Superposition Meet-in-the-Middle Attacks: Updates on Fundamental Security of AES-like Hashing

Zhenzhen Bao[2,3], Jian Guo[2], Danping Shi[1,4(✉)], and Yi Tu[2]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China shidanping@iie.ac.cn
[2] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
baozhenzhen10@gmail.com, guojian@ntu.edu.sg, TUYI0002@e.ntu.edu.sg
[3] Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing, China
[4] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. The Meet-in-the-Middle approach is one of the most powerful cryptanalysis techniques, demonstrated by its applications in preimage attacks on the full MD4, MD5, Tiger, HAVAL, and Haraka-512 v2 hash functions, and key recovery of the full block cipher KTANTAN. The success relies on the separation of a primitive into two independent chunks, where each active cell of the state is used to represent only one chunk or is otherwise considered unusable once mixed. We observe that some of such cells are linearly mixed and can be as useful as the independent ones. This leads to the introduction of superposition states and a whole suite of accompanied techniques, which we incorporate into the MILP-based search framework proposed by Bao *et al.* at EUROCRYPT 2021 and Dong *et al.* at CRYPTO 2021, and find applications on a wide range of AES-like hash functions and block ciphers.

Keywords: Whirlpool, Grøstl, AES hashing modes, MITM, 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 notion of preimage and collision resistance, i.e., it should be computationally difficult to invert the function or find two inputs map to the same digest. 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. Collision attack refers to an algorithm finding different x and x' such that $H(x) = H(x')$ with lesser evaluations than birthday attack, *i.e.*, $2^{n/2}$ computations.

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 [25], 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 [9] 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/IEC [19]. Due to the well understood security and high performance, 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 [12], PHOTON [15], and LED [16] etc.

The Meet-in-the-Middle (MITM) attack has a long history and an important role in various cryptanalysis on various primitives. It was introduced to preimage attacks on hash functions by Aumasson *et al.* [3] and Sasaki *et al.* [27] in 2008. Since then, MITM preimage attack has shown its power on many MD/SHA families of hash functions. That includes the full versions of MD4 [14], MD5 [29], Tiger [14, 32], and HAVAL [17, 27], as well as lightweight block cipher KTANTAN [7, 33].

The basic MITM idea is to find ways to split the cipher into two computational chunks and find the so-called neutral bits from each side, independent of the computation of the state of the other side. Hence, the two chunks can be computed independently but end up at a common state, where previously independent computations are finally pairwise matched. The critical point is that the independence between the two chunks allows their results to be pairwise matchable, enabling this MITM procedure to require fewer computations than a trivial enumeration.

In 2011, MITM was introduced by Sasaki [26] for the first time to preimage attack on AES hashing, invalidating the preimage resistance of the 7-round reduced version. 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, the attack was revisited in [4], and it was 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, the MITM preimage attack was introduced to meet a popular automatic tool, the Mixed-Integer-Linear-Programming (MILP) [5]. This match with MILP enables the previously invented enhancements for MITM, *e.g.*, the initial structure, the selection and cancellation of neutral bits, to evolve to a more generalized form.

The MILP models characterizing the generalized formalization of MITM, produced many improved preimage attacks on AES hashing, penetrated 8-round AES-128 hashing and full rounds of Haraka v2-512. In 2021, MITM-MILP models find more applications not only in preimage attacks but also in key-recovery and collision attacks on AES-like ciphers [10].

A common practice in all these MITM attacks is the byte-oriented decomposition of states, *i.e.*, each useful byte carries the influence of neutral bits from at most one direction. Once influences of neutral bits from both directions reach the same byte, this byte will be considered unusable for the either chunk. An immediate consequence of this practice is that the information from unusable bytes is lost, and any byte in the subsequent round computed from it becomes unusable too.

1.1 Our Contribution

Superposition Meet-in-the-Middle Attack Framework. In this work, different from the byte-oriented decomposition, we propose Superposition MITM attack framework with the help of Superposition State (SupP), under which every byte is viewed as a combination of two virtual bytes separately representing the influence of the neutral bits from one direction each. It becomes obvious to see that, the advantage of this representation is the preservation of linearity, *i.e.*, any state which is a linear combination of neutral bits from both directions will be kept so when linear operations such as `MixColumns` and `AddRoundKey` are applied. SupP opens up the possibility of new local collisions between/within the encryption and key states, thus maximizes the chances of canceling impacts on independence and enables new MITM attack configurations. Enabled by SupP, a suite of techniques is added into the MILP-based search framework proposed by Bao *et al.* [5] and Dong *et al.* [10], which greatly enlarged the solution space and led to rich results on AES-like ciphers.

Guess-and-Determine (GnD) is a popular technique and has countless applications in cryptanalysis against symmetric-key primitives such as stream ciphers [35] and block ciphers [8]. 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 then 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 [14, 30, 34], where the guess allows further computation of more state/message bits which help either extend the attack to more rounds or lower the time complexities. It is noted that GnD has never been incorporated into the MILP models for the MITM attacks in [5, 10].

Bi-direction Attribute-Propagation and Cancellation (BiDir). In previous MITM attacks [4, 5, 10, 26, 30, 34], each computation chunk propagates towards a single direction. Although the idea of adding constraints to some neutral bits in order to cancel their impact on the opposite computation has already appeared, such cancellations are only allowed in one direction in each chunk. In

this work, we allow cancellation between neutral bytes in both directions. This led us to attack configurations with lower time complexities.

Multiple Ways of `AddRoundKey` (MulAK). The identical encryption and key-schedule of Whirlpool allows the `AddRoundKey` to be moved around the `MixColumns` using an equivalent key state already involved in the key-schedule. We add the flexibility of key-addition at different positions in each round into the model, and make it possible to save the double consumption of freedom degrees between the encryption and key-schedule. This simple yet efficient strategy makes it possible to attack one more round on reduced Whirlpool.

Application Results. The superposition MITM attack framework aided by the additional techniques shows its effectiveness when applied to the popular AES-like hashing Whirlpool (an ISO/IEC standard), Grøstl (a finalist of the SHA-3 competition), AES-hashing modes (widely used, *e.g.*, AES-MMO in Zigbee protocols), and tweakable block cipher SKINNY (in its way to be included in ISO/IEC 18033). Broad improvements upon the previous best results on preimage, collision, and key-recovery attacks were obtained. The updates and a comparison with the state-of-the-art results in literature are summarized in Table 1.

For Whirlpool, the preimage resistance of its 7-round reduced version out of the total 10 rounds gets challenged for the first time, a decade after MITM itself challenging its 5-round [26] for the first time, or GnD joining MITM challenging its 6-rounds [30]. Meanwhile, the complexities of preimage attacks on the 5- and 6-round reduced versions are significantly improved. The new attack on 7-round Whirlpool relies on multiple modeling enhancements, including GnD, BiDir, and MulAK. Noticeably, in terms of preimage resistance, the presented attack reduces one round security margin out of the remaining four for this important target.

For Grøstl, there are two main instances, Grøstl-256 and Grøstl-512, named after the size of their digests. The preimage resistance of their longest attacked variants, 6-round Grøstl-256 and 8-round Grøstl-512, get updated eight years after [36]. Improvements with larger complexity reductions are also found on variants with lesser rounds. The improved attacks are achieved by allowing BiDir, in addition to the GnD. Such improvement is not possible using only MITM-GnD, and MITM-GnD is already considered in [24, 36].

In addition, the MILP models for searching preimage attacks can be directly transformed into models searching for collision attacks on hash functions and key-recovery attacks on block ciphers. With the superposition MITM-GnD models, immediate improvements were obtained on many more targets: For Grøstl-256, the collision resistance of its output transformation's longest attacked version, the 6-round, gets updated; For AES-hashing, the collision resistance of hash functions based on AES-128 reduced to 7-round get challenged, more than a decade after rebound attacks challenging its 6-round [13, 21], or two years after quantum attacks challenging its 7-round version [18]. For SKINNY, the security in terms of key-recovery attack in the single-key setting of the longest attacked

version, 23-round SKINNY-n-$3n$, gets updated. Remarkably, data complexity is drastically reduced, *e.g.*, from 2^{52} to 2^{28} for SKINNY-64-192.

On the practical impact of the presented attacks. The high complexities of some presented attacks here might look like a show-stopper for some readers. It is important to note that the gain of the attacks over brute-force search should be the focus and worthy of caution, *e.g.*, 2^{480} may look high for attacking 7-round Whirlpool, however this means a complexity gain by a factor of $2^{512-480} = 2^{32}$. Such a gain can be preserved when one tries to truncate the output digest and the truncated part does not influence the computation of the chunks in the MITM procedure of the attacks. For example, the attack on 7-round Whirlpool has the same gain if the cells in all diagonals except for the first are truncated, which is verified by our implementation. Truncating the output digest from large version hash function is commonly seen in both standard usage such as SHA-512/224 and SHA-512/256, and some engineering implementations. A gain by a factor of 2^{32} could bring the mining efforts from around 2^{72} to 2^{40}, which could be disastrous for Bitcoin.

Organization. The rest of the paper is organized as follows. Section 2 briefly introduces 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 briefly introduces the applications to collision and key-recovery attack. Some extra details on other attacks and figures are provided as Supplementary Material. Relevant source codes can be found via https://github.com/MITM-AES-like.

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 MITM Preimage Attacks, before we can introduce applications in the next sections.

The AES-like hashing in our context refers to those hash functions whose compression function (CF) or output transformation (OT) uses an AES-like round function as depicted in Figure 1, where the state can be viewed as a $\mathrm{N_{row}} \times \mathrm{N_{col}}$ matrix of c-bit cells. There are 4 general operations in order:

- `SubBytes` (SB) applies a non-linear substitution-box operation to each cell.
- `ShiftRows` (SR) cyclically shifts each row by a pre-defined number of positions.
- `MixColumns` (MC) mixes every column, *e.g.*, by multiplying an (MDS) matrix.
- `AddRoundKey` (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 SR can be a cell-permutation.

Table 1: Updated results on (pseudo-) preimage attacks

(Pseudo-) Preimage						
Cipher (Target)	#R	Time-1	Time-2	$(\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{m}^{\leftarrow}, \overrightarrow{d_{g_b}}, \overleftarrow{d_{g_r}})$	Critical Tech.	Ref.
Whirlpool (Hash)	5/10	2^{416}	2^{448}	(16, 12, 16, 0, 0)	Dedicated	[30]
	5/10	$\mathbf{2^{352}}$	$\mathbf{2^{433}}$	(20, 20, 20, 0, 0)	MILP, BiDir, MulAK	Fig. 13
	6/10	2^{448}	2^{481}	(32, 8, 32, 0, 24)	Dedicated, GnD	[30]
	6/10	$\mathbf{2^{440}}$	$\mathbf{2^{477}}$	(9, 24, 24, 15, 0)	MILP, GnD	Fig. 12
	7/10	$\mathbf{2^{480}}$	$\mathbf{2^{497}}$	(16, 4, 16, 0, 12)	MILP, GnD, MulAK	Fig. 3, 11
Grøstl-256 (CF+OT)	5/10	2^{192}	$2^{234.67}$	(8, 8, 8, 0, 0)	Dedicated	‡ [24, 36]
	5/10	$\mathbf{2^{184}}$	$\mathbf{2^{232}}$	(9, 9, 16, 0, 0)	MILP, BiDir	Fig. 14, 15
	6/10	2^{240}	2^{252}	(8, 2, 8, 0, 6)	Dedicated, GnD	‡ [24, 36]
	6/10	$\mathbf{2^{224}}$	$\mathbf{2^{245.33}}$	(4, 20, 16, 12, 0)	MILP, GnD, BiDir	Fig. 5, 6
Grøstl-512 (CF+OT)	7/14	2^{416}	2^{480}	(19, 12, 19, 0, 7)	MILP, GnD, BiDir	Fig. 16, 17
	8/14	2^{472}	2^{504}	(10, 10, 18, 5, 5)	Dedicated	† [36]
	8/14	2^{472}	$\mathbf{2^{500}}$	(9, 5, 10, 0, 4)	MILP, GnD, BiDir	Fig. 7, 8
AES-192 Hashing	9/12	2^{120}	2^{125}	(1, 1, 1, 0, 0)	MILP	[5]
	9/12	$\mathbf{2^{112}}$	$\mathbf{2^{121}}$	(2, 2, 2, 0, 0)	MILP, SupP, BiDir	Fig. 18
Kiasu-BC Hashing	8/10	2^{120}	2^{123}	(1, 4, 4, 0, 0)	Dedicated	[4]
	9/10	$\mathbf{2^{120}}$	$\mathbf{2^{125}}$	(1, 1, 1, 0, 0)	MILP, SupP, BiDir	Fig. 19

(Free-start) Collision						
Cipher (Target)	#R	Time	Mem	Setting & Type	Critical Tech.	Ref.
Grøstl-256 (OT)	6/10	2^{124}	2^{124}	classic collision	MILP	[10]
	6/10	$\mathbf{2^{116}}$	$\mathbf{2^{116}}$	classic collision	MILP, BiDir	Fig. 20
Grøstl-512 (OT)	8/14	2^{248}	2^{248}	classic collision	MILP	[10]
	8/14	$\mathbf{2^{244}}$	$\mathbf{2^{244}}$	classic collision	MILP, BiDir	Fig. 21
AES-128 Hashing	6/10	2^{56}	2^{32}	classic collision	Dedicated	[13, 21]
	7/10	$2^{42.5}$	(2^{48})	quantum collision	Dedicated	[18]
	7/10	$\mathbf{2^{56}}$	$\mathbf{2^{56}}$	classic free-start	MILP, BiDir	Fig. 22
	8/10	$\mathbf{2^{60}}$?	$\mathbf{2^{60}}$?	classic free-start	MILP, SupP, BiDir	Fig. 25
AES-192 Hashing	6/12	2^{56}	2^{32}	classic collision	Dedicated	[13, 21]
	9/12	$\mathbf{2^{56}}$?	$\mathbf{2^{56}}$?	classic free-start	MILP, SupP, BiDir	Fig. 18
AES-256 Hashing	6/14	2^{56}	2^{32}	classic collision	Dedicated	[13, 21]
	10/14	$\mathbf{2^{60}}$?	$\mathbf{2^{60}}$?	classic free-start	MILP, SupP, BiDir	Fig. 26
Kiasu-BC Hashing	6/10	2^{56}	2^{32}	classic collision	Dedicated	[13, 21]
	9/10	$\mathbf{2^{60}}$?	$\mathbf{2^{60}}$?	classic free-start	MILP, SupP, BiDir	Fig. 19

Key-recovery						
Cipher (Target)	#R	Time	Mem	Data	Critical Tech.	Ref.
SKINNY-64-192	23/40	2^{188}	2^{4}	2^{52}	MILP	[10]
	23/40	$\mathbf{2^{184}}$	2^{8}	2^{60}	MILP, SupP	Fig. 23
	23/40	2^{188}	2^{4}	$\mathbf{2^{28}}$	MILP, SupP	Fig. 24
SKINNY-128-384	23/56	2^{376}	2^{8}	2^{104}	MILP	[10]
	23/56	$\mathbf{2^{368}}$	2^{16}	2^{120}	MILP, SupP	Fig. 23
	23/56	2^{376}	2^{8}	$\mathbf{2^{56}}$	MILP, SupP	Fig. 24

– CF: Compression Function; OT: Output Transformation; Dedicated: Dedicated method;
– 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 [34] for Grøstl, respectively.
– † The presented complexities for the attacks in [36] 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 [34] with $\overline{C_2(2n, b)}$ that can be 2^0 considering the amortized complexity. Thus, all complexities are computed follow the same way.
– ‡ The 5- and 6-round attacks in [24] are on the OT of Grøstl-256. To convert to pseudo-preimage, we used the results for the case with no truncation in [24]. However, for the 6-round attack, in which guessing are required, it cannot be directly used. Thus, we combined the attack on OT in [24] with the best previous attack on the CF in [36].
?: Only hold when assuming there exist an attacker who can generate a value of neutral bytes with (amortized) computational complexity $O(1)$. Currently, such an attacker has not been found for this attack. See Remark 4 for more details.

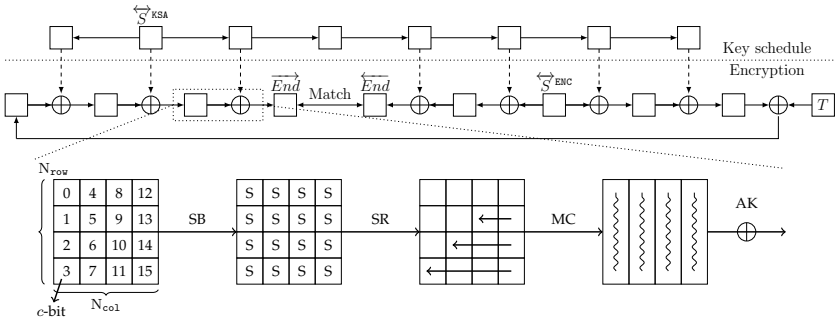


Fig. 1: Overview of AES-like hashing [5]

The MITM Preimage Attacks generally cut the entire encryption process into two chunks (*forward* and *backward chunks*), and there are few *neutral bits* from each side, so that the computation of each chunk can be done independently starting from the cutting point (*starting point*). The two computations end at a common intermediate (partial) state (*matching point*), and the computed results from the two sides can be pairwise matched via a linear relation.

Since MITM has been applied to preimage attacks on MD4, MD5, and HAVAL [3, 22,27,28], it has been developed further and applied to preimage attacks on many other hash functions. Advanced techniques in MITM preimage attacks were developed, including the *splice-and-cut* [2], (probabilistic) *initial structure* [14, 29], and (indirect) *partial matching* techniques [2, 29].

The *splice-and-cut* technique [2] views the input and output of the compression function *connected* through the feedforward operation. The *initial structure* is a few consecutive starting steps, where the two chunks overlap and two sets of neutral words (denoted by $\overrightarrow{\mathcal{N}}$ and $\overleftarrow{\mathcal{N}}$) appear simultaneously at these steps. Typically, one adds *constraints* to the values of $\overrightarrow{\mathcal{N}}$ and $\overleftarrow{\mathcal{N}}$ to limit their impact on the opposite chunk such that steps after the initial structure (forward chunk) can be computed independently of $\overleftarrow{\mathcal{N}}$ and steps before the initial structure (backward chunk) can be computed independently of $\overrightarrow{\mathcal{N}}$. The *(indirect-) partial matching* exploits any easily determined relations between the computed ending states from the two computation chunks instead of requiring values of full states. There may or may not be any *neutral bits* from the message of the compression function (or key of the underlying block cipher).

The Attack Framework. The procedure and complexity of the MITM attack depend on how the computation is decomposed into independent chunks, how neutral bits are selected and constrained, and how to match. Once these configurations have been determined, the basic MITM pseudo-preimage attack on a CF goes as follows (with initial structure and partial matching).

1. Assign arbitrary compatible values to all bytes except those that depend on the neutral bytes.

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

The Attack Complexity. Denote the size of the internal state by n, the degree of freedom in the forward and backward chunks by d_1 and d_2, and the number of bits for (partial-) matching by m, the time complexity of the attack is [4]:

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

3 MILP Model for the Configuration Search

3.1 Basic MILP Model for MITM

For searching MITM attacks using MILP, one should characterize valid attack configurations in MILP language.

To generate MILP models and search for the best MITM attack, the high-level workflow is as follows. Given a targeted cipher, enumerate all possible combinations of starting and matching points (detailed in **Searching Framework.**). For each combination, build the MILP model as follows: 1) claim variables indicating the attribute of each state cell in the input/output of each operation in each round, introduce necessary auxiliary variables; 2) define linear equations and inequalities to express relations and constraints among those variables according to the cipher specification and attack principles; 3) write the objective function corresponding to the optimal computational complexity. For each generated MILP model, use an MILP solver to search its optimal solution; Among solutions of various MILP models, select the best one; parse the solution into the attack.

The challenging parts of the modeling are to formalize new attack settings and techniques into explicit rules and then translate into linear inequalities/equations. The linear inequalities/equations should be exact such that the solutions can one-to-one correspond to valid attack configurations.

In the following, we describe the basic modeling method following the framework in [5]. As introducing the basic model, we directly introduce some natural generalizations, while other generalizations and enhancements are deferred to the sequel subsections.

Notations and Encoding. For the ease of notation, the conversational coloring scheme for describing and visualizing MITM attacks [4,5,10,26,30,34] is adopted here to characterize the attributes of state cells. As in the MILP modeling in [5], the attribute of individual state cell is encoded using two binary variables (x, y) and defined as follows. A cell is

- `Gray` (■) if and only if its value is a predefined constant, thus is known and fixed in *both* forward and backward chunks; Indicating variables $(x, y) = (1, 1)$;
- `Blue` (■) if and only if its value is 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` (■) if and only if its value is 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` (□) if and only if its value is determined by both forward and backward neutral bytes, thus can be computed independently in *neither* chunk; Indicating variables $(x, y) = (0, 0)$.

For convenience, `Black` (■) is used to represent any of the 4 cells (■, ■, ■, □). Additionally, indicating variables β and ω are defined as follows.

- $\beta = \begin{cases} 1 & \text{is inactive, do not contain degree of freedom; is } \texttt{Gray}; \\ 0 & \text{is active, is } \texttt{Blue}, \texttt{Red} \text{ or } \texttt{White}. \end{cases}$
- $\omega = \begin{cases} 1 & \text{cannot be computed in both forward and backward chunks; is } \texttt{White}; \\ 0 & \text{can be computed in at least one direction; is } \texttt{Blue}, \texttt{Red}, \text{ or } \texttt{Gray}. \end{cases}$

Searching Framework. The top-level of a search for the optimal MITM attacks is to enumerate all high-level configurations. A high-level configuration is determined by four parameters, including $\texttt{total}_r$, $\texttt{init}_r^{\mathrm{E}}$, $\texttt{init}_r^{\mathrm{K}}$, and $\texttt{match}_r$, representing the total number of targeted rounds, the location of the initial state in encryption, the location of the initial state in the key-schedule, and the location of the matching point, respectively. For the complete search of $\texttt{total}_r$-round attacks, all possible combinations of values of $\texttt{init}_r^{\mathrm{E}}$, $\texttt{init}_r^{\mathrm{K}}$, and $\texttt{match}_r$ should be tried; each combination corresponds to an independent MILP model; The following description of the modeling method is for an individual model with a fixed ($\texttt{total}_r$, $\texttt{init}_r^{\mathrm{E}}$, $\texttt{init}_r^{\mathrm{K}}$, $\texttt{match}_r$).

When building an MILP model, constraints are imposed on propagation of attributes starting from some initial states (*i.e.*, $\overleftrightarrow{S}^{\texttt{ENC}}$ and $\overleftrightarrow{S}^{\texttt{KSA}}$ in round $\texttt{init}_r^{\mathrm{E}}$ and $\texttt{init}_r^{\mathrm{K}}$) and terminating at some ending states (*i.e.*, $\overrightarrow{End}$ and $\overleftarrow{End}$ in round $\texttt{match}_r$) from two directions.

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

– In the initial states in encryption and key-schedule (*i.e.*, in $\overleftrightarrow{S}^{\texttt{ENC}}$ and $\overleftrightarrow{S}^{\texttt{KSA}}$), the attribute of each cell is constrained to be non-`White`. Thus, its indicating variables can take three assignments, *i.e.*, $(x_i, y_i) \in \{(1,0),(0,1),(1,1)\}$ for $\forall\ i \in \mathcal{N}$, where $\mathcal{N} = \{0, 1, \cdots, \mathrm{N}_{\texttt{row}} \cdot \mathrm{N}_{\texttt{col}} - 1\}$. 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. (2), where $|\mathcal{BL}^{\texttt{ENC}}|$, $|\mathcal{RD}^{\texttt{ENC}}|$, $|\mathcal{BL}^{\texttt{KSA}}|$, and $|\mathcal{RD}^{\texttt{KSA}}|$ denote the number of `Blue`, `Red` cells in $\overleftrightarrow{S}^{\texttt{ENC}}$, and `Blue`, `Red` cells $\overleftrightarrow{S}^{\texttt{KSA}}$, respectively.
– In the ending states at the matching round (*i.e.*, $\overrightarrow{End}$ and $\overleftarrow{End}$), each column is associated with a general variable since the matching is performed column-wise. Specifically, for each pair of input/output columns of the states before and after MixColumns, the associated variable $\overrightarrow{\overleftarrow{m}}_i$ indicates the degree of matching in column i and is 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. (4).

In order for `Blue`- and `Red`-attribute to independently propagate from initial states to ending states and remain matchable, special constraints might be occasionally imposed to indicate whether to cancel impact by consuming degrees of freedom. Concrete constraints on how attributes propagate and how freedom be consumed will be introduced shortly. At this moment, let's denote the accumulated consumed degree of freedom of forward by $\overrightarrow{\sigma}$ and that of backward by $\overleftarrow{\sigma}$. The remaining degrees of freedom in forward and backward (denoted by $\overrightarrow{d_b}$ and $\overleftarrow{d_r}$, respectively) after the occasionally freedom-consuming during attribute-propagation are constrained as in Eq. (3). Note that the $\overrightarrow{d_b}$ and $\overleftarrow{d_r}$ are the essential degrees of freedom for the attack, which determine the attack complexity.

According to the attack complexity in Formula 1, the search for a *valid* attack corresponds to the search for a valid attribute propagation with $\min\{\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}\} \geq 1$; the search of the *optimal* attack corresponds to the search with maximized $\min\{\overrightarrow{d_b}, \overleftarrow{d_r}, \overrightarrow{\overleftarrow{m}}\}$. Thus, the objective of the search model is to maximize a variable $\tau_{\texttt{Obj}}$, which is constrained by Eq. (5).

$$\begin{cases} \overrightarrow{\iota} = |\mathcal{BL}^{\texttt{ENC}}| + |\mathcal{BL}^{\texttt{KSA}}|, \\ \overleftarrow{\iota} = |\mathcal{RD}^{\texttt{ENC}}| + |\mathcal{RD}^{\texttt{KSA}}|. \end{cases} \quad (2) \qquad \begin{cases} \overrightarrow{d_b} = \overrightarrow{\iota} - \overrightarrow{\sigma}, \\ \overleftarrow{d_r} = \overleftarrow{\iota} - \overleftarrow{\sigma}. \end{cases} \quad (3)$$

$$\overrightarrow{\overleftarrow{m}} = \sum_{i=0}^{\mathrm{N}_{\texttt{col}}-1} \overrightarrow{\overleftarrow{m}}_i \,. \quad (4) \qquad \begin{cases} \tau_{\texttt{Obj}} \leq \overrightarrow{d_b}, \\ \tau_{\texttt{Obj}} \leq \overleftarrow{d_r}, \\ \tau_{\texttt{Obj}} \leq \overrightarrow{\overleftarrow{m}} \end{cases} . \quad (5)$$

Basic Rules for Attributes to Propagate and to Match. The attribute propagation and the matching are governed by two types of constraints. The first type is due to the specification of the targeted cipher; the second type is due to the principle of the attack. If attacks are technically improved, the

second type constraints should be adapted to the improvement. In turn, when the second type constraints are properly relaxed, the attack space is expanded so that improved attacks might be discovered.

Remark 1 (Bi-direction attribute-propagation and cancellation (BiDir)). Previously [4, 5, 10, 26, 30, 34], the constraints are imposed such that the propagation of `Red`-attribute can "make a concession" [5] to the propagation of `Blue`, but `Blue` never "gives in" to `Red` for forward computation, and vice versa. Unlike the previous work, in this work, the constraints are relaxed such that the propagation of `Blue` or `Red` attribute can make a concession (cancel its impact by consuming its degree of freedom) to the propagation of the opposite attribute in both directions. The reasons are as follows.

- In the modeling, we introduce neutral bytes into key states as well as in encryption states to bring more degree of freedom. In attribute propagation through encryption, letting `Blue`-propagation concede such that a local `Red`-cell be reserved, that might enable a remote `Red`-cell introduced from the key state be canceled such that `Blue`-propagation be possible in that remote point. That also applies to the case of `Red`-propagation.
- Besides, letting `Blue` to concede and reserve a local `Red`-cell might enable this local `Red`-cell to propagate and combine with other `Red`-cells at a remote point such that their impacts on a certain cell be mutually canceled through MixColumns and benefit `Blue`-propagation.
- In addition, once an attribute of `Blue` or `Red` propagate to the ending states no matter from which direction, it provides source of degree of matching.

Relaxing the model in this way results in attacks with bi-direction attribute-propagation; the attack space is expanded so that improved attacks are possible. (See attack configurations in Figures 3, 5, 7, 13 for examples.) However, this relaxation caused the models to be solved less efficiently. When the efficiency is acceptable, we used such relaxed model for better attacks. Otherwise, we restricted the models such that in certain rounds of the cipher, one attribute can only propagate in one direction and concede in the other direction.

In the following, we describe the MILP modeling with the relaxed model as the basic setting. The difference between this basic modeling and the modeling in [5] and how to get the restricted models will be indicated alongside.

Modeling of the Attribute Propagation through SubBytes and ShiftRows. The SubBytes operation does not change the attribute of the cells, thus is not involved

[5] Here, the use of phrases "make a concession" or "give in" is due to a view of the forward computation and backward computation be in a competition for being able to be propagated unaffected. In previous attacks, for forward computation, propagation of `Blue`-attribute is of high priority. When unaffected propagation of `Blue`-attribute becomes not straightforward due to the existence of cells of `Red`-attribute, we may try to cancel the impact by consuming the freedom of backward to ensure the propagation of `Blue`-attribute. We say such cancellation of impact by consuming freedom "concede", "make a concession " or "give in".

when building the basic model. The ShiftRows operation permutes the state cells, thus can be modeled by a set of equations or hardcoded variable substitution.

Modeling of the Attribute Propagation through AddRoundKey and `XOR` *(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 high-level principle is that `White` is the dominant attribute, `Gray` is the recessive attribute, `Blue` and `Red` are mutually exclusive attributes. The concrete rules are as follows.

- A `White` cell `XOR`ed with a cell of any attribute results in a `White` cell, *i.e.*, $(\square \oplus \blacksquare) \rightarrow \square$;
- a `Gray` cell `XOR`ed with a cell of any attribute results in the cell of the same attribute, *i.e.*, $(\blacksquare \oplus \blacksquare) \rightarrow \blacksquare$;
- a couple of `Blue` and `Red` cells results in a cell deteriorated to `White`, *i.e.*, $(\blacksquare \oplus \blacksquare) \rightarrow \square$;
- 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.*, $(\blacksquare \oplus \blacksquare) \rightarrow \blacksquare$ or $(\blacksquare \oplus \blacksquare) \xrightarrow{-1\times\blacksquare} \blacksquare$;
- 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.*, $(\blacksquare \oplus \blacksquare) \rightarrow \blacksquare$ or $(\blacksquare \oplus \blacksquare) \xrightarrow{-1\times\blacksquare} \blacksquare$.

These propagation rules can be described using only a few variables, thus can be easily translated into inequalities using convex hull computations [31].

Modeling of the Attribute Propagation through MixColumns (MC-RULE). The attribute propagation through MixColumns is governed by a set of column-wise constraints under the name `MC-RULE`. The `MC-RULE` constraints are mostly governed by the branch number ($\mathtt{Br_n}$) of the MixColumns. Again, the high-level principle is that `White` is the dominant attribute, `Gray` is the recessive attribute, `Blue` and `Red` are mutually exclusive attributes. The concrete rules are as follows:

- 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.*, $(\mathrm{N}_{\mathtt{row}} \times \blacksquare) \rightarrow (\mathrm{N}_{\mathtt{row}} \times \blacksquare)$;
- 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.*, $(i \times \blacksquare,\ j \times \blacksquare,\ k \times \blacksquare) \xrightarrow[-(i'+k')\times\blacksquare \text{ if } j\neq 0]{-(j'+k')\times\blacksquare \text{ if } i\neq 0} (i' \times \blacksquare,\ j' \times \blacksquare,\ k' \times \blacksquare,\ \ell' \times \square)$, where

$i+j+k = i'+j'+k'+\ell' = \mathrm{N}_{\mathtt{row}}$ and $\begin{cases} j'+k' < i \le \mathrm{N}_{\mathtt{row}} & \text{if } i \neq 0 \\ j'+k' = \mathrm{N}_{\mathtt{row}} & \text{otherwise} \end{cases}$, $\begin{cases} i'+k' < j \le \mathrm{N}_{\mathtt{row}} & \text{if } j \neq 0 \\ i'+k' = \mathrm{N}_{\mathtt{row}} & \text{otherwise} \end{cases}$. 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 these 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 $c_{\vec{x}}$ and $c_{\vec{y}}$ representing the consumed degree of freedom from `Blue` and `Red`, respectively. Apart from those variables, three auxiliary binary variables are introduced to indicate the following attributes of the input column:

$$\vec{\omega} = \begin{cases} 1 & \text{exists } \mathtt{White} \text{ cell}, \\ 0 & \text{otherwise}. \end{cases} \quad (6) \qquad \vec{x} = \begin{cases} 1 & \text{all are } \mathtt{Blue}/\mathtt{Gray}, \\ 0 & \text{exists } \mathtt{Red}/\mathtt{White}. \end{cases} \quad (7) \qquad \vec{y} = \begin{cases} 1 & \text{all are } \mathtt{Red}/\mathtt{Gray}, \\ 0 & \text{exists } \mathtt{Blue}/\mathtt{White}. \end{cases} \quad (8)$$

The constraints can then be formalized using inequalities listed in Eq. (15, 16, 17).

Remark 2. The set of inequalities will be simpler when one restricts the models such that one attribute can only propagate in one direction and concede in the other direction. For example, for the propagation of the `Blue` attribute to the forward, it only consumes degrees of freedom from `Red`. Thus, the last two inequalities in Eq. (16) can be $(\sum_{i=0}^{\mathrm{N}_{\mathtt{row}}-1} y_i^O) - \mathrm{N}_{\mathtt{iosum}} \cdot \vec{y} = 0$; In Eq. (17), inequalities on $c_{\vec{x}}$ can be removed.

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 those around MixColumns at the matching round, including $\overrightarrow{End}$ and $\overleftarrow{End}$ in encryption and $\overrightarrow{End}^{\mathrm{KMC}}$ or $\overleftarrow{End}^{\mathrm{K}}$ in key-schedule. Note that $\overrightarrow{End}$ and $\overleftarrow{End}$ are states that have not been added with key-state $\overrightarrow{End}^{\mathrm{KMC}}$ or $\overleftarrow{End}^{\mathrm{K}}$. Since AddRoundKey is linear, the influence of AddRoundKey for matching can be formalized using simple rules. Concretely, only a `White` cell in key state destroy the match-ability of the corresponding cell in encryption state. The `Blue` and `Red` cells in key state do not impact the match-ability but on the contrary might provide degree of matching.

Remark 3. For specific targeted cipher whose key-schedule is almost identical to the encryption, *e.g.*, Whirlpool, 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 different in most cases. 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, one should consider both ways. However, we can simply choose to use the key state with fewer `White` cells. That is because, known the propagation direction of the key-schedule (*i.e.*, $\mathtt{init}_r^{\mathrm{K}}$), between the two states $\overrightarrow{End}^{\mathrm{KMC}}$ and $\overleftarrow{End}^{\mathrm{K}}$, the one that is near to the initial key state must have set of `White` cells be subset of that in the remote state, thus has less impact.

The condition for the i-th column to have $\overrightarrow{\overleftarrow{m}}_i$ degree of matching is as follows: denote the number of known cells (*i.e.*, except `White` cells) in the input and output columns by $\overline{m_{ki}}$; when $\overline{m_{ki}} > \mathrm{N}_{\mathtt{row}}$, $\overrightarrow{\overleftarrow{m}}_i = \overline{m_{ki}} - \mathrm{N}_{\mathtt{row}}$; otherwise, $\overrightarrow{\overleftarrow{m}}_i = 0$. Denote the number of white cells by $\overline{m_{wi}}$; Since $\overline{m_{ki}} = 2{\cdot}\mathrm{N}_{\mathtt{row}} - \overline{m_{wi}}$, one have $\overrightarrow{\overleftarrow{m}}_i = \max(0, \mathrm{N}_{\mathtt{row}} - \overline{m_{wi}})$. Accordingly, the concrete system of inequalities modeling `MATCH-RULE` can be obtained as explained in Sect. A and as listed in Eq. (18).

3.2 Superposition States and Separate Attribute-Propagation

Except for allowing bi-direction attribute-propagation, the above basic modeling is in line with previous work [4, 5, 10, 26, 30, 34], where `Blue` and `Red` attributes propagate exclusively and competitively through each operation. Once the two exclusive attribute propagate into the same cell position, this cell is considered being `White`, cannot be independently computed in either forward nor backward.

However, such modeling might miss valid attacks as will be discussed shortly. In our final modeling, the two exclusive attributes `Blue` and `Red` propagate independently as long as it is possible. This is achieved by introducing *superposition states*. Superposition states are all intermediate states in encryption and key-schedule being separated into two virtual states. One virtual state carries one attribute propagation independently of the other attribute propagation. The two virtual states are combined only when going through non-linear operations. In this way, two exclusive attributes can simultaneously propagate through all linear operations in both encryption and key-schedule. See Fig. 3 and 22 for examples, and see Fig. 3 and Fig. 10 for the correspondence between superposition states and real states. This superposition setting captures the independence of the computation in a more essential way, allows new ways of local collisions between/within the encryption and key states, thus, maximize the chances of canceling impacts on independence and enables new ways of MITM decomposition. Concretely, the reasons and benefits of separating propagation with superposition states are as follows.

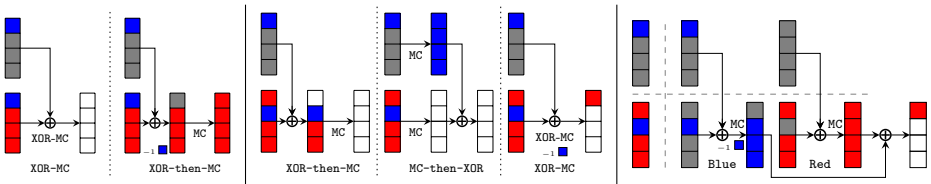
– As mentioned above, under certain constraints, `Blue` propagation can make a concession so that impacts are canceled and `Red` can propagate unaffected, and vice versa. To cancel impact, it requires consuming degrees of freedom. To be able to consume degrees of freedom, different types of operations impose different requirements on the attribute of input states. Concretely, the `XOR-RULE` and `MC-RULE` requires differently for canceling impacts. Attributes

of state cells might not meet the individual requirements but it is actually possible to cancel impacts when combining these two linear operations.
In [5], the propagation through the combination of AddRoundKey and MixColumns is characterized using the set of `XOR-MC-RULE`. In `XOR-MC-RULE`, the `OR` of the attribute indicating variables of encryption and key-state cell is used to indicate 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 their impacts on certain output cells.
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 (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 propagation with superposition states. 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).
Due to this separation with superposition states, `XOR-RULE` and `MC-RULE` without `XOR-MC-RULE` are sufficient (refer to Fig. 2 for an illustration of the separation of attribute-propagation through AddRoundKey and MixColumns).

– Additionally, the key/message-schedule of the ciphers 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 in a key state can be used to cancel impact from the `Blue` component in another key state or in an encryption state, and same for the `Red` component. Consequently, impacts that cannot be canceled in previous models can be canceled now, thus independent propagation gets more chances (see Fig. 22 for an example.)
– Moreover, at the matching point, if a key state cell is in superposition (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 (see Fig. 18 for an example.)

3.3 Multiple Ways of `AddRoundKey` (MulAK)

In Whirlpool, the key-schedule shares the same operations with the encryption except for AddRoundKey. This identity between encryption and key-schedule enables that in encryption, the AddRoundKey can be easily moved around MixColumns. Moving around MixColumns is simply switching between adding $\#\mathbf{KMC}$ or adding $\boldsymbol{k}$, where $\#\mathbf{KMC}$ and $\boldsymbol{k}$ are the states before and after MixColumns in the round function of key-schedule, that is, a switch between adding the real round-key $\boldsymbol{k}$ or an equivalent (up to MixColumns) round-key $\#\mathbf{KMC}$.



(a) `XOR-MC-RULE` does not cover a propagation

(b) `XOR-MC-RULE` is necessary if not in superposition

(c) `XOR-RULE` and `MC-RULE` are sufficient once in superposition

– 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 superposition, (`Blue` and `Red`) attribute-propagation ruled by `XOR-MC-RULE` cannot be obtained directly by `XOR-then-MC-RULE` and `MC-then-XOR-RULE`.
– In 2c, with (`Blue` and `Red`) attribute-propagation in superposition, `XOR-RULE` and `MC-RULE` are sufficient (2c achieve the same results as achieved by `XOR-MC-RULE` 2b).

Fig. 2: Combination of linear operations and superposition attribute-propagation

Moving AddRoundKey before MixColumns and using $\#\mathbf{KMC}$ can bring more advantages in some cases. Take the case where we need to reserve `Blue` by consuming `Red` for example. Let's focus on one column of the state. Suppose there is one `Red` cell in that column of $\#\mathbf{MC}$ and one `Red` cell in the same cell-position in $\#\mathbf{KMC}$; the influence of these single `Red` cells will be diffused into the whole columns in both states if there is no constraint. In such case, adding $\#\mathbf{KMC}$ with $\#\mathbf{MC}$ and letting the `Red` cell in $\#\mathbf{MC}$ be canceled by the `Red` cell in $\#\mathbf{KMC}$ achieve the same cancellation effect but consume fewer degrees of freedom than adding $\#\mathbf{AK}$ with $\boldsymbol{k}$ (take using $\#\mathbf{MC}^5 \oplus \#\mathbf{KMC}^4$ vs. $\#\mathbf{AK}^5 \oplus \boldsymbol{k}_5$ in Fig. 3 for an example). Similarly, it is also possible that adding $\#\mathbf{AK}$ with $\boldsymbol{k}$ after the MixColumns has more advantages than adding $\#\mathbf{MC}$ with $\#\mathbf{KMC}$ (take using $\#\mathbf{SB}^4 \oplus \boldsymbol{k}_3$ vs. $\#\mathbf{MC}^5 \oplus \boldsymbol{k}_2$ in Fig. 3 for an example).

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 $\#\mathbf{MC} \oplus \#\mathbf{KMC}$, `MC-AK-RULE` corresponds to using $\#\mathbf{AC} \oplus \boldsymbol{k}$; for backward computation, `AK-MC-RULE` corresponds to using $\#\mathbf{SB} \oplus \boldsymbol{k}$, and `MC-AK-RULE` corresponds to using $\#\mathbf{MC} \oplus \#\mathbf{KMC}$;

In addition, since the MixColumns is column-wise, different columns can apply different ways (apply `AK-MC-RULE` or `MC-AK-RULE`) independently. Besides, since MixColumns is linear, with superposition states, different attributes of `Blue` and `Red` can independently apply `AK-MC-RULE` or `MC-AK-RULE`.

Integrating such flexibility of choice into the MILP models expand the covered attack space but make the solving less efficient. So, to decide in which way to proceed for each column, we use the heuristic that when the key-schedule has already consumed some degree of freedom in that column, we apply `MC-AK-RULE`; otherwise, apply `AK-MC-RULE`.

3.4 Enhanced Model with Guess-and-Determine (GnD)

The Guess-and-Determine Strategy. In MITM attacks, one unfavorable situation is that a single or a few unknown cells in the input column of `MC` makes all cells in the output column unknown (refer to point 1 of `MC-RULE`). This is inevitable due to the diffusion of MixColumns, especially the property of the MDS matrix.

To turn things around in such situations, Sasaki *et al.* in [30] invent the following guess-and-determine strategy. That is, guess values of the few unknown cells to continue the propagation of attribute to reach the matching point, after (partial) matching, check the consistency of the few guessed cells. If the gained degree of matching is sufficient and the required guesswork is very little, one can still achieve a better attack complexity than a brute-force attack.

Concretely, denote 2^c by ς (where c is the number of bits in each cell). 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`. [6]

The framework of the MITM attack with GnD is as follows:

1. Assign arbitrary compatible values to all cells except for those depending on the neutral bits, and assign arbitrary values to the constants in pre-defined constraints on neutral bits;
2. Compute values $\{\overrightarrow{v}_i\}$ of forward neutral bits and values $\{\overleftarrow{v}_i\}$ of backward neutral bits fulfilling pre-defined constraints.
3. For all $\varsigma^{\overrightarrow{d_b}}$ values $\{\overrightarrow{v}_i\}$ of forward neutral bits, 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 bits, 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 compatible. For compatible 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 by changing values assigned at that step.

In the above framework of the MITM attack with GnD,

- 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 is $\varsigma^{\overrightarrow{d_b}+\overrightarrow{d_{g_b}}+\overleftrightarrow{d_{g_{br}}}+\overleftarrow{d_r}+\overleftarrow{d_{g_r}}+\overleftrightarrow{d_{g_{br}}}-\overrightarrow{m}-\overleftarrow{m}}$;

[6] Since we allow bi-direction attribute propagation in superposition states, it might bring benefit to guess a superposition cell to be simultaneously `Blue` and `Red`. Thus, here is a slight generalization of the previous GnD strategy.

– 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

$$\begin{aligned} &\varsigma^n \cdot (\varsigma^{-(\overleftarrow{d_r}-\overrightarrow{d_{g_b}})} + \varsigma^{-(\overrightarrow{d_b}-\overleftarrow{d_{g_r}})} + \varsigma^{-(\overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})}) \\ \simeq\ &\varsigma^n \cdot \max(\varsigma^{-(\overleftarrow{d_r}-\overrightarrow{d_{g_b}})}, \varsigma^{-(\overrightarrow{d_b}-\overleftarrow{d_{g_r}})}, \varsigma^{-(\overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})}) \end{aligned} \tag{9}$$

Accordingly, 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 GnD, 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 of each cell in the input of MixColumns are thus constrained together with attribute-indicating variables of the cell in output state of last operations (*e.g.*, ShiftRows or AddRoundKey). Besides, according to the complexity Eq. (9) of the attack with GnD, 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. (10) and (11).

$$\begin{cases} \overrightarrow{d_{g_b}} = \sum_{r=0,i=0}^{\mathtt{total}_r-1,n-1} {g_b}_i^r, \\ \overleftarrow{d_{g_r}} = \sum_{r=0,i=0}^{\mathtt{total}_r-1,n-1} {g_r}_i^r, \\ \overleftrightarrow{d_{g_{br}}} = \sum_{r=0,i=0}^{\mathtt{total}_r-1,n-1} {g_{br}}_i^r, \end{cases} \tag{10}$$

$$\begin{cases} \tau_{\mathtt{Obj}} \le \overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \\ \tau_{\mathtt{Obj}} \le \overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \\ \tau_{\mathtt{Obj}} \le \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}. \end{cases} \tag{11}$$

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, generally, guessing are only required around the matching point. Thus, when trade-off between search quality and search efficiency is needed, these guessing constraints can be added only for rounds around the matching point.

3.5 Transforming to Models for Searching for Collision Attacks

A MITM partial target preimage attack whose matching point is at the last round can be transformed into a collision attack, as described by Li *et al.* in [23]. Thus, the searching of MITM preimage attacks can be constraint to search for such partial target preimage attacks, and then translate into valid collision attack. Concretely, one have the follows.

Definition 1 ((t-bit) partial target pseudo preimage attack on CF [23]). *Given the value v of t bits in T, find (h', m') such that t bits in $T' := \mathsf{CF}(h', m')$ at the same position as the t bits in T, take value v.*

Let $\mathcal{A}$ be a random algorithm that can find a t-bit partial target pseudo preimage with a complexity of 2^s. This complexity can be in the average sense, which means that $\mathcal{A}$ can generate 2^r different (h', m', T') in one time with a complexity 2^{r+s}. Additionally, assume $\mathcal{A}$ output different (h', m', T')'s for different calls. Then, a free-start collision attack goes as follows.

- Set t-bit arbitrary value d' to the target. Call $\mathcal{A}$ with d' as the partial target and run it to obtain $2^{(n-t)/2}$ values of (h', m', T').
- From the $2^{(n-t)/2}$ values of (h', m', T'), find a collision on the remaining $(n-t)$ bits of T'.

According to the birthday paradox, with a high probability, the above procedure produces a valid collision pair. The total complexity of this attack is $2^{s+(n-t)/2}$.

The above complexity analysis misses the details of the MITM complexity. In the following, we re-formalize the complexity analysis of the above collision attack using its correspondence with the MITM-GnD preimage attack. Essentially, the $\mathcal{A}$ can be the core of a MITM pseudo preimage attack on CF. The t-bit partial target corresponds to $c\times \overrightarrow{\overleftarrow{m}}$ bits for partial matching in the MITM attack. Essentially, the t bits for partial matching can be a fixed t-bit linear relations on ℓ bits for $\ell \geq t$, which restrict the values of ℓ bits to a subspace of dimension $\ell - t$. Thus, matching through `MixColumns` instead of matching at exact same bit positions does not have essential influence on the effectiveness of the attack. The 2^s computational complexity of $\mathcal{A}$ corresponds to $\varsigma^{\max\left\{(\overrightarrow{d_b}+\overrightarrow{d_{g_b}}+\overleftarrow{d_{g_{br}}}),(\overleftarrow{d_r}+\overleftarrow{d_{g_r}}+\overleftarrow{d_{g_{br}}}),(\overrightarrow{d_b}+\overrightarrow{d_{g_b}}+\overleftarrow{d_{g_{br}}}+\overleftarrow{d_r}+\overleftarrow{d_{g_r}}+\overleftarrow{d_{g_{br}}}-\overrightarrow{\overleftarrow{m}})\right\}}/\varsigma^{\overrightarrow{d_b}+\overleftarrow{d_r}+\overleftarrow{d_{g_{br}}}-\overrightarrow{\overleftarrow{m}}}$, which equals to $\varsigma^{\max\left\{(\overrightarrow{d_{g_b}}-\overleftarrow{d_r}),(\overleftarrow{d_{g_r}}-\overrightarrow{d_b}),(\overrightarrow{d_{g_b}}+\overleftarrow{d_{g_r}}+\overleftarrow{d_{g_{br}}}-\overrightarrow{\overleftarrow{m}})\right\}}/\varsigma^{-\overrightarrow{\overleftarrow{m}}}$. Thus, the total complexity is $\varsigma^{\max\left\{(\overrightarrow{d_{g_b}}-\overleftarrow{d_r}),(\overleftarrow{d_{g_r}}-\overrightarrow{d_b}),(\overrightarrow{d_{g_b}}+\overleftarrow{d_{g_r}}+\overleftarrow{d_{g_{br}}}-\overrightarrow{\overleftarrow{m}})\right\}}/\varsigma^{-\overrightarrow{\overleftarrow{m}}} \times \varsigma^{\frac{n-\overrightarrow{\overleftarrow{m}}}{2}}$, *i.e.*,

$$\varsigma^{-\min\left\{(\overleftarrow{d_r}-\overrightarrow{d_{g_b}}),(\overrightarrow{d_b}-\overleftarrow{d_{g_r}}),(\overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftarrow{d_{g_{br}}})\right\}} \times \varsigma^{\frac{n+\overrightarrow{\overleftarrow{m}}}{2}}. \tag{12}$$

For a valid attack (better than birthday attack), the following should be fulfilled

$$\left\{\overrightarrow{d_b} - \overleftarrow{d_{g_r}} > \overrightarrow{\overleftarrow{m}}/2, \quad \overleftarrow{d_r} - \overrightarrow{d_{g_b}} > \overrightarrow{\overleftarrow{m}}/2, \quad \overrightarrow{d_{g_b}} + \overleftarrow{d_{g_r}} + \overleftarrow{d_{g_{br}}} < \overrightarrow{\overleftarrow{m}}/2\right\}. \tag{13}$$

For searching for the best attack, the objective function is the same as that for preimage attack, *i.e.*,

$$\max\left\{\min\left\{\overrightarrow{d_b}-\overleftarrow{d_{g_r}},\overleftarrow{d_r}-\overrightarrow{d_{g_b}},\overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}}\right\}\right\}. \tag{14}$$

Remark 4. A solution to the MILP model only corresponds to a valid attack configuration but does not formally imply a valid attack. For the attack complexity in Eq. 1, 9, and 12 to be valid, the attacker should be able to generate each value of neutral bytes with (amortized) computational complexity $O(1)$. In some obtained attack configurations, the neutral bytes are constrained in such a sophisticated manner that it is not trivial to efficiently generate their values. For such non-trivial cases, we propose to use local meet-in-the-middle procedures to solve the problem (as done for the pseudo-preimage attack on 7-round Whirlpool in Sect. 4.1, the pseudo-preimage attack on 6-round Grøstl-256 in Sect. C.2, and the free-start collision attack on 7-round AES-128 in Fig. 22). Such local meet-in-the-middle procedures might be found by manual analysis or aided by automatic tools, such as the Automatic-tool from [8]. Sometimes, to achieve amortized computational complexity $O(1)$, it is necessary to pre-compute values of neutral bytes for many fixed bytes (enumerated in the outermost loop of the attack) at once. However, for some very complex cases (marked with '?' in Table 1), even aided by automatic-tools and considering amortized complexity, it might be difficult to find pre-computation procedures with total complexity lower than the main procedure. As a theoretical problem for all attacks under this framework, this problem of efficiently generating values of neutral bits stays open.

3.6 Exploit Symmetry of the Ciphers

Integrating that many technical generalizations, the search space is greatly enlarged but the search is slow down. One needs to make a trade-off between the quality of the searching result and the efficiency of the search. Apart from the ways of trade-off mentioned alongside the introduction of the modeling, one can reduce the problem scale using symmetry of the ciphers. Specifically, in many AES-like designs, the states and operations have symmetric structures and parameters. The symmetry allows projecting attacks on small-size versions to that on large-size versions. Concretely, a state of 8×8 cells can be viewed as a 2×2 matrix of state of 4×4 cells, or a 1×2 matrix of state of 8×4 cells. Suppose the ShiftRows parameters of the 8×8 state version are $\{p_0, p_1, \ldots, p_7\}$ and $(p_{i+4} \mod 4) = p_i$ for $i \in \{0, \ldots, 3\}$, and 2 times the branch number of MixColumns of a 4×4 state version is no less than that of the 8×8 state version. Then, obtaining an attack on the 4×4 state version, cloning the state patterns four times and placing them in a 2×2 matrix, this will result in an attack on the 8×8 state version. The projection from the attack configuration in Fig. 10 to the attack configuration in Fig. 11 is one example of such correspondence. 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 Preimage Attacks on Whirlpool

Whirlpool [6] is a block-cipher based secure hash function designed by Rijmen and Barreto in 2000 and has been adopted as an ISO/IEC standard. It produces a 512-bit hash value using Miyaguchi-Preneel-mode (MP-mode) CF. The CF is defined as $\mathsf{CF}(H_i, M_i) = E_{H_i}(M_i) \oplus M_i \oplus H_i$, where E is a 10-round AES-like block cipher with 8×8-byte internal states. This underlying block cipher takes the 512-bit chaining value H_i as the key material and the 512-bit message block M_i as the input of the encryption. Both the encryption and key-schedule use round functions consisting of four operations:

- SubBytes (SB) applies the Substitution-Box to each byte.
- ShiftColumns (SC) cyclically shifts the j-column downwards by j bytes.
- MixRows (MR) multiplies each row of the state by an MDS matrix.
- AddRoundKey (AK or AddRoundConstants AC) XOR the round key (or round constants in key-schedule) to the state.

Note that the last round is a complete round which is unlike AES; a whiting key is added before the first round of encryption. However, the effect of whitening key will be canceled in the splice-and-cut MITM attacks due to the feed-forward mechanism of MP-mode. Please refer to [6] for a detailed description of Whirlpool.

4.1 New Attacks Resulted from Applying the MILP Modeling

Applying the MILP modeling 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, guess-and-determine is not required, but allowing bi-direction attribute-propagation/cancellation is essential to achieve the best complexity. For 6-round attacks, guess-and-determine is the critical technique that enables the improved results. For 7-round attacks, guess-and-determine and multiple ways of AddRoundKey (flexible choices from applying AK-MC-RULE or MC-AK-RULE) are the two critical points.

The remaining 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 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 Figures 11, 12, and 13 for visualizations of configurations of the 7-, 6-, and 5-round attacks, and refer to Sect. B for a summary of notations.

The attack on 7-round Whirlpool can be obtained by searching on a small version with $\mathrm{N_{row}} \times \mathrm{N_{col}} = 4 \times 4$ states and then projecting to attack on the 8×8-state version. In the sequel, to facilitate readers to find the correspondence between the text description and the code implementation for experimental verification of the attack, we describe it using the small version with $\mathrm{N_{row}} \times \mathrm{N_{col}} = 4 \times 4$

states that is depicted in Fig. 3. One can quickly project this attack on the small version to the real version of Whirlpool (refer to Sect. 3.6 and the correspondence between Figures 10 and 11 for illustrations). The attack complexity on the real version is the fourth power of that on the small version.

The attack configuration in Fig. 3a is found with the MILP models. With this configuration, one can conceive an equivalent configuration shown in Fig. 3b. Following both configurations, one can devise the attack, with different procedures to compute initial values of backward neutral bytes. The latter (Fig. 3b) is more direct for the computation; thus, it will be used in the following description. In the following description, all referred states are actual states that are the combination of two virtual states.

Compute initial values of neutral bytes in ***Red****.* To get the initial values of backward neutral bytes (in Red), 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 4^{th} round (k_4) will be canceled in the next round by adding $\#\mathbf{KMC}^4$ without consuming degrees of freedom.

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 $\{\#\mathbf{MC}^2, \#\mathbf{AK}^2, \#\mathbf{SB}^3, \#\mathbf{MC}^3, \#\mathbf{AK}^3\}$ (refer to Fig. 3b).

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 $\mathbf{c}$.

Note that values of Blue cells are constrained such that they have no impact on the last anti-diagonal of $\#\mathbf{MC}^2$ and the first diagonal of $\#\mathbf{AK}^3$ (are mapped to constants on these state cells). Denote the constants by $\#\mathbf{MC}^2_{\mathbf{c}}[3, 6, 9, 12]$ and $\#\mathbf{AK}^3_{\mathbf{c}}[0, 5, 10]$. 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 column-by-column using predetermined constant-impacts on cells in $\#\mathbf{MC}^2$, 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).

The Main Procedure (refer to Fig. 3b). Assign arbitrary values to those constant impacts of `Blue` on `Red` in $\#\mathbf{MC}^2$ (*i.e.*, $\#\mathbf{MC}^2_{\mathtt{C}}[3,6,9,12]$) and to $\#\mathbf{AK}^3_{\mathtt{C}}[3]$. Set $\#\mathbf{AK}^3[0,5,10,12,13,14]$ be 0, $k_3[0,5,10]$ and $\#\mathbf{SB}_4[0,5,10,15]$ be $\text{Sbox}^{-1}[0]$.

Compute the initial values of backward neutral bytes as described above and as shown in Algorithm 1, store the result in $\overrightarrow{\mathcal{T}_{\mathtt{Init}}}$.

1. For a new 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 $\#\mathbf{KMC}^4$ (this is expected to repeat ς^{11} times before terminating from Step 1(b)iiiE according to Sect. 3.4).
   (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 `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 $\overleftarrow{v}_i \oplus \text{Sbox}^{-1}[0]$, 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 `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, and *terminate*. 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, $\overrightarrow{d_b} = 4$, $\overleftarrow{d_r} = 1$, $\overleftarrow{d_{g_r}} = 3$, $\overrightarrow{\overleftarrow{m}} = 4$, $\varsigma = 2^8$, and $\overrightarrow{d_{g_b}} = \overleftrightarrow{d_{g_{br}}} = 0$. According to Eq. (9), the complexity of the whole attack on the

small version is therefore $\varsigma^n \cdot (\varsigma^{-(\overleftarrow{d_r}-\overrightarrow{d_{g_b}})} + \varsigma^{-(\overrightarrow{d_b}-\overleftarrow{d_{g_r}})} + \varsigma^{-(\overrightarrow{\overleftarrow{m}}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}})}) = \varsigma^{16} \cdot \max(\varsigma^{-1} + \varsigma^{-(4-3)} + \varsigma^{-(4-3)}) = \varsigma^{15} = 2^{120}$

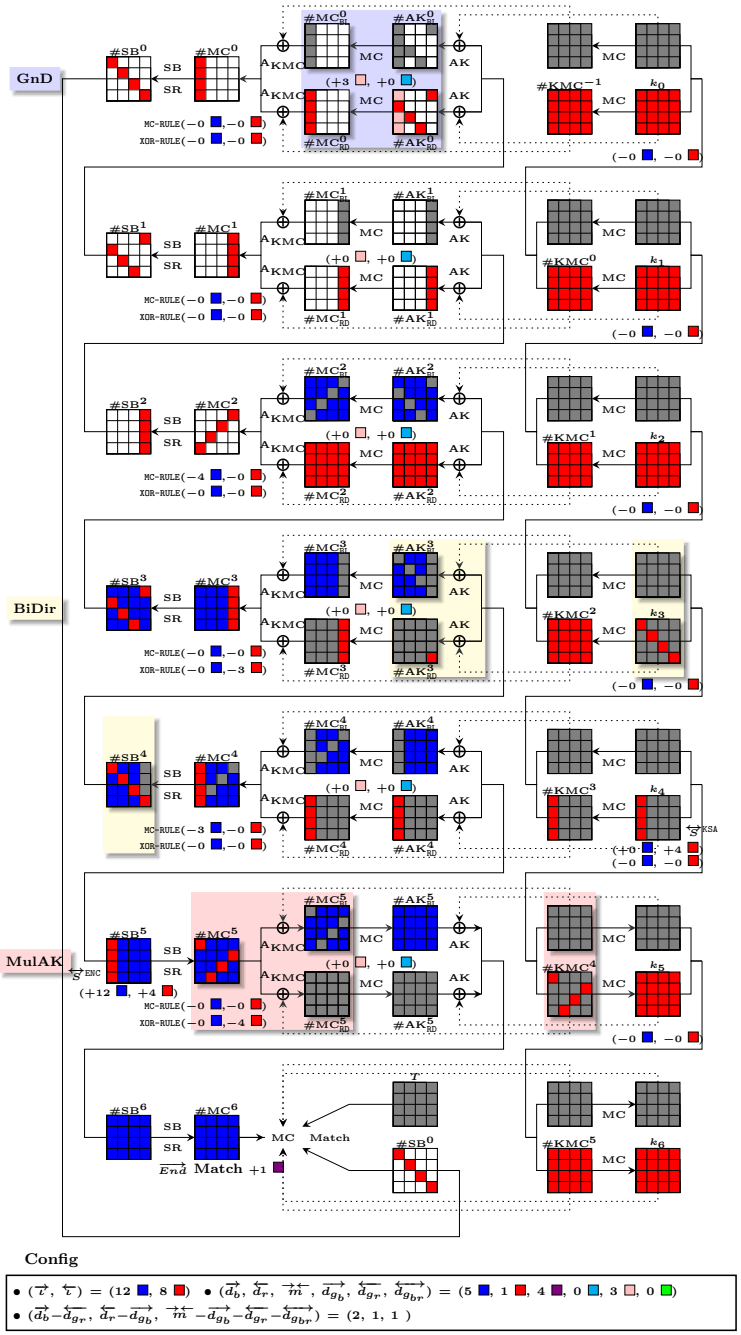
Projecting to the real version of Whirlpool ($n = \mathrm{N}_{\mathtt{row}} \times \mathrm{N}_{\mathtt{col}} = 16 \cdot 4 = 64$), the complexity will be $(2^{120})^4 = 2^{480}$. Concretely, the attack configuration will be $\overrightarrow{d_b} = 4 \cdot 4 = 16$, $\overleftarrow{d_r} = 1 \cdot 4 = 4$, $\overleftarrow{d_{g_r}} = 3 \cdot 4 = 12$, $\overrightarrow{\overleftarrow{m}} = 4 \cdot 4 = 16$, $\varsigma = 2^8$, 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 by the main procedure is the memory taken by $\overrightarrow{\mathcal{T}}$, whose size is limited by the degree of freedom for forward, that is 2^{32} for the small version and 2^{128} for the real version.

To further verify the attack and its complexity analysis, we implemented the attack on the small version with 4×4 states. The round functions of encryption and key-schedule of small Whirlpool CF is simulated using the round function of AES. To make the verification practical, the experiments aim for partial matching instead of full-state matching, while preserving the complexity gain. Concretely, the goal is to match m bits with complexity $\max\{2^{32}, 2^{m-8}\}$. Please refer to https://github.com/MITM-AES-like/Whirlpool_7R for results on $m \in \{36, 40, 44, 48\}$.

The attack on 6-round Whirlpool. When searching on the full-size version ($\mathrm{N}_{\mathtt{row}} \times \mathrm{N}_{\mathtt{col}} = 8 \times 8$), results with asymmetric patterns were found. One example is depicted in Fig. 12. Following the configuration, one can devise an attack on 6-round of Whirlpool with better complexity than previous ones. 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 $2^{24 \times 8} = 2^{192}$. 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.

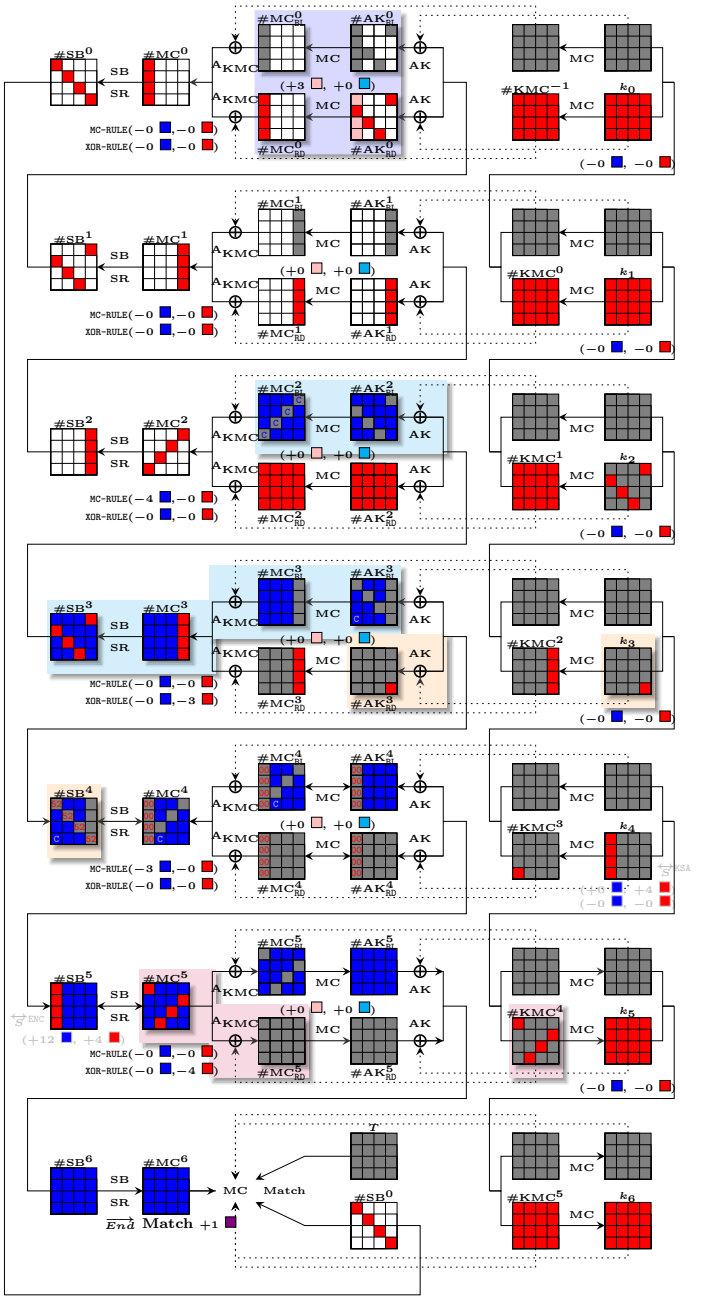
Conversion from Pseudo-Preimage to Preimage Attacks has been discussed in previous works. Here, we follow the *Two Types of Last Block Attacks* from [30,34]. Denote the time complexity of inverting the reduced-Whirlpool compression function as 2^ℓ. A random message fulfills the padding rule of Whirlpool with probability 2^{-9}, hence it costs $2^{\ell+9}$ 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 $2^{(512+\ell)/2}$ and sums to $2^{(512+\ell)/2} + 2^{\ell+9}$ to find a long and full preimage. Detailed conversion results are summarized in Table 1. We note further complexity optimizations are possible, by finding



(a) Configuration automatically found

This implies an attack on the full version (8×8) as shown in Fig. 11 with configuration $((\overrightarrow{d_b} - \overleftarrow{d_{g_r}},\ \overleftarrow{d_r} - \overrightarrow{d_{g_b}},\ \overrightarrow{m} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) = (8,\ 4,\ 4))$

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



(b) Equivalent configuration used in the experimental verification

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 (cont.)

pseudo-preimages under multi-target scenarios and utilizing them in the "unbalanced meet-in-the-middle" phase as discussed in [14].

4.2 Discussions on the New Attacks

The previous best attack on Whirlpool is up to 6-round [30]. In the 6-round attack in [30], although the freedom degree in the key states is exploited, the computational chunks between the key schedule and the encryption data-path are designed to be almost identical due to the limitation of manual analysis. In contrast, in the new attack, the computational chunks between the key schedule and the encryption data-path are largely different. 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 better.

Attacking one more round than the previously best attacked, new strategies are required on top of those appeared in [30]. In the new 7-round attack, apart from GnD, flexibly using of equivalent round-keys $\#\textsc{kmc}$ or the real round keys k (MulAK) is critical to save degrees of freedom. Moreover, complex non-linear constraints must be imposed on `Blue` cells to let `Blue` propagates towards backward besides forward (BiDir), and then cancel their impacts on `Red`-attribute propagation. Thus, efficient procedure (*e.g.*, the *local meet-in-the-middle procedure*) for obtaining initial values of neutral cells fulfilling the non-linear constraints is necessary here.

5 Application to Preimage Attacks on Grøstl

Grøstl, proposed by Gauravaram *et al.* in [12], 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 CF and OT. Please refer to Sect. C.1 or [12] for a detailed description of Grøstl.

5.1 New Attacks Resulted from Applying the MILP Modeling

Applying the MILP modeling approach described in Sect. 3 on the OT (and on the $P(H_i) \oplus H_i$ part of the CF) of Grøstl-256 and Grøstl-512, new attacks are found on 6-round and 8-round, respectively. Besides, many efficient attacks on shorter rounds are found.

For the 6-round attack on the OT of Grøstl-256, GnD is the essential that enables to cover one more attacked round than previous 5-round attacks [34]; BiDir is the essential that enables better complexity than previous 6-round attacks [24]. Besides, many inferior attacks than the presented best one on 6-round Grøstl-256 are 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, GnD is the essential that enables to achieve better complexity than the previous 8-round attack [34]. Besides, compared to the 8-round attack in [34], 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 configurations of various attacks can be found in Figures 5, 6, 7, 8, 16, 17, 14, 15. The concrete attack procedures on 6-round OT of Grøstl-256 and 8-round OT of Grøstl-512 are presented in Sect. C.

Conversion to Pseudo-Preimage Attacks. The attacks on the OT of Grøstl can be converted into pseudo-preimage attacks on Grøstl combining with similar attacks on the $P(H) \oplus H$ of the CF using the conversion method in [34]. The complexity of the converted pseudo-preimage attacks are summarized in Table 1. More details about the conversion can be found in Sect. F

5.2 Discussions on the New Attacks

An interesting feature of the best attack on 6-round Grøstl-256 depicted in Fig. 5 is that, with necessary guessing, the computation of `Blue` covers the full 6-round. That is, the `Blue` propagates to backward besides forward and contributes to degrees of matching from both sides. Besides, like the attack on 7-round Whirlpool, it requires a non-trivial local meet-in-the-middle procedure to compute initial values of neutral bytes.

Note that, obtaining the previous best attacks on Grøstl, the work in [34] has already been assisted with automatic searching, and the 5-round attack on Grøstl-256 in [34] was claimed to be optimal. However, the optimality hold only in a restricted search space, apart from lacking the consideration of the GnD technique. In contrast, the presented 5-round attacks on Grøstl-256 in Figures 14 and 15 achieve better complexity than that in [34] and is optimal in an expanded search space where BiDir and GnD are considered.

6 Applications to Collision and Key-recovery Attacks

The MILP models for searching for the best preimage attacks can be directly transformed to search for the best collision attacks on hash functions and key-recovery attacks on block ciphers. For collision attacks, according to the analysis in Sect. 3.5, what needs to be done is to simply restrict the matching point to be at the last round and add constraints shown in Eq. 13. Applying to Grøstl's OT, AES-hashing, Kiasu-BC-hashing [7], new and improved attacks were found.

For key-recovery attacks, upon the MILP models for preimage attack, one simply needs to constraint that the degrees of freedom in both forward and

[7] Kiasu-BC [20] is a tweakable block cipher, the only difference with AES-128 is XORing a 64-bit tweak value to the first two rows of the state after each AddRoundKey.

backward only source from the key states, relax the degrees of matching such that it is not included in the objective but simply be non-zero [8], and constraint that the plaintext or ciphertext contains only `Red` and `Gray` cells or only `Blue` and `Gray` cells. Besides, the objective can be set to maximize the number of `Gray` cells in the plaintext or ciphertext, which can optimize data complexity. Applying to SKINNY-n-3n, improvements in terms of time complexity (see Fig. 23) or data complexity (see Fig. 24) upon the attack in [10] on 23-round reduced version were obtained. In Sect. F, visualizations of representative attack configurations are presented. The complexity of the attacks are summarized in Table 1.

Acknowledgements

We thank anonymous reviewers for their valuable comments. This research is partially supported by the National Natural Science Foundation of China (Grants No. 62172410, 61802400, 61732021, 61802399, 61961146004), the National Key R&D Program of China (Grants No. 2018YFA0704704, 2018YFA0704701), the Youth Innovation Promotion Association of Chinese Academy of Sciences; Nanyang Technological University in Singapore under Grant 04INS000397C230, Ministry of Education in Singapore under Grants RG91/20 and MOE2019-T2-1-060; the Gopalakrishnan – NTU Presidential Postdoctoral Fellowship 2020; the Major Program of Guangdong Basic and Applied Research (Grant No. 2019B030302008), Shandong Province Key R&D Project (Nos. 2020ZLYS09 and 2019JZZY010133).

References

1. Alliance, ZigBee. ZigBee 2007 specification. *Online: http://www.zigbee.org/*, 2007.
2. 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.
3. J.-P. Aumasson, W. Meier, and F. Mendel. Preimage attacks on 3-pass HAVAL and step-reduced MD5. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 120–135. Springer, Heidelberg, Aug. 2009.
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. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 771–804. Springer, Heidelberg, Oct. 2021.
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.

[8] In MITM key-recovery attack, the degree of matching can be efficiently increased using simultaneous matching with multiple plaintext/ciphertext pairs [11].

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. C. Bouillaguet, P. Derbez, and P.-A. Fouque. Automatic search of attacks on round-reduced AES and applications. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 169–187. Springer, Heidelberg, Aug. 2011.
9. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
10. X. Dong, J. Hua, S. Sun, Z. Li, X. Wang, and L. Hu. Meet-in-the-middle attacks revisited: Key-recovery, collision, and preimage attacks. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 278–308, Virtual Event, Aug. 2021. Springer, Heidelberg.
11. T. Fuhr and B. Minaud. Match box meet-in-the-middle attack against KATAN. In C. Cid and C. Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 61–81. Springer, Heidelberg, Mar. 2015.
12. 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.
13. H. Gilbert and T. Peyrin. Super-sbox cryptanalysis: Improved attacks for AES-like permutations. In S. Hong and T. Iwata, editors, *FSE 2010*, volume 6147 of *LNCS*, pages 365–383. Springer, Heidelberg, Feb. 2010.
14. 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.
15. 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.
16. 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.
17. J. Guo, C. Su, and W. Yap. An Improved Preimage Attack against HAVAL-3. *Inf. Process. Lett.*, 115(2):386–393, 2015.
18. A. Hosoyamada and Y. Sasaki. Finding hash collisions with quantum computers by using differential trails with smaller probability than birthday bound. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 249–279. Springer, Heidelberg, May 2020.
19. 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.
20. J. Jean, I. Nikolic, and T. Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 274–288. Springer, Heidelberg, Dec. 2014.
21. M. Lamberger, F. Mendel, C. Rechberger, V. Rijmen, and M. Schläffer. Rebound distinguishers: Results on the full Whirlpool compression function. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 126–143. Springer, Heidelberg, Dec. 2009.
22. G. Leurent. MD4 is not one-way. In K. Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 412–428. Springer, Heidelberg, Feb. 2008.

23. J. Li, T. Isobe, and K. Shibutani. Converting meet-in-the-middle preimage attack into pseudo collision attack: Application to SHA-2. In A. Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 264–286. Springer, Heidelberg, Mar. 2012.
24. 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.
25. 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.
26. 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.
27. 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.
28. Y. Sasaki and K. Aoki. Preimage attacks on step-reduced MD5. In Y. Mu, W. Susilo, and J. Seberry, editors, *ACISP 08*, volume 5107 of *LNCS*, pages 282–296. Springer, Heidelberg, July 2008.
29. 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.
30. 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.
31. 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.
32. 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.
33. 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.
34. 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.
35. 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.
36. 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.

Supplementary Material

A Concrete Inequalities modeling the Propagation rules

Inequalities for `MC-RULE` *that allow each attribute (*`Blue` *and* `Red`*) conceding to the opposite attribute propagation by consuming DoF in both directions.*

$$\begin{cases} \vec{\omega} = \max\limits_{i=0}^{\mathrm{N_{row}}-1}(\omega_i^I), \\ (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} x_i^I) - \mathrm{N_{row}} \cdot \vec{x} \geq 0, \\ (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} x_i^I) - \vec{x} \leq \mathrm{N_{row}} - 1. \\ (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} {y_i}^I) - \mathrm{N_{row}} \cdot \vec{y} \geq 0, \\ (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} {y_i}^I) - \vec{y} \leq \mathrm{N_{row}} - 1. \end{cases} \quad (15) \quad \begin{cases} (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} x_i^O) + \mathrm{N_{row}} \cdot \vec{\omega} \leq \mathrm{N_{row}}, \\ (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} (x_i^I + x_i^O)) - \mathrm{N_{iosum}} \cdot \vec{x} \geq 0, \\ (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} (x_i^I + x_i^O)) - \mathtt{Br_n} \cdot \vec{x} \leq (\mathrm{N_{iosum}} - \mathtt{Br_n}), \\ (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} y_i^O) + \mathrm{N_{row}} \cdot \vec{\omega} \leq \mathrm{N_{row}}, \\ (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} (y_i^I + y_i^O)) - \mathrm{N_{iosum}} \cdot \vec{y} \geq 0, \\ (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} (y_i^I + y_i^O)) - \mathtt{Br_n} \cdot \vec{y} \leq (\mathrm{N_{iosum}} - \mathtt{Br_n}), \end{cases} \quad (16)$$

$$\begin{cases} (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} y_i^O) - \mathrm{N_{row}} \cdot \vec{y} - c_{\vec{x}} = 0 \\ (\sum\limits_{i=0}^{\mathrm{N_{row}}-1} x_i^O) - \mathrm{N_{row}} \cdot \vec{x} - c_{\vec{y}} = 0 \end{cases} \quad (17)$$

Note that, $\mathrm{N_{iosum}} - (\sum_{i=0}^{\mathrm{N_{row}}-1}(x_i^I + x_i^O))$ is the total number of `Red` or `White` cells (*i.e.*, active cells as for `Red`) in the input and output columns. Similarly, $\mathrm{N_{iosum}} - (\sum_{i=0}^{\mathrm{N_{row}}-1}(y_i^I + y_i^O))$ is the total number of `Blue` or `White` cells (*i.e.*, active cells as for `Blue`). They are constrained by the branch number $\mathtt{Br_n}$ 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, $\omega_i^{E\oplus K}$, where ω_i^I (resp. ω_i^O) indicates whether the i-th cell in the input state I (resp. output state O) of the MixColumns is `White`; ω_i^K indicates whether the i-th cell in involved key state K is `White`; and $\omega_i^{E\oplus K}$ is defined by $\omega_i^{E\oplus K} = \mathtt{OR}(\omega_i^I, \omega_i^K)$ for $i \in \{0, 1, \ldots, n-1\}$. Besides, one need the general variable $\overrightarrow{\leftarrow}_i$ to represent the degree of matching in column i, for $i \in \{0, 1, \ldots, \mathrm{N_{col}} - 1\}$.

Suppose $\overrightarrow{End}^{\text{KMC}}$ has fewer `White` cells than $\overleftarrow{End}^{\text{K}}$ and thus using $\overrightarrow{End} \oplus \overrightarrow{End}^{\text{KMC}}$ as the key addition, then, the concrete constraints on $\overrightarrow{m}_i$ are as in Eq. (18).

$$\overrightarrow{m}_i = \max(0, \text{N}_{\texttt{row}} - \sum_{j=0}^{\text{N}_{\texttt{row}}-1} (\omega^{E\oplus K}_{i\cdot \text{N}_{\texttt{row}}+j} + \omega^{O}_{i\cdot \text{N}_{\texttt{row}}+j})) \text{ for } i \in \{0, 1, \ldots, \text{N}_{\texttt{col}} - 1\} \quad (18)$$

B Summary of 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`
$\vec{x}$	a binary variable, indicating cells in a column all are `Blue`/`Gray`
$\vec{y}$	a binary variable, indicating cells in a column all are `Red`/`Gray`
$\vec{\omega}$	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`
$c_{\vec{x}}$	the general variable for the number of consumed degrees of freedom from the forward in `MC-RULE`
$c_{\vec{y}}$	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
$\overrightarrow{\overleftarrow{m}}$	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`	executing the MixColumns operation on a single column or on multiple columns
`SB`	executing the SubBytes operation on a single cell or on multiple cells

$\mathtt{SR}$	executing the ShiftRows operation on a single cell or on multiple cells
$\#\mathbf{SB}^r$	the state before going through SubBytes ($\mathtt{SB}$) at the r-th round in the encryption data-path
$\#\mathbf{SR}^r$	the state before going through ShiftRows ($\mathtt{SR}$) at the r-th round in the encryption data-path
$\#\mathbf{MC}^r$	the state before going through MixColumns ($\mathtt{MC}$) at the r-th round in the encryption data-path
$\#\mathbf{AK}^r$	the state before going through AddRoundKey ($\mathtt{AK}$) at the r-th round in the encryption data-path
$\#\mathbf{KSB}^r$	the state before going through SubBytes ($\mathtt{SB}$) at the r-th round in the key-schedule
$\#\mathbf{KSR}^r$	the state before going through ShiftRows ($\mathtt{SR}$) at the r-th round in the key-schedule
$\#\mathbf{KMC}^r$	the state before going through MixColumns ($\mathtt{MC}$) 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
$S[\mathtt{ij}]$	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 $\mathtt{MC}$ 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 $\mathtt{MC}$ to derive the value of an additional cell according to Property 1

C Concrete Procedure of the Preimage Attack on 6-round and 8-round OTs of Grøstl-256 and Grøstl-512

C.1 Description of Grøstl

Grøstl, proposed by Gauravaram *et al.* in [12], 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).

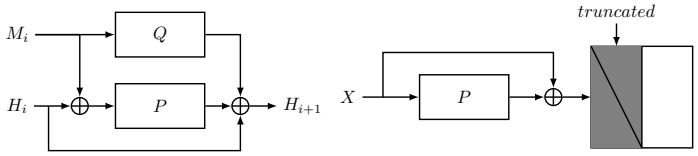


Fig. 4: Grøstl's compression function (CF) and output transformation (OT)

C.2 The preimage attack on 6-round OT of Grøstl-256.

Fig. 5 shows one of the attack configurations on 6-round OT of Grøstl-256. The following of this section describes how to use this configuration to launch concrete attack on 6-round OT of Grøstl-256.

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.*, $\#\mathbf{MC}^2_{\mathtt{col}_0}[6,7]$, $\#\mathbf{MC}^2_{\mathtt{col}_1}[5,6]$, 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.

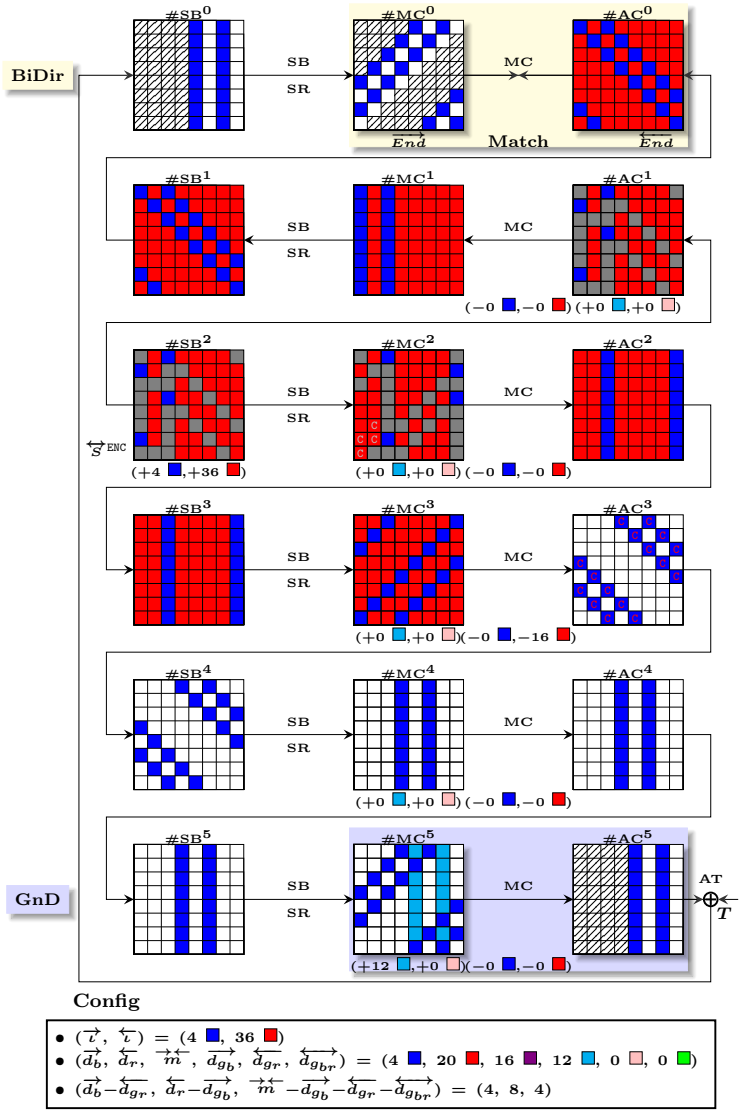


Fig. 5: An example of the MITM attack on the OT of the 6-round Grøstl-256

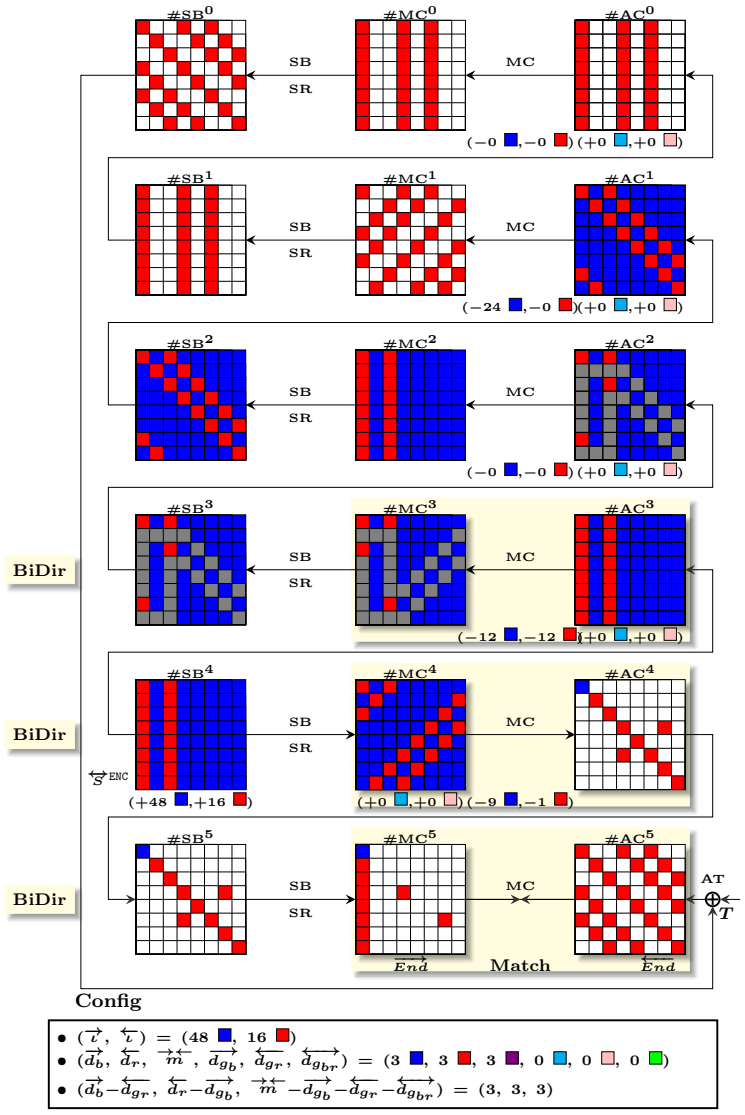


Fig. 6: An example of the MITM attack on $P(H') \oplus H'$ in the CF of the 6-round Grøstl-256 (without guessing)

The Whole Attack Procedure (refer to Fig. 5). First, fix the value of 16 `Gray` cells in $\#\mathbf{MC}^2_{\mathtt{col}_{\{0,1,3,4,5,6\}}}$ (including the four C cells $\#\mathbf{MC}^2_{\mathtt{col}_0}[6,7]$, $\#\mathbf{MC}^2_{\mathtt{col}_1}[5,6]$); 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 $\overleftarrow{\mathcal{T}}_{\mathtt{Init}}$.

1. For a new value of the 12 `Gray` cells in $\#\mathbf{MC}^2_{\mathtt{col}_{\{2,7\}}}$,
   (a) for each value $\overleftarrow{v}_i$ of `Red` neutral bytes in $\overleftarrow{\mathcal{T}}_{\mathtt{Init}}$, *i.e.*, `Red` cells in $\#\mathbf{AC}^2$,
      i. with values of `Gray` cells in $\#\mathbf{MC}^2_{\mathtt{col}_{\{2,7\}}}$ (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 $\#\mathbf{AC}^0_{\mathtt{SR}^{-1}(\mathtt{col}_{\{1,3,4,5,6,7\}})}$, derive value $\overleftarrow{v}_m$ of 16 bytes for matching through `MC` (2 bytes in each column, refer to [5, 26] 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 $\#\mathbf{AC}^5$, `XOR`ing 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), $\overrightarrow{d_{g_b}} = 12$, $\overrightarrow{m}\overleftarrow{} = 16$, $\varsigma = 2^8$, 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 done 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 done at the very beginning of the attack.

C.3 The preimage attack on 8-round OT of Grøstl-512.

Fig. 7 shows one of the attack configurations on 8-round OT of Grøstl-512. The following of this section describes how to use this configuration to launch concrete attack 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 $\#\mathbf{AC}^4[{}_3\mathtt{7}, {}_4\mathtt{7}, {}_5\mathtt{7}, {}_6\mathtt{7}, {}_8\mathtt{0}] \parallel \#\mathbf{MC}^3[{}_6\mathtt{0}, {}_6\mathtt{1}, {}_6\mathtt{7}, {}_7\mathtt{0}, {}_7\mathtt{7}, {}_8\mathtt{6}, {}_8\mathtt{7}, {}_9\mathtt{5}, {}_9\mathtt{6}, {}_9\mathtt{7}, {}_{10}\mathtt{4}, {}_{10}\mathtt{5}, {}_{10}\mathtt{6}, {}_{10}\mathtt{7}]$ [9] 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, 26]);
      iv. store $\overrightarrow{v}_i$ into a look-up table $\overrightarrow{\mathcal{T}}$, with $\overrightarrow{v}_m$ as the index.

[9] 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 $S_{\mathtt{col}_i}[j]$.

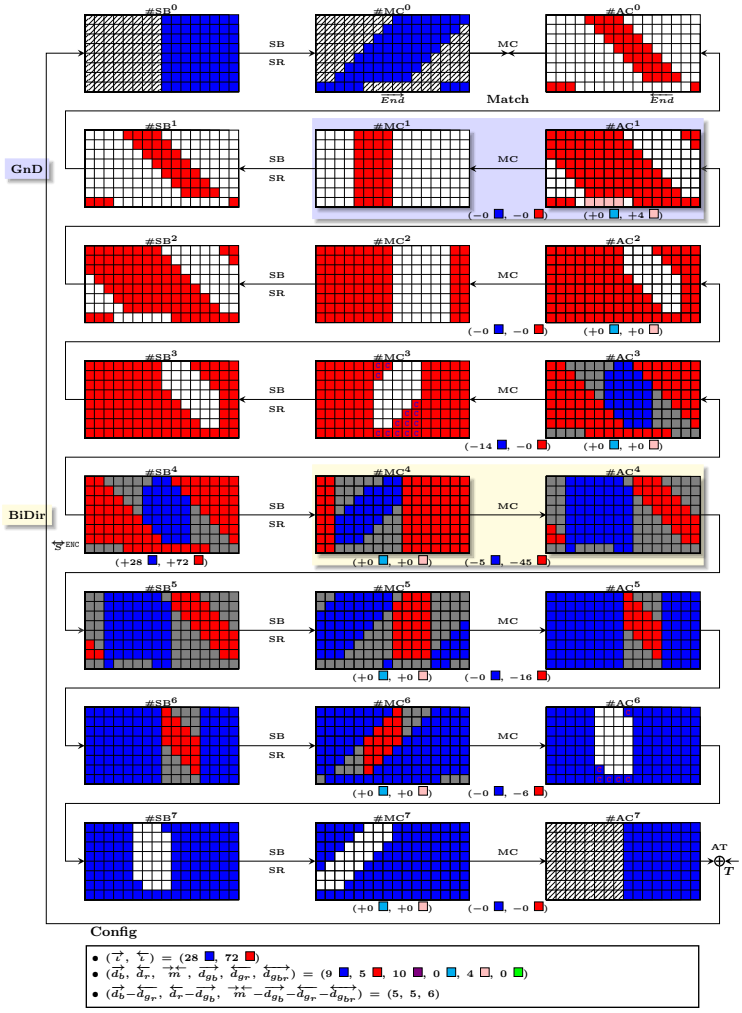


Fig. 7: An example of the MITM attack on the OT of the 8-round Grøstl-512

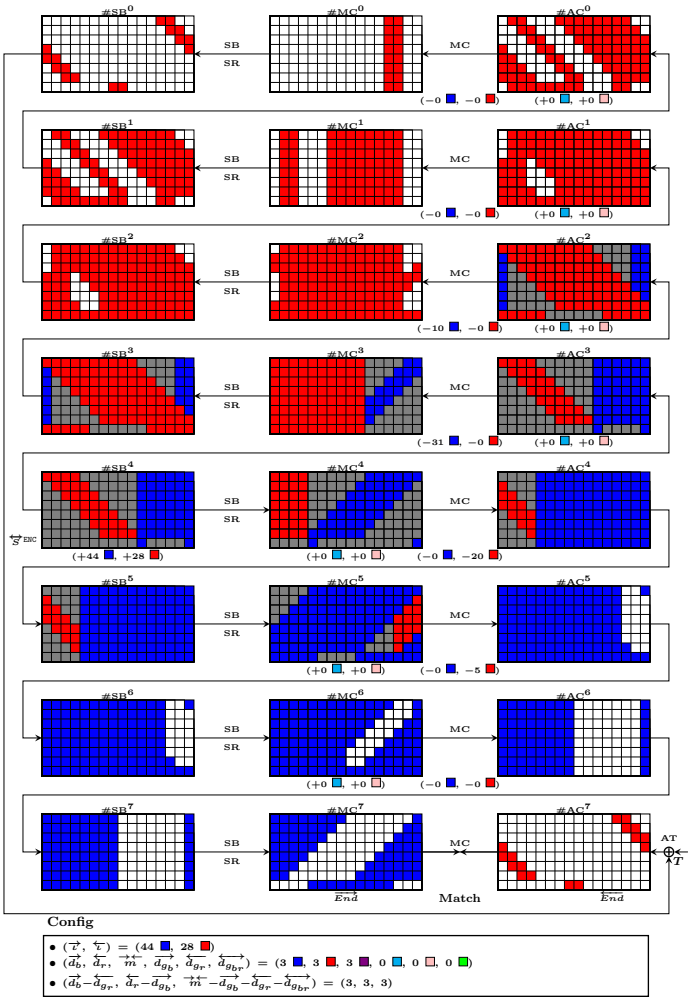


Fig. 8: An example of the MITM attack on $P(H') \oplus H'$ in the CF of the 8-round Grøstl-512 (without guessing)

(d) use the values of $\#\mathbf{AC}^6[{}_5\mathtt{6}, {}_5\mathtt{7}, {}_6\mathtt{7}, {}_7\mathtt{7}, {}_8\mathtt{0}, {}_8\mathtt{7}] \parallel \#\mathbf{MC}^5[{}_8\mathtt{0}, {}_8\mathtt{7}, {}_9\mathtt{7}, {}_{10}\mathtt{7}, {}_{11}\mathtt{7}]$ 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 $\#\mathbf{AC}^1$;

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

A. compute backward to $\#\mathbf{AC}^0$; derive the value $\overleftarrow{v}_m$ of 10 bytes from $\#\mathbf{AC}^0_{\mathtt{col}_{\{7,8,9,10\}}}$ 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 $\#\mathbf{MC}^4$, compute backward to $\#\mathbf{AC}^1$; 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 $\#\mathbf{AC}^0$, denote the value by $\overleftarrow{v}$;
- with values of `Red` and `Gray` cells in $\#\mathbf{MC}^4$, compute forward (`XOR`ing the given target at the feed-forward point) to $\#\mathbf{MC}^0$, 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 $\#\mathbf{MC}^0$ 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), $\overrightarrow{d_b} = 9$, $\overleftarrow{d_r} = 5$, $\overleftarrow{d_{g_r}} = 4$, $\overrightarrow{m}\overleftarrow{} = 10$, $\varsigma = 2^8$, and $\overleftarrow{d_{g_r}} = \overleftrightarrow{d_{g_{br}}} = 0$. Accordingly, the computational complexity of this attack on 8-round Grøstl-512 is $\varsigma^{64} \cdot \max(\varsigma^{-(9-4)}, \varsigma^{-(5)}, \varsigma^{-(10-4)}) = 2^{59 \cdot 8} = 2^{472}$.

D Compute Initial values of Neutral Bytes in the Attacks

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

Algorithm 1 Compute initial values of forward neutral bytes (in `Blue`) in the attack on 7-round Whirlpool (refer to in Fig. 3)

procedure ComputeForwardNeutralBytes
  $\overrightarrow{\mathcal{T}_{\texttt{Init}}} \leftarrow \emptyset$
  for all possible values of $\#\mathbf{AK}^3[1,2]$ **do** ▷ 2^{16}
    $\#\mathbf{MC}^3[0,1,2,3] \xleftarrow{\texttt{MC}^{-1}} \#\mathbf{AK}^3[0,1,2,3]$ ▷ $\#\mathbf{AK}^3[0,3]$ are fixed
    $\#\mathbf{SB}^3[0,5,10,15] \xleftarrow{\texttt{SB}^{-1}} \#\mathbf{MC}^3[0,1,2,3]$
    for all possible values of $\#\mathbf{SB}^3[8]$ **do** ▷ 2^8
      $\#\mathbf{SB}^3[9] \xleftarrow{\texttt{MC}_{4671_5}} (\#\mathbf{SB}^3[8,10,11], \#\mathbf{MC}^2_{\texttt{C}}[9])$ ▷ according to Property 1 [a]
      $\#\mathbf{MC}^3[8] \xleftarrow{\texttt{SB}} \#\mathbf{SB}^3[8]$
      $\#\mathbf{MC}^3[5] \xleftarrow{\texttt{SB}} \#\mathbf{SB}^3[9]$
      $L_{\#\mathbf{MC}^3_{8,5}} \xleftarrow{push} (\#\mathbf{MC}^3[8,5])$ ▷ the final size $|L_{\#\mathbf{MC}^3_{8,5}}| = 2^8$
    end for
    for all possible values of $\#\mathbf{SB}^3[13]$ **do** ▷ 2^8
      $\#\mathbf{SB}^3[14] \xleftarrow{\texttt{MC}_{4570_6}} (\#\mathbf{SB}^3[12,13,15], \#\mathbf{MC}^2_{\texttt{C}}[12])$ ▷ according to Property 1
      $\#\mathbf{MC}^3[9] \xleftarrow{\texttt{SB}} \#\mathbf{SB}^3[13]$
      $\#\mathbf{MC}^3[6] \xleftarrow{\texttt{SB}} \#\mathbf{SB}^3[14]$
      $L_{\#\mathbf{MC}^3_{9,6}} \xleftarrow{push} (\#\mathbf{MC}^3[9,6])$ ▷ the final size $|L_{\#\mathbf{MC}^3_{9,6}}| = 2^8$
    end for
    for all possible values of $\#\mathbf{SB}^3[4]$ **do** ▷ 2^8
      $\#\mathbf{SB}^3[7] \xleftarrow{\texttt{MC}_{4562_7}} (\#\mathbf{SB}^3[4,5,6], \#\mathbf{MC}^2_{\texttt{C}}[6])$ ▷ according to Property 1
      $\#\mathbf{MC}^3[4] \xleftarrow{\texttt{SB}} \#\mathbf{SB}^3[4]$
      $\#\mathbf{MC}^3[11] \xleftarrow{\texttt{SB}} \#\mathbf{SB}^3[7]$
      $L_{\#\mathbf{MC}^3_{4,11}} \xleftarrow{push} (\#\mathbf{MC}^3[4,11])$ ▷ the final size $|L_{\#\mathbf{MC}^3_{4,11}}| = 2^8$
    end for
    for all possible values of $\#\mathbf{SB}^3[2]$ **do** ▷ 2^8
      $\#\mathbf{SB}^3[3] \xleftarrow{\texttt{MC}_{4563_7}} (\#\mathbf{SB}^3[0,1,2], \#\mathbf{MC}^2_{\texttt{C}}[3])$ ▷ according to Property 1
      $\#\mathbf{MC}^3[10] \xleftarrow{\texttt{SB}} \#\mathbf{SB}^3[2]$
      $\#\mathbf{MC}^3[7] \xleftarrow{\texttt{SB}} \#\mathbf{SB}^3[3]$
      $L_{\#\mathbf{MC}^3_{10,7}} \xleftarrow{push} (\#\mathbf{MC}^3[10,7])$ ▷ the final size $|L_{\#\mathbf{MC}^3_{10,7}}| = 2^8$
    end for
    for all $(\#\mathbf{MC}^3[8,5]) \in L_{\#\mathbf{MC}^3_{8,5}}$ **do** ▷ 2^8
      for all $(\#\mathbf{MC}^3[9,6]) \in L_{\#\mathbf{MC}^3_{9,6}}$ **do** ▷ 2^8
        $a_{\#\mathbf{AK}^3[5]} \xleftarrow{\texttt{MC}} (0 \parallel \#\mathbf{MC}^3[5] \parallel \#\mathbf{MC}^3[6] \parallel 0)[1] \oplus \#\mathbf{AK}^3[5]$
        $b_{\#\mathbf{AK}^3[10]} \xleftarrow{\texttt{MC}} (\#\mathbf{MC}^3[8] \parallel \#\mathbf{MC}^3[9] \parallel 0 \parallel 0)[2] \oplus \#\mathbf{AK}^3[10]$
        $T_{\#\mathbf{MC}^3_{8,5,9,6}}[a_{\#\mathbf{AK}^3[5]} \parallel b_{\#\mathbf{AK}^3[10]}] \leftarrow (\#\mathbf{MC}^3[8,5,9,6])$ ▷ expected to have one element in each entry of $T_{\#\mathbf{MC}^3_{8,5,9,6}}$
      end for
    end for
    for all $(\#\mathbf{MC}^3[4,11]) \in L_{\#\mathbf{MC}^3_{4,11}}$ **do** ▷ 2^8
      for all $(\#\mathbf{MC}^3[10,7]) \in L_{\#\mathbf{MC}^3_{10,7}}$ **do** ▷ 2^8
        $a'_{\#\mathbf{AK}^3[5]} \xleftarrow{\texttt{MC}} (\#\mathbf{MC}^3[4] \parallel 0 \parallel 0 \parallel \#\mathbf{MC}^3[7])[1]$
        $b'_{\#\mathbf{AK}^3[10]} \xleftarrow{\texttt{MC}} (0 \parallel 0 \parallel \#\mathbf{MC}^3[10] \parallel \#\mathbf{MC}^3[11])[2]$
        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
          $\overrightarrow{\mathcal{T}_{\texttt{Init}}} \xleftarrow{push} \#\mathbf{MC}^3[0,1,2,3,4,5,6,7,8,9,10,11]$ ▷ the final size $|\overrightarrow{\mathcal{T}_{\texttt{Init}}}| = 2^{32}$
        end for
      end for
    end for
  end for
  return $\overrightarrow{\mathcal{T}_{\texttt{Init}}}$
end procedure

[a] $\texttt{MC}_{abcd_e}$ 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
  $\overleftarrow{\mathcal{T}_{\texttt{Init}}} \leftarrow \emptyset$, $\#\text{MC}^3_{\text{SR}(\texttt{col}_{\{2,7\}})} \leftarrow \vec{0}$ ▷ refer to Fig. 9 for cell index
  for all possible values of $\#\text{MC}^2[{}_0\texttt{1}, {}_0\texttt{3}, {}_0\texttt{4}, {}_0\texttt{5}, {}_1\texttt{0}, {}_1\texttt{2}, {}_1\texttt{3}, {}_1\texttt{4}]$ **do** ▷ 2^{64}
    $\#\text{AC}^2_{\texttt{col}_{\{0,1\}}} \overset{\text{MC}}{\leftarrow} \#\text{MC}^2_{\texttt{col}_{\{0,1\}}}$, $\#\text{MC}^3_{\text{SR}(\texttt{col}_{\{0,1\}})} \overset{\text{SR}\circ\text{SB}\circ\text{AC}}{\leftarrow} \#\text{AC}^2_{\texttt{col}_{\{0,1\}}}$
    for all possible values of $\#\text{MC}^3[{}_0\texttt{3}, {}_0\texttt{4}]$, $\#\text{MC}^3[{}_1\texttt{2}, {}_1\texttt{3}]$ **do** ▷ 2^{32}
      $\#\text{MC}^3[{}_0\texttt{5}, {}_0\texttt{6}] \overset{\text{MC}^*}{\leftarrow} (\#\text{MC}^3[{}_0\texttt{0}, {}_0\texttt{1}, {}_0\texttt{2}, {}_0\texttt{3}, {}_0\texttt{4}, {}_0\texttt{7}], \#\text{AC}^3_{\text{C}}[{}_0\texttt{3}, {}_0\texttt{5}])$ ▷ Property 1
      $\#\text{MC}^3[{}_1\texttt{4}, {}_1\texttt{5}] \overset{\text{MC}^*}{\leftarrow} (\#\text{MC}^3[{}_1\texttt{0}, {}_1\texttt{1}, {}_1\texttt{2}, {}_1\texttt{3}, {}_1\texttt{6}, {}_1\texttt{7}], \#\text{AC}^3_{\text{C}}[{}_1\texttt{4}, {}_1\texttt{6}])$ ▷ Property 1
      $\#\text{AC}^2[{}_3\texttt{3}, {}_4\texttt{4}, {}_5\texttt{5}, {}_6\texttt{6}, {}_3\texttt{2}, {}_4\texttt{3}, {}_5\texttt{4}, {}_6\texttt{5}] \overset{(\text{SB}\circ\text{AC})^{-1}}{\leftarrow} \#\text{MC}^2[{}_0\texttt{3}, {}_0\texttt{4}, {}_0\texttt{5}, {}_0\texttt{6}, {}_1\texttt{2}, {}_1\texttt{3}, {}_1\texttt{4}, {}_1\texttt{5}]$
      $L_{01} \overset{push}{\leftarrow} (\#\text{AC}^2[{}_3\texttt{3}, {}_4\texttt{4}, {}_5\texttt{5}, {}_6\texttt{6}, {}_3\texttt{2}, {}_4\texttt{3}, {}_5\texttt{4}, {}_6\texttt{5}])$ ▷ the final size $|L_{01}| = 2^{32}$
    end for
    for all possible values of $\#\text{MC}^3[{}_2\texttt{1}, {}_2\texttt{2}]$, $\#\text{MC}^3[{}_3\texttt{0}, {}_3\texttt{1}]$ **do** ▷ 2^{32}
      $\#\text{MC}^3[{}_2\texttt{3}, {}_2\texttt{4}] \overset{\text{MC}^*}{\leftarrow} (\#\text{MC}^3[{}_2\texttt{0}, {}_2\texttt{1}, {}_2\texttt{2}, {}_2\texttt{5}, {}_2\texttt{6}, {}_2\texttt{7}], \#\text{AC}^3_{\text{C}}[{}_2\texttt{5}, {}_2\texttt{7}])$ ▷ Property 1
      $\#\text{MC}^3[{}_3\texttt{2}, {}_3\texttt{3}] \overset{\text{MC}^*}{\leftarrow} (\#\text{MC}^3[{}_3\texttt{0}, {}_3\texttt{1}, {}_3\texttt{4}, {}_3\texttt{5}, {}_3\texttt{6}, {}_3\texttt{7}], \#\text{AC}^3_{\text{C}}[{}_3\texttt{0}, {}_3\texttt{6}])$ ▷ Property 1
      $\#\text{AC}^2[{}_3\texttt{1}, {}_4\texttt{2}, {}_5\texttt{3}, {}_6\texttt{4}, {}_3\texttt{0}, {}_4\texttt{1}, {}_5\texttt{2}, {}_6\texttt{3}] \overset{(\text{SB}\circ\text{AC})^{-1}}{\leftarrow} \#\text{MC}^2[{}_2\texttt{1}, {}_2\texttt{2}, {}_2\texttt{3}, {}_2\texttt{4}, {}_3\texttt{0}, {}_3\texttt{1}, {}_3\texttt{2}, {}_3\texttt{3}]$
      $L_{23} \overset{push}{\leftarrow} (\#\text{AC}^2[{}_3\texttt{1}, {}_4\texttt{2}, {}_5\texttt{3}, {}_6\texttt{4}, {}_3\texttt{0}, {}_4\texttt{1}, {}_5\texttt{2}, {}_6\texttt{3}])$ ▷ the final size $|L_{23}| = 2^{32}$
    end for
    for all possible values of $\#\text{MC}^3[{}_4\texttt{0}, {}_4\texttt{1}]$, $\#\text{MC}^3[{}_5\texttt{0}, {}_5\texttt{1}]$ **do** ▷ 2^{32}
      $\#\text{MC}^3[{}_4\texttt{2}, {}_4\texttt{7}] \overset{\text{MC}^*}{\leftarrow} (\#\text{MC}^3[{}_4\texttt{0}, {}_4\texttt{1}, {}_4\texttt{3}, {}_4\texttt{4}, {}_4\texttt{5}, {}_4\texttt{6}], \#\text{AC}^3_{\text{C}}[{}_4\texttt{1}, {}_4\texttt{7}])$ ▷ Property 1
      $\#\text{MC}^3[{}_5\texttt{6}, {}_5\texttt{7}] \overset{\text{MC}^*}{\leftarrow} (\#\text{MC}^3[{}_5\texttt{0}, {}_5\texttt{1}, {}_5\texttt{2}, {}_5\texttt{3}, {}_5\texttt{4}, {}_5\texttt{5}], \#\text{AC}^3_{\text{C}}[{}_5\texttt{0}, {}_5\texttt{2}])$ ▷ Property 1
      $\#\text{AC}^2[{}_4\texttt{0}, {}_5\texttt{1}, {}_6\texttt{2}, {}_3\texttt{7}, {}_5\texttt{0}, {}_6\texttt{2}, {}_3\texttt{6}, {}_4\texttt{7}] \overset{(\text{SB}\circ\text{AC})^{-1}}{\leftarrow} \#\text{MC}^2[{}_4\texttt{0}, {}_4\texttt{1}, {}_4\texttt{2}, {}_4\texttt{7}, {}_5\texttt{0}, {}_5\texttt{1}, {}_5\texttt{6}, {}_5\texttt{7}]$
      $L_{45} \overset{push}{\leftarrow} (\#\text{AC}^2[{}_4\texttt{0}, {}_5\texttt{1}, {}_6\texttt{2}, {}_3\texttt{7}, {}_5\texttt{0}, {}_6\texttt{2}, {}_3\texttt{6}, {}_4\texttt{7}])$ ▷ the final size $|L_{45}| = 2^{32}$
    end for
    for all possible values of $\#\text{MC}^3[{}_6\texttt{0}, {}_6\texttt{5}]$, $\#\text{MC}^3[{}_7\texttt{4}, {}_7\texttt{5}]$ **do** ▷ 2^{32}
      $\#\text{MC}^3[{}_6\texttt{6}, {}_6\texttt{7}] \overset{\text{MC}^*}{\leftarrow} (\#\text{MC}^3[{}_6\texttt{0}, {}_6\texttt{1}, {}_6\texttt{2}, {}_6\texttt{3}, {}_6\texttt{4}, {}_6\texttt{5}], \#\text{AC}^3_{\text{C}}[{}_6\texttt{1}, {}_6\texttt{3}])$ ▷ Property 1
      $\#\text{MC}^3[{}_7\texttt{6}, {}_7\texttt{7}] \overset{\text{MC}^*}{\leftarrow} (\#\text{MC}^3[{}_7\texttt{0}, {}_7\texttt{1}, {}_7\texttt{2}, {}_7\texttt{3}, {}_7\texttt{4}, {}_7\texttt{5}], \#\text{AC}^3_{\text{C}}[{}_7\texttt{2}, {}_7\texttt{4}])$ ▷ Property 1
      $\#\text{AC}^2[{}_6\texttt{0}, {}_3\texttt{5}, {}_4\texttt{6}, {}_5\texttt{7}, {}_3\texttt{4}, {}_4\texttt{5}, {}_5\texttt{6}, {}_6\texttt{7}] \overset{(\text{SB}\circ\text{AC})^{-1}}{\leftarrow} \#\text{MC}^2[{}_6\texttt{0}, {}_6\texttt{5}, {}_6\texttt{6}, {}_6\texttt{7}, {}_7\texttt{4}, {}_7\texttt{5}, {}_7\texttt{6}, {}_7\texttt{7}]$
      $L_{67} \overset{push}{\leftarrow} (\#\text{AC}^2[{}_6\texttt{0}, {}_3\texttt{5}, {}_4\texttt{6}, {}_5\texttt{7}, {}_3\texttt{4}, {}_4\texttt{5}, {}_5\texttt{6}, {}_6\texttt{7}])$ ▷ the final size $|L_{67}| = 2^{32}$
    end for
    for all $(\#\text{AC}^2[{}_3\texttt{3}, {}_4\texttt{4}, {}_5\texttt{5}, {}_6\texttt{6}, {}_3\texttt{2}, {}_4\texttt{3}, {}_5\texttt{4}, {}_6\texttt{5}]) \in L_{01}$ **do** ▷ 2^{32}
      for all $(\#\text{AC}^2[{}_3\texttt{1}, {}_4\texttt{2}, {}_5\texttt{3}, {}_6\texttt{4}, {}_3\texttt{0}, {}_4\texttt{1}, {}_5\texttt{2}, {}_6\texttt{3}]) \in L_{23}$ **do** ▷ 2^{32}
        $a \overset{\text{MC}}{\leftarrow} (\#\text{AC}^2[{}_3\texttt{0}, {}_3\texttt{1}, {}_3\texttt{2}, {}_3\texttt{3}] \parallel 0 \parallel 0 \parallel 0 \parallel 0)[5, 7] \oplus \#\text{MC}^2[{}_3\texttt{5}, {}_3\texttt{7}]$
        $b \overset{\text{MC}}{\leftarrow} (0 \parallel \#\text{AC}^2[{}_4\texttt{1}, {}_4\texttt{2}, {}_4\texttt{3}, {}_4\texttt{4}] \parallel 0 \parallel 0 \parallel 0)[4, 6] \oplus \#\text{MC}^2[{}_4\texttt{4}, {}_4\texttt{6}]$
        $c \overset{\text{MC}}{\leftarrow} (0 \parallel 0 \parallel \#\text{AC}^2[{}_5\texttt{2}, {}_5\texttt{3}, {}_5\texttt{4}, {}_5\texttt{5}] \parallel 0 \parallel 0)[3, 5] \oplus \#\text{MC}^2[{}_5\texttt{3}, {}_5\texttt{5}]$
        $d \overset{\text{MC}}{\leftarrow} (0 \parallel 0 \parallel 0 \parallel \#\text{AC}^2[{}_6\texttt{3}, {}_6\texttt{4}, {}_6\texttt{5}, {}_6\texttt{6}] \parallel 0)[2, 4] \oplus \#\text{MC}^2[{}_6\texttt{2}, {}_6\texttt{4}]$
        $T[a \parallel b \parallel c \parallel d] \leftarrow (\#\text{AC}^2[{}_3\texttt{3}, {}_4\texttt{4}, {}_5\texttt{5}, {}_6\texttt{6}, {}_3\texttt{2}, {}_4\texttt{3}, {}_5\texttt{4}, {}_6\texttt{5}, {}_3\texttt{1}, {}_4\texttt{2}, {}_5\texttt{3}, {}_6\texttt{4}, {}_3\texttt{0}, {}_4\texttt{1}, {}_5\texttt{2}, {}_6\texttt{3}])$
      end for
    end for
    for all $(\#\text{AC}^2[{}_4\texttt{0}, {}_5\texttt{1}, {}_6\texttt{2}, {}_3\texttt{7}, {}_5\texttt{0}, {}_6\texttt{2}, {}_3\texttt{6}, {}_4\texttt{7}]) \in L_{45}$ **do** ▷ 2^{32}
      for all $(\#\text{AC}^2[{}_6\texttt{0}, {}_3\texttt{5}, {}_4\texttt{6}, {}_5\texttt{7}, {}_3\texttt{4}, {}_4\texttt{5}, {}_5\texttt{6}, {}_6\texttt{7}]) \in L_{67}$ **do** ▷ 2^{32}
        $a' \overset{\text{MC}}{\leftarrow} (0 \parallel 0 \parallel 0 \parallel 0 \parallel \#\text{AC}^2[{}_3\texttt{4}, {}_3\texttt{5}, {}_3\texttt{6}, {}_3\texttt{7}])[5, 7]$
        $b' \overset{\text{MC}}{\leftarrow} (\#\text{AC}^2[{}_4\texttt{0}] \parallel 0 \parallel 0 \parallel 0 \parallel 0 \parallel \#\text{AC}^2[{}_4\texttt{5}, {}_4\texttt{6}, {}_4\texttt{7}])[4, 6]$
        $c' \overset{\text{MC}}{\leftarrow} (\#\text{AC}^2[{}_5\texttt{0}, {}_5\texttt{1}] \parallel 0 \parallel 0 \parallel 0 \parallel 0 \parallel \#\text{AC}^2[{}_5\texttt{6}, {}_5\texttt{7}])[3, 5]$
        $d' \overset{\text{MC}}{\leftarrow} (\#\text{AC}^2[{}_6\texttt{0}, {}_6\texttt{1}, {}_6\texttt{2}] \parallel 0 \parallel 0 \parallel 0 \parallel 0 \parallel \#\text{AC}^2[{}_6\texttt{7}] \parallel 0)[2, 4]$
        for all $\#\text{AC}^2[{}_3\texttt{3}, {}_4\texttt{4}, {}_5\texttt{5}, {}_6\texttt{6}, {}_3\texttt{2}, {}_4\texttt{3}, {}_5\texttt{4}, {}_6\texttt{5}, {}_3\texttt{1}, {}_4\texttt{2}, {}_5\texttt{3}, {}_6\texttt{4}, {}_3\texttt{0}, {}_4\texttt{1}, {}_5\texttt{2}, {}_6\texttt{3}] \in T[a' \parallel b' \parallel c' \parallel d']$ **do**
          $\overleftarrow{\mathcal{T}_{\texttt{Init}}} \overset{push}{\leftarrow} \#\text{AC}^2_{\texttt{col}_{\{0,1,3,4,5,6\}}}$ ▷ 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

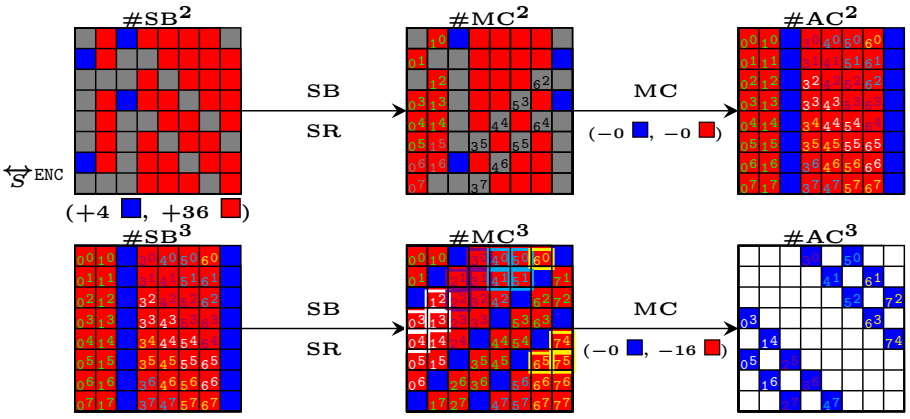


Fig. 9: Index in Algorithm 2

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

procedure ComputeForwardNeutralBytes
  $\overrightarrow{\mathcal{T}_{\texttt{Init}}} \leftarrow \emptyset$
  for all possible value $\overrightarrow{v}_i$ of the 28 `Blue` cells in $\#\text{MC}^4$ **do** ▷ $2^{2\times 28}$
    $c_0 \xleftarrow{\text{MC}} (0 \parallel 0 \parallel 0 \parallel \#\text{MC}^4_{\texttt{col}_3}[3,4,5,6] \parallel 0)[7]$
    $c_1 \xleftarrow{\text{MC}} (0 \parallel 0 \parallel \#\text{MC}^4_{\texttt{col}_4}[2,3,4,5,6] \parallel 0)[7]$
    $c_2 \xleftarrow{\text{MC}} (0 \parallel \#\text{MC}^4_{\texttt{col}_5}[1,2,3,4,5] \parallel 0 \parallel 0)[7]$
    $c_3 \xleftarrow{\text{MC}} (0 \parallel \#\text{MC}^4_{\texttt{col}_6}[1,2,3,4] \parallel 0 \parallel 0 \parallel 0)[7]$
    $c_4 \xleftarrow{\text{MC}} (\#\text{MC}^4_{\texttt{col}_8}[0,1,2] \parallel 0 \parallel 0 \parallel 0 \parallel 0 \parallel 0)[0]$
    compute backward cell-by-cell through $\texttt{SB}^{-1}$, $\texttt{SR}^{-1}$, and $\texttt{AC}^{-1}$ to get the `Blue` cells in $\#\text{AC}^3$
    $c_5 \xleftarrow{\text{MC}^{-1}} (0 \parallel \#\text{AC}^3_{\texttt{col}_6}[1,2,3,4] \parallel 0 \parallel 0 \parallel 0)[0,1,7]$
    $c_6 \xleftarrow{\text{MC}^{-1}} (\#\text{AC}^3_{\texttt{col}_7}[0,1,2,3,4,5] \parallel 0 \parallel 0)[0,7]$
    $c_7 \xleftarrow{\text{MC}^{-1}} (\#\text{AC}^3_{\texttt{col}_8}[0,1,2,3,4,5,6] \parallel 0)[6,7]$
    $c_8 \xleftarrow{\text{MC}^{-1}} (0 \parallel \#\text{AC}^3_{\texttt{col}_9}[1,2,3,4,5,6] \parallel 0)[5,6,7]$
    $c_9 \xleftarrow{\text{MC}^{-1}} (0 \parallel 0 \parallel \#\text{AC}^3_{\texttt{col}_{10}}[2,3,4,5,6] \parallel 0)[4,5,6,7]$
    $\overrightarrow{\mathcal{T}_{\texttt{Init}}}[c_0 \parallel c_1 \parallel c_2 \parallel c_3 \parallel c_4 \parallel c_5 \parallel c_6 \parallel c_7 \parallel c_8 \parallel c_9] \leftarrow \overrightarrow{v}_i$
  end for
  return $\overleftarrow{\mathcal{T}_{\texttt{Init}}}$
  ▷ Note that due to the linearity of MixColumns, values of `Gray` cells in involved columns of $\#\text{MC}^4$ and $\#\text{AC}^3$ 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 `XOR`ed together to the involved `Red` cells.
end procedure

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

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

E Visualization of More Attack Configurations

E.1 Visualization of configurations facilitated to launch (pseudo-) preimage attacks

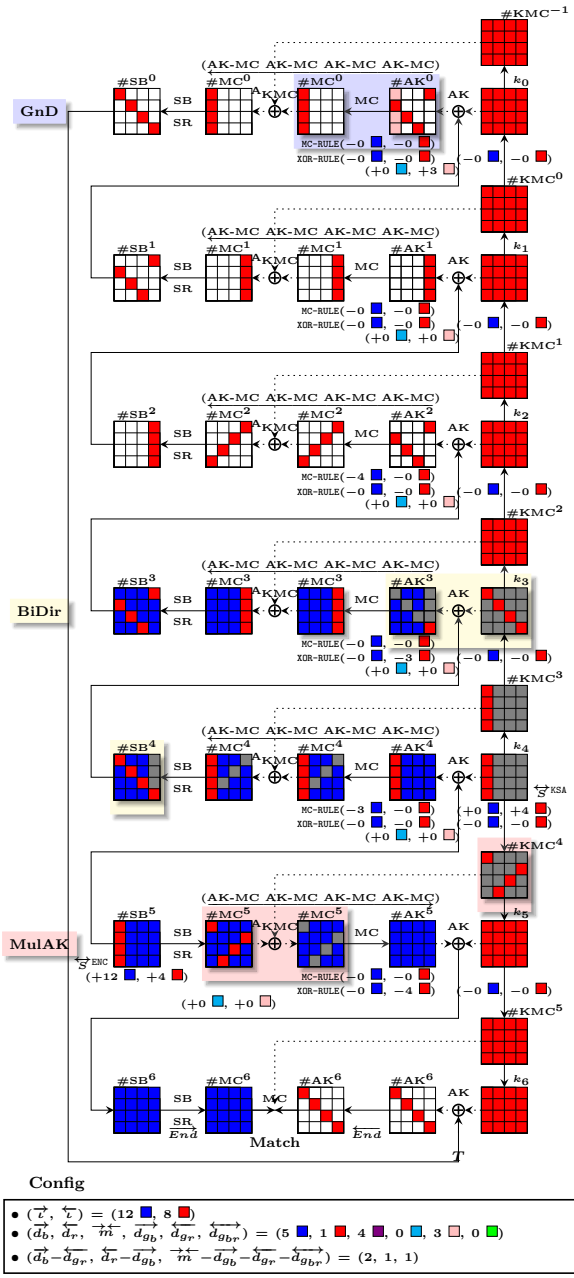


Fig. 10: The states in real attacks for superposition states in Fig. 3

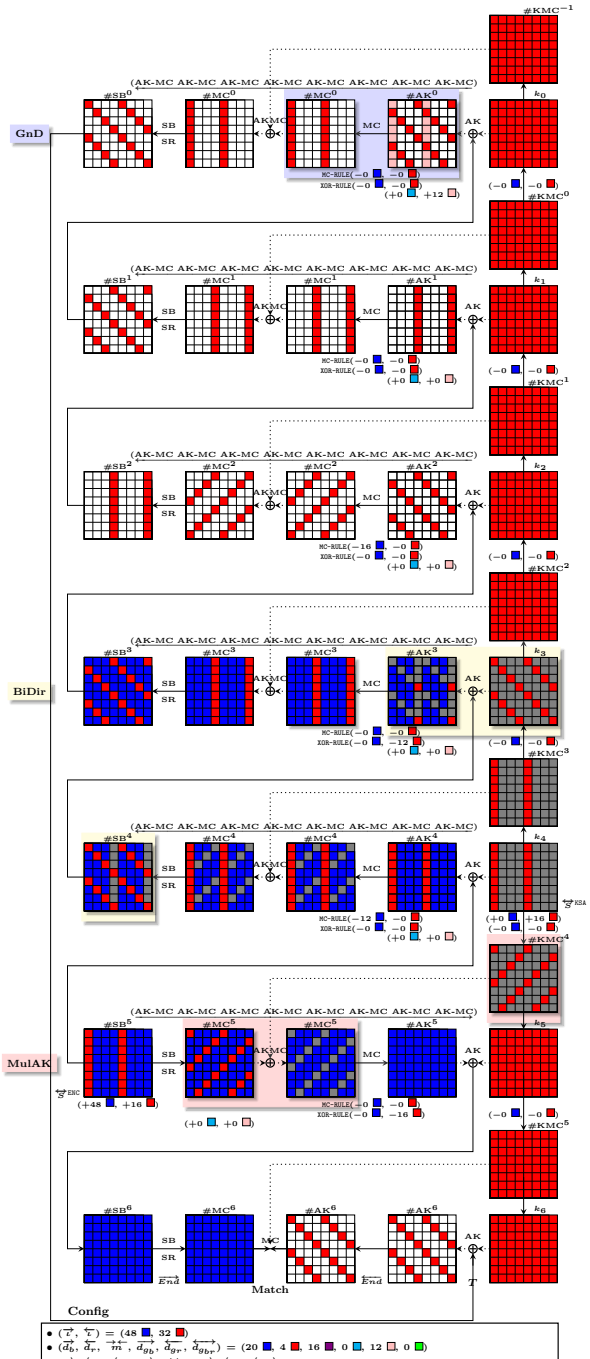


Fig. 11: The MITM attack on full-state 7-round Whirlpool (2×2 of 4×4) corresponding to the small-state configuration in Fig. 3

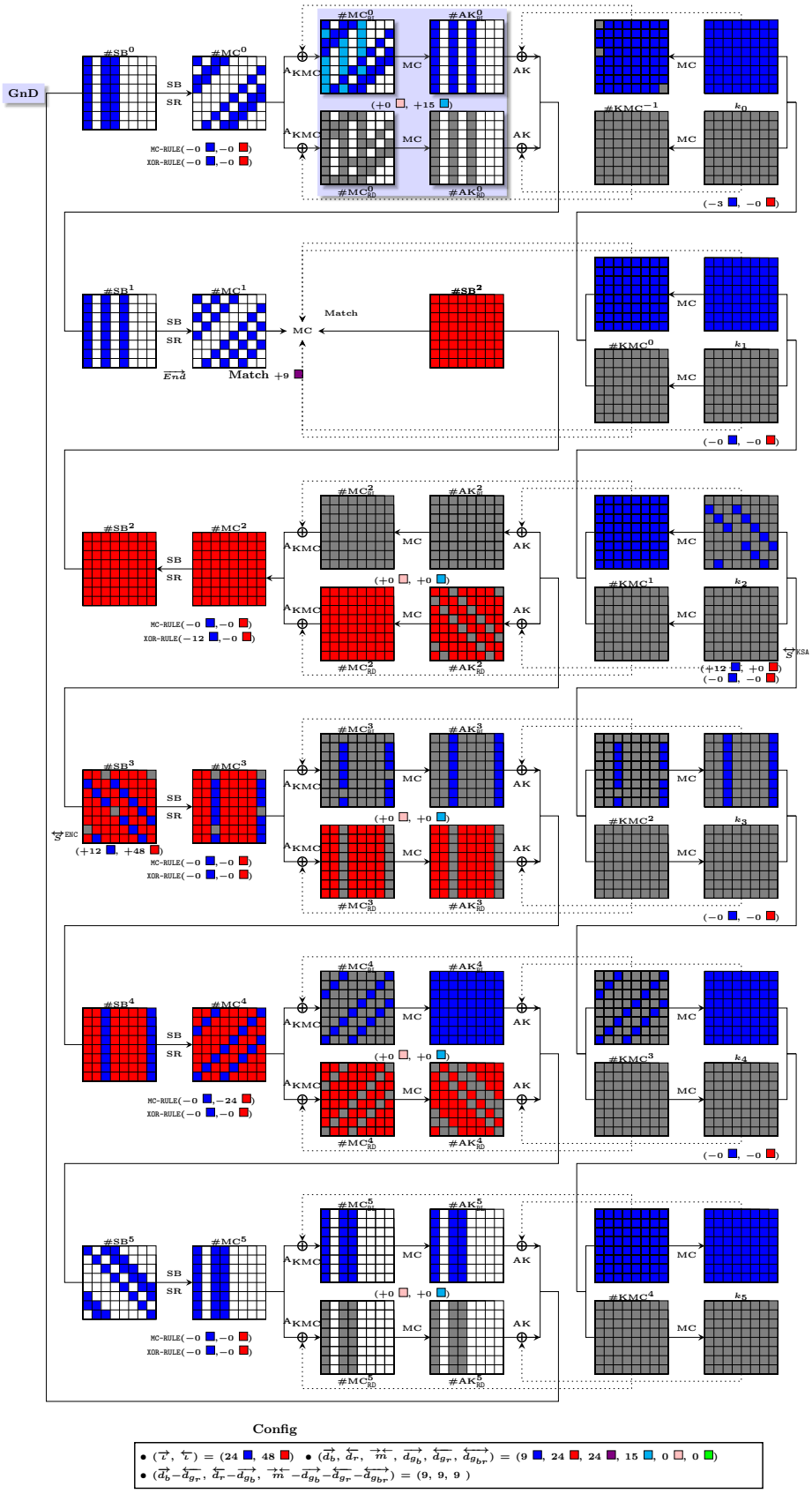
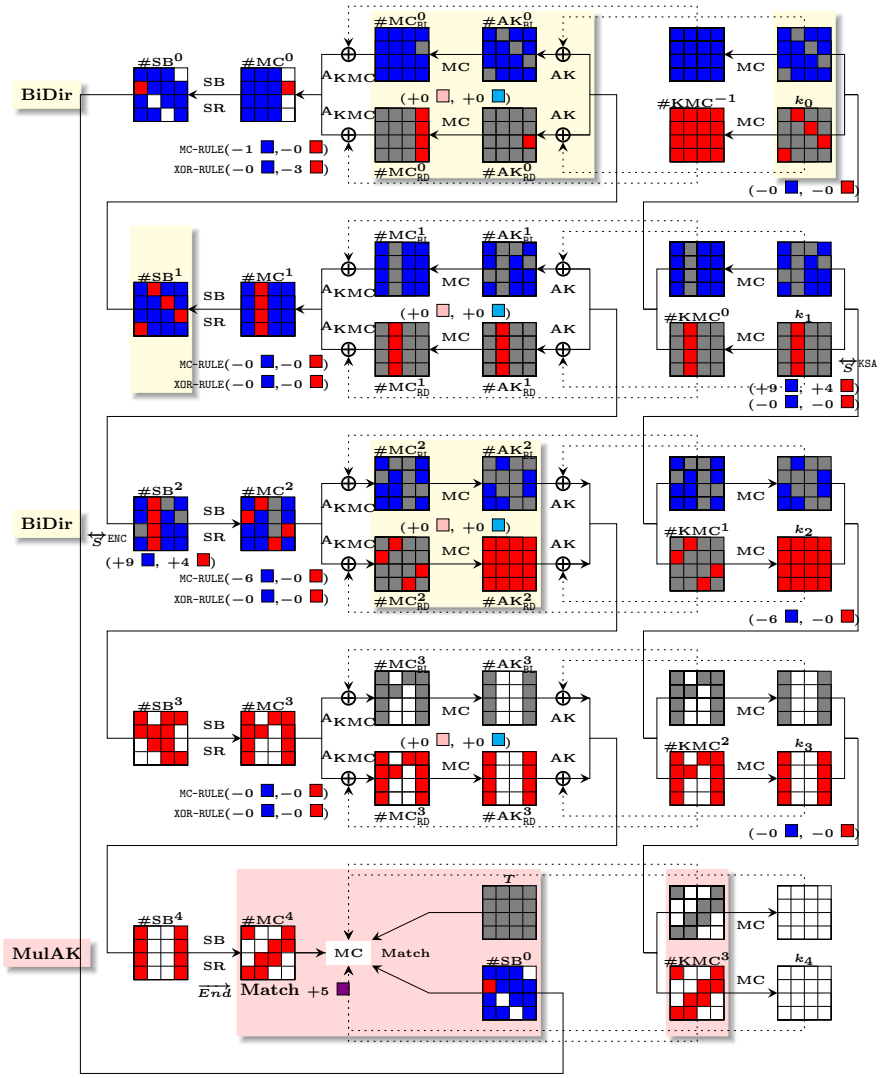


Fig. 12: An MITM attack on 6-round Whirlpool with asymmetric pattern



$\bullet\ (\overrightarrow{v},\ \overleftarrow{v}) = (18,\ 8)$ $\bullet\ (\overrightarrow{d_b},\ \overleftarrow{d_r},\ \overrightarrow{m},\ \overrightarrow{d_{g_b}},\ \overleftarrow{d_{g_r}},\ \overleftrightarrow{d_{g_{br}}}) = (5,\ 5,\ 5,\ 0,\ 0,\ 0)$

$\bullet\ (\overrightarrow{d_b}-\overleftarrow{d_{g_r}},\ \overleftarrow{d_r}-\overrightarrow{d_{g_b}},\ \overrightarrow{m}-\overrightarrow{d_{g_b}}-\overleftarrow{d_{g_r}}-\overleftrightarrow{d_{g_{br}}}) = (5,\ 5,\ 5\ )$

This implies an attack on the full version (8×8) with configuration

$((\overrightarrow{d_b} - \overleftarrow{d_{g_r}},\ \overleftarrow{d_r} - \overrightarrow{d_{g_b}},\ \overrightarrow{m} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) = (20,\ 20,\ 20))$

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

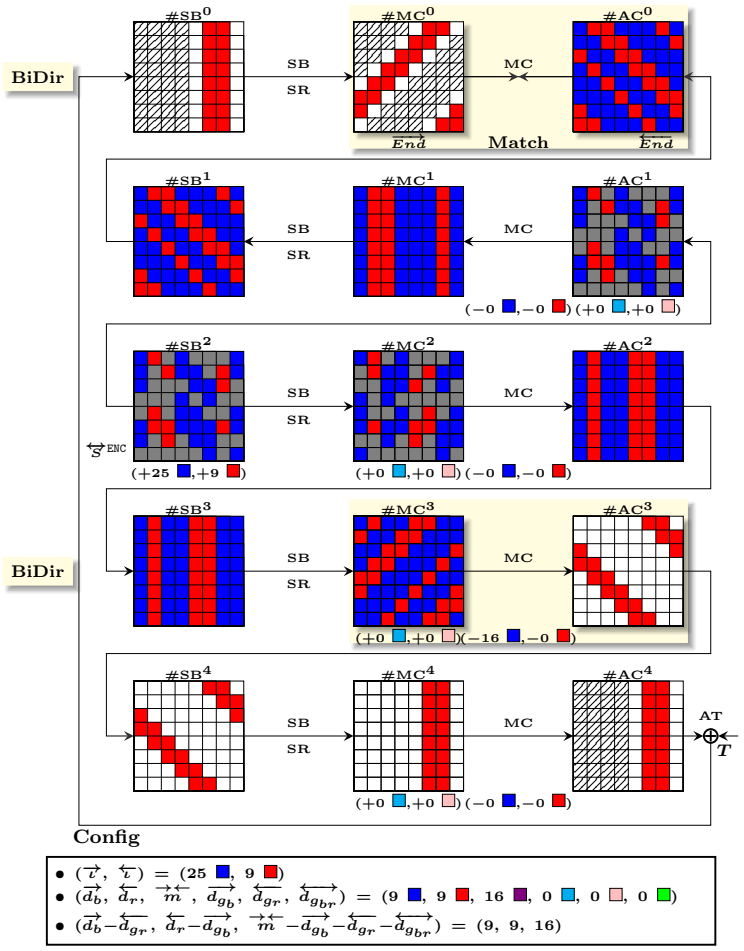


Fig. 14: An example of the MITM attack on the OT of the 5-round Grøstl-256

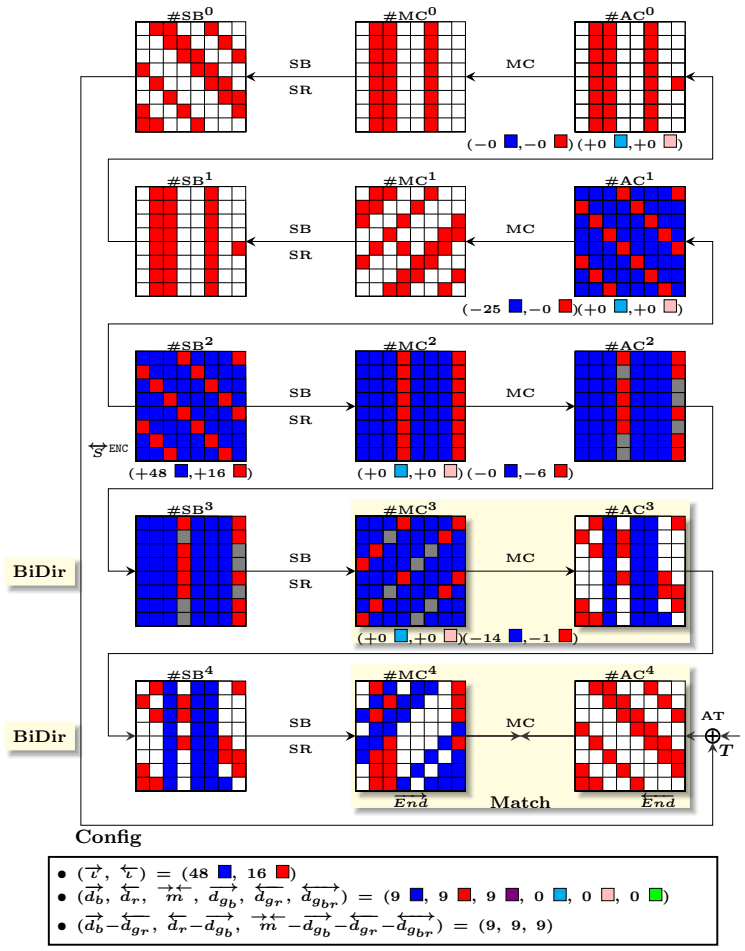


Fig. 15: An example of the MITM attack on $P(H') \oplus H'$ in the CF of the 5-round Grøstl-256 (without guessing)

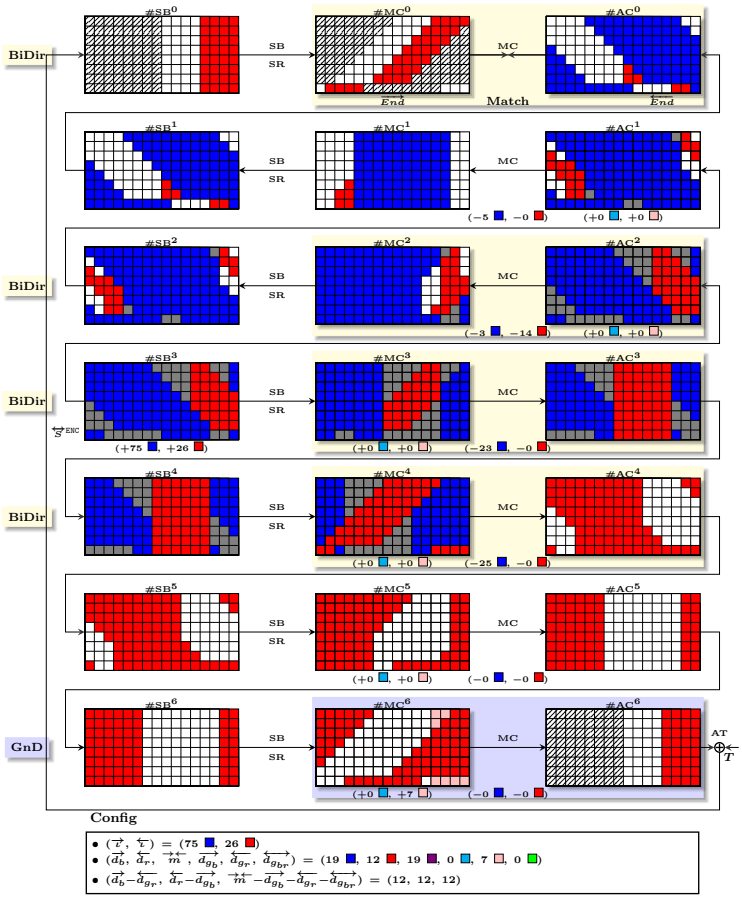


Fig. 16: An example of the MITM attack on the OT of the 7-round Grøstl-512

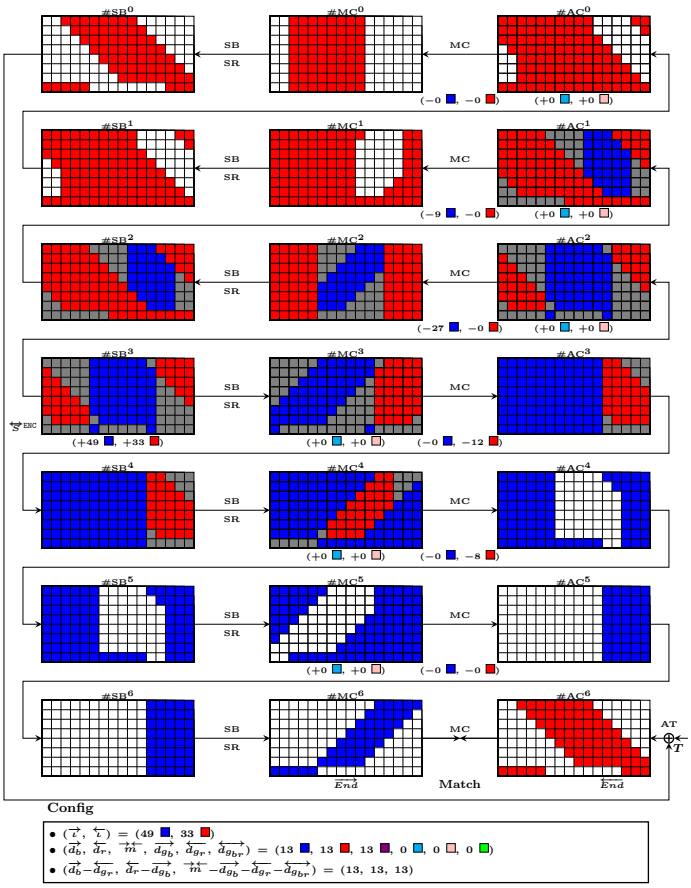
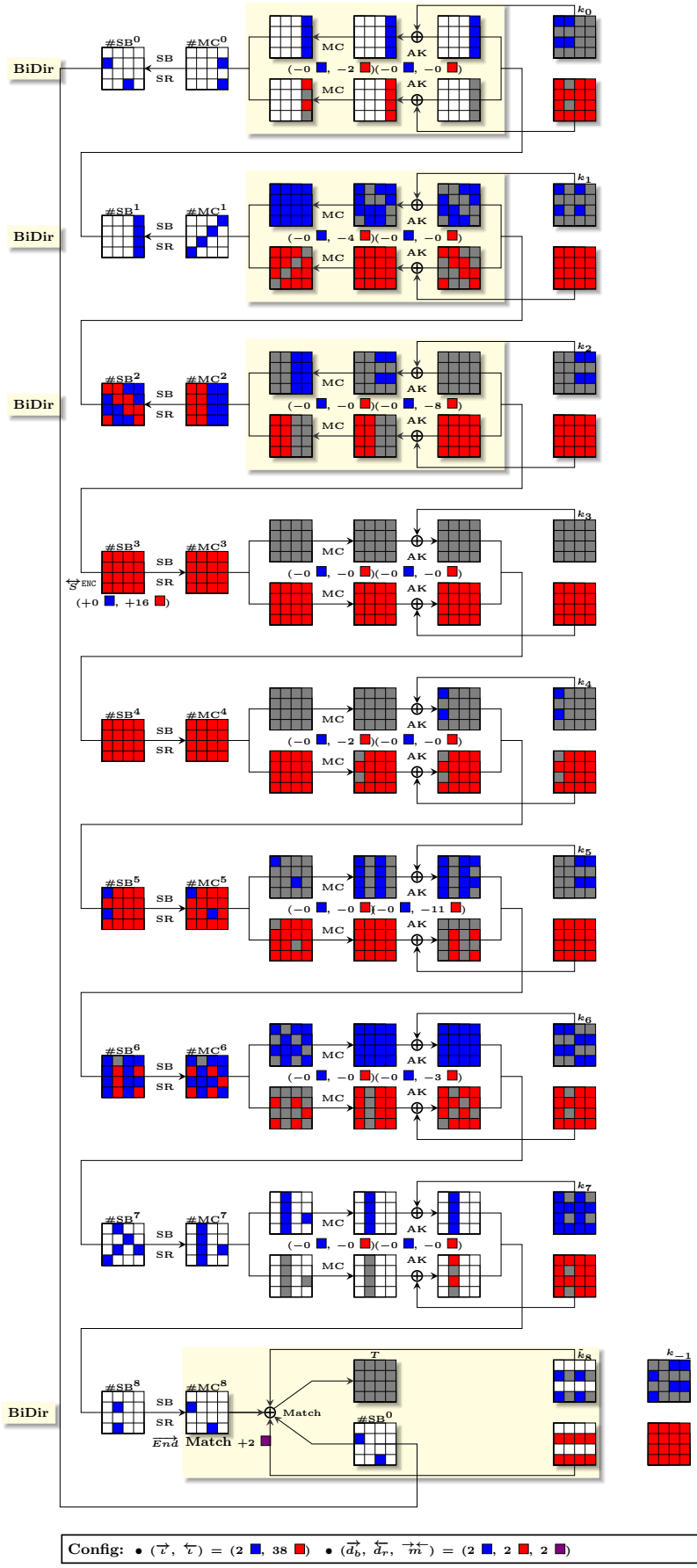
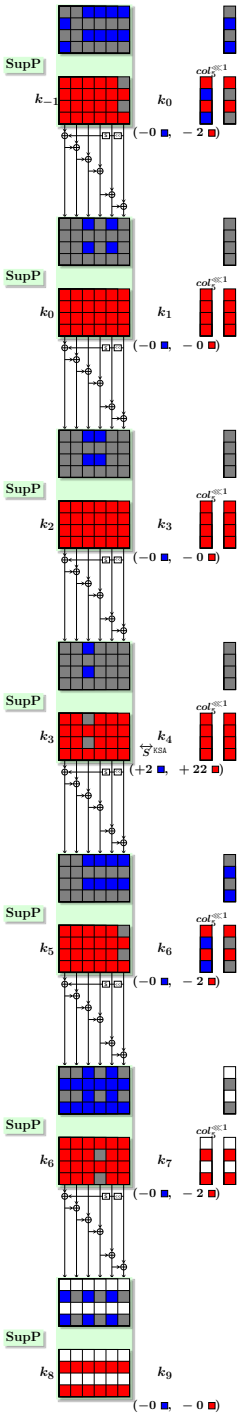


Fig. 17: An example of the MITM attack on $P(H') \oplus H'$ in the CF of the 7-round Grøstl-512 (without guessing)

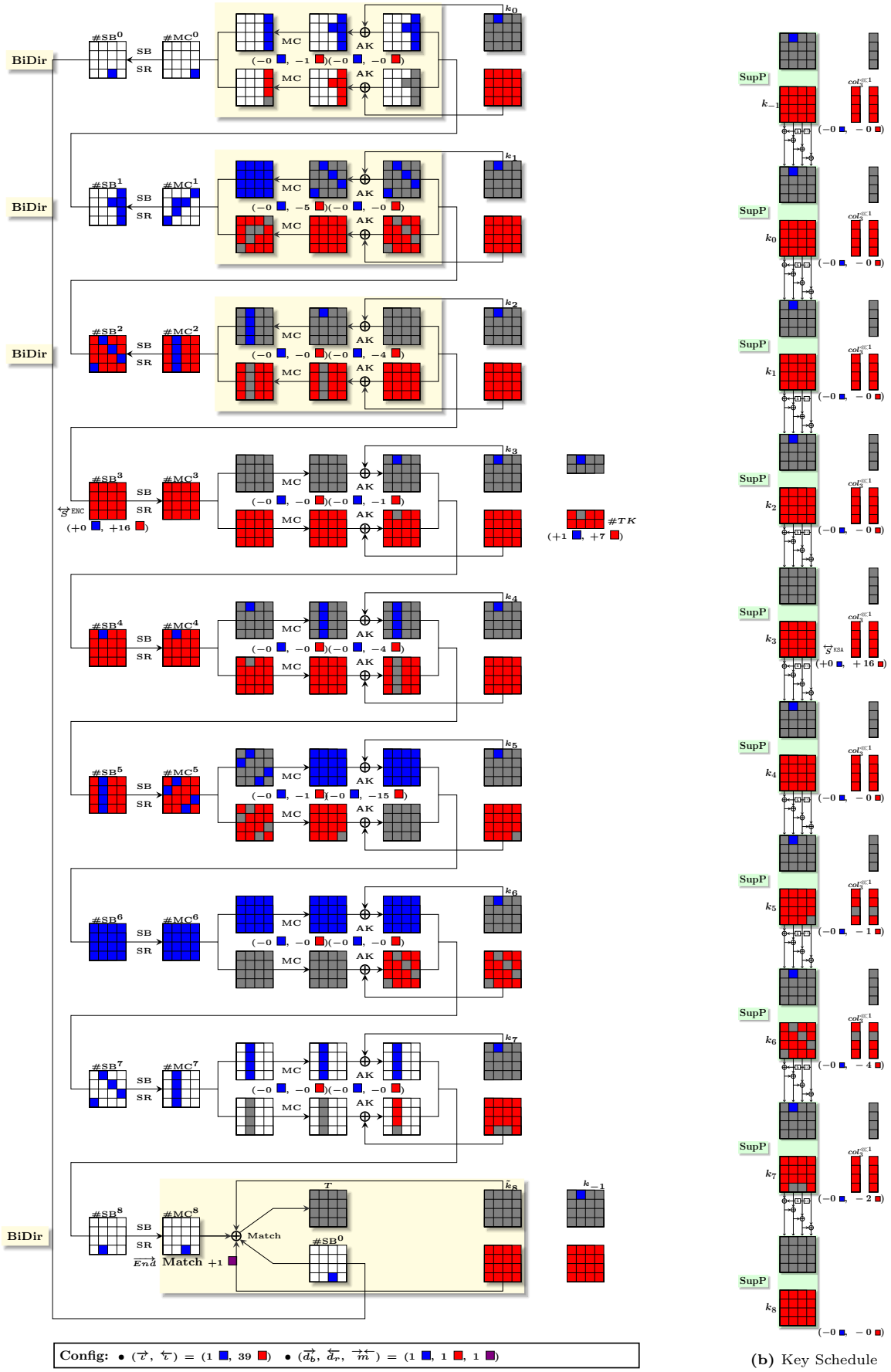


(a) Encryption States



(b) Key Schedule

Fig. 18: A configuration of MITM attack against 9-Round AES-192 hashing mode



(a) Encryption States **(b)** Key Schedule

Subkeys are already XORed with the tweak. Besides, the displayed subkey $\tilde{k_8}$ is already XORed with the whitening key k_{-1}.

Fig. 19: A configuration of MITM attack on 9-Round Kiasu-BC hashing mode

E.2 Visualization of configurations facilitated to converting to collision attacks and key-recovery attacks

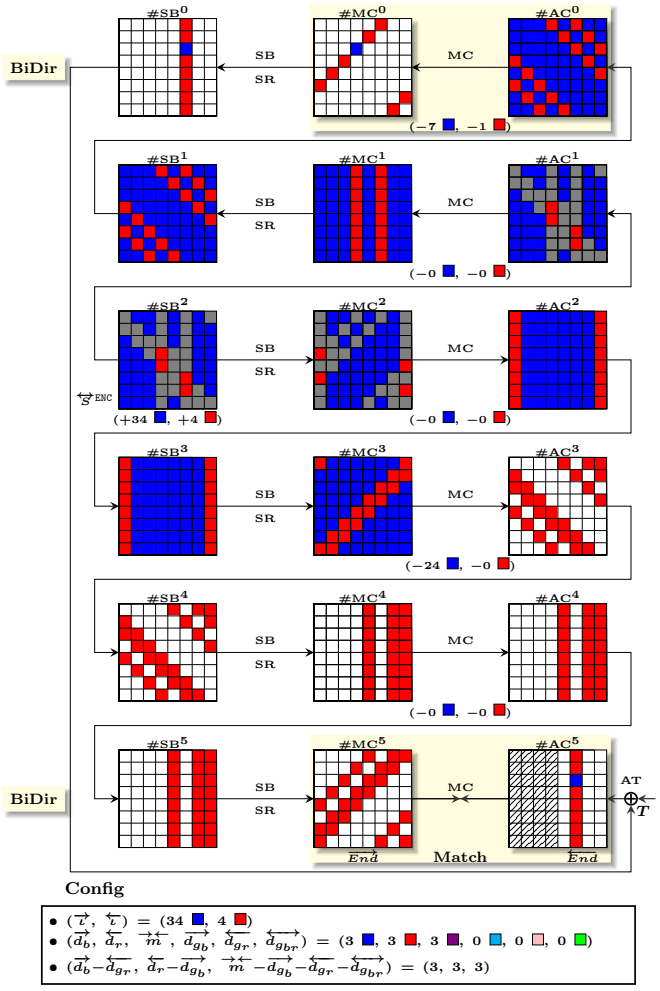


Fig. 20: An example of configuration for collision attack on the OT of 6-round Grøstl-256

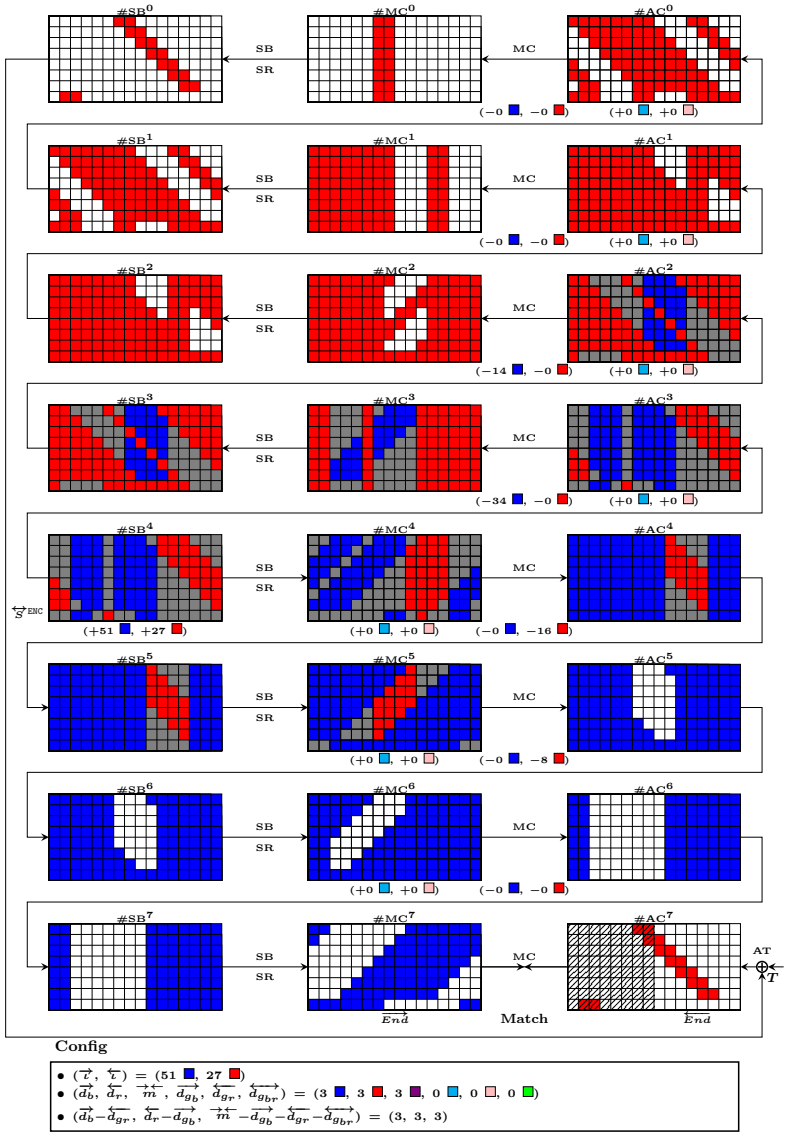
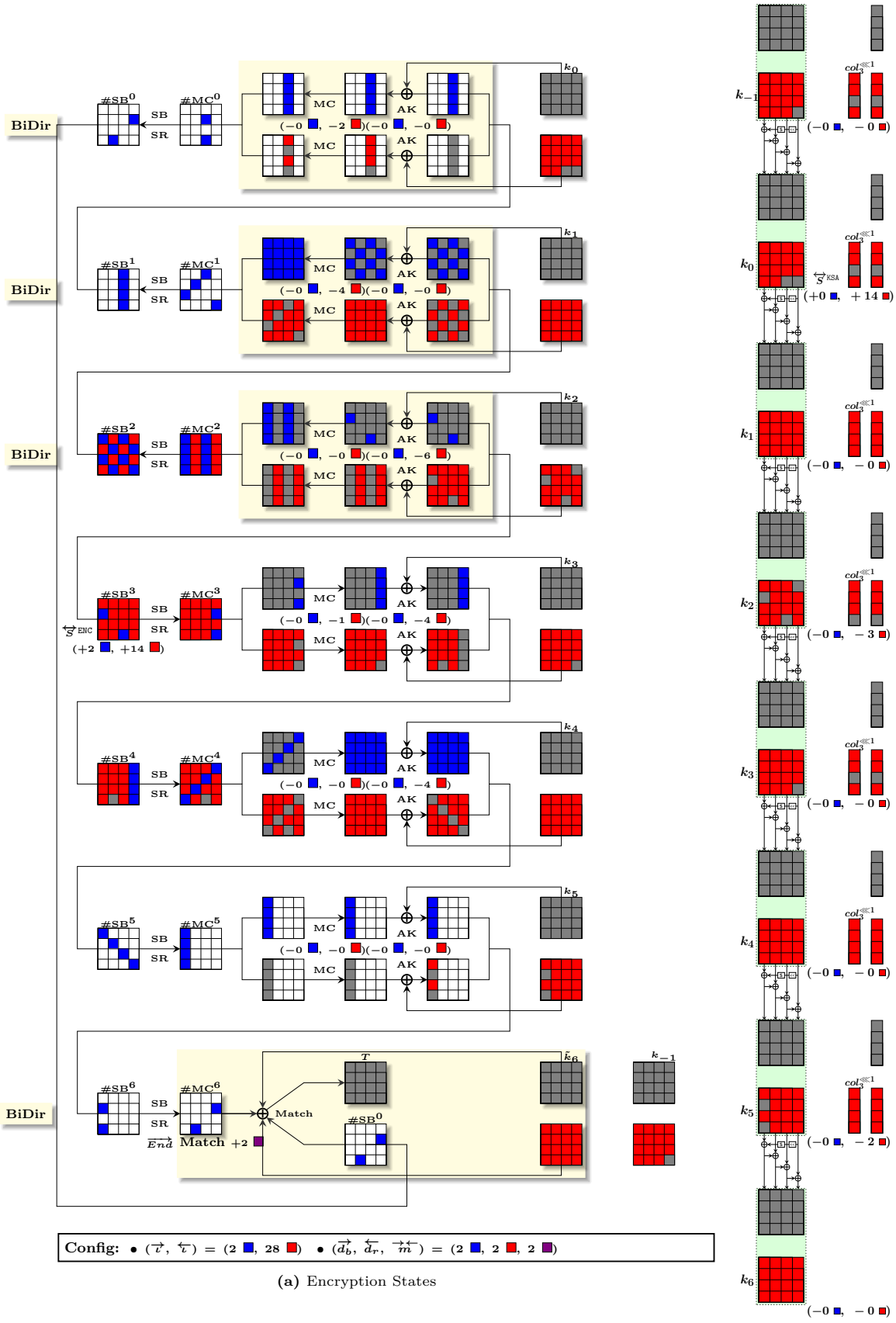


Fig. 21: An example of configuration for collision attack on the OT of 8-round Grøstl-512



(a) Encryption States

(b) Key Schedule

⋆ In this and in other attack configurations on AES-hashing, the displayed last subkey is already XORed with the whitening key k_{-1}. Besides, the key-schedule is unrolled such that variables in each subkey are directly related to the variables in $\overleftrightarrow{S}^{\text{KSA}}$. Therefore, an XOR of two `Red` (or `Blue`) cells can turn to `Gray` according to the algebraic normal form while consuming no degrees of freedom.

⋆ For converting this attack configuration to a valid collision attack, it is necessary to pre-compute values of neutral bytes (in `Red`) for many fixed bytes (in `Gray`) at once with a total complexity of no more than 2^{56}. For this, aided by the Automatic-Tool in [8], we found a guess-and-determine meet-in-the-middle procedure that generates 2^{56} values of neutral bytes for 2^{40} values of `Gray` bytes (*i.e.*, the five bytes $\#\mathtt{MC}^1_{\mathtt{Red}}[2,5,8,15]$ and $\#\mathtt{SB}^5_{\mathtt{Red}}[0]$) with a complexity of 2^{56}. The system of equations and the automatically obtained procedure can be found via https://github.com/MITM-AES-like.

Fig. 22: A configuration of MITM attack against 7-Round AES-128 hashing mode (facilitated to converting to a collision attack)

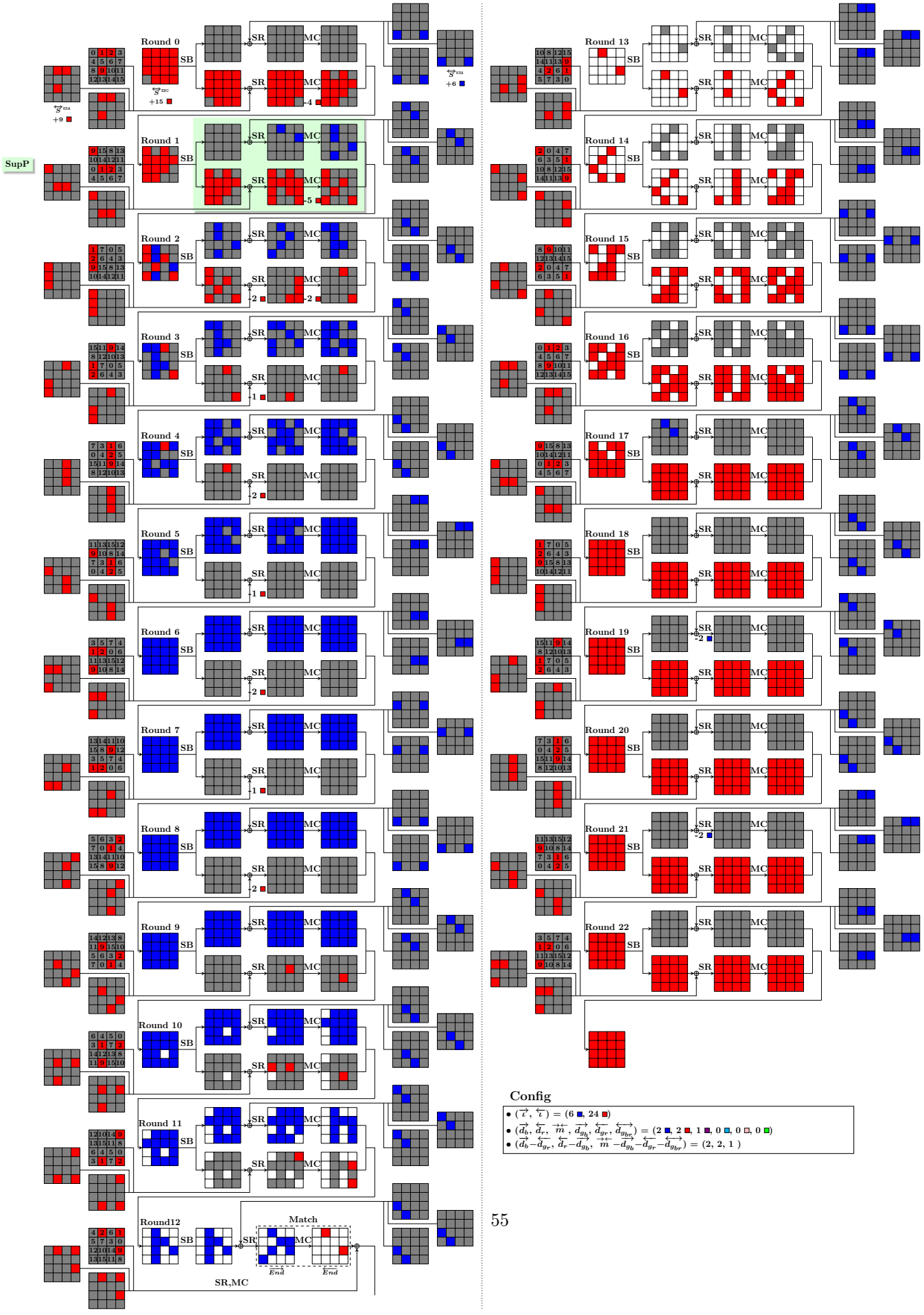


Fig. 23: A key-recovery attack on 23-round SKINNY-n-3n, optimized for time complexity

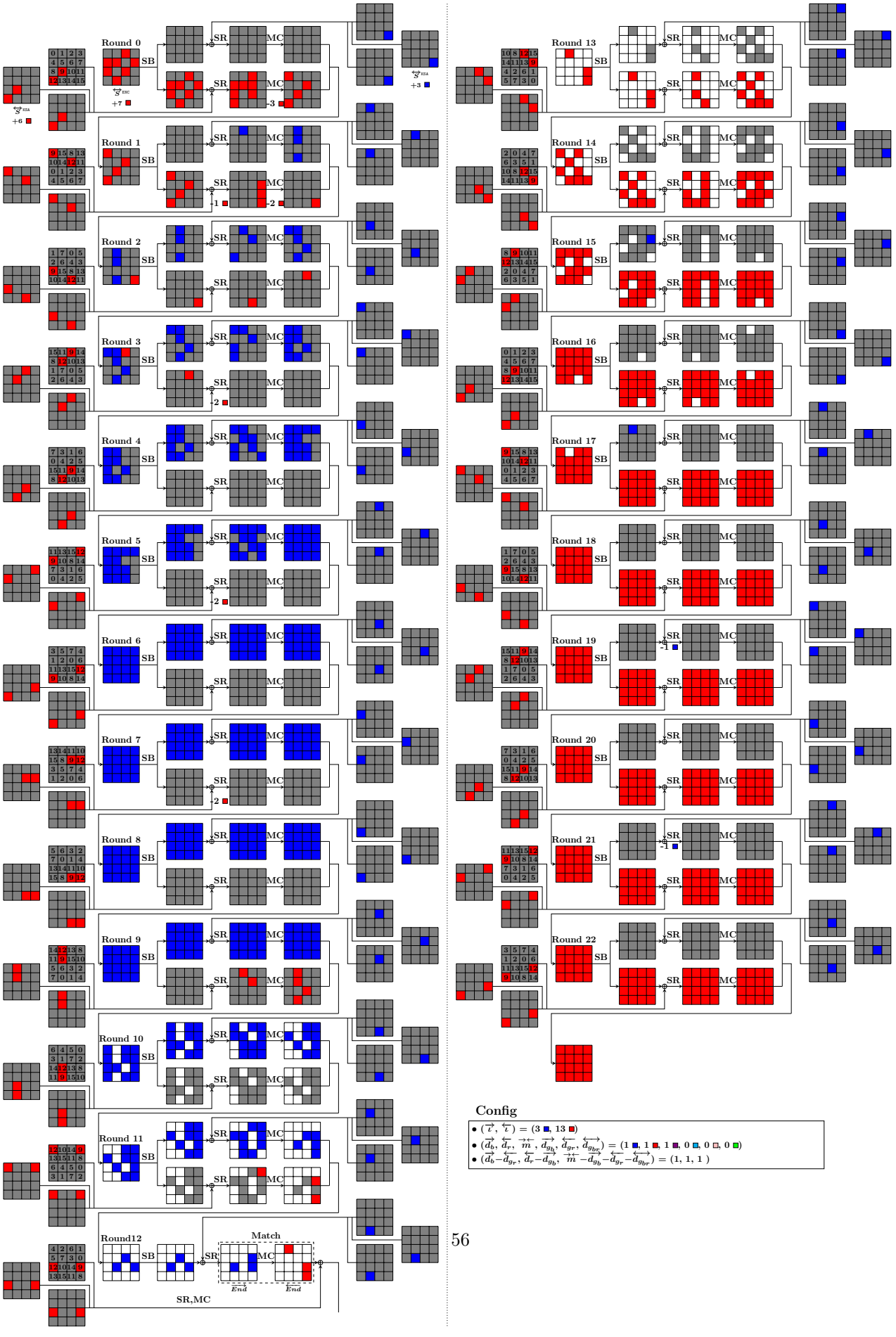
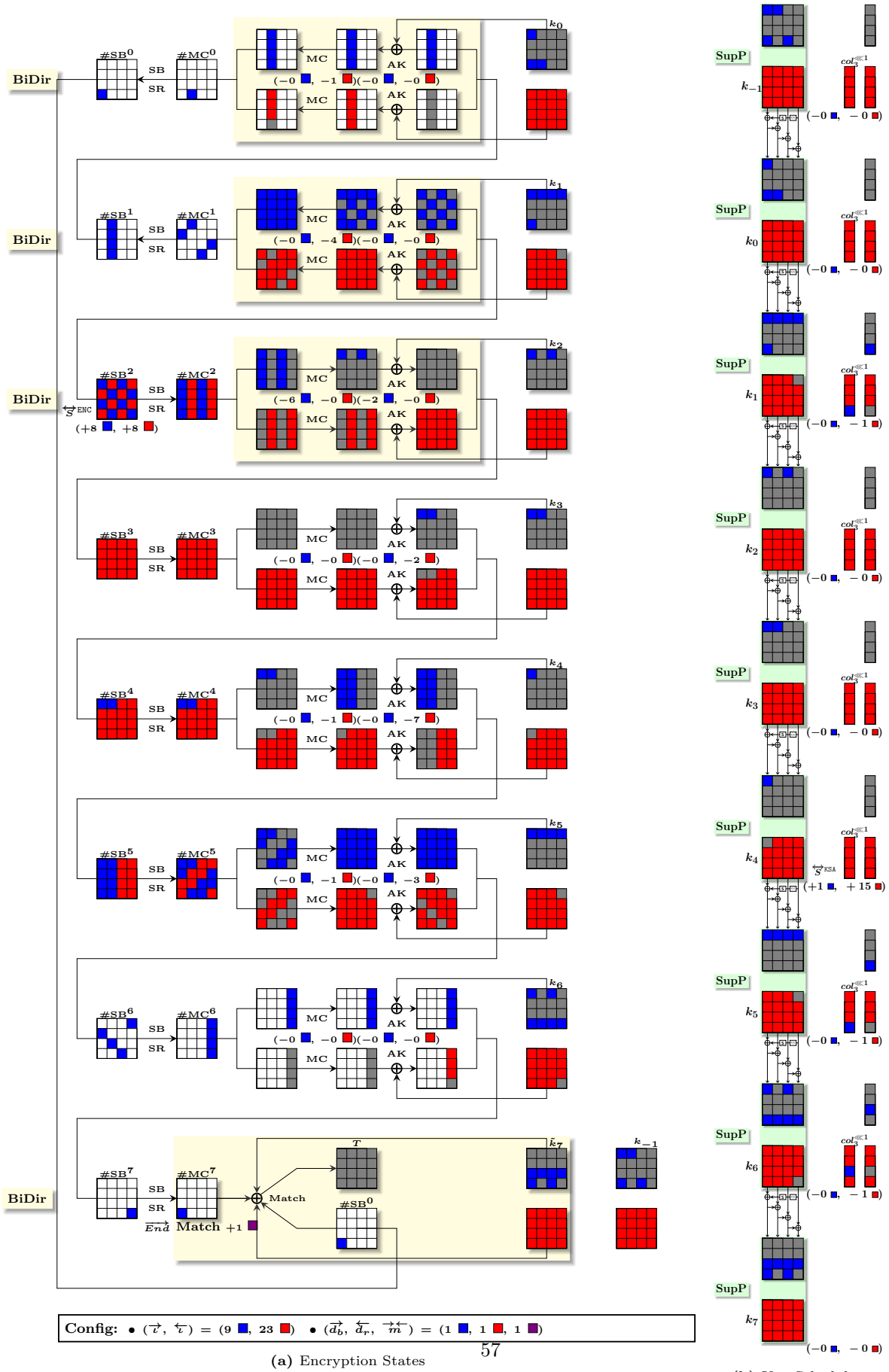


Fig. 24: A key-recovery attack on 23-round SKINNY-n-3n, optimized for data complexity

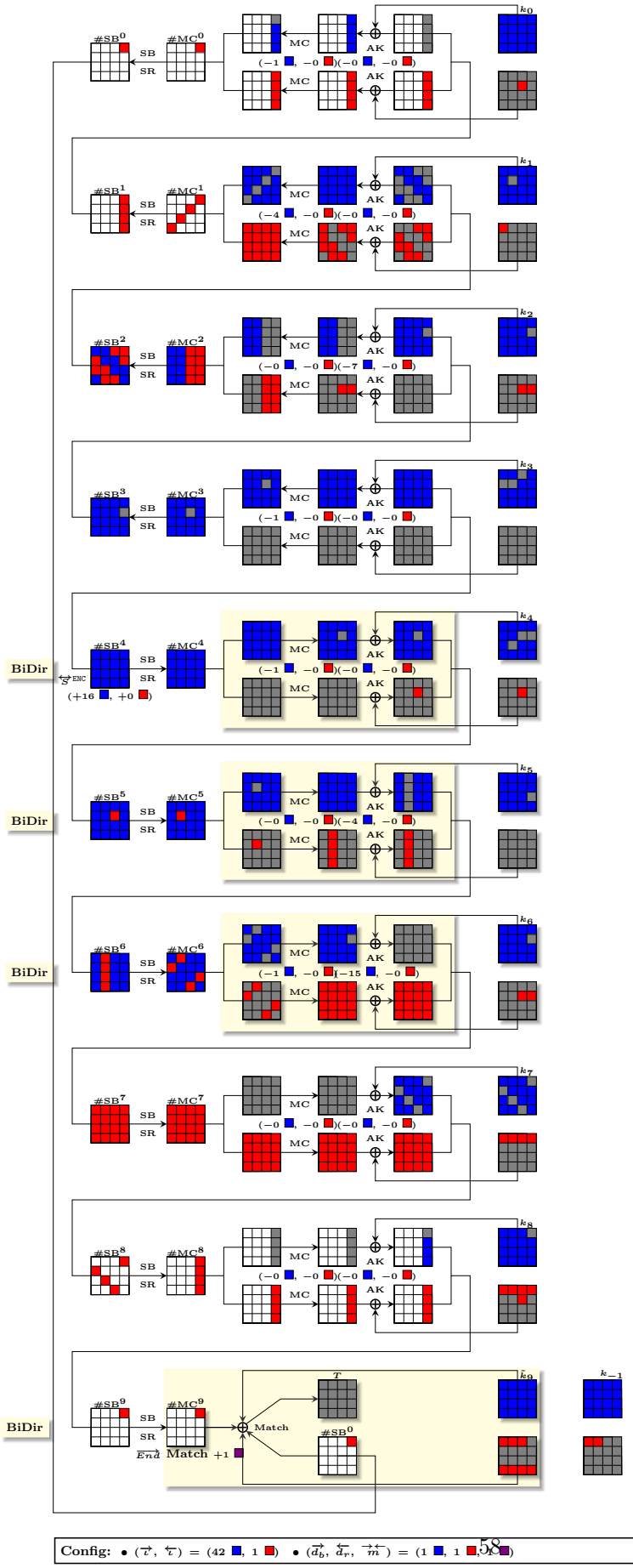


(a) Encryption States

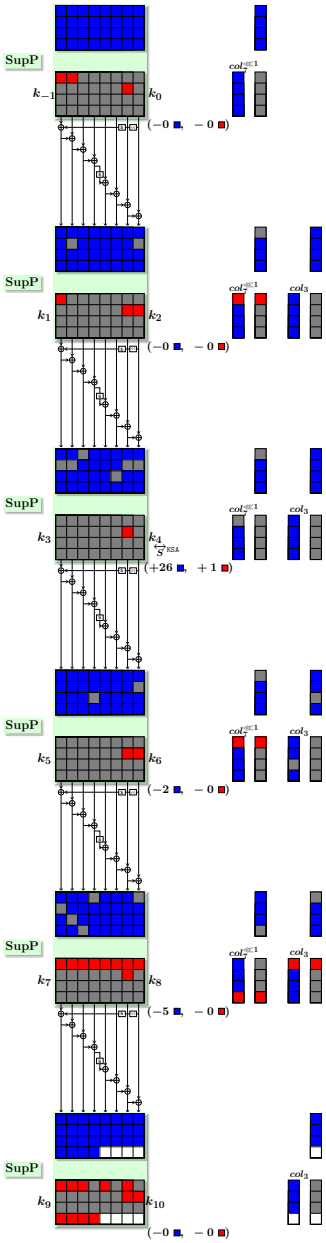
(b) Key Schedule

For converting this attack configuration to a valid collision attack, the attacker should be able to generate each value of neutral bytes with (amortized) computational complexity $O(1)$. It is necessary to pre-compute values of neutral bytes (in Red) for many fixed bytes (in Gray) at once with a total complexity of no more than 2^{60}. However, we have not found such a procedure using the same method as done for the attack on 7-round AES-128. The same problem stays open for converting attack configurations in Fig. 18, 26, and 19 to valid collision attacks.

Fig. 25: A configuration of MITM attack against 8-Round AES-128 hashing mode (facilitated to converting to a collision attack)



(a) Encryption States



(b) Key Schedule

Fig. 26: A configuration of MITM attack against 10-Round AES-256 hashing mode (facilitated to converting to a collision attack)

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 [34]. In [34], 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 [34] and is as follows (refer to Fig. 27).

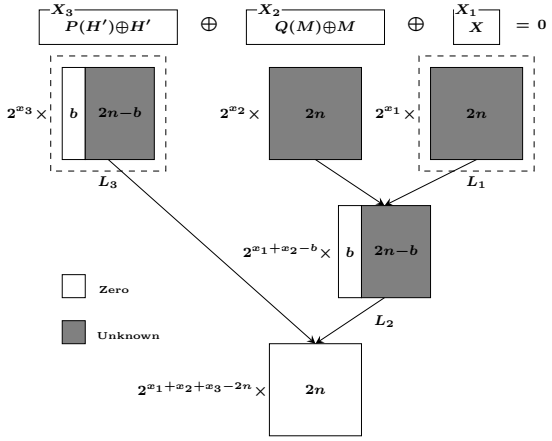


Fig. 27: Outline for pseudo-preimage attack on the Grøstl hash function [34]

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, retrieve 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 [34]); 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 [34]).

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 procedure was approximated in [34] as (after removing some constant factors)

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

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 [34] 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 $b \leq \min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*) \cdot c$, $\overline{C_2(2n,b)} = 0$ for $x_3 \geq b$, that is, the amortized complexity for finding each H' is one. When $b \geq \min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*) \cdot c$, $\overline{C_2(2n,b)} = 2^{b-\min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*)\cdot c}$, 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 $\min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\overleftarrow{m}}^*) \cdot c$ by G_2 meaning the gain over bruteforce attack on $P(H') \oplus H'$. Further, denote $\min(\overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overrightarrow{\overleftarrow{m}} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) \cdot c$ 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

$$C_T \approx \begin{cases} n - \frac{G_1}{3}, \text{ with } (x_1 = \frac{2G_1}{3}, x_2 = n - \frac{G_1}{3}, x_3 = n - \frac{G_1}{3}) & \text{if } G_2 \geq \frac{2G_1}{3}. \\ n - \frac{G_2}{2}, \text{ with } (x_1 = \frac{G_1}{2} + \frac{G_2}{4}, x_2 = n - \frac{G_1}{2} + \frac{G_2}{4}, x_3 = n - \frac{G_2}{2}) & \text{else } G_2 < \frac{2G_1}{3}. \end{cases} \tag{19}$$

The order of magnitude of the memory complexity is

$$C_M \approx \begin{cases} n + \frac{G_1}{3} - G_2, \ i.e., \ x_1 + x_2 - b, & \text{if } G_2 \geq \frac{2G_1}{3}. \\ n - \frac{G_2}{2}, \ i.e., \ x_3, & \text{if } G_2 < \frac{2G_1}{3}. \end{cases} \tag{20}$$

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	† [34]
Grøstl-256	5/10	42.67	234.67	234.67	64	64	21.33	234.67	213.33	∗ [24, 36]
Grøstl-256	5/10	48	232	232	72	72	24	232	208	Fig. 14, 15
Grøstl-256	6/10	12	248	252	16	8	4	252	252	∗ [24, 36]
Grøstl-256	6/10	21.33	245.33	245.33	32	24	10.67	245.33	242.67	Fig. 5, 6
Grøstl-512	7/14	64	480	480	96	104	32	480	440	Fig. 16, 17
Grøstl-512	8/14	12	508	508	16	8	4	508	508	† [34]
Grøstl-512	8/14	32	488	504	40	16	8	504	504	† [36]
Grøstl-512	8/14	28	496	500	40	24	12	500	500	Fig. 7, 8

$G_1 = \min(\overleftarrow{d_r} - \overrightarrow{d_{g_b}}, \overrightarrow{d_b} - \overleftarrow{d_{g_r}}, \overrightarrow{\leftarrow m} - \overrightarrow{d_{g_b}} - \overleftarrow{d_{g_r}} - \overleftrightarrow{d_{g_{br}}}) \cdot c$ meaning the gain over bruteforce attack on the OT.
$G_2 = \min(\overrightarrow{d_b}^*, \overleftarrow{d_r}^*, \overrightarrow{\leftarrow m}^*) \cdot c$ 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 [34] and [36] 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 [34] 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 [34] 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 [34] 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 [24] are on the OT of Grøstl-256. To convert to pseudo-preimage, we used the results for the case with no truncation in [24] . However, for the 6-round attack, in which guessing are required, it cannot be directly used. Thus, we combined the attack on OT in [24] with the best previous attack on the CF in [36].