymbol{k}$, and MC-AK-RULE corresponds to using #**MC** $\oplus$ #**KMC**;

In addition, since the MixColumns is column-wise, different columns can use different modeling independently. Besides, since MixColumns is linear, different attributes of Blue and Red can independently apply in different ways.

Allowing such flexibility of choice, the efficiency of the solving is impacted. So, to decide in which way to proceed for each column, we use the heuristic that when the key-schedule has consumed some degree of freedom in that column, we apply MC-AK-RULE; otherwise, apply AK-MC-RULE.

3.5 Using the Symmetry of the Ciphers

Due to the integration of many technical improvements and generalizations, the search space is greatly enlarged. One needs to make a trade-off between the quality of the searching result and the efficiency of the search. Besides trade-offs made by removing some techniques, the symmetry of the ciphers is used to reduced the problem scale.

Specifically, with respect to the MITM attacks considered in this work, many AES-like designs can be viewed as of the Matryoshka structure. Properties exploited in the attack are translation invariant in the direction of the row axis and/or column axis. Rotational parameters of rows of the large version can wrap to identical rotational parameters of rows for all small versions along the row axis. The branch number of columns of the large version can be implied by identical branch numbers of columns of all small versions along the column axis. This allows projecting attack configurations on small-size versions to that on the large-size version.

Concretely, a state of 8×8 cells can be viewed as a 2×2 matrix of state of 4×4 cells (we call this 2x2-symmetry), or a 1×2 matrix of state of 8×4 cells. Obtaining an attack configuration for the 4×4 state version, cloning the state patterns four times, and placing them in a 2×2 matrix result in a configuration for the 8×8 state version. The projection from the attack configuration in Fig. 3 to the attack configuration in Fig. 10 is one example of such correspondence. Similarly, obtaining an attack configuration for 8×4 state version, cloning the state patterns two times, and placing them side-by-side in a 1×2 matrix, also result in a configuration for the 8×8 state version. Fig. 14 shows an example of reducing 8×8 version to be an 1×2 of 8×4 version.

Exploiting such symmetry of the ciphers, the search can be efficient, while might lose asymmetric attack configurations (*e.g.*, that in Fig. 12).

4 Application to Whirlpool

4.1 Description of Whirlpool

Whirlpool, a block-cipher based secure hash function designed by Rijmen and Barreto in 2000, produces a 512-bit hash value using Miyaguchi-Preneel construction for an input message of maximum length less than 2^{256} bits. And it has been adopted as an ISO/IEC standard.

Basically, the compression function consists of the key schedule and state update transformation. The compression function uses a 10-round AES-like block cipher E_k with 8×8-byte internal states which takes a 512-bit chaining value H_i as a key and operates on a 512-bit message block of a plaintext M_i, to output $F(H_i, M_i) = E_{H_i} \oplus M_i \oplus H_i$. The $(8j + i)$-th input bytes of the message block is placed at the i-th row and j-th column of the state. Each round consists of four operations:

– SubBytes (SB) applies the Substitution-Box to each byte.

– ShiftColumns ($\mathtt{SC}$) cyclically shifts the j-column downwards by j bytes.
– MixRows ($\mathtt{MR}$) multiplies each row of the state matrix by an MDS matrix.
– AddRoundKey ($\mathtt{AK}$) $\mathtt{XOR}$ the round key to the state.

H_0 serves as the key value k_0, and then ten 512-bit subkeys k_1, k_2, $\cdots$, k_{10} are generated by the key-schedule function below:

$$k_{i+1} \leftarrow \mathtt{AC} \circ \mathtt{MR} \circ \mathtt{SC} \circ \mathtt{SB}(K_i)$$

for $i = 0, 1, 2, \cdots, 9$, where AddRoundConstants ($\mathtt{AC}$) $\mathtt{XOR}$ a 512-bit constant defined in the key schedule.

At the start of hashing, M_i serves as the plaintext p, and the whitening operation is performed to get $s_0 = k_0 \oplus p$. In each round, the output state s_{i+1} of the block cipher is updated as follows:

$$s_{i+1} \leftarrow \mathtt{AK} \circ \mathtt{MR} \circ \mathtt{SC} \circ \mathtt{SB}(s_i)$$

for $i = 0, 1, 2, \cdots, 9$.

4.2 New Attacks Resulted from Applying the MILP Modeling

Applying the MILP models described in Sect. 3 on Whirlpool, improved attacks are found for 5- and 6-round, and first attacks are found for 7-round.

For 5-round attacks (refer to Fig. 13), guess-and-determine is not required, but allowing one attribute-propagation conceding to the opposite attribute-propagation in both directions is essential for achieving the best complexity. For 6-round attacks (refer to Fig. 12), guess-and-determine is the critical technique that enables the improved results; Expanding to 7-round attacks (refer to Fig. 3, Fig. 10, Fig. 11), besides guess-and-determine, flexible choices of adding $\#\textbf{KMC}$ or adding $\boldsymbol{k}$ is an additional key point.

The following of this section describes how to use one of the resulted attack configurations to launch a concrete attack on 7-round Whirlpool. A brief description of the improved 6-round attack is then followed. In the description, ShiftRows and MixColumns instead of ShiftColumns and MixRows as specified in the design of Whirlpool are used. Thus, the states should be transposed to correspond with the specification of Whirlpool. This transposition does not influence the attacks. Please refer to Sect. C for a summary of notations.

The attack on 7-round Whirlpool. Fig. 3 and 10 show one of the attack configurations on 7-round Whirlpool. Using symmetry of the cipher, the search was done on small versions with $\mathrm{N}_{\mathtt{row}} \times \mathrm{N}_{\mathtt{col}} = 4 \times 4$. Cloning the resulted configurations four times and placing four identical small states in a 2×2 matrix, the resulted configurations with $\mathrm{N}_{\mathtt{row}} \times \mathrm{N}_{\mathtt{col}} = 8 \times 8$ is a valid one for the real version (*e.g.*, the correspondence between Fig. 3 and Fig. 10).

In the sequel, for the ease of finding correspondence with our implementation for experimental verification, we describe the attack using the small version with $\mathrm{N}_{\mathtt{row}} \times \mathrm{N}_{\mathtt{col}} = 4 \times 4$. One can quickly project this attack on the small version to the real version of Whirlpool. The complexity of the attack on the real version is to the power of four of this small version's complexity.

Compute initial values of neutral bytes in `Red`*.* The left-hand side of Fig. 3 is found by solving the MILP models. Following this configuration, one can obtain the initial values of neutral bytes in `Red` that fulfill the cancellation constraints among states ($\#\mathbf{AK}^3, k_3, \#\mathbf{SB}^3$) and ($\#\mathbf{MC}^5, \#\mathbf{KMC}^4$). The computational complexity will be no more than 2^{32}.

However, one can observe an equivalent configuration shown on the right-hand side of Fig. 3. Following both configurations, one can devise the attack, with a different procedure of computing the initial values of backward neutral bytes between the two. The latter (right-hand side of Fig. 3) is more direct in terms of computing values of `Red` neutral bytes; thus, it will be used in the following description of attack [4].

To get the initial values of backward neutral bytes, one only needs to enumerate all possible values of cell $k_3[15]$, and fix values of cells in the main anti-diagonal of $\#\mathbf{SB}^4$. That is due to the following observation. Fixing the values of cells in the first column of $\#\mathbf{MC}^4$ to be 0 (equivalently, fixing the values of cells in the main anti-diagonal of $\#\mathbf{SB}^4$ to be $\text{Sbox}^{-1}[0]$), the values of the first column of $\#\mathbf{SB}^5$ equals that of k_4. Note that the operations of the round function in encryption and key-schedule are exactly the same, excepting the former using AddRoundKey and the latter using AddRoundConstants. Thus, the first anti-diagonal of $\#\mathbf{MC}^5$ will equal that of $\#\mathbf{KMC}^4$. Consequently, the impact brought by adding `Red` cells in the 4th round (k_4) will be canceled in the next round by adding $\#\mathbf{KMC}^4$.

Compute initial values of neutral bytes in `Blue`*.* To get the initial values of forward neutral bytes (in `Blue`), one focuses on the constraints among states $\{\#\mathbf{MC}^2, \#\mathbf{AK}^2, \#\mathbf{SB}^3, \#\mathbf{MC}^3, \#\mathbf{AK}^3\}$.

There are five degrees of freedom (in bytes) for the forward (`Blue`). Using four out of the five, one can keep the same attack complexity because the degree of freedom for backward is the bottleneck. Thus, we fix the value of one `Blue` cell in $\#\mathbf{AK}^3$, *i.e.*, $\#\mathbf{AK}^3[3]$, as indicated by $\boxed{\mathtt{C}}$.

Note that values of `Blue` cells are constrained to have constant impacts on the last anti-diagonal of $\#\mathbf{MC}^2$ and constant impact on the first diagonal of $\#\mathbf{AK}^3$. Denote the constant impacts by $\#\mathbf{MC}^2_{\mathtt{C}}[3, 6, 9, 12]$ and $\#\mathbf{AK}^3_{\mathtt{C}}[0, 5, 10]$. The initial values of forward neutral bytes can be computed using a local meet-in-the-middle procedure as shown in Algorithm 1. In Algorithm 1, the procedure starts from guessing two free cells in the first column of $\#\mathbf{AK}^3$ and one cell in each of the columns in $\#\mathbf{SB}^3$, computes other undetermined cells using predetermined impacts on cells in $\#\mathbf{MC}^2$ column-by-column, and compute back pair-wisely to match at constant cells in $\#\mathbf{AK}^3$. From Algorithm 1, the computational complexity for obtaining the initial values of neutral bytes in `Blue` is 2^{32} and the memory required is 2^{32} (blocks) [5]

[4] Both procedures of computing values of backward neutral bytes are experimentally verified in our implementation.

[5] Implementation of Algorithm 1 can be found together with the implementation of the entire attack via `https://github.com/MITM-AES-like-Hashing/Whirlpool_7R`.

The Main Procedure. Fix the values of constant impacts of `Blue` on `Red` in $\#\mathbf{MC}^2$ (denoted by $\#\mathbf{MC}^2_{\mathtt{C}}[3,6,9,12]$) and the value of $\#\mathbf{AK}^3_{\mathtt{C}}[3]$ to be arbitrary. Set $\#\mathbf{AK}^3[0,5,10,12,13,14,15]$ be 0, and $k_3[0,5,10,15]$ be $\mathrm{Sbox}^{-1}[0]$.

Precompute the initial values of forward neutral bytes as shown in Algorithm 1 and store the result in $\overrightarrow{\mathcal{T}_{\mathtt{Init}}}$.

1. For each possible value of 12 `Gray` bytes $k_3[1,2,3,4,6,7,8,9,11,12,13,14]$, calculate the values of `Gray` bytes in k_4 and $\#\mathbf{KMC}^4$.
   (a) For each value $\overrightarrow{v}_i$ of `Blue` cells in $\#\mathbf{MC}^3$ stored in $\overrightarrow{\mathcal{T}_{\mathtt{Init}}}$,
      i. start from $\#\mathbf{MC}^3$, compute forward (only cells in `Blue`) with the values of `Gray` bytes in k_3 and k_4 to $\#\mathbf{MC}^5$;
      ii. `XOR` $\#\mathbf{MC}^5$ with the values of `Gray` bytes in $\#\mathbf{KMC}^4$;
      iii. set the first anti-diagonal of $\#\mathbf{MC}^5$ to be zero, and compute forward to $\#\mathbf{MC}^6$ (without AddRoundKey with k_5 but need to `XOR` the round constant RC[5], because $\#\mathbf{KMC}^4$ is used instead);
      iv. compute $\mathtt{MC}(\#\mathbf{MC}^6)$ and get the value $\overrightarrow{v}_m$ of the main diagonal
      v. store $\overrightarrow{v}_i$ in a look-up table $\overrightarrow{\mathcal{T}}$ with $\overrightarrow{v}_m$ as index;
   (b) For each value $\overleftarrow{v}_i$ of the `Red` cell in k_3, with the value of `Gray` cells,
      i. compute all values of the round keys, *i.e.*, $k_2, k_1, k_0, k_m, k_4, k_5, k_6$, where k_m is the master key;
      ii. set the `Red` cell $\#\mathbf{AK}^3[15]$ to be $\overleftarrow{v}_i \oplus \mathrm{Sbox}^{-1}[0]$, combine with the constant value in $\#\mathbf{AK}^3$, compute backward up to $\#\mathbf{AK}^0$;
      iii. For each value $\overleftarrow{v}_g$ of the `Pink` cells in $\#\mathbf{AK}^0$,
         A. with the value of `Red` cell in the first column of $\#\mathbf{AK}^0$, compute backward through feed-forward, `XOR` the given target T, compute $\#\mathbf{AK}^6$;
         B. get the value $\overleftarrow{v}_m$ of the main diagonal of $\#\mathbf{AK}^6$.
         C. for each value $\overrightarrow{v}_i$ of `Blue` cells in $\#\mathbf{MC}^3$ stored in $\overrightarrow{\mathcal{T}}[\overleftarrow{v}_m]$, combine the values of `Red` cells, restart the computation from $\#\mathbf{AK}^3$ up to $\#\mathbf{AK}^0$;
         D. If the newly computed value $\overrightarrow{v}'_g$ of the `Pink` cells in $\#\mathbf{AK}^0$ does not equal $\overleftarrow{v}_g$, go to Step 1(b)iii. Else, compute to $\#\mathbf{AK}^6$, denote the value by $\overleftarrow{v}$.
         E. Start from $\#\mathbf{AK}^3$ with the combined knowledge of values of both `Blue` and `Red` cells, compute to $\mathtt{MC}(\#\mathbf{MC}^6)$, if its value $\overrightarrow{v}$ equals $\overleftarrow{v}$, a full-state match is found, output the state $\#\mathbf{SB}^0$ and the key state k_m. Otherwise, go to Step 1.

Complexity. As analyzed above, the computational and memory complexity of the precomputation of the initial value of forward neutral bytes is 2^{32}. As for the complexity of the main procedure, the attack configuration on the small version ($n = \mathrm{N}_{\mathtt{row}} \times \mathrm{N}_{\mathtt{col}} = 4 \times 4 = 16$) is, $\overrightarrow{d_b} = 4$, $\overleftarrow{d_r} = 1$, $\overleftarrow{d_{g_r}} = 3$, $\overset{\rightarrow\leftarrow}{m} = 4$, $\varsigma = 2^8$, and $\overrightarrow{d_{g_b}} = \overleftrightarrow{d_{g_{br}}} = 0$. According to Eq. (8), the complexity of the whole attack on the small version is therefore $\varsigma^n \cdot (\varsigma^{-(\overleftarrow{d_r} - \overrightarrow{d_{g_b}})} + \varsigma^{-(\overrightarrow{d_b} - \overleftarrow{d_{g_r}})} + \varsigma^{-(\overset{\rightarrow\leftarrow}{m} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}})}) = \varsigma^{16} \cdot \max(\varsigma^{-1} + \varsigma^{-(4-3)} + \varsigma^{-(4-3)}) = \varsigma^{15} = 2^{120}$

Projecting to the real version of Whirlpool ($n = \mathrm{N}_{\texttt{row}} \times \mathrm{N}_{\texttt{col}} = 16 \cdot 4 = 64$) (refer to Fig. 10 in which fix four extra `Blue` cells in $\#\mathbf{AK}^3$), the complexity will be $(2^{120})^4 = 2^{480}$. Concretely, the attack configuration will be $\overrightarrow{d_b} = 4 \cdot 4 = 16$, $\overleftarrow{d_r} = 1 \cdot 4 = 4$, $\overleftarrow{d_{g_r}} = 3 \cdot 4 = 12$, $\overrightarrow{\overleftarrow{m}} = 4 \cdot 4 = 16$, $\varsigma = 2^8$, and $\overrightarrow{d_{g_b}} = \overleftrightarrow{d_{g_{br}}} = 0$. Accordingly, the attack complexity on the real version of 7-round Whirlpool is $\varsigma^{64} \cdot \max(\varsigma^{-4} + \varsigma^{-(16-12)} + \varsigma^{-(16-12)}) = 2^{480}$.

The memory required in the main procedure is that taken by $\overrightarrow{\mathcal{T}}$, whose size is also limited by the degree of freedom for forward, that is 2^{32} as for the small version, and 2^{128} for the real version.

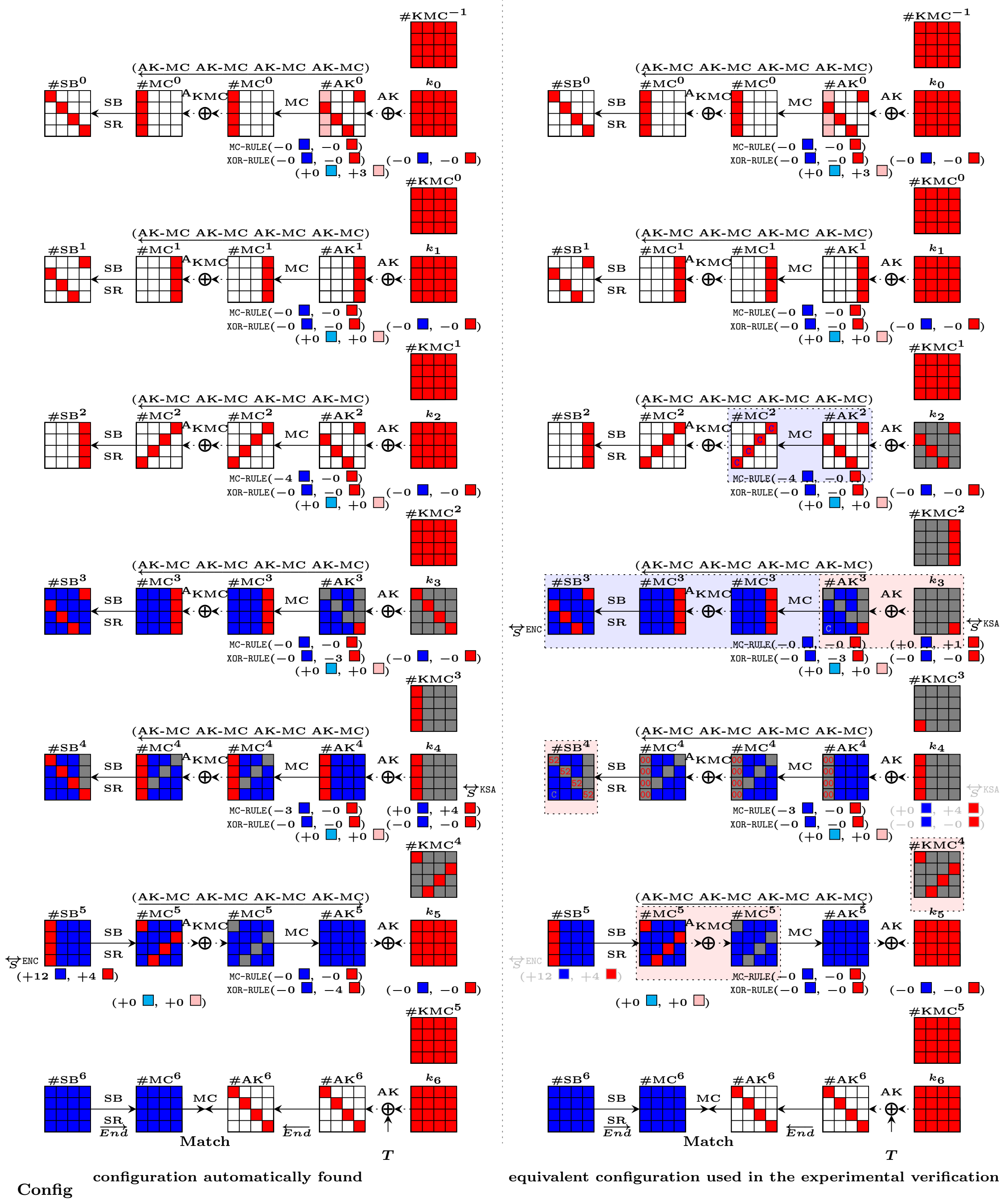
To further verify the attack and its complexity analysis, we implemented the full attack on the small version with $\mathrm{N}_{\texttt{row}} \times \mathrm{N}_{\texttt{col}} = 4 \times 4$. Round function of AES is used to simulate the round function and key schedule of the small version of Whirlpool compression function (round constants are from full Whirlpool). To make the verification practical, the experiments aim for partial matching instead of full matching between the matching states, while preserving the 2^8 complexity gain. Concretely, the goal is to match m bits instead of 128 bits with verified complexity $\max\{2^{32}, 2^{m-8}\}$. Please refer to `https://github.com/MITM-AES-like-Hashing/Whirlpool_7R` for results on $m \in \{36, 40, 44, 48\}$.

Remark 3. Besides the presented attack, there are several different attack configurations generated by solving MILP models, *e.g.*, the example illustrated in Fig. 11. These results commonly require to guess at least three cells in at least one direction and use the flexibility of choosing `XOR` $\#\mathbf{KMC}$ or $\boldsymbol{k}$.

The attack on 6-round Whirlpool. When using the symmetry of the cipher and search on small versions (with $\mathrm{N}_{\texttt{row}} \times \mathrm{N}_{\texttt{col}} = 4\times4$, and $\mathrm{N}_{\texttt{row}} \times \mathrm{N}_{\texttt{col}} = 8\times4$), the best attack configurations imply attacks on the real version with a configuration $\overrightarrow{d_b} = 8$, $\overleftarrow{d_r} = 8$, $\overrightarrow{\overleftarrow{m}} = 8$, $\varsigma = 2^8$, and $\overrightarrow{d_{g_b}} = \overleftarrow{d_{g_r}} = \overleftrightarrow{d_{g_{br}}} = 0$. Thus, the complexity is 2^{448} (refer to Fig. 14).

When searching on the full-size version ($\mathrm{N}_{\texttt{row}} \times \mathrm{N}_{\texttt{col}} = 8\times8$), better results are found with asymmetric patterns. One example is depicted in Fig. 12. Following the configuration depicted in Fig. 12, one can devise a better attack on 6-round of Whirlpool. The procedures to compute the initial values of neutral bytes for both directions are relatively simpler than that of the above attack on 7-round. That is because, the cancellation constraints on the `Blue` is at a single point, *i.e.*, ($\#\mathbf{MC}^2, \#\mathbf{AK}^2$). The cancellation constraints on the `Red` is also at a single point, *i.e.*, ($\#\mathbf{MC}^4, \#\mathbf{AK}^4$). As for such constraints, column-by-column independent computations can be used to derive the initial values of neutral bytes for both directions. The concrete attack configuration is $\overrightarrow{d_b} = 9$, $\overleftarrow{d_r} = 24$, $\overrightarrow{d_{g_b}} = 15$, $\overrightarrow{\overleftarrow{m}} = 24$, $\varsigma = 2^8$, and $\overleftarrow{d_{g_r}} = \overleftrightarrow{d_{g_{br}}} = 0$. Accordingly, the attack complexity on the full-size version of 6-round Whirlpool is $\varsigma^{64} \cdot \max(\varsigma^{-24-15} + \varsigma^{-(9)} + \varsigma^{-(24-15)}) = 2^{440}$. The memory required is $2^{24\times8} = 2^{192}$.

Conversion from Pseudo-Preimage to Preimage Attacks. This has been discussed in previous works, and we follow the *Two Types of Last Block Attacks*


configuration automatically found
equivalent configuration used in the experimental verification
Match
T

Config

- $(\overrightarrow{\iota}, \overleftarrow{\iota}) = (12$ 🟦, 8 🟥) • $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}}) = (5$ 🟦, 1 🟥, 4 🟪, 0 🟦, 3 🟥, 0 🟩)
- $(\overrightarrow{d_b}-\overleftarrow{d_{g_r}}, \overleftarrow{d_r}-\overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}}) = (2, 1, 1)$

This implies an attack on the full version (8×8) as shown in Fig. 10 with configuration
$((\overrightarrow{d_b} - \overleftarrow{d_{gr}}, \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{gr}} - \overleftrightarrow{d_{g_{br}}}) = (8, 4, 4))$

Fig. 3: An example of using $(2 \times 2$ of $4 \times 4)$ to search the MITM attack on 7-round Whirlpool and an equivalent configuration used in the experimental verification

from [25, 29]. Denote the time complexity of inverting the reduced-Whirlpool compression function as 2^l. A random message fulfills the padding rule of Whirlpool with probability 2^{-9}, hence it costs 2^{l+9} to find a right last block. Then an unbalanced meet-in-the-middle is carried out between the initial value and the input chaining value of the last block, which costs $2^{(512+l)/2}$ and sums to $2^{(512+l)/2} + 2^{l+9}$ to find a long and full preimage. Detailed calculations are summarized in Table 1. We note further complexity optimizations are possible, by finding pseudo-preimages under multi-target scenarios and utilizing them in the "unbalanced meet-in-the-middle" phase as discussed in [12].

4.3 Discussions on the New Attacks

The previous best attack on Whirlpool is up to 6-round [25], in which guess-and-determine is the essential technique that enables expanding one more attacked round than the 5-round attack in [29]. Besides, in the 6-round attack in [25], freedom degree in the key is exploited to reduce the complexity. However, the computational chunks between the key schedule and the encryption data-path are designed to be almost identical (due to the feasibility of manual analysis). Concretely, in the attack on 6-round Whirlpool in [25], to continue the propagation of `Red`, values of 24 cells ($\overleftarrow{d_{g_r}} = 24$) are guessed. The freedom in `Blue` is 32 bytes ($\overrightarrow{d_b} = 32$); the freedom in `Red` is 8 bytes ($\overleftarrow{d_r} = 8$); and the degree of matching is 32 bytes ($\overrightarrow{\overleftarrow{m}} = 32$). Thus, the total complexity is determined by $\min(\overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) = \min(32-24, 8-0, 32-24) = 8$, which is 2^{448}.

In contrast, in the new attack automatically found by the MILP model, as can be seen in Fig. 12, the computational chunks between the key schedule and the encryption data-path are largely different (but still share common patterns in the starting rounds). Thus, the degree of freedom in `Blue` and `Red` can be relatively more balanced, and the required number of guessing bytes is relatively less. Consequently, the complexity is further improved (to 2^{440}).

To extend one more attacked round to 7-round, additional strategies are required on top of those appeared in [25]. As can be seen in Fig. 3 and 10, in round 5, the equivalent round-key #KMC should be added before MixColumns to enable the cancellation of impacts from `Red` cells on `Blue`-attribute propagation to the forward. Moreover, complex constraints (cover two rounds across non-linear operations) must be imposed on `Blue` cells to cancel their impacts on the `Red`-attribute propagation to the backward. Non-trivial procedure (*e.g.*, the local meet-in-the-middle procedure in Algorithm 1) for obtaining initial values of `Blue` neutral cells fulfilling the non-linear constraints is also necessary here.

Overall, the improvements obtained over [5] are the combined effort of the new techniques including guess-and-determine and relaxed model, and model optimization utilizing the similarity between key-schedule and encryption.

5 Application to Grøstl

5.1 Description of Grøstl

Grøstl, proposed by Gauravaram *et al.* is one of the five finalists of SHA-3 competition hosted by NIST. Grøstl adopts a double-pipe design, *i.e.*, the size of the chaining value, which is $2n$-bit, is twice as the hash size, which is n-bit. For Grøstl-256, the hash size is 256 bits, and for Grøstl-512, it is 512 bits.

Two $2n$-bit AES-like permutations P and Q are employed to build the compression function (CF) and output transformation (OT). The P and Q work on 8×8 sized state for Grøstl-256 and 8×16 sized state for Grøstl-512. For Grøstl-256, they consist of 10 AES-like rounds; and for Grøstl-512, they consist of 14 AES-like rounds. Concretely, the round function of P and Q is made up of 4 operations, *i.e.*, SubBytes (SB), ShiftRows (SR), MixColumns (MC), AddRoundConstants (AC). The ShiftRows (SR) of P cyclically shifts the i-th row leftwards for j bytes. For P of Grøstl-256, j is in (0,1,2,3,4,5,6,7), while for P of Grøstl-512, j is in (0,1,2,3,4,5,6,11). Since the ShiftRows (SR) operation for P and Q is different and Q is not involved in our attack, details of Q are skipped.

The compression function of Grøstl built from P and Q is written as: $H_i = P(H_{i-1} \oplus m_i) \oplus Q(m_i) \oplus H_{i-1}$ $(i \geq 1)$, where H_i is the chaining value, and m_i is the message block. After processing all the message blocks, the last chaining value serves as the input of the output transformation, which is

$$\Omega(X) = Trunc_n(P(X) \oplus X).$$

The right half of $P(x) \oplus X$ is the output hash value (refer to Fig. 4).

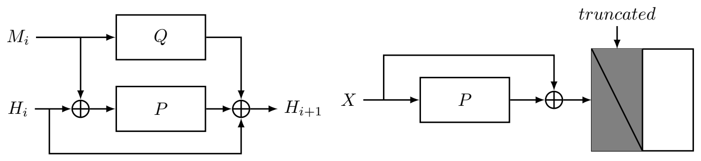


Fig. 4: Grøstl's compression function (CF) and output transformation (OT)

5.2 New Attacks Resulted from Applying the MILP Modeling

Applying the MILP models described in Sect. 3 on the OT (resp. $P(H_i) \oplus H_i$ in the CF) of Grøstl-256 and Grøstl-512 [6], new attacks are found on 6-round (refer to Fig. 5, 6) and 8-round (refer to Fig. 7, 8), respectively. Besides, many

[6] Should omit the modeling for AddRoundKey and add constraints for truncation. Besides, because of the truncation, matching point has only three choices, *i.e.*, the first, the second last, or the last rounds.

efficient attacks on shorter rounds are also found (refer to Fig. 15, 16 for 5-round Grøstl-256, Fig. 19, 20 for 6-round Grøstl-512, Fig. 17, 18 for 7-round Grøstl-512).

For the 6-round attack on the OT of Grøstl-256, guess-and-determine is the essential that enables to cover one more attacked round than previous 5-round attack in [29]; allowing one attribute (Blue and Red) propagating in both directions is the essential that enables better complexity than previous 6-round attack in [18]. Besides, many inferior attacks than the presented best one on 6-round Grøstl-256 are also found. For those inferior attacks, the computation of initial values of neutral bytes is relatively easier, but the complexity of the entire attack is higher. Thus, a non-trivial procedure to compute the initial structure (initial values of neutral bytes) is essential for achieving the best complexity.

For the 8-round attack on the OT of Grøstl-512, guess-and-determine is the essential that enables to achieve better complexity than the previous 8-round attack [29]. Besides, compared to the 8-round attack in [29], in the presented attack, the initial structure covers one more round (4 rounds) by allowing one attribute-propagation conceding to the opposite attribute-propagation in both directions.

The following of this section describes how to use the resulted attack configurations to launch concrete attacks on 6-round OT of Grøstl-256 and 8-round OT of Grøstl-512. Brief discussions on conversions to pseudo-preimage attacks on the hash function using both attacks on OT and on CF are than followed. Please refer to Sect.C for a summary of notations.

The attack on 6-round OT of Grøstl-256. Fig. 5 shows one of the attack configurations on 6-round OT of Grøstl-256. Firstly, one launch a precomputation to compute initial values of neutral bytes.

To compute initial values of neutral bytes in ***Red*** one can adopt a local meet-in-the-middle procedure as specified in Algorithm 2, which is similar to Algorithm 1.

Concretely, in the configuration of the attack in Fig. 5, the degree of freedom for the forward computation is the bottleneck; there are four bytes extra degrees of freedom for the backward that can be fixed for the ease of computing initial values of backward neutral bytes (in Red). Thus, we fix the value of four Red cells in $\#\mathbf{MC}^2$, *i.e.*, $\#\mathbf{MC}^2_{\texttt{col}_0}[6,7]$, $\#\mathbf{MC}^2_{\texttt{col}_1}[5,6]$, as indicated by 🟥. In Algorithm 2, the focus is on cancellation constraints among states $\{\#\mathbf{MC}^2, \#\mathbf{AC}^2, \#\mathbf{SB}^3, \#\mathbf{MC}^3, \#\mathbf{AC}^3\}$. The procedure starts from guessing eight free cells in the first two columns of $\#\mathbf{MC}^2$ and four cells in each pair of columns in $\#\mathbf{MC}^3$, computes other undetermined cells using constant impacts in $\#\mathbf{AK}^3$ column-by-column, and compute back octuple-wisely to match at constant cells in $\#\mathbf{MC}^2$. The complexity of this procedure is 2^{128} computations and 2^{128} blocks of memory.

The Whole Attack Procedure (refer to Fig. 5). First, fix the value of 16 Gray cells in $\#\mathbf{MC}^2_{\texttt{col}_{\{0,1,3,4,5,6\}}}$ (including the four 🟥 cells $\#\mathbf{MC}^2_{\texttt{col}_0}[6,7]$, $\#\mathbf{MC}^2_{\texttt{col}_1}[5,6]$); fix the value of 16 impacts on C-marked Blue cells in $\#\mathbf{AC}^3$. With these constants, compute the initial values of neutral bytes in Red by Algorithm 2, and obtain a hash table $\overleftarrow{\mathcal{T}}_{\texttt{Init}}$.

1. For each possible value of the 12 Gray cells in $\#\mathbf{MC}^2_{\texttt{col}_{\{2,7\}}}$,
   (a) for each value $\overleftarrow{v}_i$ of Red neutral bytes in $\overleftarrow{\mathcal{T}}_{\texttt{Init}}$, *i.e.*, Red cells in $\#\mathbf{AC}^2$,
      i. with values of Gray cells in $\#\mathbf{MC}^2_{\texttt{col}_{\{2,7\}}}$ (the constraints imposed by other Gray cells in $\#\mathbf{MC}^2$ have already been fulfilled during the precomputation of $\overleftarrow{v}_i$), compute backward to $\#\mathbf{AC}^0$;
      ii. from $\#\mathbf{AC}^0_{\texttt{SR}^{-1}(\texttt{col}_{\{1,3,4,5,6,7\}})}$, derive value $\overleftarrow{v}_m$ of 16 bytes for matching through MC (2 bytes in each column, refer to [5, 22] for matching through MC);
      iii. store $\overleftarrow{v}_i$ into a look-up table $\overleftarrow{\mathcal{T}}$, with $\overleftarrow{v}_m$ as the index.
   (b) for each value $\overrightarrow{v}_i$ of the four Blue neutral bytes in $\#\mathbf{MC}^2$,
      i. with values of Gray cells in $\#\mathbf{MC}^2$, compute backward to get values of Blue cells in $\#\mathbf{AC}^0$;
      ii. with values of Gray cells in $\#\mathbf{MC}^2$, compute forward to get values of Blue cells in $\#\mathbf{AC}^2$ and $\#\mathbf{AC}^3$;
      iii. XOR the value of constant impacts on C-marked Blue cells in $\#\mathbf{AC}^3$; compute forward to $\#\mathbf{MC}^5$,
      iv. for each value $\overrightarrow{v}_g$ of the 12 Cyan cells in $\#\mathbf{MC}^5$,
         A. compute the two Blue columns, XOR with the given target, compute forward the Blue cells through feed-forward to $\#\mathbf{MC}^0$;
         B. from values of Blue cells in both $\#\mathbf{MC}^0$ and $\#\mathbf{AC}^0$, derive a value $\overrightarrow{v}_m$ of 16 bytes for matching (*i.e.*, 2 bytes in each column);
         C. for each value of Red cells in $\#\mathbf{AC}^2$ that is stored in entry $\overleftarrow{\mathcal{T}}[\overrightarrow{v}_m]$,
         - with values of Blue cells in $\#\mathbf{AC}^2$, compute forward to $\#\mathbf{MC}^5$; get the value $\overrightarrow{v}'_g$ of cells at the position of Cyan cells, if $\overrightarrow{v}'_g \neq \overrightarrow{v}_g$, go to Step 1(b)iv;
         - compute forward to get values of all cells without hatching-mark in $\#\mathbf{AC}^5$, XORing the given target at the feed-forward point), compute through feed-forward to $\#\mathbf{MC}^0$, - if the values of White cells without hatching-marks are fully matched with values of Blue and Red cells through MixColumns (values of cells with hatching-marks in $\#\mathbf{MC}^0$ can be chosen), a full truncated-state match found, output and return; otherwise, go to Step 1;

Complexity. The memory complexity is 2^{128} (blocks). As for the computational complexity, it is determined by the following configuration of this attack, *i.e.*, $n = 8 \times 4 = 32$ (truncated 4 out of the 8 columns), $\overrightarrow{d_b} = 4$, $\overleftarrow{d_r} = 16$ (minus 4 from that shown in Fig. 5), $\overrightarrow{d_{g_b}} = 12$, $\overrightarrow{\overleftarrow{m}} = 16$, $\varsigma = 2^8$, and $\overleftarrow{d_{g_r}} = \overleftrightarrow{d_{g_{br}}} = 0$. Accordingly, the computational complexity of this attack on 6-round Grøstl-256 is $\varsigma^{32} \cdot \max(\varsigma^{-(4)}, \varsigma^{-(16-12)} + \varsigma^{-(16-12)}) = 2^{28 \cdot 8} = 2^{224}$.

Note that, the required number of repetitions of the main MITM procedure is $\varsigma^{n-\overrightarrow{d_b}-\overleftarrow{d_r}}$, *i.e.*, ς^{12} for this attack configuration. Thus, enumerating all possible values of 12 `Gray` cells in $\#\mathbf{MC}^{2}_{\mathtt{col}_{\{2,7\}}}$, as did in Step 1, is sufficient. This condition enables to fix values of all other `Gray` cells in $\#\mathbf{MC}^2$ and all 16 constant impacts on `C`-marked `Blue` cells in $\#\mathbf{AC}^3$, as did at the very beginning of the attack.

The attack on 8-round OT of Grøstl-512. Fig. 7 shows one of the attack configurations on 8-round OT of Grøstl-512.

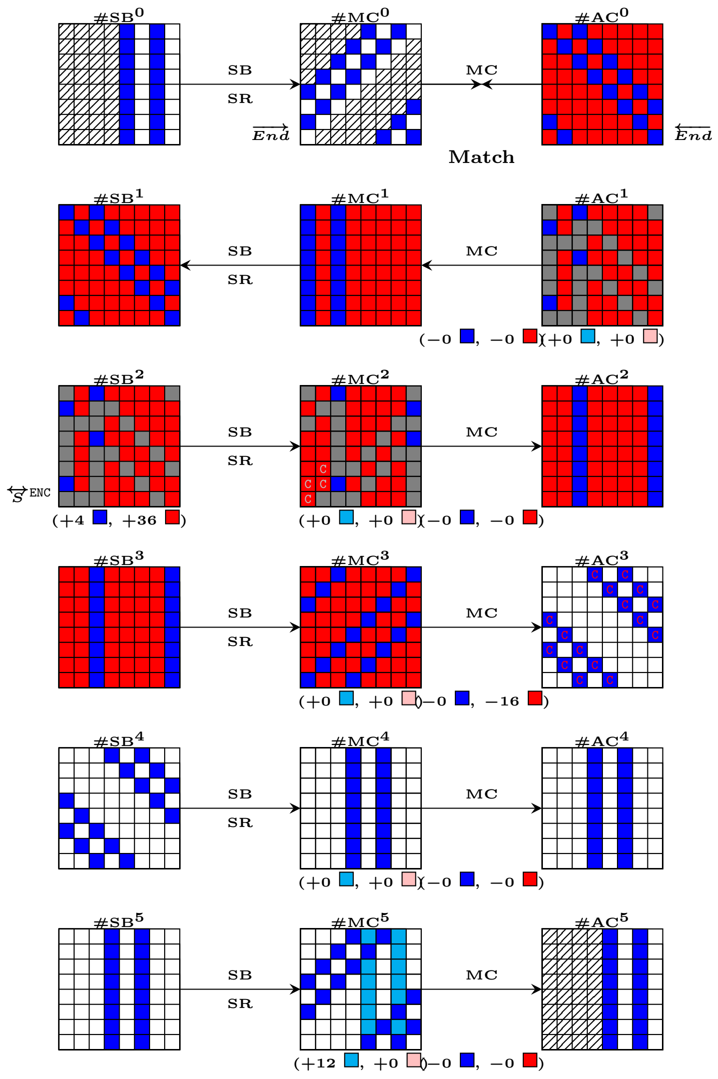
The whole attack requires a precomputation phase to obtain initial values of neutral bytes in `Blue` and in `Red` for various values of `Gray` cells and values of impacts on cells of opposite attributes. Compared to the procedure of Algorithm 2 in the attack on 6-round Grøstl-256, the precomputation procedure in here is simpler. Concrete procedures can be found in Algorithm 3 and 4. The complexity of the precomputation for obtaining the values of neutral bytes in `Blue` is $2^{28\cdot 8}$, *i.e.*, 2^{224} computations and 2^{224} blocks of memory. That for neutral bytes in `Red` is $2^{16\cdot 8}$, *i.e.*, 2^{128} computations and 2^{128} blocks of memory.

During the main attack procedure, the resulted look-up tables $\overrightarrow{\mathcal{T}_{\mathtt{Init}}}$ and $\overleftarrow{\mathcal{T}_{\mathtt{Init}}}$ from Algorithm 3 and 4 will be used to retrive the initial values of neutral bytes under various values of `Gray` bytes and values of impacts.

The Main Attack Procedure (refer to Fig. 7).

1. For each possible value of `Gray` cells in $\#\mathbf{MC}^4$, $\#\mathbf{AC}^4$, $\#\mathbf{AC}^5$, and value of impacts from `Blue` on `C`-marked `Red` cells in $\#\mathbf{MC}^3$, and value of impacts from `Red` on `C`-marked `Blue` cells in $\#\mathbf{AC}^6$,
   (a) compute the value of `Gray` cells in $\#\mathbf{MC}^5$ from that in $\#\mathbf{AC}^4$ (cell-by-cell)
   (b) use the value of $\#\mathbf{AC}^4[{}_{3}7, {}_{4}7, {}_{5}7, {}_{6}7, {}_{8}0]$ $\|$ $\#\mathbf{MC}^3[{}_{6}0, {}_{6}1, {}_{6}7, {}_{7}0, {}_{7}7, {}_{8}6, {}_{8}7, {}_{9}5, {}_{9}6, {}_{9}7, {}_{10}4, {}_{10}5, {}_{10}6, {}_{10}7]$ [7] to look up the table $\overrightarrow{\mathcal{T}_{\mathtt{Init}}}$, retrive the value of 28 `Blue` neutral cells in $\#\mathbf{MC}^4$.
   (c) for each value $\overrightarrow{v}_i$ of the 28 `Blue` neutral cells in $\#\mathbf{MC}^4$,
      i. with the value of `Gray` cells in $\#\mathbf{MC}^4$, $\#\mathbf{AC}^4$, $\#\mathbf{AC}^5$, compute forward to $\#\mathbf{AC}^6$; `XOR` with values of pre-determined impacts on `C`-marked cells in $\#\mathbf{AC}^6$, compute forward to $\#\mathbf{AC}^7$;
      ii. `XOR` with the value of given target T (truncated out half of the state), compute through feed-forward to $\#\mathbf{MC}^0$;
      iii. derive the value $\overrightarrow{v}_m$ of 10 (*i.e.*, 2+3+3+2) bytes from $\#\mathbf{MC}^{0}_{\mathtt{col}_{\{7,8,9,10\}}}$ for matching through `MC` (refer to [5, 22]);
      iv. store $\overrightarrow{v}_i$ into a look-up table $\overrightarrow{\mathcal{T}}$, with $\overrightarrow{v}_m$ as the index.
   (d) use the values of $\#\mathbf{AC}^6[{}_{5}6, {}_{5}7, {}_{6}7, {}_{7}7, {}_{8}0, {}_{8}7]$ $\|$ $\#\mathbf{MC}^5[{}_{8}0, {}_{8}7, {}_{9}7, {}_{10}7, {}_{11}7]$ to look up the table $\overleftarrow{\mathcal{T}_{\mathtt{Init}}}$, retrive the values of 16 `Red` neutral cells in $\#\mathbf{AC}^5$.
   (e) for each value $\overleftarrow{v}_i$ of `Red` neutral cells in $\#\mathbf{AC}^5$,
      i. with the value of `Gray` cells in $\#\mathbf{AC}^5$, $\#\mathbf{MC}^5$, $\#\mathbf{AC}^4$, $\#\mathbf{MC}^4$, compute backward to $\#\mathbf{MC}^3$; `XOR` with values of pre-determined impacts on `C`-marked cells in $\#\mathbf{MC}^3$, compute forward to $\#\mathbf{AC}^1$;

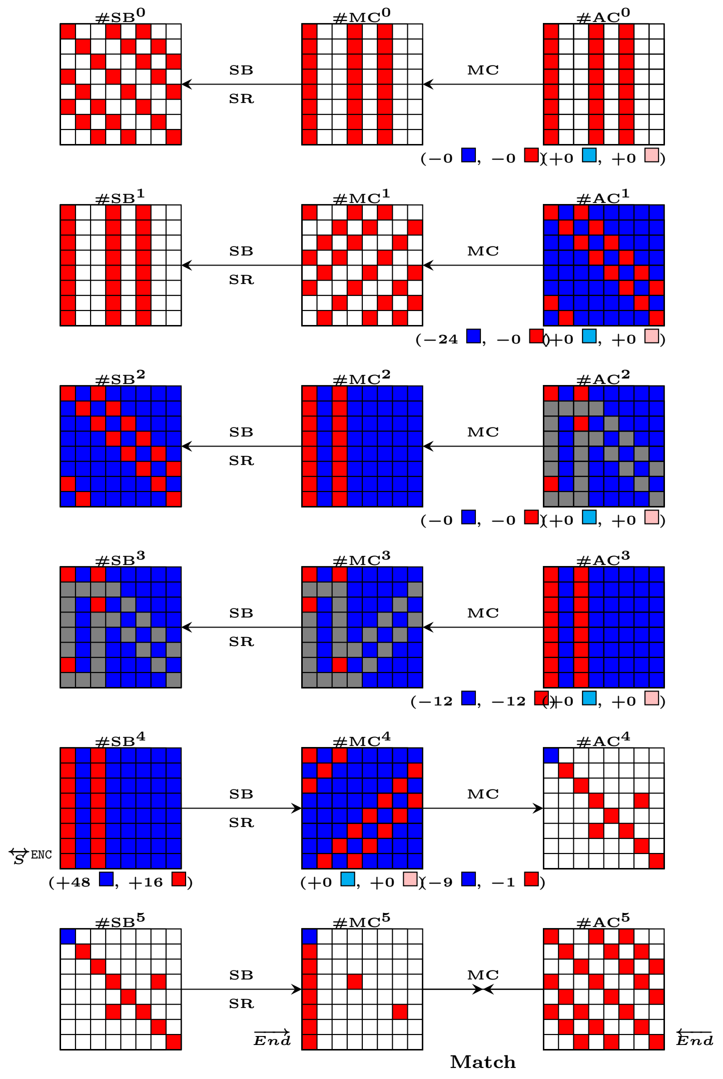
[7] The notation $S[{}_{\mathtt{i}}\mathtt{j}]$ is used to represent the j-th cell in the i-th column of a state S, which is the shorten for $S_{\mathtt{col}_i}[j]$.



Config

- $(\overrightarrow{\iota}, \overleftarrow{\iota}) = (4$ 🟦, 36 🟥$)$
- $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}}) = (4$ 🟦, 20 🟥, 16 🟪, 12 🟦, 0 🟥, 0 🟩$)$
- $(\overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) = (4, 8, 4)$

Fig. 5: An example of the MITM attack on the OT of the 6-round Grøstl-256



Config

- $(\overrightarrow{\iota}, \overleftarrow{\iota}) = (48$ 🟦, 16 🟥$)$
- $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}}) = (3$ 🟦, 3 🟥, 3 🟪, 0 🟦, 0 🟥, 0 🟩$)$
- $(\overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) = (3, 3, 3)$

Fig. 6: An example of the MITM attack on $P(H') \oplus H'$ in the CF of the 6-round Grøstl-256 (without guessing)

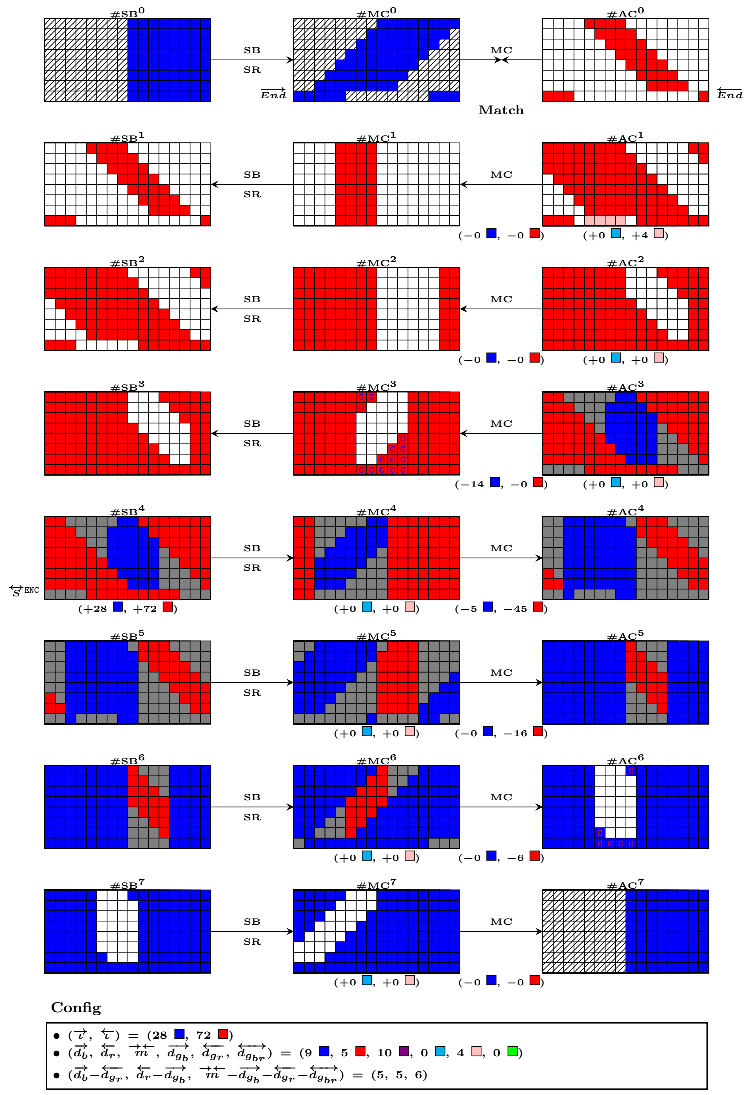

Config
• $(\overrightarrow{\iota}, \overleftarrow{\iota})$ = (28 🟦, 72 🟥)
• $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}})$ = (9 🟦, 5 🟥, 10 🟪, 0 🟦, 4 🟥, 0 🟩)
• $(\overrightarrow{d_b}-\overrightarrow{d_{g_r}}, \overleftarrow{d_r}-\overleftarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})$ = (5, 5, 6)

Fig. 7: An example of the MITM attack on the OT of the 8-round Grøstl-512

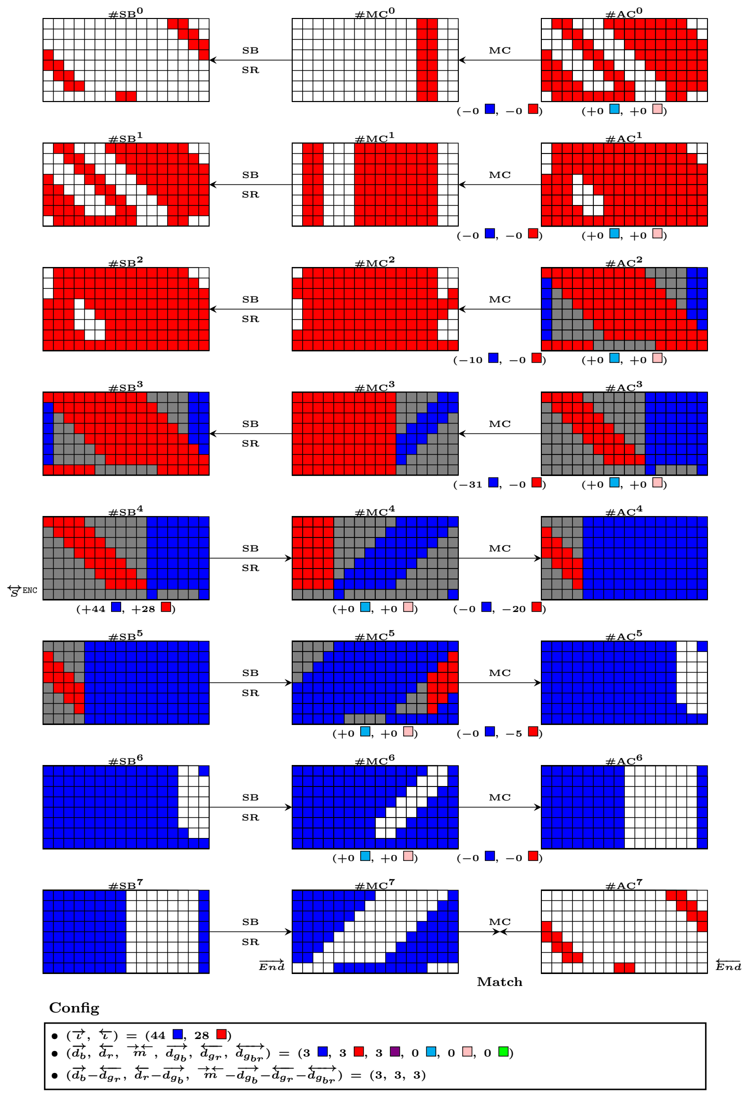

Config
• $(\overrightarrow{\iota}, \overleftarrow{\iota})$ = (44 🟦, 28 🟥)
• $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}})$ = (3 🟦, 3 🟥, 3 🟪, 0 🟦, 0 🟥, 0 🟩)
• $(\overrightarrow{d_b}-\overrightarrow{d_{g_r}}, \overleftarrow{d_r}-\overleftarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})$ = (3, 3, 3)

Fig. 8: An example of the MITM attack on $P(H') \oplus H'$ in the CF of the 8-round Grøstl-512 (without guessing)

ii. for each value $\overleftarrow{v}_g$ of the 4 Pink cells in $\#\mathrm{AC}^1$

A. compute backward to $\#\mathrm{AC}^0$; derive the value $\overleftarrow{v}_m$ of 10 bytes from $\#\mathrm{AC}^0_{\mathtt{col}_{\{7,8,9,10\}}}$ for matching through MC;

B. for each value $\overrightarrow{v}_i$ stored in entry $\overrightarrow{\mathcal{T}}[\overleftarrow{v}_m]$,

- with values of Red and Gray cells in $\#\mathrm{MC}^4$, compute backward to $\#\mathrm{AC}^1$; get the value $\overleftarrow{v}'_g$ of cells at the position of Pink cells, if $\overleftarrow{v}'_g \neq \overleftarrow{v}_g$, go to Step 1(e)ii;
- compute backward to $\#\mathrm{AC}^0$, denote the value by $\overleftarrow{v}$;
- with values of Red and Gray cells in $\#\mathrm{MC}^4$, compute forward (XORing the given target at the feed-forward point) to $\#\mathrm{MC}^0$, denote the value of cells without hatching-marks by $\overrightarrow{v}$; if $\overrightarrow{v}$ and $\overleftarrow{v}$ can be fully matched through MixColumns (values of cells with hatching-marks in $\#\mathrm{MC}^0$ can be chosen), a full truncated-state match found, output and return; otherwise, go to Step 1;

Complexity. Due to the precomputation, the memory complexity is 2^{224} (blocks). As for the computational complexity, it is determined by the following configuration of this attack. $n = 8 \times 8 = 64$ (truncated 8 out of the 16 columns), $\overrightarrow{d_b} = 9$, $\overleftarrow{d_r} = 5$, $\overleftarrow{d_{g_r}} = 4$, $\overrightarrow{\overleftarrow{m}} = 10$, $\varsigma = 2^8$, and $\overleftarrow{d_{g_r}} = \overleftrightarrow{d_{g_{br}}} = 0$. Accordingly, the computational complexity of this attack on 8-round Grøstl-512 is $\varsigma^{64} \cdot \max(\varsigma^{-(9-4)}, \varsigma^{-(5)}, \varsigma^{-(10-4)}) = 2^{59 \cdot 8} = 2^{472}$.

Conversion to Pseudo-Preimage Attacks. The attack procedures presented above are on the OT of Grøstl. They can be converted into pseudo-preimage attacks on Grøstl combining with similar attack procedures on the $P(H) \oplus H$ of the CF using the conversion method in [29]. The complexity of the converted pseudo-preimage attacks are summarized in Table 1. More details can be found in F.

5.3 Discussions on the New Attacks

An interesting feature of the presented attack on 6-round Grøstl-256 is that, with necessary guessing, the computation of Blue covers the full 6-round. That is, the propagation of Blue also requires computing to backward and contributes to increasing degrees of matching. Besides, like the attack on 7-round Whirlpool, a non-trivial local meet-in-the-middle procedure to compute initial values of neutral bytes is also necessary.

Note that, obtaining the previous best attacks on Grøstl, the work in [29] has already been assisted with automatic searching and the 5-round attack on Grøstl-256 in [29] was claimed to be optimal. However, except for lacking of the guess-and-determine technique, the search space is also limited. The presented 5-round attacks on Grøstl-256 in Fig. 15 and 16 achieve better compelxity than that in [29] due to allowing the propagation of Blue and Red to both directions while do not involve guessing. Overall, the improvements on Grøstl in this work are results of the combination of the integration of new techniques, relaxed model, and a powerful off-the-shelf solver.

6 Conclusions

In [5] Bao *et al.* introduced the Mixed-Integer-Linear-Programming into the search of best configurations for the MITM preimage attacks against AES-like hashing. Based on their work, we introduced more techniques such as guess-and-determine, relaxed modeling, and round-dependent modeling in order to enlarge the configuration search space and hence to find better attacks. As a result, we improved the best preimage attacks against Whirlpool and AES-256 from 6 to 7 and 9 to 10 rounds, and lowered the attack complexities of 5 and 6-round Whirlpool, 5 and 6-round Grøstl-256, and 8-round Grøstl-512. The results are possible due the combination of new techniques which enlarged the search space, and a set of model optimizations for the MILP program to output in real time.

References

1. Alliance, ZigBee. ZigBee 2007 specification. *Online: http://www.zigbee.org/*, 2007.
2. R. AlTawy and A. M. Youssef. Preimage Attacks on Reduced-Round Stribog. In D. Pointcheval and D. Vergnaud, editors, *AFRICACRYPT 14*, volume 8469 of *LNCS*, pages 109–125. Springer, Heidelberg, May 2014.
3. 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.
4. Z. Bao, L. Ding, J. Guo, H. Wang, and W. Zhang. Improved Meet-in-the-Middle Preimage Attacks against AES Hashing Modes. *IACR Trans. Symm. Cryptol.*, 2019(4):318–347, 2019.
5. Z. Bao, X. Dong, J. Guo, Z. Li, D. Shi, S. Sun, and X. Wang. Automatic Search of Meet-in-the-Middle Preimage Attacks on AES-like Hashing. Cryptology ePrint Archive, Report 2020/467, 2020. https://eprint.iacr.org/2020/467.
6. P. S. L. M. Barreto and V. Rijmen. The WHIRLPOOL Hashing Function. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.529.3184&rep=rep1&type=pdf, 2000. Revised in 2003.
7. A. Bogdanov and C. Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *SAC 2010*, volume 6544 of *LNCS*, pages 229–240. Springer, Heidelberg, Aug. 2011.
8. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
9. B. Debraize and L. Goubin. Guess-and-Determine Algebraic Attack on the Self-Shrinking Generator. In K. Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 235–252. Springer, Heidelberg, Feb. 2008.
10. O. Dunkelman, N. Keller, and A. Shamir. Improved Single-Key Attacks on 8-Round AES-192 and AES-256. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 158–176. Springer, Heidelberg, Dec. 2010.
11. 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.

12. 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.
13. J. Guo, T. Peyrin, and A. Poschmann. The PHOTON Family of Lightweight Hash Functions. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 222–239. Springer, Heidelberg, Aug. 2011.
14. J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The LED Block Cipher. In B. Preneel and T. Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 326–341. Springer, Heidelberg, Sept. / Oct. 2011.
15. J. Guo, C. Su, and W. Yap. An Improved Preimage Attack against HAVAL-3. *Inf. Process. Lett.*, 115(2):386–393, 2015.
16. P. Hawkes and G. G. Rose. Guess-and-Determine Attacks on SNOW. In K. Nyberg and H. M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 37–46. Springer, Heidelberg, Aug. 2003.
17. 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.
18. B. Ma, B. Li, R. Hao, and X. Li. Improved (Pseudo) Preimage Attacks on Reduced-Round GOST and Grøstl-256 and Studies on Several Truncation Patterns for AES-like Compression Functions. In K. Tanaka and Y. Suga, editors, *IWSEC 15*, volume 9241 of *LNCS*, pages 79–96. Springer, Heidelberg, Aug. 2015.
19. R. C. Phan. Mini Advanced Encryption Standard (Mini-AES): a Testbed for Cryptanalysis Students. *Cryptologia*, 26(4):283–306, 2002.
20. 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.
21. S. Sadeghi, T. Mohammadi, and N. Bagheri. Cryptanalysis of Reduced round SKINNY Block Cipher. *IACR Trans. Symm. Cryptol.*, 2018(3):124–162, 2018.
22. 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.
23. 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.
24. 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.
25. Y. Sasaki, L. Wang, S. Wu, and W. Wu. Investigating Fundamental Security Requirements on Whirlpool: Improved Preimage and Collision Attacks. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 562–579. Springer, Heidelberg, Dec. 2012.
26. 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.
27. 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.

28. L. Wei, C. Rechberger, J. Guo, H. Wu, H. Wang, and S. Ling. Improved Meet-in-the-Middle Cryptanalysis of KTANTAN (Poster). In U. Parampalli and P. Hawkes, editors, *ACISP 11*, volume 6812 of *LNCS*, pages 433–438. Springer, Heidelberg, July 2011.
29. 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.
30. B. Zhang and D. Feng. New Guess-and-Determine Attack on the Self-Shrinking Generator. In X. Lai and K. Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 54–68. Springer, Heidelberg, Dec. 2006.
31. J. Zou, W. Wu, S. Wu, and L. Dong. Improved (Pseudo) Preimage Attack and Second Preimage Attack on Round-Reduced Grostl Hash Function. *J. Inf. Sci. Eng.*, 30(6):1789–1806, 2014.

A Concrete Inequalities modeling the Propagation rules

Inequalities for `MC-RULE` *that allowing each attribute of* `Blue` *and* `Red` *conceding to the opposite attribute propagation by consuming freedom of itself in both directions.*

$$\begin{cases} \boldsymbol{\omega} = \max_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} (\omega_i^I), \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} x_i^I) - \mathrm{N}_{\mathtt{row}} \cdot \boldsymbol{x} \geq 0, \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} x_i^I) - \boldsymbol{x} \leq \mathrm{N}_{\mathtt{row}} - 1. \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} {y_i}^I) - \mathrm{N}_{\mathtt{row}} \cdot \boldsymbol{y} \geq 0, \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} {y_i}^I) - \boldsymbol{y} \leq \mathrm{N}_{\mathtt{row}} - 1. \end{cases} \tag{11}$$

$$\begin{cases} (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} x_i^O) + \mathrm{N}_{\mathtt{row}} \cdot \boldsymbol{\omega} \leq \mathrm{N}_{\mathtt{row}}, \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} y_i^O) + \mathrm{N}_{\mathtt{row}} \cdot \boldsymbol{\omega} \leq \mathrm{N}_{\mathtt{row}}, \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} (x_i^I + x_i^O)) - \mathtt{Br}_{\mathtt{n}} \cdot \boldsymbol{x} \leq (\mathrm{N}_{\mathtt{iosum}} - \mathtt{Br}_{\mathtt{n}}), \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} (x_i^I + x_i^O)) - \mathrm{N}_{\mathtt{iosum}} \cdot \boldsymbol{x} \geq 0, \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} (y_i^I + y_i^O)) - \mathtt{Br}_{\mathtt{n}} \cdot \boldsymbol{y} \leq (\mathrm{N}_{\mathtt{iosum}} - \mathtt{Br}_{\mathtt{n}}), \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} (y_i^I + y_i^O)) - \mathrm{N}_{\mathtt{iosum}} \cdot \boldsymbol{y} \geq 0, \end{cases} \tag{12}$$

$$\begin{cases} (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} y_i^O) - c_{\boldsymbol{x}} \geq 0 \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} y_i^O) - \mathrm{N}_{\mathtt{row}} \cdot \boldsymbol{y} - c_{\boldsymbol{x}} \leq 0 \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} y_i^O) + \mathrm{N}_{\mathtt{row}} \cdot \boldsymbol{y} + c_{\boldsymbol{x}} \leq \mathrm{N}_{\mathtt{iosum}} \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} x_i^O) - c_{\boldsymbol{y}} \geq 0 \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} x_i^O) - \mathrm{N}_{\mathtt{row}} \cdot \boldsymbol{x} - c_{\boldsymbol{y}} \leq 0 \\ (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} x_i^O) + \mathrm{N}_{\mathtt{row}} \cdot \boldsymbol{x} + c_{\boldsymbol{y}} \leq \mathrm{N}_{\mathtt{iosum}} \end{cases} \tag{13}$$

Note that, $\mathrm{N}_{\mathtt{iosum}} - (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} (x_i^I + x_i^O))$ is the total number of `Red` or `White` cells (*i.e.*, active cells as for `Red`) in the input and output columns. Similarly, $\mathrm{N}_{\mathtt{iosum}} - (\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} (y_i^I + y_i^O))$ is the total number of `Blue` or `White` cells (*i.e.*, active cells as for `Blue`). They are constrained by the branch number $\mathtt{Br}_{\mathtt{n}}$ of MixColumns.

Inequalities for `MATCH-RULE`*.* For concrete formalization of `MATCH-RULE`, the involved variables include the binary variables that indicate the attribute of each cell in the input and output columns (the corresponding columns in $\overrightarrow{End}$

and $\overleftarrow{End}$ and also the key state $\overleftarrow{End}^{\mathrm{K}}$ or $\overrightarrow{End}^{\mathrm{KMC}}$), *i.e.*, (x_i^I, y_i^I), (x_i^O, y_i^O), (x_i^K, y_i^K), and auxiliary binary variables ω_i^I, ω_i^O, ω_i^K, $\omega_i^{E\oplus K}$, $x_i^{E\oplus K}$, $y_i^{E\oplus K}$ for $i \in \{0, 1, \cdots, \mathrm{N}_{\texttt{row}} - 1\}$. Besides, one need the general variable $\overset{\rightarrow\leftarrow}{m}_i$ to represent the degree of matching in column i. Apart from those variables, the following auxiliary binary variables are introduced to indicating the following attributes of input/output columns (after the last AddRoundKey):

- $\overrightarrow{e_b}$: whether exist `Blue` cells (🟦);
- $\overleftarrow{e_r}$: whether exist `Red` cells (🟥);
- $\overline{e_m}$: whether exist more than $\mathrm{N}_{\texttt{row}}$ known cells (🟦, 🟥, or ⬛);
- $\overset{\rightarrow\leftarrow}{e_f}$: whether exist filtering ability, *i.e.*, $\overset{\rightarrow\leftarrow}{e_f} = \texttt{AND}(\overrightarrow{e_b}, \overleftarrow{e_r}, \overline{e_m})$.

Suppose $\overrightarrow{End}^{\mathrm{KMC}}$ has fewer `White` cells than $\overleftarrow{End}^{\mathrm{K}}$ and thus using $\overrightarrow{End} \oplus \overrightarrow{End}^{\mathrm{KMC}}$ as the key addition, then, the concrete constraints on $\overset{\rightarrow\leftarrow}{m}_i$ are as in Eq. (14).

$$\begin{cases} \omega_i^{E\oplus K} = \texttt{OR}(\omega_i^I, \omega_i^K), \\ x_i^{E\oplus K} = \texttt{AND}(x_i^I, x_i^K), \\ y_i^{E\oplus K} = \texttt{AND}(y_i^I, y_i^K). \end{cases} \quad \overset{\rightarrow\leftarrow}{m}_i = \begin{cases} \mathrm{N}_{\texttt{row}} - \sum_{i=0}^{\mathrm{N}_{\texttt{row}}-1} (\omega_i^{E\oplus K} + \omega_i^O), & \overset{\rightarrow\leftarrow}{e_f} = 1; \\ 0, & \overset{\rightarrow\leftarrow}{e_f} = 0. \end{cases}$$

$$\begin{cases} \overrightarrow{e_b} + \sum_{i=0}^{\mathrm{N}_{\texttt{row}}-1} (\omega_i^{E\oplus K} + \omega_i^O + y_i^{E\oplus K} + y_i^O) \le \mathrm{N}_{\texttt{iosum}}, \\ \mathrm{N}_{\texttt{iosum}} \cdot \overrightarrow{e_b} + \sum_{i=0}^{\mathrm{N}_{\texttt{row}}-1} (\omega_i^{E\oplus K} + \omega_i^O + y_i^{E\oplus K} + y_i^O) \ge \mathrm{N}_{\texttt{iosum}}, \\ \overleftarrow{e_r} + \sum_{i=0}^{\mathrm{N}_{\texttt{row}}-1} (\omega_i^{E\oplus K} + \omega_i^O + x_i^{E\oplus K} + x_i^O) \le \mathrm{N}_{\texttt{iosum}}, \\ \mathrm{N}_{\texttt{iosum}} \cdot \overleftarrow{e_r} + \sum_{i=0}^{\mathrm{N}_{\texttt{row}}-1} (\omega_i^{E\oplus K} + \omega_i^O + x_i^{E\oplus K} + x_i^O) \ge \mathrm{N}_{\texttt{iosum}}, \\ (\mathrm{N}_{\texttt{row}} + 1) \cdot \overline{e_m} + \sum_{i=0}^{\mathrm{N}_{\texttt{row}}-1} (\omega_i^{E\oplus K} + \omega_i^O) \le \mathrm{N}_{\texttt{iosum}}, \\ \mathrm{N}_{\texttt{row}} \cdot \overline{e_m} + \sum_{i=0}^{\mathrm{N}_{\texttt{row}}-1} (\omega_i^{E\oplus K} + \omega_i^O) \ge \mathrm{N}_{\texttt{row}}, \\ \overset{\rightarrow\leftarrow}{e_f} = \texttt{AND}(\overrightarrow{e_b}, \overleftarrow{e_r}, \overline{e_m}). \end{cases} \tag{14}$$

B Compute Initial values of Neutral Bytes in the Attacks

Property 1 (Property of MDS matrix). Known any $\mathrm{N}_{\texttt{row}}$ out of the $2 \cdot \mathrm{N}_{\texttt{row}}$ input and output elements of the MDS matrix, the remaining $\mathrm{N}_{\texttt{row}}$ elements can be computed.

Algorithm 1 Compute initial values of forward neutral bytes (in `Blue`) in the attack in Fig. 3

```
1: procedure ComputeForwardNeutralBytes
2:     $\overrightarrow{\mathcal{T}_{\mathtt{Init}}} \leftarrow \emptyset$
3:     for all possible values of $\#\mathbf{AK}^3[1,2]$ do                                   ▷ $2^{16}$
4:         $\#\mathbf{MC}^3[0,1,2,3] \xleftarrow{\mathtt{MC}^{-1}} \#\mathbf{AK}^3[0,1,2,3]$            ▷ $\#\mathbf{AK}^3[0,3]$ are fixed
5:         $\#\mathbf{SB}^3[0,5,10,15] \xleftarrow{\mathtt{SB}^{-1}} \#\mathbf{MC}^3[0,1,2,3]$
6:         for all possible values of $\#\mathbf{SB}^3[8]$ do                                ▷ $2^8$
7:             $\#\mathbf{SB}^3[9] \xleftarrow{\mathtt{MC}_{4671_5}} (\#\mathbf{SB}^3[8,10,11], \#\mathbf{MC}^2_{\mathtt{C}}[9])$   ▷ according to Property 1 [a]
8:             $\#\mathbf{MC}^3[8] \xleftarrow{\mathtt{SB}} \#\mathbf{SB}^3[8]$
9:             $\#\mathbf{MC}^3[5] \xleftarrow{\mathtt{SB}} \#\mathbf{SB}^3[9]$
10:            $L_{\#\mathbf{MC}^3_{8,5}} \xleftarrow{push} (\#\mathbf{MC}^3[8,5])$                 ▷ the final size $|L_{\#\mathbf{MC}^3_{8,5}}| = 2^8$
11:        end for
12:        for all possible values of $\#\mathbf{SB}^3[13]$ do                               ▷ $2^8$
13:            $\#\mathbf{SB}^3[14] \xleftarrow{\mathtt{MC}_{4570_6}} (\#\mathbf{SB}^3[12,13,15], \#\mathbf{MC}^2_{\mathtt{C}}[12])$  ▷ according to Property 1
14:            $\#\mathbf{MC}^3[9] \xleftarrow{\mathtt{SB}} \#\mathbf{SB}^3[13]$
15:            $\#\mathbf{MC}^3[6] \xleftarrow{\mathtt{SB}} \#\mathbf{SB}^3[14]$
16:            $L_{\#\mathbf{MC}^3_{9,6}} \xleftarrow{push} (\#\mathbf{MC}^3[9,6])$                 ▷ the final size $|L_{\#\mathbf{MC}^3_{9,6}}| = 2^8$
17:        end for
18:        for all possible values of $\#\mathbf{SB}^3[4]$ do                                ▷ $2^8$
19:            $\#\mathbf{SB}^3[7] \xleftarrow{\mathtt{MC}_{4562_7}} (\#\mathbf{SB}^3[4,5,6], \#\mathbf{MC}^2_{\mathtt{C}}[6])$        ▷ according to Property 1
20:            $\#\mathbf{MC}^3[4] \xleftarrow{\mathtt{SB}} \#\mathbf{SB}^3[4]$
21:            $\#\mathbf{MC}^3[11] \xleftarrow{\mathtt{SB}} \#\mathbf{SB}^3[7]$
22:            $L_{\#\mathbf{MC}^3_{4,11}} \xleftarrow{push} (\#\mathbf{MC}^3[4,11])$               ▷ the final size $|L_{\#\mathbf{MC}^3_{4,11}}| = 2^8$
23:        end for
24:        for all possible values of $\#\mathbf{SB}^3[2]$ do                                ▷ $2^8$
25:            $\#\mathbf{SB}^3[3] \xleftarrow{\mathtt{MC}_{4563_7}} (\#\mathbf{SB}^3[0,1,2], \#\mathbf{MC}^2_{\mathtt{C}}[3])$        ▷ according to Property 1
26:            $\#\mathbf{MC}^3[10] \xleftarrow{\mathtt{SB}} \#\mathbf{SB}^3[2]$
27:            $\#\mathbf{MC}^3[7] \xleftarrow{\mathtt{SB}} \#\mathbf{SB}^3[3]$
28:            $L_{\#\mathbf{MC}^3_{10,7}} \xleftarrow{push} (\#\mathbf{MC}^3[10,7])$               ▷ the final size $|L_{\#\mathbf{MC}^3_{10,7}}| = 2^8$
29:        end for
30:        for all $(\#\mathbf{MC}^3[8,5]) \in L_{\#\mathbf{MC}^3_{8,5}}$ do                           ▷ $2^8$
31:            for all $(\#\mathbf{MC}^3[9,6]) \in L_{\#\mathbf{MC}^3_{9,6}}$ do                       ▷ $2^8$
32:                $a_{\#\mathbf{AK}^3[5]} \xleftarrow{\mathtt{MC}} (0 \parallel \#\mathbf{MC}^3[5] \parallel \#\mathbf{MC}^3[6] \parallel 0)[1] \oplus \#\mathbf{AK}^3[5]$
33:                $b_{\#\mathbf{AK}^3[10]} \xleftarrow{\mathtt{MC}} (\#\mathbf{MC}^3[8] \parallel \#\mathbf{MC}^3[9] \parallel 0 \parallel 0)[2] \oplus \#\mathbf{AK}^3[10]$
34:                $T_{\#\mathbf{MC}^3_{8,5,9,6}}[a_{\#\mathbf{AK}^3[5]} \parallel b_{\#\mathbf{AK}^3[10]}] \leftarrow (\#\mathbf{MC}^3[8,5,9,6])$ ▷ expected to
    have one element in each entry of $T_{\#\mathbf{MC}^3_{8,5,9,6}}$
35:            end for
36:        end for
37:        for all $(\#\mathbf{MC}^3[4,11]) \in L_{\#\mathbf{MC}^3_{4,11}}$ do                         ▷ $2^8$
38:            for all $(\#\mathbf{MC}^3[10,7]) \in L_{\#\mathbf{MC}^3_{10,7}}$ do                     ▷ $2^8$
39:                $a'_{\#\mathbf{AK}^3[5]} \xleftarrow{\mathtt{MC}} (\#\mathbf{MC}^3[4] \parallel 0 \parallel 0 \parallel \#\mathbf{MC}^3[7])[1]$
40:                $b'_{\#\mathbf{AK}^3[10]} \xleftarrow{\mathtt{MC}} (0 \parallel 0 \parallel \#\mathbf{MC}^3[10] \parallel \#\mathbf{MC}^3[11])[2]$
41:                for all $\#\mathbf{MC}^3[8,5,9,6] \in T_{\#\mathbf{MC}^3_{8,5,9,6}}[a'_{\#\mathbf{AK}^3[5]} \parallel b'_{\#\mathbf{AK}^3[10]}]$ do    ▷
    expected to have one element in the entry
42:                    $\overrightarrow{\mathcal{T}_{\mathtt{Init}}} \xleftarrow{push} \#\mathbf{MC}^3[0,1,2,3,4,5,6,7,8,9,10,11]$        ▷ the final size
    $|\overrightarrow{\mathcal{T}_{\mathtt{Init}}}| = 2^{32}$
43:                end for
44:            end for
45:        end for
46:    end for
47:    return $\overrightarrow{\mathcal{T}_{\mathtt{Init}}}$
48: end procedure
```

[a] $\mathtt{MC}_{abcd_e}$ represents using values of the a-th, b-th, c-th, d-th cells in the array of (input $\parallel$ output) of the `MC` to derive the value of the e-th cell according to Property 1.

Algorithm 2 Compute values of backward neutral bytes (in `Red`) in Fig. 5

```
procedure ComputeBackwardNeutralBytes
    $\overleftarrow{\mathcal{T}}_{\texttt{Init}} \leftarrow \emptyset$, $\#\mathrm{MC}^3_{\texttt{SR}(\texttt{col}_{\{2,7\}})} \leftarrow \mathbf{0}$        ▷ refer to Fig. 9 for cell index
    for all possible values of $\#\mathrm{MC}^2[{}_01, {}_03, {}_04, {}_05, {}_10, {}_12, {}_13, {}_14]$ do        ▷ $2^{64}$
        $\#\mathrm{AC}^2_{\texttt{col}_{\{0,1\}}} \xleftarrow{\texttt{MC}} \#\mathrm{MC}^2_{\texttt{col}_{\{0,1\}}}$, $\#\mathrm{MC}^3_{\texttt{SR}(\texttt{col}_{\{0,1\}})} \xleftarrow{\texttt{SR}\circ\texttt{SB}\circ\texttt{AC}} \#\mathrm{AC}^2_{\texttt{col}_{\{0,1\}}}$
        for all possible values of $\#\mathrm{MC}^3[{}_03, {}_04]$, $\#\mathrm{MC}^3[{}_12, {}_13]$ do        ▷ $2^{32}$
            $\#\mathrm{MC}^3[{}_05, {}_06] \xleftarrow{\texttt{MC}^*} (\#\mathrm{MC}^3[{}_00, {}_01, {}_02, {}_03, {}_04, {}_07], \#\mathrm{AC}^3_{\texttt{C}}[{}_03, {}_05])$        ▷ Property 1
            $\#\mathrm{MC}^3[{}_14, {}_15] \xleftarrow{\texttt{MC}^*} (\#\mathrm{MC}^3[{}_10, {}_11, {}_12, {}_13, {}_16, {}_17], \#\mathrm{AC}^3_{\texttt{C}}[{}_14, {}_16])$        ▷ Property 1
            $\#\mathrm{AC}^2[{}_33, {}_44, {}_55, {}_66, {}_32, {}_43, {}_54, {}_65] \xleftarrow{(\texttt{SB}\circ\texttt{AC})^{-1}} \#\mathrm{MC}^2[{}_03, {}_04, {}_05, {}_06, {}_12, {}_13, {}_14, {}_15]$
            $L_{01} \xleftarrow{push} (\#\mathrm{AC}^2[{}_33, {}_44, {}_55, {}_66, {}_32, {}_43, {}_54, {}_65])$        ▷ the final size $|L_{01}| = 2^{32}$
        end for
        for all possible values of $\#\mathrm{MC}^3[{}_21, {}_22]$, $\#\mathrm{MC}^3[{}_30, {}_31]$ do        ▷ $2^{32}$
            $\#\mathrm{MC}^3[{}_23, {}_24] \xleftarrow{\texttt{MC}^*} (\#\mathrm{MC}^3[{}_20, {}_21, {}_22, {}_25, {}_26, {}_27], \#\mathrm{AC}^3_{\texttt{C}}[{}_25, {}_27])$        ▷ Property 1
            $\#\mathrm{MC}^3[{}_32, {}_33] \xleftarrow{\texttt{MC}^*} (\#\mathrm{MC}^3[{}_30, {}_31, {}_34, {}_35, {}_36, {}_37], \#\mathrm{AC}^3_{\texttt{C}}[{}_30, {}_36])$        ▷ Property 1
            $\#\mathrm{AC}^2[{}_31, {}_42, {}_53, {}_64, {}_30, {}_41, {}_52, {}_63] \xleftarrow{(\texttt{SB}\circ\texttt{AC})^{-1}} \#\mathrm{MC}^2[{}_21, {}_22, {}_23, {}_24, {}_30, {}_31, {}_32, {}_33]$
            $L_{23} \xleftarrow{push} (\#\mathrm{AC}^2[{}_31, {}_42, {}_53, {}_64, {}_30, {}_41, {}_52, {}_63])$        ▷ the final size $|L_{23}| = 2^{32}$
        end for
        for all possible values of $\#\mathrm{MC}^3[{}_40, {}_41]$, $\#\mathrm{MC}^3[{}_50, {}_51]$ do        ▷ $2^{32}$
            $\#\mathrm{MC}^3[{}_42, {}_47] \xleftarrow{\texttt{MC}^*} (\#\mathrm{MC}^3[{}_40, {}_41, {}_43, {}_44, {}_45, {}_46], \#\mathrm{AC}^3_{\texttt{C}}[{}_41, {}_47])$        ▷ Property 1
            $\#\mathrm{MC}^3[{}_56, {}_57] \xleftarrow{\texttt{MC}^*} (\#\mathrm{MC}^3[{}_50, {}_51, {}_52, {}_53, {}_54, {}_55], \#\mathrm{AC}^3_{\texttt{C}}[{}_50, {}_52])$        ▷ Property 1
            $\#\mathrm{AC}^2[{}_40, {}_51, {}_62, {}_37, {}_50, {}_62, {}_36, {}_47] \xleftarrow{(\texttt{SB}\circ\texttt{AC})^{-1}} \#\mathrm{MC}^2[{}_40, {}_41, {}_42, {}_47, {}_50, {}_51, {}_56, {}_57]$
            $L_{45} \xleftarrow{push} (\#\mathrm{AC}^2[{}_40, {}_51, {}_62, {}_37, {}_50, {}_62, {}_36, {}_47])$        ▷ the final size $|L_{45}| = 2^{32}$
        end for
        for all possible values of $\#\mathrm{MC}^3[{}_60, {}_65]$, $\#\mathrm{MC}^3[{}_74, {}_75]$ do        ▷ $2^{32}$
            $\#\mathrm{MC}^3[{}_66, {}_67] \xleftarrow{\texttt{MC}^*} (\#\mathrm{MC}^3[{}_60, {}_61, {}_62, {}_63, {}_64, {}_65], \#\mathrm{AC}^3_{\texttt{C}}[{}_61, {}_63])$        ▷ Property 1
            $\#\mathrm{MC}^3[{}_76, {}_77] \xleftarrow{\texttt{MC}^*} (\#\mathrm{MC}^3[{}_70, {}_71, {}_72, {}_73, {}_74, {}_75], \#\mathrm{AC}^3_{\texttt{C}}[{}_72, {}_74])$        ▷ Property 1
            $\#\mathrm{AC}^2[{}_60, {}_35, {}_46, {}_57, {}_34, {}_45, {}_56, {}_67] \xleftarrow{(\texttt{SB}\circ\texttt{AC})^{-1}} \#\mathrm{MC}^2[{}_60, {}_65, {}_66, {}_67, {}_74, {}_75, {}_76, {}_77]$
            $L_{67} \xleftarrow{push} (\#\mathrm{AC}^2[{}_60, {}_35, {}_46, {}_57, {}_34, {}_45, {}_56, {}_67])$        ▷ the final size $|L_{67}| = 2^{32}$
        end for
        for all $(\#\mathrm{AC}^2[{}_33, {}_44, {}_55, {}_66, {}_32, {}_43, {}_54, {}_65]) \in L_{01}$ do        ▷ $2^{32}$
            for all $(\#\mathrm{AC}^2[{}_31, {}_42, {}_53, {}_64, {}_30, {}_41, {}_52, {}_63]) \in L_{23}$ do        ▷ $2^{32}$
                $a \xleftarrow{\texttt{MC}} (\#\mathrm{AC}^2[{}_30, {}_31, {}_32, {}_33] \,\|\, 0 \,\|\, 0 \,\|\, 0 \,\|\, 0)[5, 7] \oplus \#\mathrm{MC}^2[{}_35, {}_37]$
                $b \xleftarrow{\texttt{MC}} (0 \,\|\, \#\mathrm{AC}^2[{}_41, {}_42, {}_43, {}_44] \,\|\, 0 \,\|\, 0 \,\|\, 0)[4, 6] \oplus \#\mathrm{MC}^2[{}_44, {}_46]$
                $c \xleftarrow{\texttt{MC}} (0 \,\|\, 0 \,\|\, \#\mathrm{AC}^2[{}_52, {}_53, {}_54, {}_55] \,\|\, 0 \,\|\, 0)[3, 5] \oplus \#\mathrm{MC}^2[{}_53, {}_55]$
                $d \xleftarrow{\texttt{MC}} (0 \,\|\, 0 \,\|\, 0 \,\|\, \#\mathrm{AC}^2[{}_63, {}_64, {}_65, {}_66] \,\|\, 0)[2, 4] \oplus \#\mathrm{MC}^2[{}_62, {}_64]$
                $T[a \,\|\, b \,\|\, c \,\|\, d] \leftarrow (\#\mathrm{AC}^2[{}_33, {}_44, {}_55, {}_66, {}_32, {}_43, {}_54, {}_65, {}_31, {}_42, {}_53, {}_64, {}_30, {}_41, {}_52, {}_63])$
            end for
        end for
        for all $(\#\mathrm{AC}^2[{}_40, {}_51, {}_62, {}_37, {}_50, {}_62, {}_36, {}_47]) \in L_{45}$ do        ▷ $2^{32}$
            for all $(\#\mathrm{AC}^2[{}_60, {}_35, {}_46, {}_57, {}_34, {}_45, {}_56, {}_67]) \in L_{67}$ do        ▷ $2^{32}$
                $a' \xleftarrow{\texttt{MC}} (0 \,\|\, 0 \,\|\, 0 \,\|\, 0 \,\|\, \#\mathrm{AC}^2[{}_34, {}_35, {}_36, {}_37])[5, 7]$
                $b' \xleftarrow{\texttt{MC}} (\#\mathrm{AC}^2[{}_40] \,\|\, 0 \,\|\, 0 \,\|\, 0 \,\|\, 0 \,\|\, \#\mathrm{AC}^2[{}_45, {}_46, {}_47])[4, 6]$
                $c' \xleftarrow{\texttt{MC}} (\#\mathrm{AC}^2[{}_50, {}_51] \,\|\, 0 \,\|\, 0 \,\|\, 0 \,\|\, 0 \,\|\, \#\mathrm{AC}^2[{}_56, {}_57])[3, 5]$
                $d' \xleftarrow{\texttt{MC}} (\#\mathrm{AC}^2[{}_60, {}_61, {}_62] \,\|\, 0 \,\|\, 0 \,\|\, 0 \,\|\, 0 \,\|\, \#\mathrm{AC}^2[{}_67] \,\|\, 0)[2, 4]$
                for all $\#\mathrm{AC}^2[{}_33, {}_44, {}_55, {}_66, {}_32, {}_43, {}_54, {}_65, {}_31, {}_42, {}_53, {}_64, {}_30, {}_41, {}_52, {}_63] \in T[a' \,\|\, b' \,\|\, c' \,\|\, d']$ do
                    $\overleftarrow{\mathcal{T}}_{\texttt{Init}} \xleftarrow{push} \#\mathrm{AC}^2_{\texttt{col}_{\{0,1,3,4,5,6\}}}$        ▷ the final size $|\overleftarrow{\mathcal{T}}_{\texttt{Init}}| = 2^{128}$
                end for
            end for
        end for
    end for
    return $\overleftarrow{\mathcal{T}}_{\texttt{Init}}$
end procedure
```

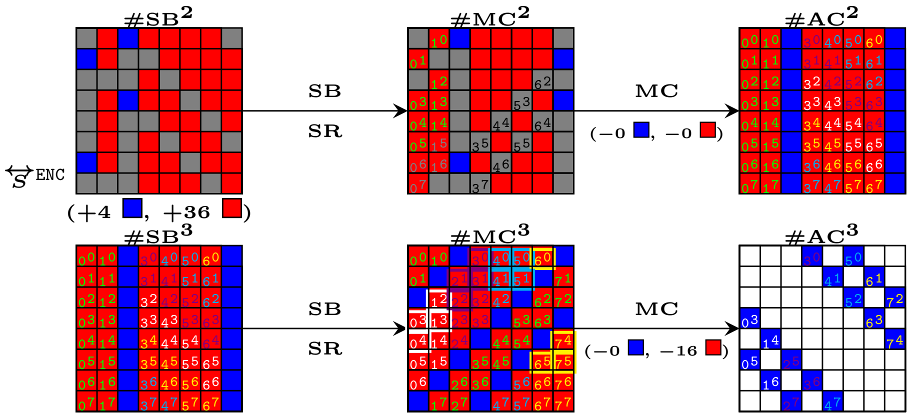


Fig. 9: Index in Algorithm 2

Algorithm 3 Compute values of forward neutral bytes (in `Blue`) in Fig. 7

```
1: procedure ComputeForwardNeutralBytes
2:     $\overrightarrow{\mathcal{T}_{\texttt{Init}}} \leftarrow \emptyset$
3:     for all possible value $\overrightarrow{v}_i$ of the 28 Blue cells in $\#\textbf{MC}^4$ do        ▷ $2^{2\times 28}$
4:         $c_0 \xleftarrow{\texttt{MC}} (0 \| 0 \| 0 \| \#\textbf{MC}^4_{\texttt{col}_3}[3,4,5,6] \| 0)[7]$
5:         $c_1 \xleftarrow{\texttt{MC}} (0 \| 0 \| \#\textbf{MC}^4_{\texttt{col}_4}[2,3,4,5,6] \| 0)[7]$
6:         $c_2 \xleftarrow{\texttt{MC}} (0 \| \#\textbf{MC}^4_{\texttt{col}_5}[1,2,3,4,5] \| 0 \| 0)[7]$
7:         $c_3 \xleftarrow{\texttt{MC}} (0 \| \#\textbf{MC}^4_{\texttt{col}_6}[1,2,3,4] \| 0 \| 0 \| 0)[7]$
8:         $c_4 \xleftarrow{\texttt{MC}} (\#\textbf{MC}^4_{\texttt{col}_8}[0,1,2] \| 0 \| 0 \| 0 \| 0 \| 0)[0]$
9:         compute backward cell-by-cell through $\texttt{SB}^{-1}$, $\texttt{SR}^{-1}$, and $\texttt{AC}^{-1}$ to get the Blue
   cells in $\#\textbf{AC}^3$
10:        $c_5 \xleftarrow{\texttt{MC}^{-1}} (0 \| \#\textbf{AC}^3_{\texttt{col}_6}[1,2,3,4] \| 0 \| 0 \| 0)[0,1,7]$
11:        $c_6 \xleftarrow{\texttt{MC}^{-1}} (\#\textbf{AC}^3_{\texttt{col}_7}[0,1,2,3,4,5] \| 0 \| 0)[0,7]$
12:        $c_7 \xleftarrow{\texttt{MC}^{-1}} (\#\textbf{AC}^3_{\texttt{col}_8}[0,1,2,3,4,5,6] \| 0)[6,7]$
13:        $c_8 \xleftarrow{\texttt{MC}^{-1}} (0 \| \#\textbf{AC}^3_{\texttt{col}_9}[1,2,3,4,5,6] \| 0)[5,6,7]$
14:        $c_9 \xleftarrow{\texttt{MC}^{-1}} (0 \| 0 \| \#\textbf{AC}^3_{\texttt{col}_{10}}[2,3,4,5,6] \| 0)[4,5,6,7]$
15:        $\overrightarrow{\mathcal{T}_{\texttt{Init}}}[c_0 \| c_1 \| c_2 \| c_3 \| c_4 \| c_5 \| c_6 \| c_7 \| c_8 \| c_9] \leftarrow \overrightarrow{v}_i$
16:    end for
17:    return $\overleftarrow{\mathcal{T}_{\texttt{Init}}}$
18:                                                    ▷ Note that due to the linearity
   of MixColumns, values of Gray cells in involved columns of $\#\textbf{MC}^4$ and $\#\textbf{AC}^3$ can
   be set to zero in this precomputation phase. During the main attack phase, in the
   backward computation, the values of Gray cells will be integrated; constant impacts
   from these values of Gray cells together with the pre-determined impacts from the
   Blue cells are needed to be XORed together to the involved Red cells.
19: end procedure
```

Algorithm 4 Compute values of backward neutral bytes (in `Red`) in Fig. 7

```
1: procedure ComputeBackwardNeutralBytes
2:     $\overleftarrow{\mathcal{T}_{\texttt{Init}}} \leftarrow \emptyset$
3:     for all possible value $\overleftarrow{v}_i$ of the 16 Red cells in $\#\textbf{AC}^5$ do        ▷ $2^{2\times 16}$
4:         compute forward cell-by-cell through AC, SB, and SR to get the Red cells in
   $\#\textbf{MC}^6$
5:         $c_0 \xleftarrow{\texttt{MC}} (0 \| 0 \| 0 \| \#\textbf{MC}^6_{\texttt{col}_5}[3,4,5,6] \| 0)[6,7]$
6:         $c_1 \xleftarrow{\texttt{MC}} (0 \| 0 \| \#\textbf{MC}^6_{\texttt{col}_6}[2,3,4,5] \| 0 \| 0)[7]$
7:         $c_2 \xleftarrow{\texttt{MC}} (0 \| \#\textbf{MC}^6_{\texttt{col}_7}[1,2,3,4] \| 0 \| 0 \| 0)[7]$
8:         $c_3 \xleftarrow{\texttt{MC}} (\#\textbf{MC}^6_{\texttt{col}_8}[0,1,2,3] \| 0 \| 0 \| 0 \| 0)[0,7]$
9:         $c_4 \xleftarrow{\texttt{MC}^{-1}} (\#\textbf{AC}^5_{\texttt{col}_8}[0,1,2,3] \| 0 \| 0 \| 0 \| 0)[0,7]$
10:        $c_5 \xleftarrow{\texttt{MC}^{-1}} (0 \| \#\textbf{AC}^5_{\texttt{col}_9}[1,2,3,4] \| 0 \| 0 \| 0)[7]$
11:        $c_6 \xleftarrow{\texttt{MC}^{-1}} (0 \| 0 \| \#\textbf{AC}^5_{\texttt{col}_{10}}[2,3,4,5] \| 0 \| 0)[7]$
12:        $c_7 \xleftarrow{\texttt{MC}^{-1}} (0 \| 0 \| 0 \| \#\textbf{AC}^5_{\texttt{col}_{11}}[3,4,5,6] \| 0)[7]$
13:        $\overleftarrow{\mathcal{T}_{\texttt{Init}}}[c_0 \| c_1 \| c_2 \| c_3 \| c_4 \| c_5 \| c_6 \| c_7] \leftarrow \overleftarrow{v}_i$
14:    end for
15:    return $\overleftarrow{\mathcal{T}_{\texttt{Init}}}$
16: end procedure
```

C Notations.

Notations	
`Blue` (🟦)	active in the forward chunk
`Red` (🟥)	active in the backward chunk
`Gray` (▩)	known constant in both chunks
`White` (□)	unknown in both chunks
`Black` (■)	any of `Blue` (🟦), `Red` (🟥), `Gray` (▩), `White` (□)
`Cyan` (🟦)	guessed for forward chunks
`Pink` (🟥)	guessed for backward chunks
`Green` (🟩)	guessed for both forward and backward chunks
`Violet` (🟪)	degree of matching
$\overleftrightarrow{S}^{\texttt{ENC}}$	starting state in the encryption data path
$\overleftrightarrow{S}^{\texttt{KSA}}$	starting state in the key-schedule data path
$\overrightarrow{End}$	ending state for the forward computation
$\overleftarrow{End}$	ending state for the backward computation
$\mathcal{BL}^{\texttt{ENC}}$	subset of $\mathcal{N}$, index of `Blue` cells (🟦) in $\overleftrightarrow{S}^{\texttt{ENC}}$, $\mathcal{BL}^{\texttt{ENC}} \cap \mathcal{RD}^{\texttt{ENC}} = \emptyset$
$\mathcal{BL}^{\texttt{KSA}}$	subset of $\mathcal{N}$, index of `Blue` cells (🟦) in $\overleftrightarrow{S}^{\texttt{KSA}}$, $\mathcal{BL}^{\texttt{KSA}} \cap \mathcal{RD}^{\texttt{KSA}} = \emptyset$
$\mathcal{RD}^{\texttt{ENC}}$	subset of $\mathcal{N}$, index of `Red` cells (🟥) in $\overleftrightarrow{S}^{\texttt{ENC}}$, $\mathcal{RD}^{\texttt{ENC}} \cap \mathcal{BL}^{\texttt{ENC}} = \emptyset$
$\mathcal{RD}^{\texttt{KSA}}$	subset of $\mathcal{N}$, index of `Red` cells (🟥) in $\overleftrightarrow{S}^{\texttt{KSA}}$, $\mathcal{BL}^{\texttt{KSA}} \cap \mathcal{RD}^{\texttt{KSA}} = \emptyset$
$\mathcal{GY}^{\texttt{ENC}}$	subset of $\mathcal{N}$, index of `Gray` cells (▩) in $\overleftrightarrow{S}^{\texttt{ENC}}$, $\mathcal{GY}^{\texttt{ENC}} = \mathcal{N} - \mathcal{BL}^{\texttt{ENC}} \cup \mathcal{RD}^{\texttt{ENC}}$
$\mathcal{GY}^{\texttt{KSA}}$	subset of $\mathcal{N}$, index of `Gray` cells (▩) in $\overleftrightarrow{S}^{\texttt{KSA}}$, $\mathcal{GY}^{\texttt{KSA}} = \mathcal{N} - \mathcal{BL}^{\texttt{KSA}} \cup \mathcal{RD}^{\texttt{KSA}}$
Encoding	
x_i^S and y_i^S	0-1 variables to encode the color (attribute) of the ith cell of a state S
	$(1,0)$: `Blue` cell (🟦)
	$(0,1)$: `Red` cell (🟥)
	$(1,1)$: `Gray` cell (▩)
	$(0,0)$: `White` cell (□)
x	a binary variable, indicating a cell can be computed in the forward chunk; is `Blue` or `Gray`
y	a binary variable, indicating a cell can be computed in the backward chunk; is `Red` or `Gray` `White`
β	a binary variable, indicating a cell is `Gray` (▩)
ω	a binary variable, indicating a cell is `White`
$\boldsymbol{x}$	a binary variable, indicating cells in a column all are `Blue`/`Gray`
$\boldsymbol{y}$	a binary variable, indicating cells in a column all are `Red`/`Gray`
$\boldsymbol{\omega}$	a binary variable, indicating a column exists `White` cell
$\overrightarrow{\iota}$	$\overrightarrow{\iota} = \lvert\mathcal{BL}^{\texttt{ENC}}\rvert + \lvert\mathcal{BL}^{\texttt{KSA}}\rvert$, the initial degrees of freedom for the forward
$\overleftarrow{\iota}$	$\overleftarrow{\iota} = \lvert\mathcal{RD}^{\texttt{ENC}}\rvert + \lvert\mathcal{RD}^{\texttt{KSA}}\rvert$, the initial degrees of freedom for the backward
$\overrightarrow{\sigma}$	$\overrightarrow{d_b} = \overrightarrow{\iota} - \overrightarrow{\sigma}$, the accumulated coniosumed degrees of freedom from the forward

$\overleftarrow{\sigma}$	$\overleftarrow{d_r} = \overleftarrow{\iota} - \overleftarrow{\sigma}$, the accumulated coniosumed degrees of freedom from the backward
c_x	the binary variable indicating whether to coniosume a degree of freedom from the forward in `XOR-RULE`
c_y	the binary variable indicating whether to consume a degree of freedom from the backward in `XOR-RULE`
$c_{\boldsymbol{x}}$	the general variable for the number of consumed degrees of freedom from the forward in `MC-RULE`
$c_{\boldsymbol{y}}$	the general variable for the number of consumed degrees of freedom from the backward in `MC-RULE`
$\overrightarrow{d_b}$	the degrees of freedom for the forward computation
$\overleftarrow{d_r}$	the degrees of freedom for the backward computation
$\overset{\rightarrow\leftarrow}{m}$	the degrees of matching
$\overrightarrow{d_{g_b}}$	the number of cells only guessed to be `Blue` (forward computation)
$\overleftarrow{d_{g_r}}$	be the number of cells only guessed to be `Red` (backward computation)
$\overleftrightarrow{d_{g_{br}}}$	be the number of cells guessed to be both `Blue` and `Red`
$\tau_{\texttt{Obj}}$	the auxiliary variable to set the objective function
	Cipher Specification
$\mathrm{N}_{\texttt{row}}$	the number of rows in the state of the compression function
$\mathrm{N}_{\texttt{col}}$	the number of columns in the state of the compression function
c	the number of bits in each cell of the state of the compression function; in targeted ciphers of this work, it is 8
ς	the number of possible values of a cell, *i.e.*, $\varsigma = 2^c$; in targeted ciphers of this work, it is 2^8
n	$n = \mathrm{N}_{\texttt{row}} \cdot \mathrm{N}_{\texttt{col}}$, the number of cells in the state of the compression function
$\mathrm{N}_{\texttt{iosum}}$	the total number of cells in the input and output of the MC on one column, $\mathrm{N}_{\texttt{iosum}} = 2 \cdot \mathrm{N}_{\texttt{row}}$.
$\texttt{Br}_{\texttt{n}}$	$\texttt{Br}_{\texttt{n}} = \mathrm{N}_{\texttt{row}} + 1$, the branch number of the MDS matrix used in the compression function
	Notations used in descriptions of concrete attacks
`MC`	excuating the MixColumns operation on a single column or on multiple columns
`SB`	excuating the SubBytes operation on a single cell or on multiple cells
`SR`	excuating the ShiftRows operation on a single cell or on multiple cells
$\#\mathbf{SB}^r$	the state before go through SubBytes (`SB`) at the r-th round in the encryption data-path
$\#\mathbf{SR}^r$	the state before go through ShiftRows (`SR`) at the r-th round in the encryption data-path
$\#\mathbf{MC}^r$	the state before go through MixColumns (`MC`) at the r-th round in the encryption data-path
$\#\mathbf{AK}^r$	the state before go through AddRoundKey (`AK`) at the r-th round in the encryption data-path

$\#\mathbf{KSB}^r$	the state before go through SubBytes (SB) at the r-th round in the key-schedule
$\#\mathbf{KSR}^r$	the state before go through ShiftRows (SR) at the r-th round in the key-schedule
$\#\mathbf{KMC}^r$	the state before go through MixColumns (MC) at the r-th round in the key-schedule
$\boldsymbol{k}^r$	the r-th round-key
$S[i, j, \cdots]$	the i-th, j-th, $\cdots$ cells of a state (in the order of column first)
$S[i]$	the i-th cell of a state (in the order of column first)
$S_{\mathtt{col}_i}$	the i-th column of a state
$S_{\mathtt{col}_{\{i,j,\cdots\}}}$	the i-th, j-th, $\cdots$, columns of a state S
$S_{\mathtt{col}_i}[j]$	the j-th cell in the i-th column of a state S
$S_{\mathtt{col}_i}[j_1, j_2, \cdots]$	the j_1-th, j_2-th, $\cdots$, cells in the i-th column of a state S
$S[\mathtt{i}\,\mathtt{j}]$	the j-th cell in the i-th column of a state S, shorten for $S_{\mathtt{col}_i}[j]$
$\|$	concatenation
$S_{\mathtt{SR}^{-1}(\mathtt{col}_i)}$	the cells in the i-th diagonal of the state S
$S_{\mathtt{SR}(\mathtt{col}_i)}$	the cells in the i-th anti-diagonal of the state S
$\mathtt{MC}_{abcd_e}$	using values of the a-th, b-th, c-th, d-th cells in the array of (input $\|$ output) of the MC to derive the value of the e-th cell according to Property 1
$\mathtt{MC}^*$	using values of the $\mathrm{N}_{\mathtt{row}}$ cells in the array of (input $\|$ output) of the MC to derive the value of an additional cell according to Property 1

D Visualization of more Examples of Attack Configurations

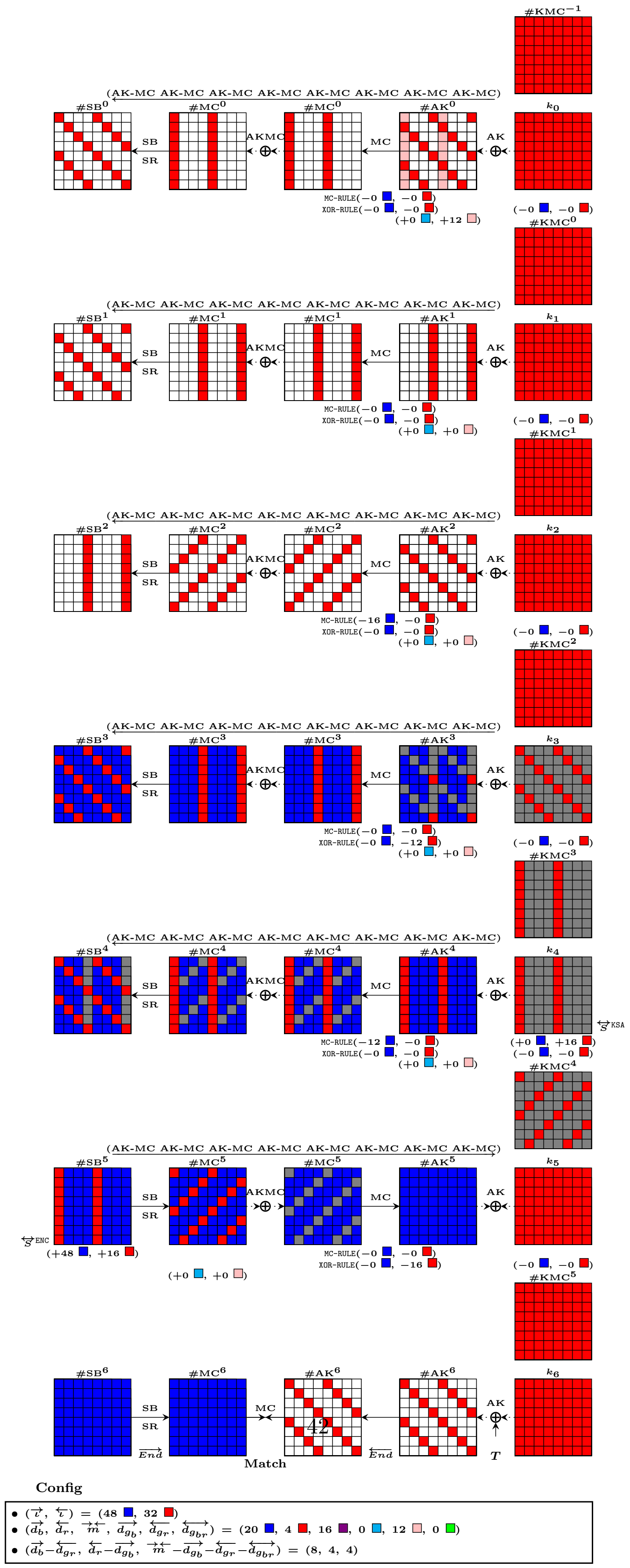
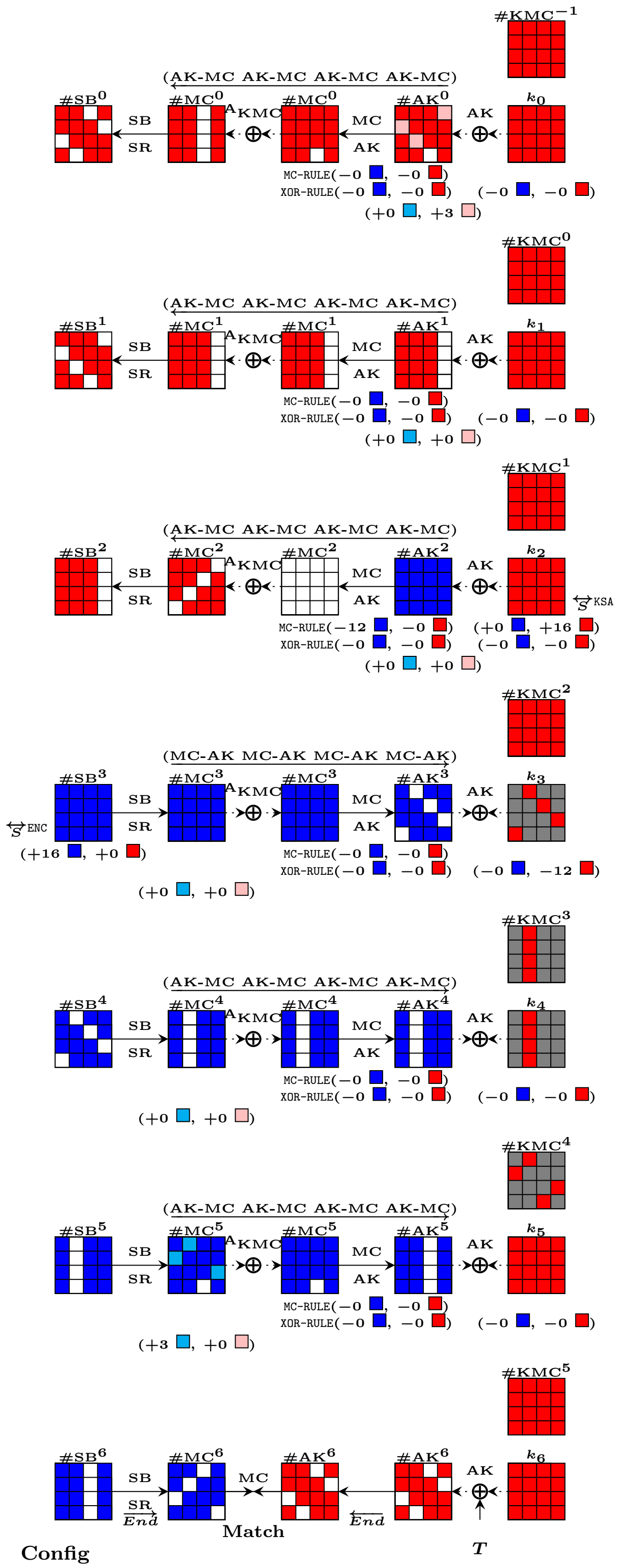

#KMC^{-1}
(AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC)
#SB^0 #MC^0 #MC^0 #AK^0 k_0
SB SR AKMC MC AK
MC-RULE(−0 🟦, −0 🟥)
XOR-RULE(−0 🟦, −0 🟥)
(+0 🟦, +12 🟥)
(−0 🟦, −0 🟥)
#KMC^0
#SB^1 #MC^1 #MC^1 #AK^1 k_1
#KMC^1
#SB^2 #MC^2 #MC^2 #AK^2 k_2
MC-RULE(−16 🟦, −0 🟥)
#KMC^2
#SB^3 #MC^3 #MC^3 #AK^3 k_3
XOR-RULE(−0 🟦, −12 🟥)
#KMC^3
#SB^4 #MC^4 #MC^4 #AK^4 k_4
MC-RULE(−12 🟦, −0 🟥)
$\overleftrightarrow{S}$ KSA
(+0 🟦, +16 🟥)
#KMC^4
#SB^5 #MC^5 #MC^5 #AK^5 k_5
$\overleftrightarrow{S}$ ENC
(+48 🟦, +16 🟥)
XOR-RULE(−0 🟦, −16 🟥)
#KMC^5
#SB^6 #MC^6 #AK^6 #AK^6 k_6
$\overrightarrow{End}$ Match $\overleftarrow{End}$ T
Config
• $(\overrightarrow{\iota}, \overleftarrow{\iota})$ = (48 🟦, 32 🟥)
• $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}})$ = (20 🟦, 4 🟥, 16 🟪, 0 🟦, 12 🟥, 0 🟩)
• $(\overrightarrow{d_b}-\overleftarrow{d_{g_r}}, \overleftarrow{d_r}-\overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})$ = (8, 4, 4)

Fig. 10: The full version of the configuration in example Fig. 3 for the MITM attack on 7-round Whirlpool (2 × 2 of 4 × 4)



Config

- $(\overrightarrow{\iota},\ \overleftarrow{\iota})\ =\ (16$ 🟦, 16 🟥)
- $(\overrightarrow{d_b},\ \overleftarrow{d_r},\ \overrightarrow{\overleftarrow{m}},\ \overrightarrow{d_{g_b}},\ \overleftarrow{d_{g_r}},\ \overleftrightarrow{d_{g_{br}}})\ =\ (4$ 🟦, 4 🟥, 8 🟪, 3 ▩, 3 ▢, 0 🟩)
- $(\overrightarrow{d_b}-\overleftarrow{d_{g_r}},\ \overleftarrow{d_r}-\overrightarrow{d_{g_b}},\ \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})\ =\ (1,\ 1,\ 2)$

This implies an attack on the full version (8×8) with configuration
$((\overrightarrow{d_b} - \overleftarrow{d_{g_r}},\ \overleftarrow{d_r} - \overrightarrow{d_{g_b}},\ \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}})\ =\ (4,\ 4,\ 8))$

Fig. 11: Another example of using $(2 \times 2$ of $4 \times 4)$ to search the MITM attack on 7-round Whirlpool

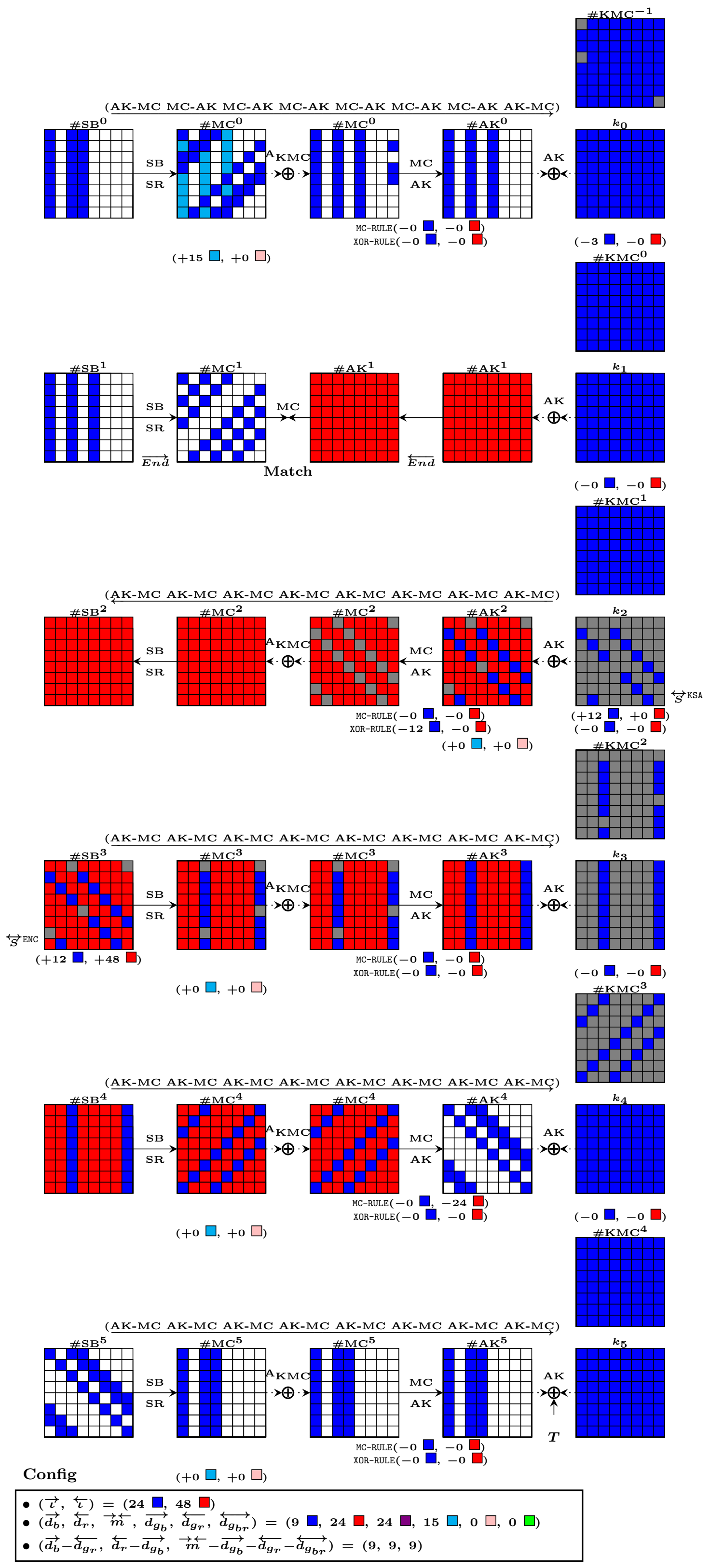
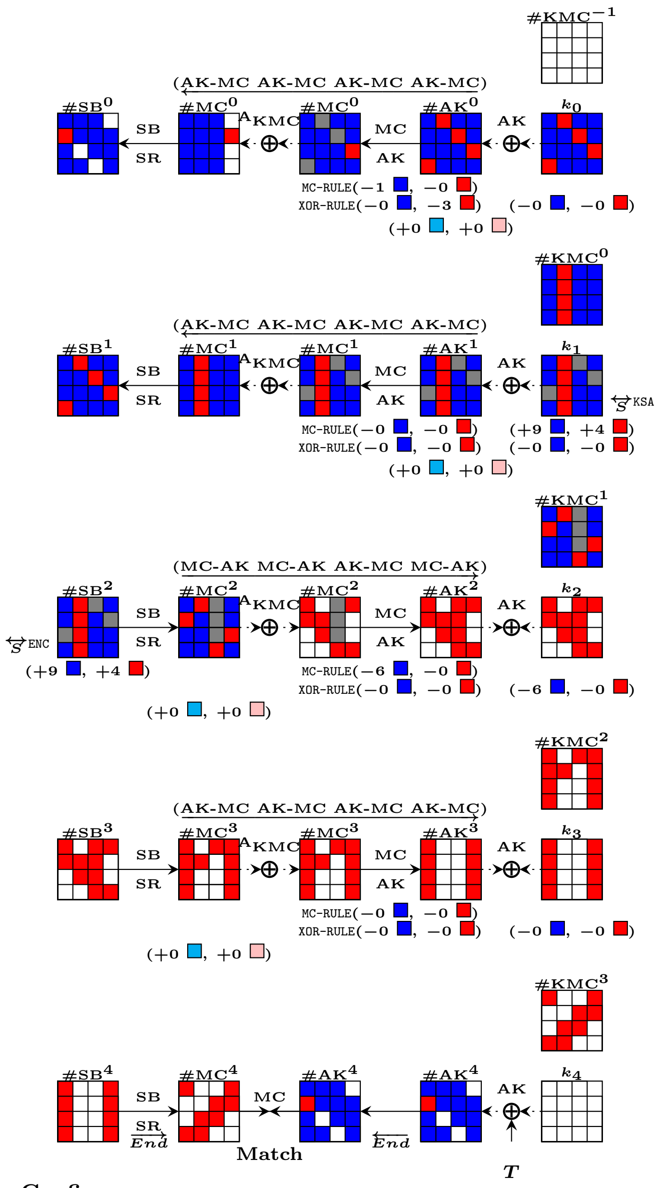

#KMC^{-1}
(AK-MC MC-AK MC-AK MC-AK MC-AK MC-AK MC-AK AK-MC)
#SB^0
#MC^0
#MC^0
#AK^0
k_0
SB
SR
KMC
MC
AK
MC-RULE(−0 🟦, −0 🟥)
XOR-RULE(−0 🟦, −0 🟥)
(+15 🟦, +0 🟥)
(−3 🟦, −0 🟥)
#KMC^0
#SB^1
#MC^1
#AK^1
#AK^1
k_1
$\overrightarrow{End}$
$\overleftarrow{End}$
Match
(−0 🟦, −0 🟥)
#KMC^1
(AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC)
#SB^2
#MC^2
#MC^2
#AK^2
k_2
$\overleftrightarrow{S}$ KSA
MC-RULE(−0 🟦, −0 🟥)
XOR-RULE(−12 🟦, −0 🟥)
(+0 🟦, +0 🟥)
(+12 🟦, +0 🟥)
(−0 🟦, −0 🟥)
#KMC^2
(AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC)
#SB^3
#MC^3
#MC^3
#AK^3
k_3
$\overleftrightarrow{S}$ ENC
(+12 🟦, +48 🟥)
MC-RULE(−0 🟦, −0 🟥)
XOR-RULE(−0 🟦, −0 🟥)
(+0 🟦, +0 🟥)
(−0 🟦, −0 🟥)
#KMC^3
(AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC)
#SB^4
#MC^4
#MC^4
#AK^4
k_4
MC-RULE(−0 🟦, −24 🟥)
XOR-RULE(−0 🟦, −0 🟥)
(+0 🟦, +0 🟥)
(−0 🟦, −0 🟥)
#KMC^4
(AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC AK-MC)
#SB^5
#MC^5
#MC^5
#AK^5
k_5
T
MC-RULE(−0 🟦, −0 🟥)
XOR-RULE(−0 🟦, −0 🟥)
(+0 🟦, +0 🟥)
Config
• $(\overrightarrow{\iota}, \overleftarrow{\iota})$ = (24 🟦, 48 🟥)
• $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{m}\overleftarrow{\ }, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}})$ = (9 🟦, 24 🟥, 24 🟪, 15 🟦, 0 🟥, 0 🟩)
• $(\overrightarrow{d_b}-\overleftarrow{d_{g_r}}, \overleftarrow{d_r}-\overrightarrow{d_{g_b}}, \overrightarrow{m}\overleftarrow{\ }-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})$ = (9, 9, 9)

Fig. 12: An example of the MITM attack on 6-round Whirlpool

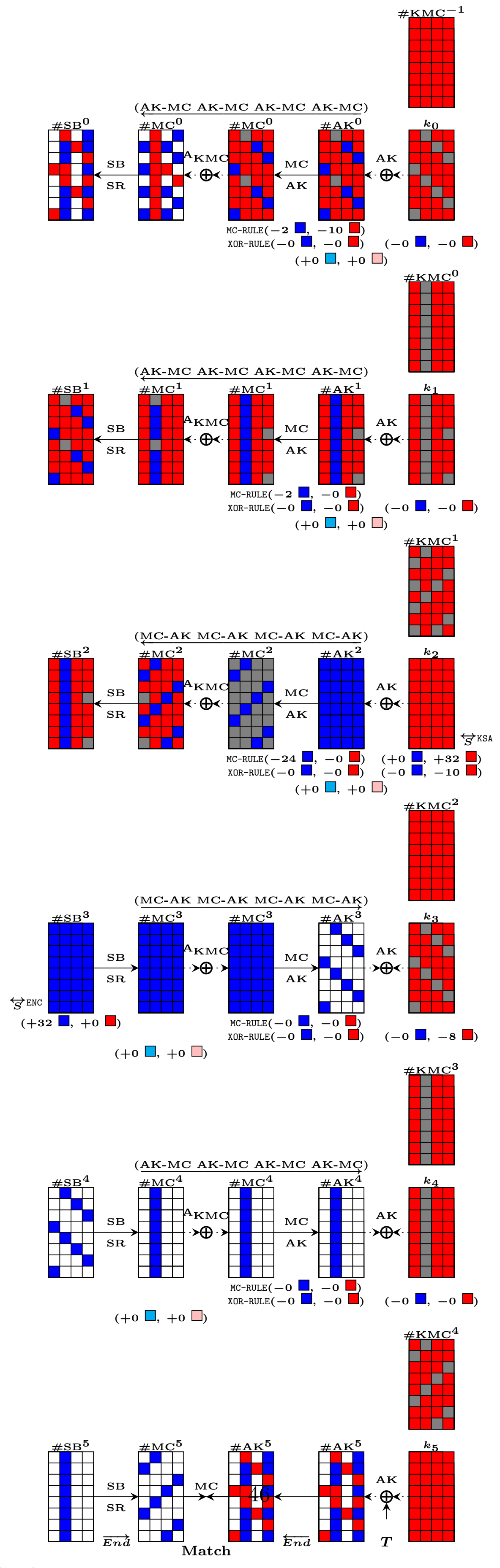

#KMC^{-1}
(AK-MC AK-MC AK-MC AK-MC)
#SB^0 SB SR #MC^0 AKMC #MC^0 MC AK #AK^0 AK k_0
MC-RULE(−1 🟦, −0 🟥)
XOR-RULE(−0 🟦, −3 🟥)
(−0 🟦, −0 🟥)
(+0 🟦, +0 🟥)
#KMC^0
(AK-MC AK-MC AK-MC AK-MC)
#SB^1 SB SR #MC^1 AKMC #MC^1 MC AK #AK^1 AK k_1
KSA
MC-RULE(−0 🟦, −0 🟥)
XOR-RULE(−0 🟦, −0 🟥)
(+9 🟦, +4 🟥)
(−0 🟦, −0 🟥)
(+0 🟦, +0 🟥)
#KMC^1
(MC-AK MC-AK AK-MC MC-AK)
#SB^2 SB SR #MC^2 AKMC #MC^2 MC AK #AK^2 AK k_2
ENC
(+9 🟦, +4 🟥)
MC-RULE(−6 🟦, −0 🟥)
XOR-RULE(−0 🟦, −0 🟥)
(−6 🟦, −0 🟥)
(+0 🟦, +0 🟥)
#KMC^2
(AK-MC AK-MC AK-MC AK-MC)
#SB^3 SB SR #MC^3 AKMC #MC^3 MC AK #AK^3 AK k_3
MC-RULE(−0 🟦, −0 🟥)
XOR-RULE(−0 🟦, −0 🟥)
(−0 🟦, −0 🟥)
(+0 🟦, +0 🟥)
#KMC^3
#SB^4 SB SR End #MC^4 MC #AK^4 End #AK^4 AK k_4
Match
T

Config

- $(\overrightarrow{\iota}, \overleftarrow{\iota}) = (18$ 🟦, 8 🟥$)$
- $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}}) = (5$ 🟦, 5 🟥, 5 🟪, 0 🟦, 0 🟥, 0 🟩$)$
- $(\overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) = (5, 5, 5)$

This implies an attack on the full version (8×8) with configuration
$((\overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) = (20, 20, 20))$

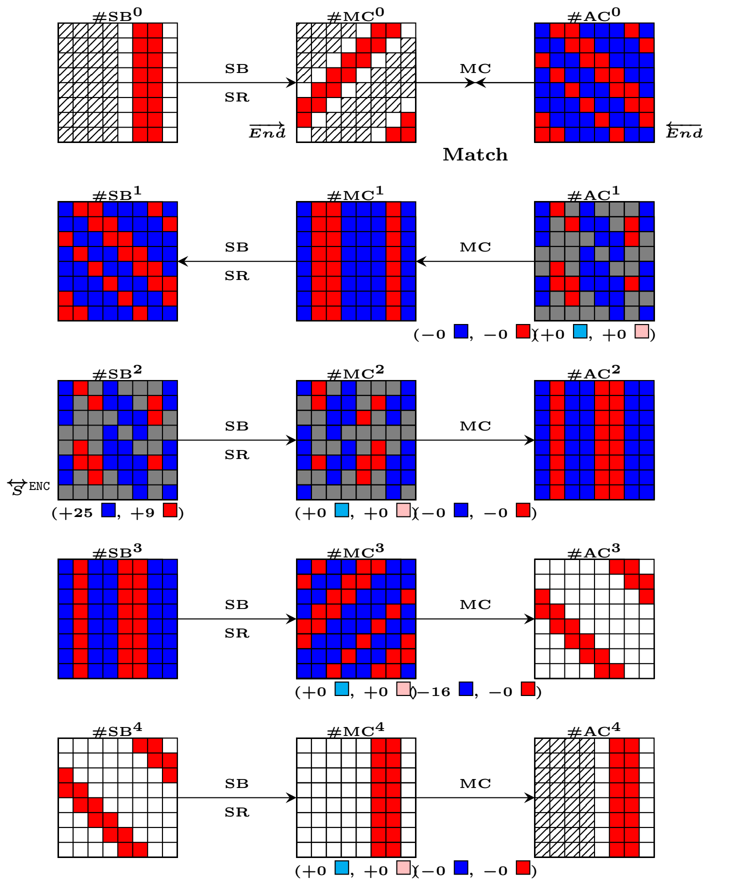
Fig. 13: An example of using $(2 \times 2$ of $4 \times 4)$ to search the MITM attack on 5-round Whirlpool



Config

- $(\overrightarrow{\iota},\ \overleftarrow{\iota}) = (32$ 🟦, 32 🟥$)$
- $(\overrightarrow{d_b},\ \overleftarrow{d_r},\ \overrightarrow{\overleftarrow{m}},\ \overrightarrow{d_{g_b}},\ \overleftarrow{d_{g_r}},\ \overleftrightarrow{d_{g_{br}}}) = (4$ 🟦, 4 🟥, 4 🟪, 0 🟦, 0 🟥, 0 🟩$)$
- $(\overrightarrow{d_b}-\overleftarrow{d_{g_r}},\ \overleftarrow{d_r}-\overrightarrow{d_{g_b}},\ \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}}) = (4,\ 4,\ 4)$

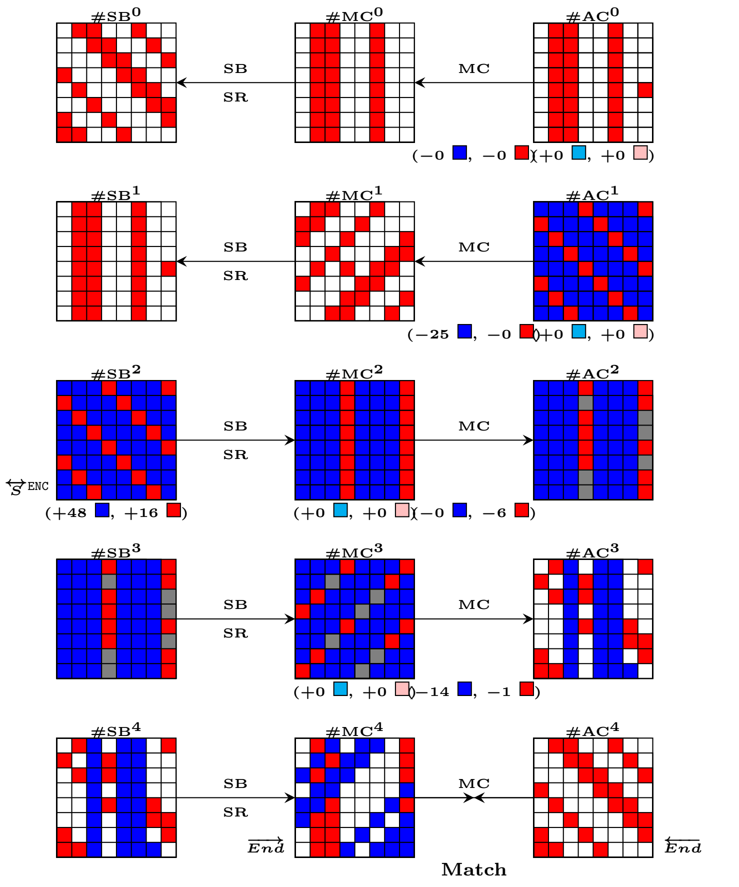
This implies an attack on the full version (8×8) with configuration $((\overrightarrow{d_b} - \overleftarrow{d_{g_r}},\ \overleftarrow{d_r} - \overrightarrow{d_{g_b}},\ \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) = (8,\ 8,\ 8))$



Config

- $(\overrightarrow{\iota}, \overleftarrow{\iota}) = (25$ 🟦, 9 🟥$)$
- $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{m}\overleftarrow{\ }, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}}) = (9$ 🟦, 9 🟥, 16 🟪, 0 🩵, 0 🩷, 0 🟩$)$
- $(\overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{m}\overleftarrow{\ } - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) = (9, 9, 16)$

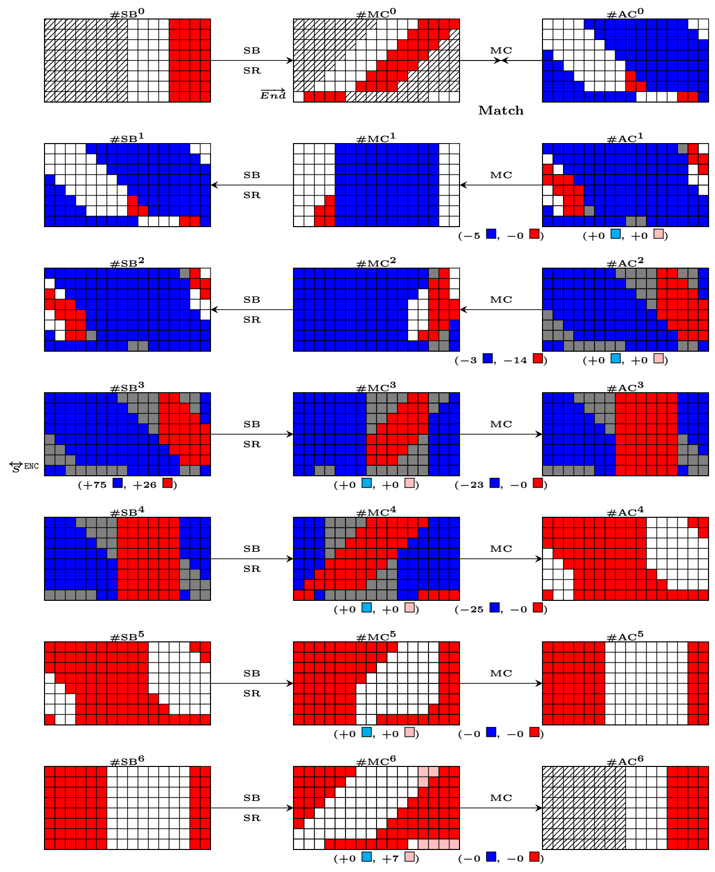
Fig. 15: An example of the MITM attack on the OT of the 5-round Grøstl-256



Config

- $(\overrightarrow{\iota}, \overleftarrow{\iota}) = (48$ 🟦, 16 🟥$)$
- $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{m}\overleftarrow{\ }, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}}) = (9$ 🟦, 9 🟥, 9 🟪, 0 🩵, 0 🩷, 0 🟩$)$
- $(\overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{m}\overleftarrow{\ } - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) = (9, 9, 9)$

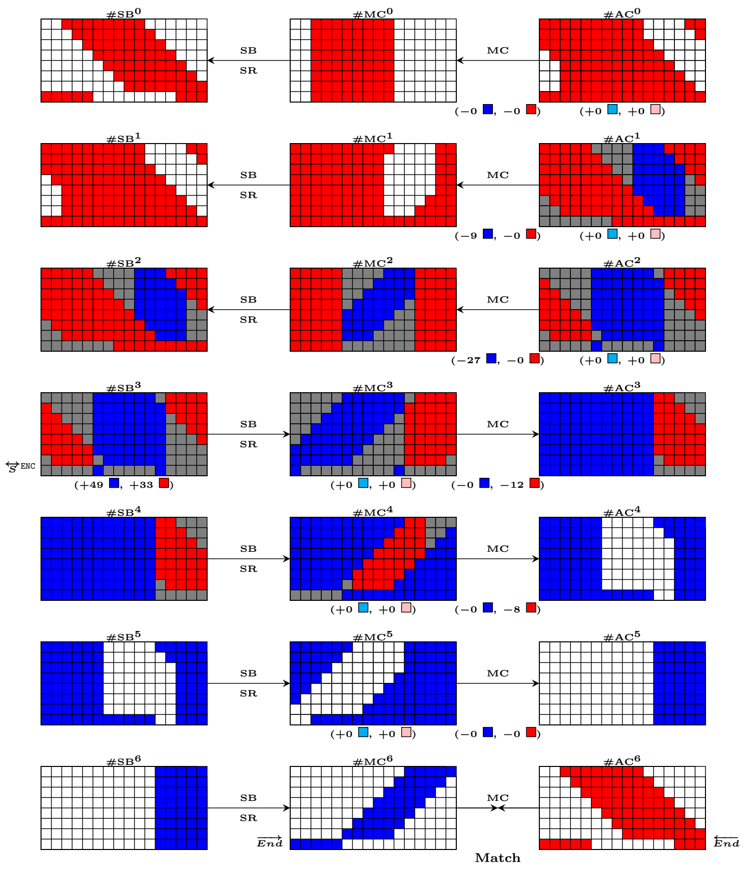
Fig. 16: An example of the MITM attack on $P(H') \oplus H'$ in the CF of the 5-round Grøstl-256 (without guessing)



Config

- $(\overrightarrow{\iota}, \overleftarrow{\iota})$ = (75 🟦, 26 🟥)
- $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}})$ = (19 🟦, 12 🟥, 19 🟪, 0 🟦, 7 🟥, 0 🟩)
- $(\overrightarrow{d_b}-\overleftarrow{d_{g_r}}, \overleftarrow{d_r}-\overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})$ = (12, 12, 12)

Fig. 17: An example of the MITM attack on the OT of the 7-round Grøstl-512



Config

- $(\overrightarrow{\iota}, \overleftarrow{\iota})$ = (49 🟦, 33 🟥)
- $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}})$ = (13 🟦, 13 🟥, 13 🟪, 0 🟦, 0 🟥, 0 🟩)
- $(\overrightarrow{d_b}-\overleftarrow{d_{g_r}}, \overleftarrow{d_r}-\overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})$ = (13, 13, 13)

Fig. 18: An example of the MITM attack on $P(H') \oplus H'$ in the CF of the 7-round Grøstl-512 (without guessing)

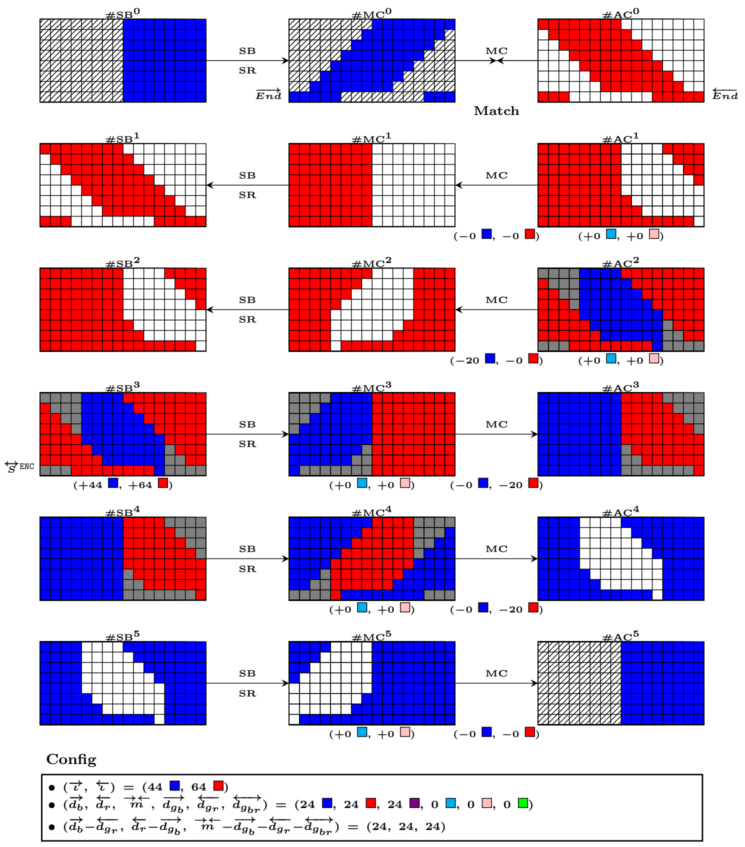

#SB0
#MC0
#AC0
SB
SR
MC
$\overrightarrow{End}$
$\overleftarrow{End}$
Match
#SB1
#MC1
#AC1
(−0 🟦, −0 🟥)
(+0 🟦, +0 🟥)
#SB2
#MC2
#AC2
(−20 🟦, −0 🟥)
(+0 🟦, +0 🟥)
#SB3
#MC3
#AC3
$\overleftrightarrow{S}$ENC
(+44 🟦, +64 🟥)
(+0 🟦, +0 🟥)
(−0 🟦, −20 🟥)
#SB4
#MC4
#AC4
(+0 🟦, +0 🟥)
(−0 🟦, −20 🟥)
#SB5
#MC5
#AC5
(+0 🟦, +0 🟥)
(−0 🟦, −0 🟥)
Config
• $(\overrightarrow{\iota}, \overleftarrow{\iota})$ = (44 🟦, 64 🟥)
• $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}})$ = (24 🟦, 24 🟥, 24 🟪, 0 🟦, 0 🟥, 0 🟩)
• $(\overrightarrow{d_b}-\overleftarrow{d_{g_r}}, \overleftarrow{d_r}-\overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})$ = (24, 24, 24)

Fig. 19: An example of the MITM attack on the OT of the 6-round Grøstl-512

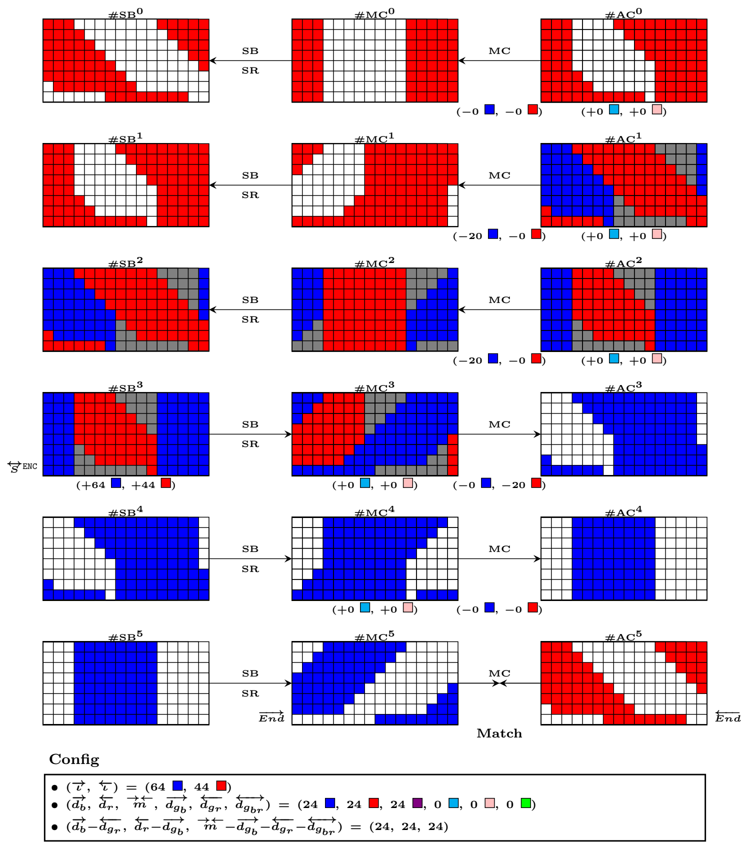

#SB0
#MC0
#AC0
SB
SR
MC
(−0 🟦, −0 🟥)
(+0 🟦, +0 🟥)
#SB1
#MC1
#AC1
(−20 🟦, −0 🟥)
(+0 🟦, +0 🟥)
#SB2
#MC2
#AC2
(−20 🟦, −0 🟥)
(+0 🟦, +0 🟥)
#SB3
#MC3
#AC3
$\overleftrightarrow{S}$ENC
(+64 🟦, +44 🟥)
(+0 🟦, +0 🟥)
(−0 🟦, −20 🟥)
#SB4
#MC4
#AC4
(+0 🟦, +0 🟥)
(−0 🟦, −0 🟥)
#SB5
#MC5
#AC5
$\overrightarrow{End}$
$\overleftarrow{End}$
Match
Config
• $(\overrightarrow{\iota}, \overleftarrow{\iota})$ = (64 🟦, 44 🟥)
• $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}})$ = (24 🟦, 24 🟥, 24 🟪, 0 🟦, 0 🟥, 0 🟩)
• $(\overrightarrow{d_b}-\overleftarrow{d_{g_r}}, \overleftarrow{d_r}-\overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})$ = (24, 24, 24)

Fig. 20: An example of the MITM attack on $P(H') \oplus H'$ in the CF of the 6-round Grøstl-512 (without guessing)

E 10-Round Attack on AES-256 Hashing Modes

We applied our tool set to the hashing modes based on AES-256 reduced to 10 out of the total 14 rounds, and found a configuration leading to an attack with 2^8 time complexity gain, as depicted in Figure 21. It is interesting to note, although the tool set is equipped with the guess-and-determine technique, the result does not require it nor the symmetry properties, *i.e.*, $\overleftarrow{d_{g_r}} = \overrightarrow{d_{g_b}} = \overleftrightarrow{d_{g_{br}}} = 0$. The configuration $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}) = (2, 1, 1)$ gives a pseudo-preimage attack with time complexity 2^{120} and memory complexity 2^8.

F Conversion of the Attacks on OT to Pseudo-Preimage Attacks on Grøstl

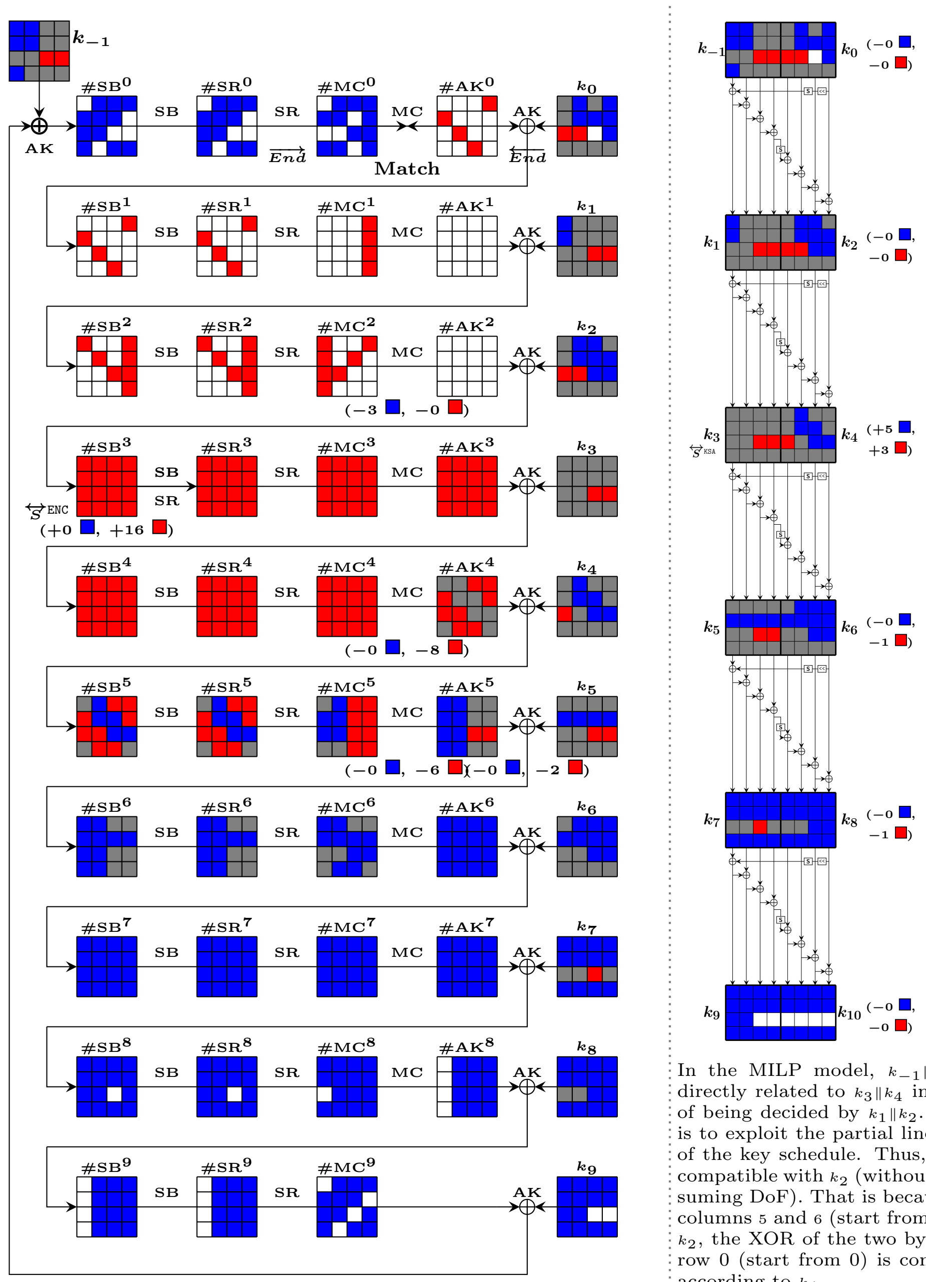
The attack procedures presented in the main body are on the OT of Grøstl. They can be converted into pseudo-preimage attacks on Grøstl combining with similar attack procedures on the $P(H_i) \oplus H_i$ of the CF using the conversion method in [29]. In [29], the psedudo preimage attack, *i.e.*, finding (H, M) such that $X = P(H \oplus M) \oplus H \oplus Q(M)$ and $Trunc_n(P(X) \oplus X) = T$ for a given T, is turn into solving a three-sum problem $X_1 \oplus X_2 \oplus X_3 = 0$, where $X_1 = P(H') \oplus H' \oplus Q(M) \oplus M$, $X_2 = Q(M) \oplus M$, $X_3 = P(H') \oplus H'$ while $H' = H \oplus M$ and X_1 satisfies $Trunc_n(P(X_1) \oplus X_1) = T$. A procedure similar to the generalized birthday attack was used in [29] and is as follows (refer to Fig. 22).

1. Find 2^{x_1} preimages X_1 given the target T of the OT (use the MITM procedure on OT) and store in a lookup table L_1.
2. Choose 2^{x_2} random M with correct padding and calculate $X_2 = Q(M) \oplus M$. Find partial matches on the leftmost b bits between 2^{x_2} X_2's and 2^{x_1} X_1's in L_1, and store the partial matches $(X_1 \oplus X_2, M)$ in a lookup table L_2, of which the size is expected to be $2^{x_1+x_2-b}$.
3. Find 2^{x_3} H' such that the leftmost b bits of X_3 are all zero (use the MITM procedure on $P(H') \oplus H'$ of the CF) , where $X_3 = P(H') \oplus H'$. Store (X_3, H') in a lookup table L_3.
4. Find full matches ($2n$ bits) between the $2^{x_1+x_2-b}$ $X_1 \oplus X_2$'s in L_2 and the 2^{x_3} X_3's in L_3, retrive the corresponding M and H', output $(H' \oplus M, M)$.

Essentially, the b bits for partial matching can locate at any positions in the state instead of the leftmost (noted in [29]); Further, they can be any fixed b-bit linear relations on ℓ bits for $\ell \geq b$, which restrict the values of ℓ bits to a subspace of dimension $\ell - b$, instead of b zero's (this is not used in [29]).

Let the complexity of finding one $2n$-bit X_1 given the n-bit target T be $2^{C_1(2n,n)}$. Let the complexity of finding one $2n$-bit H' given the b-bit restriction on $P(H') \oplus H'$ be $2^{C_2(2n,b)}$. Then, the complexity of the above precedure was approximated in [29] as (after removing some constant factors)

$$2^{x_1+C_1(2n,n)} + 2^{x_2} + 2^{x_1+x_2-b} + 2^{x_3+C_2(2n,b)}.$$



k_{-1}
AK
#SB^0 SB #SR^0 SR #MC^0 MC #AK^0 AK k_0
$\overrightarrow{End}$
Match
$\overleftarrow{End}$
#SB^1 SB #SR^1 SR #MC^1 MC #AK^1 AK k_1
#SB^2 SB #SR^2 SR #MC^2 MC #AK^2 AK k_2
(−3 🟦, −0 🟥)
#SB^3 SB #SR^3 SR #MC^3 MC #AK^3 AK k_3
SR
$\overleftrightarrow{S}^{\text{ENC}}$
(+0 🟦, +16 🟥)
#SB^4 SB #SR^4 SR #MC^4 MC #AK^4 AK k_4
(−0 🟦, −8 🟥)
#SB^5 SB #SR^5 SR #MC^5 MC #AK^5 AK k_5
(−0 🟦, −6 🟥)(−0 🟦, −2 🟥)
#SB^6 SB #SR^6 SR #MC^6 MC #AK^6 AK k_6
#SB^7 SB #SR^7 SR #MC^7 MC #AK^7 AK k_7
#SB^8 SB #SR^8 SR #MC^8 MC #AK^8 AK k_8
#SB^9 SB #SR^9 SR #MC^9 AK k_9
k_{-1} k_0 (−0 🟦, −0 🟥)
k_1 k_2 (−0 🟦, −0 🟥)
k_3 $\overleftrightarrow{S}^{\text{KSA}}$ k_4 (+5 🟦, +3 🟥)
k_5 k_6 (−0 🟦, −1 🟥)
k_7 k_8 (−0 🟦, −1 🟥)
k_9 k_{10} (−0 🟦, −0 🟥)

In the MILP model, $k_{-1}\|k_0$ is directly related to $k_3\|k_4$ instead of being decided by $k_1\|k_2$. This is to exploit the partial linearity of the key schedule. Thus, k_0 is compatible with k_2 (without consuming DoF). That is because in columns 5 and 6 (start from 0) of k_2, the XOR of the two bytes in row 0 (start from 0) is constant according to k_4.

Config

- $(\overrightarrow{\iota}, \overleftarrow{\iota}) = (5$ 🟦, 19 🟥) • $(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}}, \overleftrightarrow{d_{g_{br}}}) = (2$ 🟦, 1 🟥, 1 🟪, 0 🩵, 0 🩷, 0 🟩)
- $(\overrightarrow{d_b}-\overleftarrow{d_{g_r}}, \overleftarrow{d_r}-\overrightarrow{d_{g_b}}, \overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}}) = (2, 1, 1)$

Fig. 21: An example of the MITM attack on 10-round AES-256

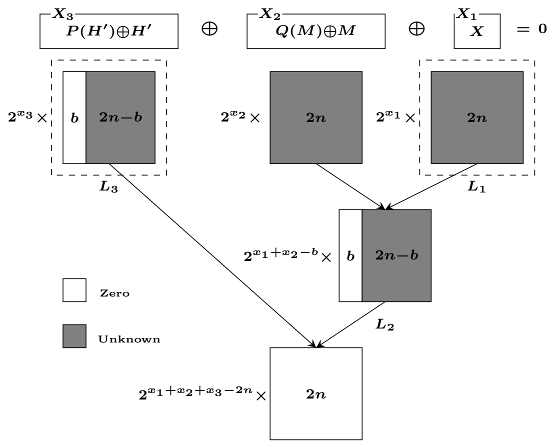


Fig. 22: Outline for pseudo-preimage attack on the Grøstl hash function [29]

Suppose the configuration for attacking $P(H')\oplus H'$ be $(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*)$ (in bytes), when the target is of b-bit, the degree of matching $\overrightarrow{\overleftarrow{m}}^*(b) = \min(\overrightarrow{\overleftarrow{m}}^*, \lfloor b/c \rfloor)$. The complexity $2^{C_2(2n,b)} = \begin{cases} 2^{b-\min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*(b))\cdot c} & b >= 2\cdot\min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*)\cdot c \\ 2^{b-\min(b/2, \overrightarrow{\overleftarrow{m}}^*(b)\cdot c)} & b < 2\cdot\min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*)\cdot c \end{cases}$, where, $b/2$ in the second case is because one cannot fully use all the freedom degrees in that case.

However, we note that it is possible for Step 3 that, finding 2^{x_3} H' requires much less computations than $2^{x_3+C_2(2n,b)}$. This is because one may find a large number of H' within a single MITM procedure when the degree of freedom is large and the number b of bits for partial preimage attack is small. For example, for attacking 6-round $P(H')\oplus H'$ of Grøstl-256 using configuration Fig. 6 with $(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*) = (3,3,3)$, when $b = 3\times 8$, with $2^{3\times 8}$ computations, one can find $2^{3\times 8}$ H' partially matching on b bits. Thus, the amortized complexity of finding one H' is 2^0. When x_3 is larger than b, the complexity of the above Step 3 is 2^{x_3}. Thus, unlike in [29] that limits the lower bound of $2^{C_2(2n,b)}$ to a birthday bound $2^{b/2}$, we use $\overline{C_2(2n,b)}$ to represent the amortized complexity of finding one H', and the complexity is then

$$2^{x_1+C_1(2n,n)} + 2^{x_2} + 2^{x_3+\overline{C_2(2n,b)}} + 2^{x_1+x_2-b}.$$

Because $x_1+x_2+x_3 = 2n$, the complexity is

$$2^{x_1+C_1(2n,n)} + 2^{x_2} + 2^{x_3+\overline{C_2(2n,b)}} + 2^{2n-(x_3+b)}.$$

When $b \le \min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*)\cdot c$, $\overline{C_2(2n,b)} = 0$ for $x_3 \ge b$, that is, the amortized complexity for finding each H' is one. When $b \ge \min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*)\cdot c$, $\overline{C_2(2n,b)} = 2^{b-\min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*)\cdot c}$, increasing b has the same effect as increasing x_3. Thus, we can always take $b = \min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*)\cdot c$ and adjusting x_3.

Denote $\min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overset{\rightarrow\leftarrow *}{m}) \cdot c$ by G_2 meaning the gain over bruteforce attack on $P(H') \oplus H'$. Further, denote $\min(\overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overset{\rightarrow\leftarrow}{m} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) \cdot c$ by G_1 meaning the gain over bruteforce attack on the OT, which equals $n - C_1(2n, n)$. Then, the complexity is

$$2^{x_1+n-G_1} + 2^{x_2} + 2^{x_3} + 2^{x_1+x_2-G_2}.$$

In this formula, there are three parameters x_1, x_2, x_3 with $x_1 + x_2 + x_3 = 2n$. Given G_1 and G_2, one can find the best complexity by adjusting x_1, x_2, and x_3. The order of magnitude of the total complexity is

$$C_T \approx \begin{cases} n - \frac{G_1}{3}, \ \text{with } (x_1 = \frac{2G_1}{3}, x_2 = n - \frac{G_1}{3}, x_3 = n - \frac{G_1}{3}) & \text{if } G_2 \geq \frac{2G_1}{3}. \\ n - \frac{G_2}{2}, \ \text{with } (x_1 = \frac{G_1}{2} + \frac{G_2}{4}, x_2 = n - \frac{G_1}{2} + \frac{G_2}{4}, x_3 = n - \frac{G_2}{2}) & \text{else } G_2 < \frac{2G_1}{3}. \end{cases} \quad (15)$$

The order of magnitude of the memory complexity is

$$C_M \approx \begin{cases} n + \frac{G_1}{3} - G_2, \ \textit{i.e.}, \ x_1 + x_2 - b, & \text{if } G_2 \geq \frac{2G_1}{3}. \\ n - \frac{G_2}{2}, \ \textit{i.e.}, \ x_3, & \text{if } G_2 < \frac{2G_1}{3}. \end{cases} \quad (16)$$

Concrete values of parameters for obtaining the best complexity for various attacks are summarized in Table 3.

Table 3: Concrete parameters for complexity of pseudo-preimage attacks on Grøstl (with recomputed complexities for previous works)

	#R	x_1	x_2	x_3	G_1	G_2	G	C_T	C_M	Ref.
Grøstl-256	5/10	32	240	240	48	40	16	240	232	† [29]
Grøstl-256	5/10	42.67	234.67	234.67	64	64	21.33	234.67	213.33	∗ [18, 31]
Grøstl-256	5/10	48	232	232	72	72	24	232	208	Fig. 15, 16
Grøstl-256	6/10	12	248	252	16	8	4	252	252	∗ [18, 31]
Grøstl-256	6/10	21.33	245.33	245.33	32	24	10.67	245.33	242.67	Fig. 5, 6
Grøstl-512	6/14	128	448	448	192	192	64	448	384	Fig. 19, 20
Grøstl-512	7/14	64	480	480	96	104	32	480	440	Fig. 17, 18
Grøstl-512	8/14	12	508	508	16	8	4	508	508	† [29]
Grøstl-512	8/14	32	488	504	40	16	8	504	504	† [31]
Grøstl-512	8/14	28	496	500	40	24	12	500	500	Fig. 7, 8

$G_1 = \min(\overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overset{\rightarrow\leftarrow}{m} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) \cdot c$ meaning the gain over bruteforce attack on the OT.
$G_2 = \min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overset{\rightarrow\leftarrow *}{m}) \cdot c$ meaning the gain over bruteforce attack on $P(H') \oplus H'$.
G: the gain over bruteforce attack on CF + OT (pseudo-preimage attack).
C: the order of the total complexity.
† The presented parameters for the attacks in [29] and [31] are recomputed by removing constant factors (*e.g.*, the cost C_{TL} for lookup table is replaced by 1) and replacing $C_2(2n, b)$ that is lower bounded by $b/2$ in [29] with $\overline{C_2(2n, b)}$ that can be 2^0 considering the amortized complexity. Thus, all complexities in this table are computed follow the same way.
The claimed complexity of the 5-round attack on Grøstl-256 in [29] is $2^{244.85}$, with $x_1 = 36.93$, $x_2 = 244.93$, $x_3 = 230.13$, $C_1(2n, n) = 206$, $b = 31$.
The claimed complexity of the 8-round attack on Grøstl-512 in [29] is $2^{507.32}$, with $x_1 = 10.50$, $x_2 = 506.50$, $x_3 = 507.00$, $C_1(2n, n) = 495$, $b = 0$.
∗ The 5- and 6-round attacks in [18] are on the OT of Grøstl-256. To convert to pseudo-preimage, we used the results for the case with no truncation in [18] . However, for the 6-round attack, in which guessing are required, it cannot be directly used. Thus, we combined the attack on OT in [18] with the best previous attack on the CF in [31].