Meet-in-the-Middle Attack on Primitives with Binary Matrix Linear Layer

Qingliang Hou[1], Kuntong Li[1], Guoyan Zhang[1,2(✉)], Yanzhao Shen[2], Qidi You[3,4], and Xiaoyang Dong[5]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, China
{qinglianghou, likuntong}@mail.sdu.edu.cn, guoyanzhang@sdu.edu.cn
[2] Shandong Institute of Blockchain, Jinan, China
shenyanzhao@sdibc.cn
[3] State Key Laboratory of Space-Ground Integrated Information Technology
youqd@spacestar.com.cn
[4] Space star Technology Co., Ltd
[5] Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing, China
xiaoyangdong@tsinghua.edu.cn

Abstract. Meet-in-the-middle (MitM) is a powerful approach for the cryptanalysis of symmetric primitives. In recent years, MitM has led to many improved records about key recovery, preimage and collision attacks with the help of automated tools. However, most of the previous work target AES-like hashing where the linear layer is an MDS matrix. And we observe that their automatic model for MDS matrix is not suitable for primitives using a binary matrix as their linear layer.

In this paper, we propose the n-XOR model to describe the XOR operation with an arbitrary number of inputs. And it can be applied to primitives with a binary matrix of arbitrary size. Then, we propose a check model to eliminate the possible inaccuracies caused by n-XOR. But the check model is limited by the input size (not greater than 4). Combined with the two new models, we find a MitM key recovery attack on 11-round Midori64. When the whitening keys are excluded, a MitM key recovery attack can be mounted on the 12-round Midori64. Compared with the previous best work, both of the above results have distinct advantages in terms of reducing memory and data complexity. At last, we apply the n-XOR model to the hashing modes of primitives with large size binary matrix. The preimage attack on weakened Camellia-MMO (without FL/FL^{-1} and whitening layers) and Aria-DM are both improved by 1 round.

Keywords: Meet-in-the-Middle · Binary Matrix · Key Recovery · Preimage · Midori64 · Camellia· Aria.

First Author and Second Author contributed equally to this work.

1 Introduction

The Meet-in-the-middle (MitM) is a powerful cryptanalysis strategy first proposed by Diffie and Hellman to attack Double DES [12]. The core idea is to identify two disjoint neutral sets of unknown values. Then, the whole computation path can be divided into two independent chunks, which are determined by two neutral sets and denoted by *forward chunk* and *backward chunk*, respectively. At last, the two chunks will meet at a common internal state where the consistency is checked to filter out candidate assignments of unknown values. From then on, MitM and its variants have been successfully applied to many block ciphers [9,32,18,29]. At SAC 2008, Aumasson *et al.* [3] first introduced the theory of MitM into preimage attacks on step-reduced MD5 and 3-pass HAVAL. Sequentially, many refined techniques were proposed to enhance the power of MitM, such as splice-and-cut [2], initial structure [30], bicliques [8], and so on. At FSE 2011, Sasaki [26] applied such MitM preimage attack to the PGV [25] hashing modes of AES and presented the first preimage attack on 7-round AES-MMO/MP/DM together with the partial indirect matching technique. Interestingly, these enhancements were finally found to be applicable in the key recovery attack on block ciphers. At ACISP 2011, Wei *et al.* [37] broke the full round KTANTAN using the splice-and-cut technique by connecting the plaintext and ciphertext with encryption or decryption oracles with only 4 chosen plaintexts.

Despite being clear that a MitM attack is entirely determined by its *characteristic*, i.e., the configuration for two chunks, it's still complicated and error-prone to explore the whole configuration space. Recently, automated tools were introduced to find the best characteristic by solving an optimization problem. At Eurocrypt 2021, Bao *et al.* [6] proposed an MILP-based MitM preimage attack on AES-like hash and Haraka v2. At CRYPTO 2021, Dong *et al.* [13] extended the automatic model into key-recovery and collision attacks and introduced a table-based method to solve the non-linear constraints imposed on neutral sets. At CRYPTO 2022, Bao *et al.* [7] considered the MitM attack in a view of superposition (SupP) states and bi-directional attribute propagation (BiDir) such that neutral sets are treated independently and can be imposed constraints in both computation paths. At Asiacrypt 2023, Hou *et al.* [17] introduced the SupP framework into Feistel-based hash functions. At Eurocrypt 2024, Chen *et al.* [10] considered the linearization of the S-Box in AES and allowed a linear combination of two neutral sets in the initial structure. Different from the above work, Schrottenloher and Stevens [33] studied a simple top-down modeling paradigm for both classical and quantum preimage attacks against permutations and was later extended to key recovery attack on block ciphers with simple key schedules [34]. The simplified attack excluded many details. In this paper, we adopt the bottom-up MitM framework in [7] and the table-based method in [13].

In the previous work, the targets are most built by a block cipher with an MDS matrix. Through the diffusion layer, each output cell is related to all the input cells. However, the primitives with binary matrix are rarely studied, where each output cell is represented as the XOR of partial input cells. In [13], Dong *et al.* introduced the 3-XOR model for SKINNY-n-$3n$. In their model, the number of input cells is fixed to be 4. All valid cases can be easily exhausted to form a system of inequalities using the convex hull method [36]. However, if more input cells are involved, the number of valid cases will increase extremely leading to larger size of system of inequalities, which can make model infeasible to compute. Hence, there is a gap to find an accurate and effective method to describe the MitM attribute propagation through a binary matrix of arbitrary size.

Our Contributions. In this paper, we propose a novel model called n-XOR under the encoding scheme in [7], to describe the propagation of MitM attributes through an XOR operation with an arbitrary number of input cells. And the number of inequalities formed by n-XOR is fixed, independent of the number of inputs. Hence, n-XOR can be applied to large binary matrices effectively. However, we also observe that only applying n-XOR will lead to subtle inaccuracies. An extremely explicit case is that the constraint on the same neutral bits may be double counted in two different n-XOR operations. Besides, there are more implicit cases depending on the specific linear layer. Hence, we propose an additional check model to eliminate these inaccuracies. But this model is limited by the input size n, that is, $n \leq 4$ in our paper.

As a low-energy lightweight cryptography, Midori [5] is well-suited for constrained environments, like the edge gateways and end devices in the blockchain on-chain and off-chain interactions. As a proof of work, we first apply the two new models to Midori64 [5], with a 4×4 binary matrix as linear layer. Then, an 11-round key recovery attack is found with time complexity of 2^{124}. The data and memory complexity are 2^{36} and 2^{6}, respectively. When omitting the whitening layer, a 12-round MitM characteristic for weakened Midori64 is found with time complexity of 2^{120}. The data and memory cost are 2^{48} and $2^{10.6}$, respectively. Besides, the data and memory complexity can be further reduced if the time complexity is relaxed to 2^{124}. Compared to the previous best records of Midori64 [23,35,22], despite a little higher time complexity, our results have distinct advantages in reducing data and memory complexity.

It's a practical design strategy to build hash functions on widely used block cipher with a longstanding record of cryptanalysis. And AES-MMO was even internationally standardized by ISO [19]. Since Camellia [1] was also standardized by ISO [20] and Aria [21] was standardized by Korean Standard (KS X1213), the hashing modes of Camellia or Aria may be potential candidates used in practice. Indeed, their security have been evaluated in a series of works [31,27,16,4]. In this paper, we apply the n-XOR to describe the MitM attributes propagation through the large binary matrix of Camellia and Aria. Finally, we find a preimage attack on 14-round weakened Camellia-MMO (without FL/FL^{-1} and whitening layers) and a preimage attack on 6-round Aria-DM. Compared to the previous best records [28,16], the attack rounds are both improved by 1 round.

Our results are also summarized in Table 1 and Table 2. For the source code, please refer to https://github.com/wenny-kt/MITM-Binary-Matrix.

The rest of this paper is organized as follows. In Section 2, we give an overview of how the automated MitM attacks are deployed, along with some enhanced

Table 1: Single Key attacks on `Midori64`, where ID and $\mathcal{DS}$-MitM denote impossiable differential and Demirci-Selçuk MitM attack, respectively.

Target	Rounds	Data	Memory(Bytes)	Time(Enc.)	Technique	Ref.
`Midori64`	11	2^{60}	$2^{95.8}$	$2^{116.6}$	ID	[23]
	11	2^{53}	$2^{92.2}$	2^{122}	$\mathcal{DS}$-MitM	[22]
	11	$\mathbf{2^{36}}$	$\mathbf{2^{6}}$	2^{124}	MitM	Section 4.1
	12	$2^{55.5}$	2^{109}	$2^{125.5}$	$\mathcal{DS}$-MitM	[22]
	$12^{\dagger}$	$2^{61.9}$	2^{44}	$2^{90.5}$	ID	[35]
	$12^{\dagger}$	$\mathbf{2^{48}}$	$\mathbf{2^{10.6}}$	2^{120}	MitM	Section 4.2
	$12^{\dagger}$	$\mathbf{2^{36}}$	$\mathbf{2^{5.6}}$	2^{124}	MitM	Section 4.2

† Weakened version without whitening layers.

Table 2: A Summary of the MitM Attacks on Hashing Modes.

Target	Attacks	Rounds	Time1	Time2	Memory	Technique	Ref.
`Camellia-MMO`	Preimage	$13^{\ddagger}$	2^{120}	2^{125}	2^{8}	MitM	[28]
		$14^{\ddagger}$	2^{120}	2^{125}	2^{8}	MitM	Section 5
`Aria-DM`	Preimage	5	2^{120}	2^{125}	2^{8}	MitM	[16]
		6	2^{120}	2^{125}	2^{112}	MitM	Section 6

- ‡ Weakened version without FL/FL^{-1} and whitening layers.
- Time1 represents the time complexity of pseudo-preimage. Time2 represents the time complexity of preimage attack converted from the pseudo-preimage attack according to [24, Fact9.99].

techniques. In Section 3, we introduce two new improved models embedded in the automated MitM framework, called `n-XOR` and check model. The applications to `Midori64`, `Camellia-MMO` and `Aria-DM` are presented in Sects. 4, 5 and 6, respectively. Finally, we conclude in Section 7.

2 Preliminaries: Automated Meet-in-the-Middle Attack

In this section, we provide an overview of how the MitM attack framework is constructed, and how it is encoded into the MILP language with specified configurations for the preimage and key recovery attack. Then, we recall two enhanced techniques to improve the power of MitM attack. The first one is the *table-based method* introduced in [13] to solving the non-linear constraints. Another one is the *Superposition (SupP) States and Bi-direction Attribute-Propagation (BiDir)* introduced in [7] to preserving more valid solutions.

2.1 Framework of the Meet-in-the-Middle Attack

The MitM attack framework is illustrated in Figure 1. $\mathcal{S}^{\texttt{ENC}}$ and $\mathcal{S}^{\texttt{KEY}}$ are the starting states where there are $\lambda_{\mathcal{B}}^{\texttt{ENC}}$ and $\lambda_{\mathcal{B}}^{\texttt{KEY}}$ neutral bits for forward computation denoted by 🟦, and there are $\lambda_{\mathcal{R}}^{\texttt{ENC}}$ and $\lambda_{\mathcal{R}}^{\texttt{KEY}}$ neutral bits for backward computation denoted by 🟥. After imposing $l_{\mathcal{R}}^{\texttt{ENC}}$ and $l_{\mathcal{R}}^{\texttt{KEY}}$ constraints on $\lambda_{\mathcal{R}}^{\texttt{ENC}}$ and $\lambda_{\mathcal{R}}^{\texttt{KEY}}$ backward neutral bits, respectively, 🟦 can be propagated to the matching

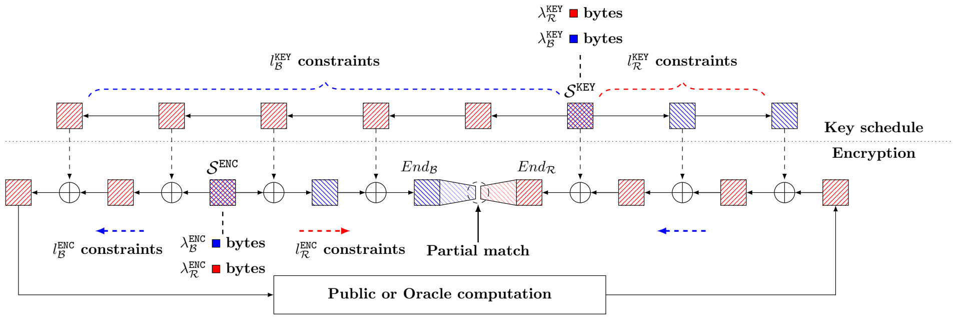

$\lambda_{\mathcal{R}}^{\texttt{KEY}}$ 🟥 bytes
$\lambda_{\mathcal{B}}^{\texttt{KEY}}$ 🟦 bytes
$l_{\mathcal{B}}^{\texttt{KEY}}$ constraints
$l_{\mathcal{R}}^{\texttt{KEY}}$ constraints
$\mathcal{S}^{\texttt{KEY}}$
Key schedule
Encryption
$\mathcal{S}^{\texttt{ENC}}$
$End_{\mathcal{B}}$
$End_{\mathcal{R}}$
$l_{\mathcal{B}}^{\texttt{ENC}}$ constraints
$\lambda_{\mathcal{B}}^{\texttt{ENC}}$ 🟦 bytes
$\lambda_{\mathcal{R}}^{\texttt{ENC}}$ 🟥 bytes
$l_{\mathcal{R}}^{\texttt{ENC}}$ constraints
Partial match
Public or Oracle computation

Fig. 1: A high-level overview of the MITM attacks [13]

points $End_{\mathcal{B}}$ independent of the 🟥 bits. The degree of freedom (DoF) for the 🟥 neutral space is computed by $d_{\mathcal{R}} = \lambda_{\mathcal{R}}^{\texttt{ENC}} + \lambda_{\mathcal{R}}^{\texttt{KEY}} - l_{\mathcal{R}}^{\texttt{ENC}} - l_{\mathcal{R}}^{\texttt{KEY}}$. Similarly, forward neutral bits are imposed on $l_{\mathcal{B}}^{\texttt{ENC}}$ and $l_{\mathcal{B}}^{\texttt{KEY}}$ constraints to cancel the effect of 🟦 in the backward computation. The DoF of the 🟦 neutral space can be computed by $d_{\mathcal{B}} = \lambda_{\mathcal{B}}^{\texttt{ENC}} + \lambda_{\mathcal{B}}^{\texttt{KEY}} - l_{\mathcal{B}}^{\texttt{ENC}} - l_{\mathcal{B}}^{\texttt{KEY}}$. Through a feed-forward mechanism or querying a public Encryption-Decryption oracle, $End_{\mathcal{R}}$ can be derived by 🟥. Instead of requiring the full states, the partial matching exploits the filtering ability derived by the deterministic relation "$End_{\mathcal{B}} = End_{\mathcal{R}}$" and denoted by d_m.

With the configurations of $\left(\lambda_{\mathcal{B}}^{\texttt{ENC}}, \lambda_{\mathcal{B}}^{\texttt{KEY}}, \lambda_{\mathcal{R}}^{\texttt{ENC}}, \lambda_{\mathcal{R}}^{\texttt{KEY}}, l_{\mathcal{B}}^{\texttt{ENC}}, l_{\mathcal{B}}^{\texttt{KEY}}, l_{\mathcal{R}}^{\texttt{ENC}}, l_{\mathcal{R}}^{\texttt{KEY}}, d_m\right)$, the basic attack procedure goes as follows:

1. Choose constants in $\mathcal{S}^{\texttt{ENC}}$ and $\mathcal{S}^{\texttt{KEY}}$ and $l_{\mathcal{B}}^{\texttt{ENC}} + l_{\mathcal{B}}^{\texttt{KEY}} + l_{\mathcal{R}}^{\texttt{ENC}} + l_{\mathcal{R}}^{\texttt{KEY}}$ constraints.
2. For $2^{d_{\mathcal{B}}}$ values of 🟦 neutral space, compute forward to $End_{\mathcal{B}}$ from the starting states, and store the values of 🟦 in table $L_{\mathcal{B}}[End_{\mathcal{B}}]$.
3. For $2^{d_{\mathcal{R}}}$ values of 🟥 neutral space, compute backward to $End_{\mathcal{R}}$ from the starting states, and store the values of 🟥 in table $L_{\mathcal{R}}[End_{\mathcal{R}}]$.
4. According to the indices, check the match between $L_{\mathcal{B}}$ and $L_{\mathcal{R}}$.
5. For the surviving pairs that pass the match, check for a full-state match.

Complexity analysis. The above steps 2-5 form a MitM *episode*. To find an h-bit full match, $2^{h-(d_{\mathcal{B}}+d_{\mathcal{R}})}$ episodes are needed. Since each episode is performed with a time of $2^{\max\{d_{\mathcal{B}},d_{\mathcal{R}}\}} + 2^{d_{\mathcal{B}}+d_{\mathcal{R}}-d_m}$, the total time complexity is:

$$2^{h-(d_{\mathcal{B}}+d_{\mathcal{R}})} \cdot \left(2^{\max\{d_{\mathcal{B}},d_{\mathcal{R}}\}} + 2^{d_{\mathcal{B}}+d_{\mathcal{R}}-d_m}\right) \approx 2^{h-\min\{d_{\mathcal{B}},d_{\mathcal{R}},d_m\}} \tag{1}$$

Apparently, a MitM characteristic is valid, if and only if $\min\{d_{\mathcal{B}}, d_{\mathcal{R}}, d_m\} \geq 1$. For MitM key recovery attack, additional constraints must be fulfilled to ensure that the internal states in $\mathcal{S}^{\texttt{ENC}}$ can be totally determined by $\mathcal{S}^{\texttt{KEY}}$. This is equivalent to using up the DoFs of $\mathcal{S}^{\texttt{ENC}}$, i.e., $\lambda_{\mathcal{B}}^{\texttt{ENC}} - l_{\mathcal{B}}^{\texttt{ENC}} = 0$ and $\lambda_{\mathcal{R}}^{\texttt{ENC}} - l_{\mathcal{R}}^{\texttt{ENC}} = 0$. Besides, there should exists only one type of neutral bit in the plaintext or ciphertext, and at least 1-bit constant in the plaintext or ciphertext to avoid using up the full codebook. In [6], Bao *et al.* encoded the type of each byte in `AES` with a pair of boolean variables:

1. 🟥 $\mathcal{R}$, $(x, y) = (0, 1)$: Known byte only with backward computation.
2. 🟦 $\mathcal{B}$, $(x, y) = (1, 0)$: Known byte only with forward computation.
3. ▨ $\mathcal{G}$, $(x, y) = (1, 1)$: Constant byte and known in both forward and backward computations.
4. □ $\mathcal{W}$, $(x, y) = (0, 0)$: Unknown byte in forward and backward computations.

Then, the propagation rules for `XOR` and `MixColumns` can be described as a system of inequalities based on the above definitions. A valid MitM characteristic is defined as a solution solved by the off-the-shelf MILP solvers, like Gurobi [15], with the objective function that maximizes the $\min\{d_{\mathcal{B}}, d_{\mathcal{R}}, d_m\}$. For the detailed MILP models of these propagation rules, please refer to [6] or Appendix A.

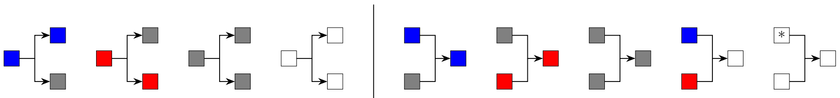
2.2 Enhanced Techniques

Table-based method solving non-linear constraints. Note that Equation (1) holds mostly when the constraints imposed on neutral bits can be solved in $O(1)$ time, such as linear equations. However, there are many practice MitM characteristics with non-linear constrained neutral bits, which can not be solved efficiently. In [13], Dong *et al.* proposed a precomputation method to compute the value of the constraints by enumerating the neutral bits. Specifically, after setting the value of constants in starting states, do as follows:

1. For $2^{\lambda_{\mathcal{B}}^{\mathtt{ENC}}+\lambda_{\mathcal{B}}^{\mathtt{KEY}}}$ 🟦 values, compute the values of $l_{\mathcal{B}}^{\mathtt{ENC}} + l_{\mathcal{B}}^{\mathtt{KEY}}$ constraints (denoted by $\mathfrak{c}_{\mathcal{B}} \in \mathbb{F}_2^{l_{\mathcal{B}}^{\mathtt{ENC}}+l_{\mathcal{B}}^{\mathtt{KEY}}}$) and store the $\lambda_{\mathcal{B}}^{\mathtt{ENC}} + \lambda_{\mathcal{B}}^{\mathtt{KEY}}$ 🟦 bits in $U[\mathfrak{c}_{\mathcal{B}}]$.
2. For $2^{\lambda_{\mathcal{R}}^{\mathtt{ENC}}+\lambda_{\mathcal{R}}^{\mathtt{KEY}}}$ 🟥 values, compute the values of $l_{\mathcal{R}}^{\mathtt{ENC}} + l_{\mathcal{R}}^{\mathtt{KEY}}$ constraints (denoted by $\mathfrak{c}_{\mathcal{R}} \in \mathbb{F}_2^{l_{\mathcal{R}}^{\mathtt{ENC}}+l_{\mathcal{R}}^{\mathtt{KEY}}}$) and store the $\lambda_{\mathcal{R}}^{\mathtt{ENC}} + \lambda_{\mathcal{R}}^{\mathtt{KEY}}$ 🟥 bits in $V[\mathfrak{c}_{\mathcal{R}}]$.

Then, in each MitM episode, for a given $\mathfrak{c}_{\mathcal{B}}$ and $\mathfrak{c}_{\mathcal{R}}$, the values in $U[\mathfrak{c}_{\mathcal{B}}]$ and $V[\mathfrak{c}_{\mathcal{R}}]$ can be searched in time $O(1)$. The time and memory cost for one precomputation phase are both $2^{\lambda_{\mathcal{B}}^{\mathtt{ENC}}+\lambda_{\mathcal{B}}^{\mathtt{KEY}}} + 2^{\lambda_{\mathcal{R}}^{\mathtt{ENC}}+\lambda_{\mathcal{R}}^{\mathtt{KEY}}}$.

SupP States and BiDir. In the SupP MitM framework of [7], neutral cells from both directions can be separated into two virtual states, called SupP states, to keep the linearity through linear operations. Then, 🟦 and 🟥 will be treated independently through linear operations, and the initial DoFs can be consumed in both directions. After a series of linear operations, two SupP states are finally combined before the next nonlinear operation. The color patterns and how the states are separated and combined are visualized in Figure 2. BiDir allows neutral cells to be consumed in both two directions, but this may lead to dependency between one type of neutral cell with non-linear constraints imposed on another. In [11], Degré proposed a more generic table-based method to cancel this dependency. Combined with the SupP states and BiDir methods, the solution space is greatly enlarged, such that some attack configurations with lower time complexities may be found. In the rest of this paper, we simplify the representation of SupP states. The virtual states of pure 🟥/🟦/▨/□ are omitted. And we denote the SupP states by the 🟩 cell in which the blue cell and red cell occur simultaneously.



(a) Rules for separation

(b) Rules for combination

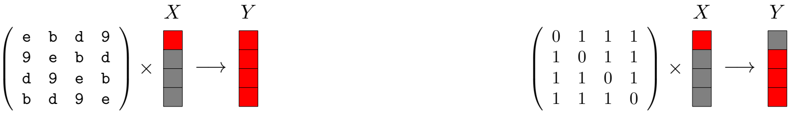
Fig. 2: Rules for separation and combination, where "$*$" means any color

3 New Models for Linear Layer with Binary Matrix

In this section, we first propose an effective method to build an MILP model to describe the MitM attributes propagation through a `n-XOR` operation with SupP states. Interestingly, the number of input cells involved in the `XOR` operation can be arbitrary, but the size of MILP model will not increase. However, we also observe that this may lead to double counting of constraints on the same neutral cells. Then, we show that the inaccuracy can be easily eliminated by adding an additional check model.

3.1 N-XOR Model

To simulate the MitM attributes propagation through the linear layer, Bao *et al.* proposed the `MC-RULE` for the MDS matrix in `AES`-like hashing [6,7]. As shown in Figure 3(a), each input cell has an effect on all output cells in MDS matrix. However, some primitives adopt a binary matrix in the diffusion layer where each output cell is computed by the `XOR` of partial input cells. As the `Midori64`'s binary matrix shown in Figure 3(b), the first output cell is only related to the last three input cells. Apparently, this will lead to inaccurate propagation if we apply the `MC-RULE` for MDS matrix on binary matrix directly since one output cell is not related to all input cells.



(a) Coloring pattern of `MC-RULE` for MDS matrix

(b) Coloring pattern for binary matrix

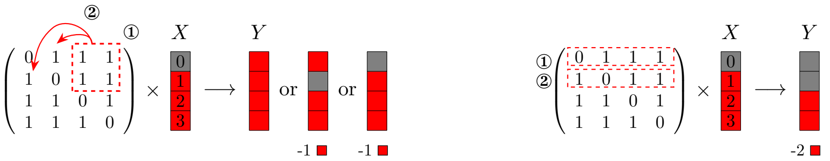
Fig. 3: A case of the difference of color pattern between MDS and binary matrix

In [13], Dong *et al.* proposed the `3-XOR-RULE` to model the key addition in `SKINNY`-n-$3n$. By enumerating four input cells, one output cell and one indicator variable for DoF cost, all valid color patterns can be restricted to a subset of $\mathbb{F}_2^{11}$, which can be described into a system of inequalities using the convex hull technique [36]. If we directly extend the strategy of `3-XOR-RULE` to the `XOR` operation with n input cells, then the enumeration scope will be restricted to a subset of $\mathbb{F}_2^{2n+3}$. When n is large, it's complicated and error-prone to enumerate all valid color patterns. And the size of the system of inequalities may be large, which renders the model infeasible to compute.

An alternative strategy is to apply the `XOR-RULE` in [6,7] for two-input `XOR` consecutively. This strategy is valid but may miss some valid patterns by introducing additional auxiliary variables. We take the attribute propagation through `Midori64`'s diffusion layer to state this fact as shown in Figure 4. In the first step of Figure 4(a), an auxiliary variable `auxi` is needed to carry on the output of $X[2] \oplus X[3]$. For the second step, $X[1]$ and $X[0]$ are `XOR`ed with `auxi` to compute $Y[0]$ and $Y[1]$, respectively. Then, one of the following cases will occur,

- If `auxi` is ■ by consuming one DoF, then $Y[0]$ will always be 🟥, and $Y[1]$ will always be ■.
- If `auxi` is 🟥, then $Y[1]$ will always be 🟥. $Y[0]$ can be either 🟥 or ■ by consuming one DoF.

However, with the `n-XOR` model in Figure 4(b), step 1 and step 2 can be executed independently without correlated variables. Then, $Y[0]$ and $Y[1]$ can be ■ simultaneously by consuming 2 DoFs of 🟥, which can not be captured by the first strategy.



(a) Model of consecutive `XOR` (b) `n-XOR` Model

Fig. 4: The advantage of `n-XOR` model compared with consecutive `XOR`

In the following, we show how to convert the propagation of 🟥 cells through the `n-XOR` operation under SupP states into MILP language. All coloring patterns can be specified by the following set of rules denoted by `n-XOR-RULE`$^-$. The `n-XOR-RULE`$^+$ for 🟦 can be obtained in a similar way by exchanging 🟦 and 🟥 since they are dual.

- `n-XOR-RULE`$^-$-1. If there is at least one □ in input, then the output is □.
- `n-XOR-RULE`$^-$-2. If all cells of the input are ■, then the output must be ■.
- `n-XOR-RULE`$^-$-3. If there are 🟥 and ■ cells but no □ cell in the input, then one of the following situations will occur:
  - The output is 🟥 cell and no DoF is consumed.
  - The output is ■ by consuming one DoF of 🟥.

Let $(A[1], A[2], \cdots, A[n])$ be the input of `n-XOR` where $A[i] = (x_i^A, y_i^A)$. Let B be the output where $B = (x^B, y^B)$. Like [6], we introduce three boolean indicator variables μ, ν and η in the model. $\mu = 1$ if and only if there exists $i \in [1, 2, \cdots, n]$ such that $(x_i^A, y_i^A) = (0, 0)$. That is, `n-XOR-RULE`$^-$-1 is fulfilled. $\nu = 1$ if and only if $x_i^A = y_i^A = 1$ for all $1 \leq i \leq n$, which corresponds to `n-XOR-RULE`$^-$-2. When $\mu = \nu = 0$, `n-XOR-RULE`$^-$-3 is fulfilled. Besides, $\eta = 1$ when there exists one constraint imposed on input 🟥 cells. With the help of indicator variables, the `n-XOR-RULE`$^-$ can be converted into a system of inequalities shown in Equation (2) and Equation (3).

$$\begin{cases} \sum_{i=0}^{n-1} y_i^A + \mu \leq n \\ \sum_{i=0}^{n-1} y_i^A + n \cdot \mu \geq n \\ \sum_{i=0}^{n-1} x_i^A - \nu \leq n - 1 \\ \sum_{i=0}^{n-1} x_i^A - n \cdot \nu \geq 0 \end{cases} \tag{2}$$

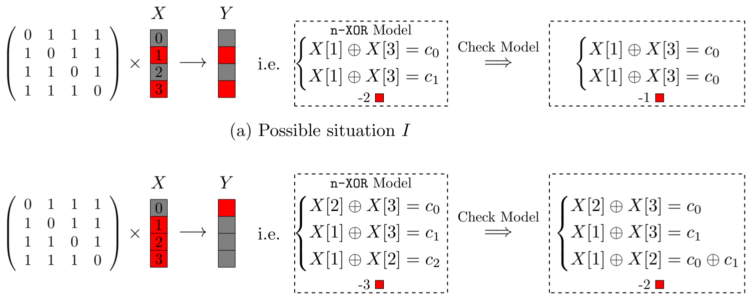
$$\begin{cases} y^B + \mu = 1 \\ x^B + \mu \leq 1 \\ \eta - x^B + \nu = 0 \\ \sum_{i=0}^{n-1} x_i^A + x^B - 2 \cdot \nu \leq n - 1 \\ \sum_{i=0}^{n-1} x_i^A + x^B - (n+1) \cdot \nu \geq 0 \end{cases} \tag{3}$$

At the end, we must emphasize that, in addition to preserving more valid coloring patterns, another advantage of `n-XOR` is that the size of model is fixed, independent of the number of input cells. And this makes it possible to describe the attributes propagation for primitives with large binary matrices, like `Camellia` and `Aria`.

3.2 Check Model: More Accurate Consumption of DoFs

We also observe that `n-XOR` model may lead to some subtle inaccuracies. We still take a possible propagation of `Midori64`'s diffusion layer as an example to state this fact. A particularly explicit case is that the constraint on the same neutral cells may be double counted due to the independent computation of each output cell as shown in Figure 5(a). Besides, there are some more implicit cases leading to inaccuracy as shown in Figure 5(b).

Then, we introduce the check model to show how the inaccuracy can be eliminated, and describe it in the MILP language. We still state this by considering the 🟥 propagation through the `n-XOR` operation under SupP states. Let $A[j] = (x_j^A, y_j^A)$, for $1 \leq j \leq n$, be the input of the $n \times n$ binary matrix M. After the `n-XOR` Model, we can get $\boldsymbol{\eta} = (\eta_1, \cdots, \eta_n)$ denoted by the degree consumption vector where η_i is the indicator variable introduced in Equation (3) and $\eta_i = 1$ means there exists one constraint imposed on the input 🟥 cells for the i-th row of M. Since only 🟥 cells are needed to be considered for DoF consumption, we introduce another $n \times n$ binary matrix M' to intuitively mark which 🟥 cells contribute to the DoF consumption. Then, M' is generated as follows :



(a) Possible situation I

(b) Possible situation II

Fig. 5: Possible situations in our models

- If $\eta_i = 1$ and $M_{i,j} = 1$ and $x_j^A = 0$, then $M'_{i,j} = 1$.
- If the first case is not satisfied, then $M'_{i,j} = 0$.

For the first case, $\eta_i = 1$ means no □ in the involved input cells, and $M_{i,j} = 1$ and $x_j^A = 0$ means $A[j]$ is a 🟥 cell involved in the i-th XOR operation. We introduce a general variable η' to denote the rank of M', which equals to the accurate DoF consumption theoretically. Since M is a fixed matrix, we can conclude that the accurate DoF consumption can be determined by the other $2n$ variables $(x_1^A, \cdots, x_n^A, \eta_1, \cdots, \eta_n)$. Finally, the subset $(x_1^A, \cdots, x_n^A, \eta_1, \cdots, \eta_n, \eta')$ of $\mathbb{F}_2^{2n} \times \mathbb{F}_{n+1}$ can be restricted to a system of linear inequalities using the convex hull technique [36]. Different with the origin framework, the configuration $l_{\mathcal{R}}^{\mathtt{ENC}} + l_{\mathcal{R}}^{\mathtt{KEY}}$ should be calculated by accumulating the accurate DoF consumption determined by the n-XOR and check model, along with extra constraints imposed by other operations, such as KeyAddition. The configuration $l_{\mathcal{B}}^{\mathtt{ENC}} + l_{\mathcal{B}}^{\mathtt{KEY}}$ for degree consumption of 🟦 can also be gotten in the similar way due to the duality [7].

However, it should be noted that the cost of exhaustion to determine the accurate DoF consumption is still affected by the number of input cells. Hence, check model can not be applied to large binary matrix ($n > 4$ in this paper). Although it's trivial to compute the rank of a general matrix in $O(n^3)$, there is still no effective way to implement it in MILP model. Besides, in addition to finding out better modeling methods or more suitable optimizers, we can still combine theoretical models and manually checking to deal with large matrices, such as Section 5 and Section 6. In practice, by relaxing the constraint to $\min\{d_{\mathcal{B}}, d_{\mathcal{R}}, d_m\} \geq 1 - i$, where $i \geq 1$, we check the feasible solutions to find out valid characteristic. It also should be noted that the final results derived by the manually checking method may not be the optimal solution.

4 MitM Key Recovery Attack on Midori64

Midori64 is an SPN-based lightweight block cipher, consisting of 64-bit block and a 128-bit key. The state is seen as a 4×4 matrix of 4-bit cells, and its diffusion layer is 4×4 boolean matrix. The detailed specification is provided in Appendix B.1.

In this section, we present an 11-round MitM key recovery attack on `Midori64` with a time complexity of 2^{124}. For the weakened version of `Midori64`, without whitening key, a 12-round MitM characteristic is found with a time complexity of 2^{120}. Despite a little higher time complexity, the above two attacks can be applied with extremely low data and memory cost compared to the previous best work [23,35]. Besides, the data and memory of the attack on 12-round weakened `Midori64` can be further reduced if the time complexity is relaxed to 2^{124}.

4.1 MitM Key Recovery Attack on 11-round Midori64

As shown in Figure 6 and Figure 7, an 11-round MitM key recovery attack is identified, where $|\mathcal{S}^{\mathtt{ENC}}| = 16$ independent bytes in the encryption data path are set to be 0 as Line 1-2 in Algorithm 1, to ensure the values of all the other bytes are totally determined by the given key. And at least one 0 byte in the ciphertext C to avoid using the full codebook. The starting states are C and $(K^{(0)}, K^{(1)})$. The encryption data path provides $\lambda_{\mathcal{R}}^{\mathtt{ENC}} = 9$ and $\lambda_{\mathcal{B}}^{\mathtt{ENC}} = 0$ DoFs for 🟥 and 🟦, respectively. And the $\lambda_{\mathcal{R}}^{\mathtt{ENC}} = 9$ 🟥 cells are used up when computing $A_{\mathtt{ShC}}^{(9)}$ through an `MC` operation and $A_{\mathtt{MC}}^{(8)}$ through an `XOR` operation in the backward computation path. For $(K^{(0)}, K^{(1)})$, the initial DoFs for 🟥 and 🟦 are $\lambda_{\mathcal{R}}^{\mathtt{KEY}} = 3$ and $\lambda_{\mathcal{B}}^{\mathtt{KEY}} = 2$, respectively. In the key schedule, $K^{(0)}[1] \oplus K^{(0)}[9]$ and $K^{(0)}[1] \oplus K^{(0)}[13]$ are restricted to constants, i.e., $l_{\mathcal{R}}^{\mathtt{KEY}} = 2$. Hence, we get $\mathrm{DoF}_{\mathcal{R}} = \lambda_{\mathcal{R}}^{\mathtt{KEY}} - l_{\mathcal{R}}^{\mathtt{KEY}} = 1$. Similarly, $K^{(0)}[5] \oplus K^{(1)}[5]$ is imposed on $l_{\mathcal{B}}^{\mathtt{KEY}} = 1$ constraint, and then $\mathrm{DoF}_{\mathcal{B}} = \lambda_{\mathcal{B}}^{\mathtt{KEY}} - l_{\mathcal{B}}^{\mathtt{KEY}} = 1$. The matching phase happens at the `MC` operation between $A_{\mathtt{ShC}}^{(3)}$ and $A_{\mathtt{MC}}^{(3)}$, providing $d_m = 1$ degree of matching by Equation (4).

$$A_{\mathtt{ShC}}^{(3)}[2] \oplus A_{\mathtt{ShC}}^{(3)}[10] = A_{\mathtt{MC}}^{(3)}[2] \oplus A_{\mathtt{MC}}^{(3)}[10] \tag{4}$$

According to Equation (1), the overall time complexity is $2^{4\times(32-\min\{1,1,1\})} \approx 2^{124}$. The data complexity is 2^{36} by traversing the $16 - 7 = 9$ non-constant cells in C. A detailed attack procedure is given in Algorithm 1. The memory cost is about 2^6 bytes to store $(\mathcal{S}_{\mathcal{R}}, \mathcal{S}_{\mathcal{B}}, L)$.

4.2 MitM Key Recovery Attack on 12-round Weakened Midori64

In this section, we focus on the weakened version of `Midori64` omitting the whitening layers. And we found a MitM key recovery attack on the 12-round `Midori64` as shown in Figure 8. As explained above, $|\mathcal{S}^{\mathtt{ENC}}| = 16$ independent $\boxed{0}$ bytes in the encryption data path are set as 0. The starting states are ciphertext C and two sub-key $(K^{(0)}, K^{(1)})$. In ciphertext, there are $\lambda_{\mathcal{R}}^{\mathtt{ENC}} = 12$ and $\lambda_{\mathcal{B}}^{\mathtt{ENC}} = 0$ initial DoFs for 🟥 and 🟦, respectively. And the DoFs of 🟥 are used up when computing $A_{\mathtt{ShC}}^{(10)}$ through an `MC` operation and $A_{\mathtt{MC}}^{(9)}$ through an `XOR` operation. The two sub-key $(K^{(0)}, K^{(1)})$ provide $\lambda_{\mathcal{R}}^{\mathtt{KEY}} = 6$ and $\lambda_{\mathcal{B}}^{\mathtt{KEY}} = 2$ initial DoFs for 🟥 and 🟦, respectively. For the key schedule, $K^{(0)}[0] \oplus K^{(0)}[4]$, $K^{(0)}[0] \oplus K^{(0)}[8]$,

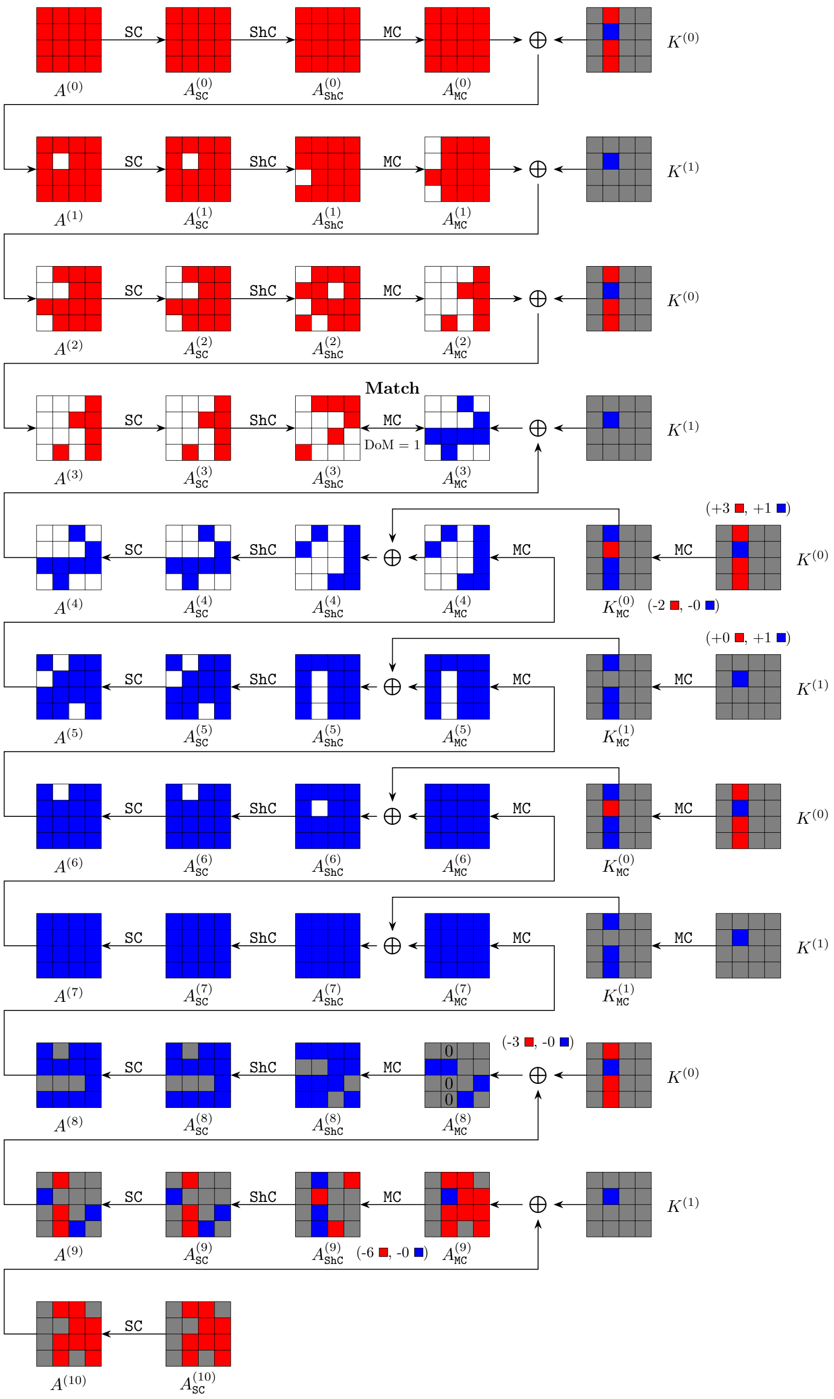


Fig. 6: Meet-in-the-Middle key recovery attack on 11-round `Midori64`

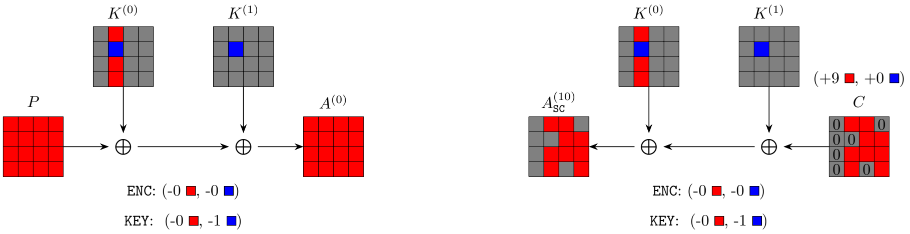


Fig. 7: The MitM characteristic through whitening layers of 11-round `Midori64`

Algorithm 1: MitM Key Recovery Attack on 11-round `Midori64`

```
Set the $\boxed{0}$ bytes to be 0, i.e., $C[0,3,4,5,8,12,14] \leftarrow 0$, $A^{(8)}_{\mathtt{MC}}[1,9,13] \leftarrow 0$
$A^{(9)}_{\mathtt{MC}}[1] \oplus A^{(9)}_{\mathtt{MC}}[9] \leftarrow 0$, $A^{(9)}_{\mathtt{MC}}[1] \oplus A^{(9)}_{\mathtt{MC}}[13] \leftarrow 0$, $A^{(9)}_{\mathtt{MC}}[2] \oplus A^{(9)}_{\mathtt{MC}}[6] \leftarrow 0$,
  $A^{(9)}_{\mathtt{MC}}[2] \oplus A^{(9)}_{\mathtt{MC}}[10] \leftarrow 0$, $A^{(9)}_{\mathtt{MC}}[7] \oplus A^{(9)}_{\mathtt{MC}}[11] \leftarrow 0$, $A^{(9)}_{\mathtt{MC}}[7] \oplus A^{(9)}_{\mathtt{MC}}[15] \leftarrow 0$
Collecting plaintext-ciphertext pairs by traversing the non-constant $16 - 7 = 9$
  cells in $C$, and storing them in table $H$
for all possible values of the ▨ cells in $K^{(0)}$ and $K^{(1)}$ do
    $A^{(10)}_{\mathtt{SC}}[0,3,4,5,8,12,14] \leftarrow (K^{(0)} \oplus K^{(1)})[0,3,4,5,8,12,14]$
    for $(\mathfrak{c}_{\mathcal{R},1}, \mathfrak{c}_{\mathcal{R},2}, \mathfrak{c}_{\mathcal{B}}) \in \mathbb{F}_2^{3\times 4}$ do
        Derive the solution space $\mathcal{S}_{\mathcal{R}}$ of 🟥 cells by
            $\begin{cases} K^{(0)}[1] \oplus K^{(0)}[9] &= \mathfrak{c}_{\mathcal{R},1} \\ K^{(0)}[1] \oplus K^{(0)}[13] &= \mathfrak{c}_{\mathcal{R},2} \end{cases}$
        Derive the solution space $\mathcal{S}_{\mathcal{B}}$ of 🟦 cells by $K^{(0)}[5] \oplus K^{(1)}[5] = \mathfrak{c}_{\mathcal{B}}$
        $L \leftarrow [\,]$
        for $v_{\mathcal{R}} \in \mathcal{S}_{\mathcal{R}}$ do
            Compute $A^{(3)}_{\mathtt{ShC}}[2,10]$ along the forward computation path:
            $A^{(8)}_{\mathtt{MC}} \rightarrow C \rightarrow Dec_K(C) \rightarrow A^{(3)}_{\mathtt{ShC}}$ by accessing $H$
            $L[A^{(3)}_{\mathtt{ShC}}[2] \oplus A^{(3)}_{\mathtt{ShC}}[10]] \leftarrow v_{\mathcal{R}}$
        end
        for $v_{\mathcal{B}} \in \mathcal{S}_{\mathcal{B}}$ do
            Compute $A^{(3)}_{\mathtt{MC}}[2,10]$ along the backward computation path:
              $C \rightarrow A^{(3)}_{\mathtt{MC}}$
            for Candidate keys in $L[A^{(3)}_{MC}[2] \oplus A^{(3)}_{MC}[10]]$ do
                Test the guessed key with several plaintext-ciphertext pairs
            end
        end
    end
end
```

$K^{(0)}[1] \oplus K^{(0)}[5]$ and $K^{(0)}[1] \oplus K^{(0)}[13]$ are restricted to constants, i.e., $l_{\mathcal{R}}^{\mathtt{KEY}} = 4$. Hence, we get $\mathrm{DoF}_{\mathcal{R}} = \lambda_{\mathcal{R}}^{\mathtt{KEY}} - l_{\mathcal{R}}^{\mathtt{KEY}} = 2$ and $\mathrm{DoF}_{\mathcal{B}} = \lambda_{\mathcal{B}}^{\mathtt{KEY}} = 2$. The matching phase happens at the `MC` operation between $A_{\mathtt{ShC}}^{(4)}$ and $A_{\mathtt{MC}}^{(4)}$, providing $d_m = 1$ degree of matching by Equation (5).

$$A_{\mathtt{ShC}}^{(4)}[4] \oplus A_{\mathtt{ShC}}^{(4)}[12] = A_{\mathtt{MC}}^{(4)}[4] \oplus A_{\mathtt{MC}}^{(4)}[12] \tag{5}$$

In [14], Fuhr *et al.* proposed the *simultaneous matching* to decrease $2^{d_{\mathcal{B}}+d_{\mathcal{R}}-d_m}$ in Equation (1) exponentially by testing the surviving keys with multiple plaintext-ciphertext pairs in parallel. Hence, the overall time is dominated by $2^{4\times(32-\min\{2,2\})} \approx 2^{120}$. The data complexity is 2^{48} by traversing the $16-4$ non-constant cells in C. A detailed attack procedure is given in Algorithm 2. The memory cost is $2^{10.6}$ bytes to store $(\mathcal{S}_{\mathcal{R}}, L)$.

When considering optimization for data complexity, we found a MitM key recovery attack on 12-round `Midori64` with data complexity of 2^{36} by relaxing the time complexity to 2^{124}. The figure and algorithm are given in Figure 17 and Algorithm 4 in Appendix C.

5 MitM Preimage Attack on Weakened Camellia

`Camellia` is a Feistel-based block cipher with 128-bit block. The diffusion layer is a 8×8 boolean matrix. In this work, we only target on the version with a 128-bit key. The detailed specification is provided in Appendix B.2.

5.1 The MitM Characteristic of 14-round weakened Camellia

We first applied the `n-XOR` model to describe the attributes propagation through the diffusion layer. However, the check model can not be deployed since the large size of the diffusion layer. We relaxed the constraint to $\min\{d_{\mathcal{B}}, d_{\mathcal{R}}, d_m\} \geq 1-i$, where $i \geq 1$, as stated in Section 3.2, and manually checked the solution files to find out valid solutions (may not be optimal).

The final valid configuration of the pseudo-preimage MitM attack on 14-round weakened `Camellia-MMO` without FL/FL^{-1} and whitening layers is shown in Figure 9. We deploy the `n-XOR` model by considering the `MixColumns` and `XOR` as a whole. The attack starts at $A^{(9)}$ and $B^{(9)}$ illustrated in Figure 9(a), in which the initial DoFs for 🟦 and 🟥 are $\lambda_{\mathcal{B}} = \lambda_{\mathcal{R}} = 7$. In the forward computation path, in order to facilitate the propagation of 🟦 cells, there are $l_{\mathcal{R}} = 6$ linear constraints imposed on $A_{\mathtt{SB}}^{(9)}[7] \oplus B^{(9)}[i]$, for $i \in \{0,1,2,4,5,6\}$. Similarly, in the backward computation path, $l_{\mathcal{B}} = 6$ linear constraints are imposed on $A_{\mathtt{SB}}^{(8)}[7] \oplus A^{(9)}[i]$, for $i \in \{0,1,2,4,5,6\}$, to facilitate the propagation of 🟥 cells. Hence, we get $d_{\mathcal{B}} = \lambda_{\mathcal{B}} - l_{\mathcal{B}} = 1$ and $d_{\mathcal{R}} = \lambda_{\mathcal{R}} - l_{\mathcal{R}} = 1$.

Around the feed-forward mechanism of `MMO` mode, we set global constraints on round keys $(k_0, k_1, k_{12}, k_{13})$ to preserve some attributes like [28]. Specifically, for the given target $H_0 \| H_1$, $A_{\mathtt{SB}}^{(0)}$ equals to $A_{\mathtt{SB}}^{(13)}$ by setting $k_0 = k_{13} \oplus H_0$ globally. Since $B^{(0)} = \mathtt{MC}(A_{\mathtt{SB}}^{(13)}) \oplus A^{(12)} \oplus H_1$ and $A^{(1)} = B^{(0)} \oplus \mathtt{MC}(A_{\mathtt{SB}}^{(0)})$, then we can get

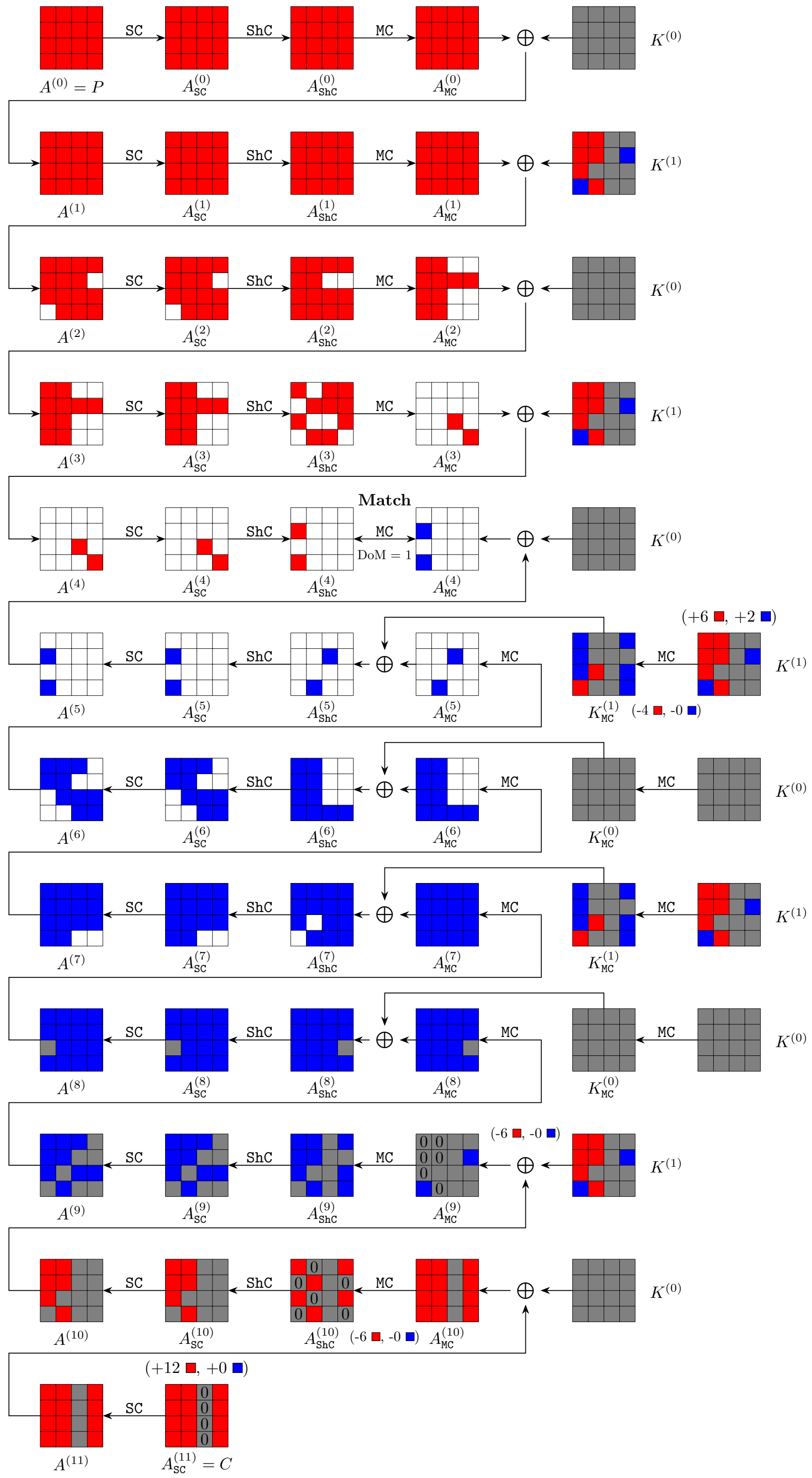


Fig. 8: Meet-in-the-Middle key recovery attack on 12-round weakened `Midori64`, optimized for time complexity

Algorithm 2: MitM Key Recovery Attack on 12-round weakened `Midori64`, optimized for time complexity

```
$C[2,6,10,14] \leftarrow 0,\ A^{(10)}_{\texttt{ShC}}[1,4,7,9,12,15] \leftarrow 0,\ A^{(9)}_{\texttt{MC}}[0,1,4,5,8,13] \leftarrow 0$
Collecting plaintext-ciphertext pairs by traversing the non-constant
  $16-4=12$ cells in $C$, and storing them in table $H$
for all possible values of the ■ cells in $K^{(0)}$ and $K^{(1)}$ do
    for $(\mathfrak{c}_{\mathcal{R},1}, \mathfrak{c}_{\mathcal{R},2}, \mathfrak{c}_{\mathcal{R},3}, \mathfrak{c}_{\mathcal{R},4}) \in \mathbb{F}_2^{4\times 4}$ do
        Derive the solution space $\mathcal{S}_{\mathcal{R}}$ of 🟥 cells by
            $\begin{cases} K^{(0)}[0] \oplus K^{(0)}[4] = \mathfrak{c}_{\mathcal{R},1} \quad K^{(0)}[0] \oplus K^{(0)}[8] = \mathfrak{c}_{\mathcal{R},2} \\ K^{(0)}[1] \oplus K^{(0)}[5] = \mathfrak{c}_{\mathcal{R},3} \quad K^{(0)}[1] \oplus K^{(0)}[13] = \mathfrak{c}_{\mathcal{R},4} \end{cases}$
        $L \leftarrow [\ ]$
        for $v_{\mathcal{R}} \in \mathcal{S}_{\mathcal{R}}$ do
            Compute $A^{(4)}_{\texttt{ShC}}[4,12]$ along the forward computation path:
            $A^{(9)}_{\texttt{MC}} \rightarrow C \rightarrow Dec_K(C) \rightarrow A^{(4)}_{\texttt{ShC}}$ by accessing $H$
            $L[A^{(4)}_{\texttt{ShC}}[4] \oplus A^{(4)}_{\texttt{ShC}}[12]] \leftarrow v_{\mathcal{R}}$
        end
        for $2^{2\times 4}$ possible values of $K^{(1)}[7,12]$ do
            Compute $A^{(4)}_{\texttt{MC}}[4,12]$ along the backward computation path:
              $C \rightarrow A^{(4)}_{\texttt{MC}}$
            for Candidate keys in $L[A^{(4)}_{\texttt{MC}}[4] \oplus A^{(4)}_{\texttt{MC}}[12]]$ do
                Test the guessed key with several plaintext-ciphertext pairs
            end
        end
    end
end
```

$A^{(1)} = A^{(12)} \oplus H_1$. Similarly, $A^{(2)}$ equals to $B^{(12)} \oplus H_0$ by setting $k_1 = k_{12} \oplus H_1$. The cost to determine such proper subkeys is given in Section 5.2 and will not exceed the time complexity of main MitM procedure.

The matching points are $A^{(5)}$ and $B^{(5)}$ in Figure 9(c). At first glance, there are no degree for the direct matching. However, after applying a linear transformation P^{-1} to $B^{(5)}$ as in Figure 10, two-byte degree of match are derived. Since $d_{\mathcal{B}} = d_{\mathcal{R}} = 1$, we only use one-byte for match, i.e., $d_m = 1$. The specific matching equation is Equation (6).

$$\bigoplus_{i\in[0,1,2,4,5,6]} B^{(3)}[i] \oplus A^{(3)}_{\texttt{SB}}[3] = \bigoplus_{i\in[0,1,2,4,5,6]} A^{(6)}[i] \oplus A^{(5)}_{\texttt{SB}}[3] \qquad (6)$$

According to Equation (1), the total time complexity is bounded by $2^{8\times(16-\min\{1,1,1\})} \approx 2^{120}$. A detailed attack procedure is given in Algorithm 3. The memory complexity of a hash table L is 2^8. And this attack can be converted to a second preimage attack with a time complexity of 2^{125} according to [24, Fact9.99].

5.2 The Cost to Determine a Proper Key

The key schedule of `Camellia` with 128-bit key is shown in Figure 15. As explained above, we only need to focus on $(k_0, k_1, k_{12}, k_{13})$ [1],

$$k_0 \leftarrow K'_A,\ \ k_1 \leftarrow K''_A,\ \ k_{12} \leftarrow K''[30{-}63]\|K'[0{-}29],\ \ k_{13} \leftarrow K'[30{-}63]\|K''[0{-}29].$$

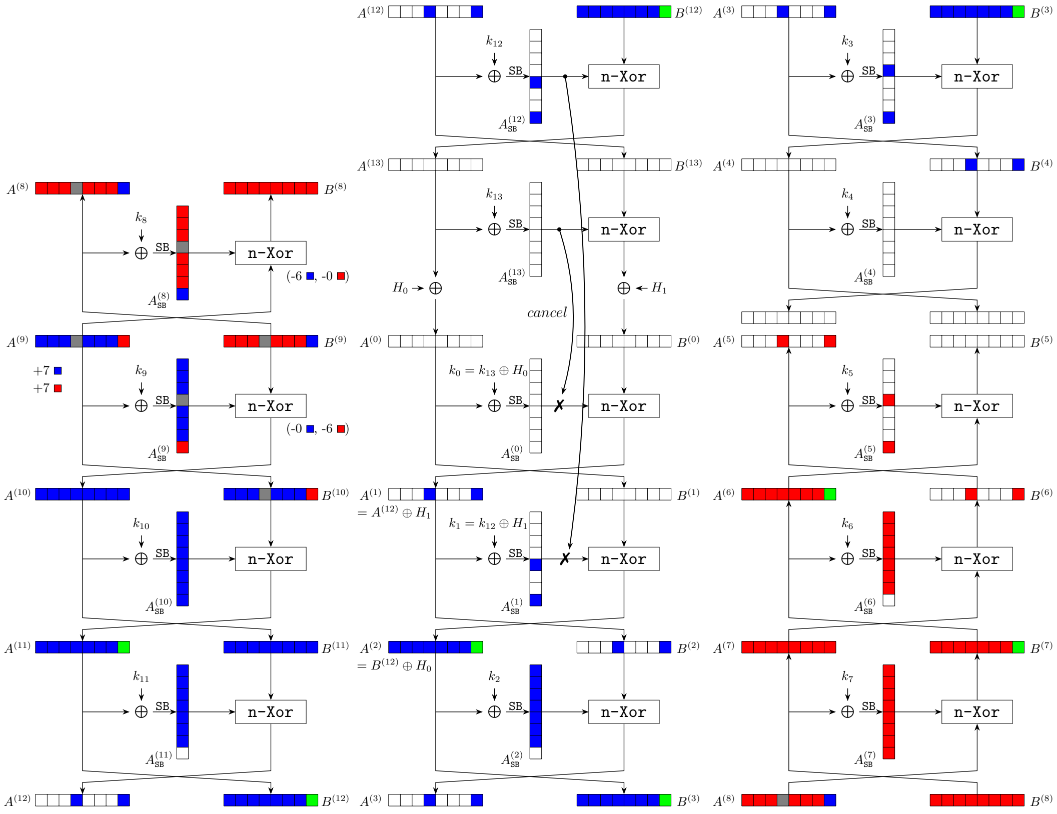
As shown in Figure 15, every internal state can be derived for given K' and S_0. Hence, we get $K'' = F_0(K') \oplus S_0$ and $K''_A = F_2(F_1(S_0)) \oplus F_0(K')$. According to the global constraints $k_0 = k_{13} \oplus H_0$ and $k_1 = k_{12} \oplus H_1$, the relation between K' and S_0 can be represented as Equation (7).

$$F_2(F_1(S_0)) \oplus F_0(K') = \big(F_0(K') \oplus S_0\big)\,[30-63]\|K'[0-29] \oplus H_1 \qquad (7)$$

Besides, we note that K' and S_0 can be placed at two sides of Equation (8), respectively. The left-hand-side of Equation (8) only contains variables in terms of K', while the right-hand-side of Equation (8) depends on S_0.

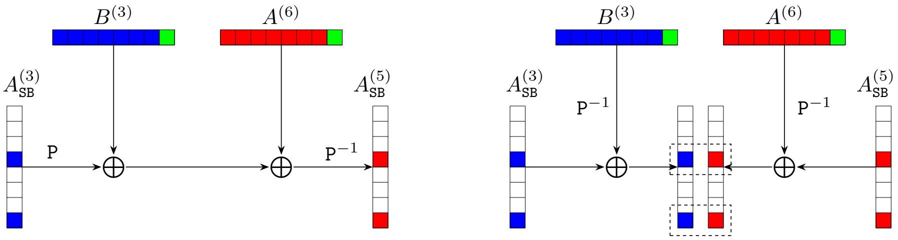
$$F_0(K') \oplus F_0(K')[30{-}63]\|K'[0{-}29] \ = \ F_2(F_1(S_0)) \oplus S_0[30{-}63]\|\overbrace{0\cdots0}^{30} \oplus H_1 \quad (8)$$

Then, an algebraic meet-in-the-middle attack can be mounted by enumerating K' and S_0 independently to filter out valid pairs according to Equation (8), i.e. $d_{\mathcal{B}} = d_{\mathcal{R}} = d_m = 64$. The time and memory complexity are both 2^{64}. Besides, the memory cost can be further reduced by extracting partial x bits of K' and S_0 as global variables. Then, the memory can be reduced by a fraction of 2^x, while the total time is bounded by 2^{64+x}. To avoid exceeding the time cost of main MitM procedure, $64 + x \leq 120$ should be fulfilled, i.e., x can take 56 at most. The corresponding memory cost is 2^8.



(a) Initial structure (b) 4-round shrink and link (c) Matching point

Fig. 9: Meet-in-the-Middle pseudo-preimage attack on 14-round weakened Camellia-MMO



(a) Original (b) Transformed

Fig. 10: The matching process of 14-round weakened Camellia-MMO

Algorithm 3: MitM Pseudo-Preimage Attack on 14-round weakened `Camellia-MMO`

```
Setting a global key satisfying $k_0 = k_{13} \oplus H_0$, $k_1 = k_{12} \oplus H_1$;
for $2^{16}$ values of the ■ bytes in $A^{(9)}[3] \| B^{(9)}[3]$ do
    for $\mathfrak{c}_{\mathcal{B}} \in \mathbb{F}_2^{8\times 6}$ do
        for $\mathfrak{c}_{\mathcal{R}} \in \mathbb{F}_2^{8\times 6}$ do
            $L \leftarrow [\ ]$
            Solve the following system of equations to find the solution space
                $\mathcal{S}_{\mathcal{B}}$ of 🟦 in $A^{(9)}$ and $B^{(9)}$; /* $|\mathcal{S}_{\mathcal{B}}| = 2^{8\times(7-6)} = 2^8$ */
                $A^{(8)}_{\mathtt{SB}}[7] \oplus A^{(9)}[0] = \mathfrak{c}_{\mathcal{B}}[0]$, $A^{(8)}_{\mathtt{SB}}[7] \oplus A^{(9)}[1] = \mathfrak{c}_{\mathcal{B}}[1]$, $A^{(8)}_{\mathtt{SB}}[7] \oplus A^{(9)}[2] = \mathfrak{c}_{\mathcal{B}}[2]$,
                $A^{(8)}_{\mathtt{SB}}[7] \oplus A^{(9)}[4] = \mathfrak{c}_{\mathcal{B}}[3]$, $A^{(8)}_{\mathtt{SB}}[7] \oplus A^{(9)}[5] = \mathfrak{c}_{\mathcal{B}}[4]$, $A^{(8)}_{\mathtt{SB}}[7] \oplus A^{(9)}[6] = \mathfrak{c}_{\mathcal{B}}[5]$.
            Solve the following system of equations to find the solution space
                $\mathcal{S}_{\mathcal{R}}$ of 🟥 in $A^{(9)}$ and $B^{(9)}$; /* $|\mathcal{S}_{\mathcal{B}}| = 2^{8\times(7-6)} = 2^8$ */
                $A^{(9)}_{\mathtt{SB}}[7] \oplus B^{(9)}[0] = \mathfrak{c}_{\mathcal{R}}[0]$, $B^{(9)}_{\mathtt{SB}}[7] \oplus A^{(9)}[1] = \mathfrak{c}_{\mathcal{R}}[1]$, $A^{(9)}_{\mathtt{SB}}[7] \oplus B^{(9)}[2] = \mathfrak{c}_{\mathcal{R}}[2]$,
                $A^{(9)}_{\mathtt{SB}}[7] \oplus B^{(9)}[4] = \mathfrak{c}_{\mathcal{R}}[3]$, $A^{(9)}_{\mathtt{SB}}[7] \oplus B^{(9)}[5] = \mathfrak{c}_{\mathcal{R}}[4]$, $A^{(9)}_{\mathtt{SB}}[7] \oplus B^{(9)}[6] = \mathfrak{c}_{\mathcal{R}}[5]$.
            for $v_{\mathcal{B}} \in \mathcal{S}_{\mathcal{B}}$ do
                Compute forward to $A^{(3)}$ and $B^{(3)}$, derive 1-byte $End_{\mathcal{B}}$ by
                    $End_{\mathcal{B}} \leftarrow P^{-1}\left(B^{(3)}\right)[3] \oplus A^{(3)}_{\mathtt{SB}}[3]$
                $L[End_{\mathcal{B}}] \leftarrow v_{\mathcal{B}}$;
            end
            for $v_{\mathcal{R}} \in \mathcal{S}_{\mathcal{R}}$ do
                Compute backward to $A^{(6)}$ and $B^{(6)}$, derive 1-byte $End_{\mathcal{R}}$ by
                    $End_{\mathcal{R}} \leftarrow P^{-1}\left(A^{(6)}\right)[3] \oplus A^{(5)}_{\mathtt{SB}}[3]$
                for $v_{\mathcal{B}} \in L[End_{\mathcal{R}}]$ do
                    Reconstruct the (candidate) message $X$;
                        /* $2^{8\times(1+1-1)} = 2^8$ values passed the filter */
                    if $X$ is a preimage then
                        Output $X$ and stop;
                    end
                end
            end
        end
    end
end
```

6 MitM Preimage Attack on 6-Round Aria

`Aria` is an SPN-based block cipher that supports a 128-bit block. In this work, we target on the version with a 128-bit key. The state is treated as a 4×4 matrix. And the diffusion layer is a 16×16 boolean matrix. The detailed specification of `Aria` is presented in Appendix B.3.

Since the large size diffusion layer, only the `n-XOR` model can be applied to describe the MitM attribution propagation through the diffusion layer. By relaxing the constraint to $\min\{d_{\mathcal{B}}, d_{\mathcal{R}}, d_m\} \geq 1 - i$, where $i \geq 1$, as stated in Section 3.2, we finally found out a valid configuration of the pseudo-preimage MitM attack on 6-round `Aria-DM` as shown in Figure 11 (may not be optimal). The attack starts at $A^{(1)}$ in which the initial DoFs for 🟦 and 🟥 are $\lambda_{\mathcal{B}} = 1, \lambda_{\mathcal{R}} = 14$, respectively. Since there are non-linear constraints on 🟥 cells to compute $A^{(2)}_{\mathtt{DL}}$ through the `DL` operation. We use the table-based method in [13] to solve such non-linear constraints.

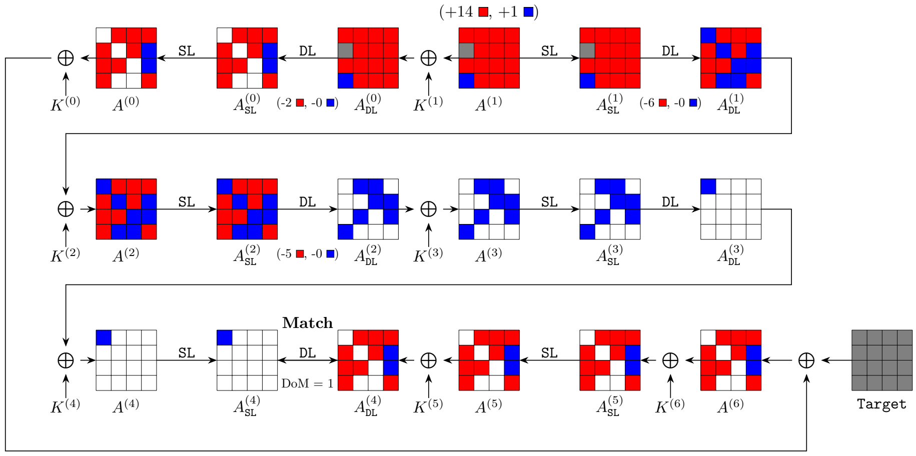


Fig. 11: Meet-in-the-Middle pseudo-preimage attack on 6-round `Aria-DM`

Precomputation of red initial values. By enumerating the 🟥 cells in $A^{(1)}$, in the backward computation path, two constraints imposed on 🟥 cells can be computed as follows:

$$\begin{cases} A^{(0)}_{\mathtt{DL}}[0] \oplus A^{(0)}_{\mathtt{DL}}[6] \oplus A^{(0)}_{\mathtt{DL}}[7] \oplus A^{(0)}_{\mathtt{DL}}[8] \oplus A^{(0)}_{\mathtt{DL}}[10] \oplus A^{(0)}_{\mathtt{DL}}[13] = \mathfrak{c}[0] \\ A^{(0)}_{\mathtt{DL}}[0] \oplus A^{(0)}_{\mathtt{DL}}[4] \oplus A^{(0)}_{\mathtt{DL}}[5] \oplus A^{(0)}_{\mathtt{DL}}[9] \oplus A^{(0)}_{\mathtt{DL}}[11] \oplus A^{(0)}_{\mathtt{DL}}[14] = \mathfrak{c}[1] \end{cases}$$

In the forward computation path, there are 11 constraints imposed on the 🟥 cells. During the `DL` operation in the 2nd round, 6 constraints are imposed on the 🟥 cells. The specific expression of the constraints is shown in as follows:

$$\begin{cases}
A_{\mathtt{SL}}^{(1)}[4] \oplus A_{\mathtt{SL}}^{(1)}[6] \oplus A_{\mathtt{SL}}^{(1)}[8] \oplus A_{\mathtt{SL}}^{(1)}[9] \oplus A_{\mathtt{SL}}^{(1)}[13] \oplus A_{\mathtt{SL}}^{(1)}[14] = \mathfrak{c}[2] \\
A_{\mathtt{SL}}^{(1)}[4] \oplus A_{\mathtt{SL}}^{(1)}[9] \oplus A_{\mathtt{SL}}^{(1)}[10] \oplus A_{\mathtt{SL}}^{(1)}[14] \oplus A_{\mathtt{SL}}^{(1)}[15] = \mathfrak{c}[3] \\
A_{\mathtt{SL}}^{(1)}[2] \oplus A_{\mathtt{SL}}^{(1)}[5] \oplus A_{\mathtt{SL}}^{(1)}[6] \oplus A_{\mathtt{SL}}^{(1)}[8] \oplus A_{\mathtt{SL}}^{(1)}[13] \oplus A_{\mathtt{SL}}^{(1)}[15] = \mathfrak{c}[4] \\
A_{\mathtt{SL}}^{(1)}[0] \oplus A_{\mathtt{SL}}^{(1)}[6] \oplus A_{\mathtt{SL}}^{(1)}[7] \oplus A_{\mathtt{SL}}^{(1)}[8] \oplus A_{\mathtt{SL}}^{(1)}[10] \oplus A_{\mathtt{SL}}^{(1)}[13] = \mathfrak{c}[5] \\
A_{\mathtt{SL}}^{(1)}[5] \oplus A_{\mathtt{SL}}^{(1)}[7] \oplus A_{\mathtt{SL}}^{(1)}[10] \oplus A_{\mathtt{SL}}^{(1)}[11] = \mathfrak{c}[6] \\
A_{\mathtt{SL}}^{(1)}[10] \oplus A_{\mathtt{SL}}^{(1)}[11] \oplus A_{\mathtt{SL}}^{(1)}[12] \oplus A_{\mathtt{SL}}^{(1)}[15] = \mathfrak{c}[7]
\end{cases}$$

Based on the above 6 constraints $\left(\mathfrak{c}[2], \mathfrak{c}[3], \mathfrak{c}[4], \mathfrak{c}[5], \mathfrak{c}[6], \mathfrak{c}[7]\right)$, the effect of the 🟥 cells on the 7 cells $A_{\mathtt{DL}}^{(1)}[0, 5, 7, 10, 11, 13, 14]$ can be cancelled as follows:

$$\begin{cases}
A_{\mathtt{SL}}^{(1)}[4] \oplus A_{\mathtt{SL}}^{(1)}[6] \oplus A_{\mathtt{SL}}^{(1)}[8] \oplus A_{\mathtt{SL}}^{(1)}[9] \oplus A_{\mathtt{SL}}^{(1)}[13] \oplus A_{\mathtt{SL}}^{(1)}[14] = \mathfrak{c}[2] \\
A_{\mathtt{SL}}^{(1)}[4] \oplus A_{\mathtt{SL}}^{(1)}[9] \oplus A_{\mathtt{SL}}^{(1)}[10] \oplus A_{\mathtt{SL}}^{(1)}[14] \oplus A_{\mathtt{SL}}^{(1)}[15] = \mathfrak{c}[3] \\
A_{\mathtt{SL}}^{(1)}[6] \oplus A_{\mathtt{SL}}^{(1)}[8] \oplus A_{\mathtt{SL}}^{(1)}[11] \oplus A_{\mathtt{SL}}^{(1)}[12] \oplus A_{\mathtt{SL}}^{(1)}[13] = \mathfrak{c}[2] \oplus \mathfrak{c}[3] \oplus \mathfrak{c}[7] \\
A_{\mathtt{SL}}^{(1)}[2] \oplus A_{\mathtt{SL}}^{(1)}[5] \oplus A_{\mathtt{SL}}^{(1)}[6] \oplus A_{\mathtt{SL}}^{(1)}[8] \oplus A_{\mathtt{SL}}^{(1)}[13] \oplus A_{\mathtt{SL}}^{(1)}[15] = \mathfrak{c}[4] \\
A_{\mathtt{SL}}^{(1)}[2] \oplus A_{\mathtt{SL}}^{(1)}[4] \oplus A_{\mathtt{SL}}^{(1)}[7] \oplus A_{\mathtt{SL}}^{(1)}[9] \oplus A_{\mathtt{SL}}^{(1)}[12] \oplus A_{\mathtt{SL}}^{(1)}[14] = \mathfrak{c}[2] \oplus \mathfrak{c}[4] \oplus \mathfrak{c}[6] \oplus \mathfrak{c}[7] \\
A_{\mathtt{SL}}^{(1)}[0] \oplus A_{\mathtt{SL}}^{(1)}[6] \oplus A_{\mathtt{SL}}^{(1)}[7] \oplus A_{\mathtt{SL}}^{(1)}[8] \oplus A_{\mathtt{SL}}^{(1)}[10] \oplus A_{\mathtt{SL}}^{(1)}[13] = \mathfrak{c}[5] \\
A_{\mathtt{SL}}^{(1)}[0] \oplus A_{\mathtt{SL}}^{(1)}[4] \oplus A_{\mathtt{SL}}^{(1)}[5] \oplus A_{\mathtt{SL}}^{(1)}[9] \oplus A_{\mathtt{SL}}^{(1)}[11] \oplus A_{\mathtt{SL}}^{(1)}[14] = \mathfrak{c}[2] \oplus \mathfrak{c}[5] \oplus \mathfrak{c}[6]
\end{cases}$$

In a similar way, the 5 constraints $\left(\mathfrak{c}[8], \mathfrak{c}[9], \mathfrak{c}[10], \mathfrak{c}[11], \mathfrak{c}[12]\right)$ imposed on the 🟥 cells through the DL in the 3rd round are enough to cancel the effect of the 🟥 cells on the 6 cells $A_{\mathtt{DL}}^{(2)}[4, 6, 8, 9, 13, 14]$. For the specific expression of the constraints, please refer to Algorithm 5 in Appendix C. And the detailed DoFs consumption process is illustrated as follows:

$$\begin{cases}
A_{\mathtt{SL}}^{(2)}[2] \oplus A_{\mathtt{SL}}^{(2)}[8] \oplus A_{\mathtt{SL}}^{(2)}[15] = \mathfrak{c}[8] \\
A_{\mathtt{SL}}^{(2)}[2] \oplus A_{\mathtt{SL}}^{(2)}[9] \oplus A_{\mathtt{SL}}^{(2)}[12] = \mathfrak{c}[8] \oplus \mathfrak{c}[12] \\
A_{\mathtt{SL}}^{(2)}[1] \oplus A_{\mathtt{SL}}^{(2)}[4] \oplus A_{\mathtt{SL}}^{(2)}[15] = \mathfrak{c}[9] \\
A_{\mathtt{SL}}^{(2)}[1] \oplus A_{\mathtt{SL}}^{(2)}[6] \oplus A_{\mathtt{SL}}^{(2)}[12] = \mathfrak{c}[9] \oplus \mathfrak{c}[11] \\
A_{\mathtt{SL}}^{(2)}[3] \oplus A_{\mathtt{SL}}^{(2)}[6] \oplus A_{\mathtt{SL}}^{(2)}[8] = \mathfrak{c}[10] \\
A_{\mathtt{SL}}^{(2)}[3] \oplus A_{\mathtt{SL}}^{(2)}[4] \oplus A_{\mathtt{SL}}^{(2)}[9] = \mathfrak{c}[10] \oplus \mathfrak{c}[11] \oplus \mathfrak{c}[12]
\end{cases}$$

In summary, the values of $l_{\mathcal{R}} = 13$ constraints can be determined for given values of $\lambda_{\mathcal{R}} = 14$ 🟥 cells in $A^{(1)}$. Hence, we get $d_{\mathcal{B}} = 1$, $d_{\mathcal{R}} = \lambda_{\mathcal{R}} - l_{\mathcal{R}} = 1$.

Matching process. The matching points are $A_{\mathtt{SL}}^{(4)}, A_{\mathtt{DL}}^{(4)}$, indirect matching through the DL provides one-byte match, i.e., DoM $= 1$. The specific matching process is Equation (9).

$$A_{\mathtt{SL}}^{(4)}[0] \oplus A_{\mathtt{DL}}^{(4)}[13] \oplus A_{\mathtt{DL}}^{(4)}[14] = A_{\mathtt{DL}}^{(4)}[3] \oplus A_{\mathtt{DL}}^{(4)}[4] \oplus A_{\mathtt{DL}}^{(4)}[6] \oplus A_{\mathtt{DL}}^{(4)}[8] \oplus A_{\mathtt{DL}}^{(4)}[9] \tag{9}$$

Based on the above MitM framework, combined with the table-based technique for solving nonlinear constrained neutral words [13], Algorithm 5 gives a detailed attack procedure in Appendix C.

Complexity. The nonlinear constraints imposed on 🟥 cells are solved in Lines 2-8 of Algorithm 5. That is, 14 🟥 cells of $A^{(1)}[0, 2, 4\text{-}15]$ are traversed to compute the exact values of $\mathfrak{c}_{\mathcal{R}}[0\text{-}12]$. Then, the values of $A^{(1)}[0, 2, 4\text{-}15]$ are stored in a hash table V under the index of $\mathfrak{c}_{\mathcal{R}}[0\text{-}12]$. Hence, the time complexity of the precomputation phase is $2^{8\times 14} = 2^{112}$. The memory complexity is also 2^{112} to store table V.

Lines 10-24 of Algorithm 5 stand for one MitM episode. With the parameters $(d_{\mathcal{B}}, d_{\mathcal{R}}, d_m) = (1, 1, 1)$, there are a total of $2^{8\times(1+1-1)} = 2^8$ solutions that can be filtered out according to Equation (9). In order to find a full match of 128-bit, it's expected to repeat $2^{120-8} = 2^{112}$ MitM episodes. By traversing the ◼ in $A^{(1)}$ at the outer loop and enumerating the 13 constraints imposed on 🟥 cells, it is sufficient to find a full match. According to Equation (1), The total time complexity of the attack phase is

$$2^8 \times 2^{112} + 2^{8\times(16-\min\{1,1,1\})} \approx 2^{120}.$$

The memory complexity is dominated by the table V of 2^{112}. And this attack can be converted to a preimage attack with a time complexity of 2^{125} according to [24, Fact9.99].

7 Conclusion

In this paper, we propose the `n-XOR` model to simulate the `XOR` operation with an arbitrary number of input cells. Specifically, the size of `n-XOR` model is independent of the number of input cells, and thus it is well suitable for primitives with a binary matrix as the diffusion layer. To eliminate the subtle inaccuracies caused by `n-XOR` model, we introduce another check model to determine the exact DoFs consumption of MitM attributes propagation. However, the size of the check model is still limited by the number of input cells n and does not work well when $n > 4$ in this paper. We expect that there will be more elegant and efficient techniques to overcome this defect and we leave this as an open problem.

We apply the above two new models to a MitM key recovery attack on 11-round `Midori64` with low data and memory. Besides, when omitting the whitening layers, two 12-round MitM characteristics for key recovery attack are found for optimizing time and data, respectively. For hash functions, we obtain improved preimage attack on 14-round weakened `Camellia-MMO` and 6-round `Aria-DM`. Both attacks are improved by 1 round compared to previous best records.

References

1. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit block cipher suitable for multiple platforms — design and-

analysis. In: Selected Areas in Cryptography. pp. 39–56. Springer Berlin Heidelberg (2001). https://doi.org/10.1007/3-540-44983-3_4

2. Aoki, K., Sasaki, Y.: Preimage attacks on one-block md4, 63-step md5 and more. In: Selected Areas in Cryptography. pp. 103–119. Springer Berlin Heidelberg (2009). https://doi.org/10.1007/978-3-642-04159-4_7
3. Aumasson, J., Meier, W., Mendel, F.: Preimage attacks on 3-pass HAVAL and step-reduced MD5. In: SAC. vol. 5381, pp. 120–135 (2008). https://doi.org/10.1007/978-3-642-04159-4_8
4. Baek, S., Kim, J.: Quantum rebound attacks on reduced-round ARIA-based hash functions. Cryptology ePrint Archive, Paper 2022/1604 (2022), https://eprint.iacr.org/2022/1604
5. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: ASIACRYPT 2015. pp. 411–436. Springer Berlin Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_17
6. Bao, Z., Dong, X., Guo, J., Li, Z., Shi, D., Sun, S., Wang, X.: Automatic search of meet-in-the-middle preimage attacks on aes-like hashing. In: EUROCRYPT 2021. pp. 771–804. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-77870-5_27
7. Bao, Z., Guo, J., Shi, D., Tu, Y.: Superposition meet-in-the-middle attacks: updates on fundamental security of aes-like hashing. In: Annual International Cryptology Conference. pp. 64–93. Springer (2022). https://doi.org/10.1007/978-3-031-15802-5_3
8. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full aes. In: Advances in Cryptology – ASIACRYPT 2011. pp. 344–371. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_19
9. Bogdanov, A., Rechberger, C.: A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher ktantan. In: SAC. pp. 229–240. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-19574-7_16
10. Chen, S., Guo, J., List, E., Shi, D., Zhang, T.: Diving deep into the preimage security of aes-like hashing. Cryptology ePrint Archive, Paper 2024/300 (2024), https://eprint.iacr.org/2024/300
11. Degré, M., Derbez, P., Lahaye, L., Schrottenloher, A.: New models for the cryptanalysis of ascon. Cryptology ePrint Archive, Paper 2024/298 (2024), https://eprint.iacr.org/2024/298
12. Diffie, W., Hellman, M.E.: Special feature exhaustive cryptanalysis of the NBS data encryption standard. Computer **10**(6), 74–84 (1977). https://doi.org/10.1109/C-M.1977.217750
13. Dong, X., Hua, J., Sun, S., Li, Z., Wang, X., Hu, L.: Meet-in-the-middle attacks revisited: Key-recovery, collision, and preimage attacks. In: CRYPTO 2021. pp. 278–308. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-84252-9_10
14. Fuhr, T., Minaud, B.: Match box meet-in-the-middle attack against KATAN. In: FSE 2014. pp. 61–81 (2014). https://doi.org/10.1007/978-3-662-46706-0_4
15. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023), https://www.gurobi.com
16. Hong, D., Koo, B., Kim, D.C.: Preimage and second-preimage attacks on pgv hashing modes of round-reduced aria, camellia, and serpent. IEICE T Fund Electr **95**(1), 372–380 (2012), https://api.semanticscholar.org/CorpusID:19830401

17. Hou, Q., Dong, X., Qin, L., Zhang, G., Wang, X.: Automated meet-in-the-middle attack goes to feistel. In: ASIACRYPT 2023. pp. 370–404. Springer Nature Singapore (2023). https://doi.org/10.1007/978-981-99-8727-6_13
18. Isobe, T.: A single-key attack on the full gost block cipher. Journal of cryptology **26**, 172–189 (2013). https://doi.org/10.1007/s00145-012-9118-5
19. ISO/IEC: 10118-2:2010 Information technology — Security techniques - Hash-functions - Part 2: Hash-functions using an n-bit block cipher, 3rd edn (2010)
20. ISO/IEC 18033-3:2010 Information technology-Security techniques-EncryptionAlgorithms-Part 3: Block ciphers (2010)
21. Kwon, D., Kim, J., Park, S., Sung, S.H., Sohn, Y., Song, J.H., Yeom, Y., Yoon, E.J., Lee, S., Lee, J., et al.: New block cipher: Aria. In: Inscrypt. pp. 432–445. Springer (2003). https://doi.org/10.1007/978-3-540-24691-6_32
22. Lin, L., Wu, W.: Meet-in-the-middle attacks on reduced-round midori64. IACR ToSC pp. 215–239 (2017). https://doi.org/10.13154/tosc.v2017.i1.215-239
23. Liu, Y., Xiang, Z., Chen, S., Zhang, S., Zeng, X.: A novel automatic technique based on milp to search for impossible differentials. In: ACNS. pp. 119–148. Springer Nature Switzerland (2023). https://doi.org/10.1007/978-3-031-33488-7_5
24. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. CRC Press, Inc., USA, 1st edn. (1996)
25. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: a synthetic approach. In: Advances in Cryptology — CRYPTO' 93. pp. 368–378. Springer Berlin Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_31
26. Sasaki, Y.: Meet-in-the-middle preimage attacks on aes hashing modes and an application to whirlpool. In: FSE. pp. 378–396. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-21702-9_22
27. Sasaki, Y.: Preimage attacks on feistel-sp functions: Impact of omitting the last network twist. In: ACNS. pp. 170–185. Springer Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-38980-1_11
28. Sasaki, Y.: Preimage attacks on feistel-sp functions: Impact of omitting the last network twist. IEICE T Fund Electr **98**(1), 61–71 (2015). https://doi.org/10.1587/transfun.E98.A.61
29. Sasaki, Y.: Integer linear programming for three-subset meet-in-the-middle attacks: Application to GIFT. In: IWSEC 2018. vol. 11049, pp. 227–243 (2018). https://doi.org/10.1007/978-3-319-97916-8_15
30. Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In: EUROCRYPT 2009, Proceedings. vol. 5479, pp. 134–152. Springer (2009). https://doi.org/10.1007/978-3-642-01001-9_8
31. Sasaki, Y., Emami, S., Hong, D., Kumar, A.: Improved known-key distinguishers on feistel-sp ciphers and application to camellia. In: Information Security and Privacy. pp. 87–100. Springer Berlin Heidelberg (2012). https://doi.org/10.1007/978-3-642-31448-3_7
32. Sasaki, Y., Wang, L., Sakai, Y., Sakiyama, K., Ohta, K.: Three-subset meet-in-the-middle attack on reduced xtea. In: AFRICACRYPT 2012. pp. 138–154. Springer Berlin Heidelberg (2012). https://doi.org/10.1007/978-3-642-31410-0_9
33. Schrottenloher, A., Stevens, M.: Simplified mitm modeling for permutations: New (quantum) attacks. In: CRYPTO 2022. pp. 717–747. Springer Nature Switzerland (2022). https://doi.org/10.1007/978-3-031-15982-4_24
34. Schrottenloher, A., Stevens, M.: Simplified modeling of mitm attacks for block ciphers: New (quantum) attacks. IACR Transactions on Symmetric Cryptology **2023**, 146–183 (2023). https://doi.org/10.46586/tosc.v2023.i3.146-183

35. Shahmirzadi, A.R., Azimi, S.A., Salmasizadeh, M., Mohajeri, J., Aref, M.R.: Impossible differential cryptanalysis of reduced-round midori64 block cipher. In: ISCISC. pp. 99–104 (Sep 2017). https://doi.org/10.1109/ISCISC.2017.8488362

36. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In: ASIACRYPT 2014. pp. 158–178 (2014). https://doi.org/10.1007/978-3-662-45611-8_9

37. Wei, L., Rechberger, C., Guo, J., Wu, H., Wang, H., Ling, S.: Improved meet-in-the-middle cryptanalysis of ktantan (poster). In: Information Security and Privacy. pp. 433–438. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-22497-3_31

A Details of MILP Models for MitM Attack

In this section, we briefly recall the MILP model for MC and XOR operation of AES in [6].

The MC. The rules of the MC are formalized in two different directions in [6]. Taking the forward computation as an example, the set of rules is given as follows:

1. If there is at least one □ in the input column, all the outputs are □;
2. If there are 🟦 but no □ and 🟥 in the input column, then all the outputs are 🟦;
3. If all the inputs are ▨, then all the outputs are ▨;
4. If there are 🟥 and 🟦 but no □ in the input column, each output must be 🟦 or □. Moreover, the sum of the numbers of 🟦 and ▨ in the input and output columns must be no more than 3;
5. If there are 🟥 but no □ and 🟦 in the input column, then each output must be 🟥 or ▨. Moreover, the number of ▨ in the input and output columns must be no more than 3.

Some examples of valid coloring schemes of the MC-RULE in the forward computation are shown in Figure 12.

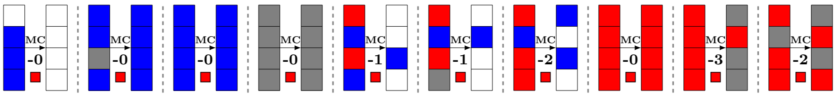
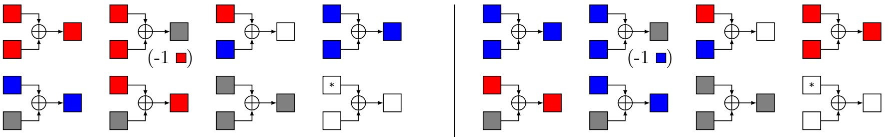


Fig. 12: Some valid coloring schemes for MC in forward computation in [6]

Let $(\alpha[0], \alpha[1], \alpha[2], \alpha[3])^T$ and $(\beta[0], \beta[1], \beta[2], \beta[3])^T$ be the input and output columns. In [6], Bao *et al.* use three 0-1 indicator variables μ, υ, ω for the input column to fulfill different rules auxiliary. Let $\mu = 1$ if and only if there exists $i \in \{0,1,2,3\}$ such that $(x_i^\alpha, y_i^\alpha) = (0,0)$. Let $\upsilon = 1$ if and only if $x_i^\alpha = 1$ for each $i \in \{0,1,2,3\}$. Let $\omega = 1$ if and only if $y_i^{\alpha} = 1$ for each $i \in \{0,1,2,3\}$. Then, with the help of μ, υ, ω, the MC-RULE in the forward computation can be described as a system of inequalities:

$$\begin{cases} \sum_{i=0}^{3} x_i^{\alpha} - 4\upsilon \geq 0; \\ \sum_{i=0}^{3} x_i^{\alpha} - \upsilon \leq 3. \end{cases} \Bigg| \begin{cases} \sum_{i=0}^{3} x_i^{\beta} + 4\mu \leq 4; \\ \sum_{i=0}^{3} y_i^{\beta} + 4\mu \leq 4; \\ \sum_{i=0}^{3} y_i^{\beta} - 4\omega = 0; \end{cases} \begin{cases} \sum_{i=0}^{3} (x_i^{\alpha} + x_i^{\beta}) - 5\upsilon \leq 3; \\ \sum_{i=0}^{3} (x_i^{\alpha} + x_i^{\beta}) - 8\upsilon \geq 0. \end{cases}$$

The XOR. For the XOR operation in two different directions, the coloring schemes of the input and output cells are shown in Figure 13.



(a) For the forward computation

(b) For the backward computation

Fig. 13: The XOR in [6], where a "*" means that the cell can be any color

Let $\alpha[i]$, $\beta[i]$ denote the input cells and $\gamma[i]$ denote the output cell, where $0 \leq i \leq 15$. Let a boolean variable d_i indicate the consumption of DoF, where $d_i = 1$ means that one DoF is consumed to let the corresponding output be ■. The set of rules restrict $(x_i^{\alpha}, y_i^{\alpha}, x_i^{\beta}, y_i^{\beta}, x_i^{\gamma}, y_i^{\gamma}, d_i)$ to a subset of $\mathbb{F}_2^7$, which can be described by a system of linear inequalities with the convex hull technique in [36].

B Descriptions of Midori, Camellia and Aria

B.1 Specification of Midori

Midori is a family of SPN-based lightweight block cipher designed by Banik *et al.* at ASIACRYPT 2015 [5]. With its low energy consumption, it is suitable for deployment in edge gateways and end devices to facilitate blockchain on-chain and off-chain interactions. Two versions of Midori use a 64-bit and a 128-bit internal state, respectively. In this work, we focus on the 64-bit version denoted by Midori64. The internal state of Midori64 can be represented as a 4×4 array as shown in Figure 14. Midori64 is of 16 iterated rounds and each round function consists of four operations:

- `SubCell (SC)`: Apply the 4-bit non-linear involution S-box on each nibble.
- `ShuffleCell (ShC)`: Update the position of each nibble by a pre-defined permutation.
- `MixColumn (MC)`: Each column is left multiplied by a 4×4 binary matrix M as follows.

$$M = \begin{pmatrix} 0\,1\,1\,1 \\ 1\,0\,1\,1 \\ 1\,1\,0\,1 \\ 1\,1\,1\,0 \end{pmatrix}.$$

- `KeyAdd (KA)`: A round key is `XOR`ed to the internal state.

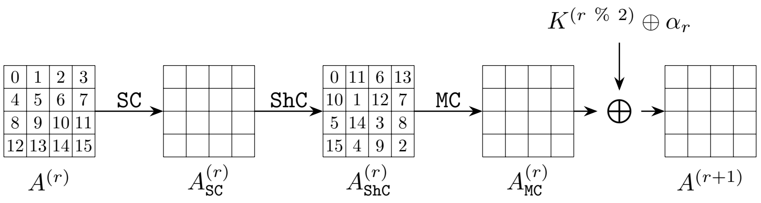


Fig. 14: One full round function of `Midori64`

For the last round, the operations `ShC`, `MC` and `KA` are omitted. Two sub-keys $K^{(0)} \| K^{(1)}$ are derived from the 128-bit master key K and the round keys are generated by $K^{(r\%2)} \oplus \alpha_r$ alternatively, where $0 \leq r \leq 14$ and α_r is a round constant. Besides, additional `KA` operations are applied with a whitening key $WK = K^{(0)} \oplus K^{(1)}$ before the first round and after the last round.

B.2 Specification of Camellia

`Camellia` is a Feistel-based block cipher designed by NTT and Mitsubishi Electric Corporation [1] and has been specified in ISO/IEC 18033-3:2010 [20]. This work only targets on the weakened version of `Camellia` with 128 bits block and key size, where the FL/FL^{-1} transformations and whitening layers are omitted. The iterated round function consists of `AddRoundKey` (`AK`), `SubBytes` (`SB`) and `MixColumns` (`MC`) as shown in Figure 15. The linear layer of `MC` is a 8×8 binary matrix described as follows.

$$P = \begin{pmatrix} 1\,0\,1\,1\,0\,1\,1\,1 \\ 1\,1\,0\,1\,1\,0\,1\,1 \\ 1\,1\,1\,0\,1\,1\,0\,1 \\ 0\,1\,1\,1\,1\,1\,1\,0 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 0\,1\,1\,0\,1\,0\,1\,1 \\ 0\,0\,1\,1\,1\,1\,0\,1 \\ 1\,0\,0\,1\,1\,1\,1\,0 \end{pmatrix}.$$

The key schedule takes a 128-bit key $K = K'\|K''$ as the input of 4-round Feistel structure, as shown in Figure 15, to compute another 128-bit key $K_A = K'_A\|K''_A$. The round function is borrowed from the encryption, where the round keys are pre-defined constants. Then, each round key k_i can be derived from the rotation of K or K_A. Since we only focus on $(k_0, k_1, k_{12}, k_{13})$, we omit detailed key schedule here.

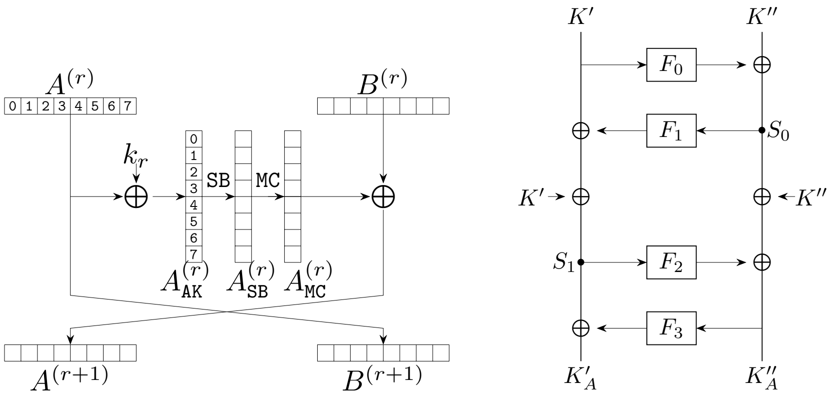


Fig. 15: One full round function of `Camellia` and the key schedule of `Camellia`

B.3 Specification of Aria

`Aria` was proposed by Korean researchers at ICISC 2003 [21] and the version 1.2 was subsequently included in the Korean Standard (KS X1213) in 2004. In this paper, we focus our attention on `Aria`-128, which refers to both the block and key sizes are 128 bits, and which we henceforth abbreviate as `Aria`. `Aria` is based on SPN structure with 12 rounds, and each round except the last one consists of `Substitution-Layer` (`SL`), `Diffusion-Layer` (`DL`) and `AddRoundKey` (`AK`) as shown in Figure 16. In the last round, the `DL` is omitted. Before the first round, a whitening key is `XOR`ed to the plaintext. The updated matrix P used in

DL is a 16×16 binary matrix described as follows.

$$P = \begin{pmatrix}
0&0&0&1&1&0&1&0&1&1&0&0&0&1&1&0\\
0&0&1&0&0&1&0&1&1&1&0&0&1&0&0&1\\
0&1&0&0&1&0&1&0&0&0&1&1&1&0&0&1\\
1&0&0&0&0&1&0&1&0&0&1&1&0&1&1&0\\
1&0&1&0&0&1&0&0&1&0&0&1&0&0&1&1\\
0&1&0&1&1&0&0&0&0&1&1&0&0&0&1&1\\
1&0&1&0&0&0&0&1&0&1&1&0&1&1&0&0\\
0&1&0&1&0&0&1&0&1&0&0&1&1&1&0&0\\
1&1&0&0&1&0&0&1&0&0&1&0&0&1&0&1\\
1&1&0&0&0&1&1&0&0&0&0&1&1&0&1&0\\
0&0&1&1&0&1&1&0&1&0&0&0&0&1&0&1\\
0&0&1&1&1&0&0&1&0&1&0&0&1&0&1&0\\
0&1&1&0&0&0&1&1&0&1&0&1&1&0&0&0\\
1&0&0&1&0&0&1&1&1&0&1&0&0&1&0&0\\
1&0&0&1&1&1&0&0&0&1&0&1&0&0&1&0\\
0&1&1&0&1&1&0&0&1&0&1&0&0&0&0&1
\end{pmatrix}.$$

In this paper, we target on the preimage attack on Aria-DM. Since the key is usually fixed as a constant in the DM hashing mode, we omit the description of the key schedule here.

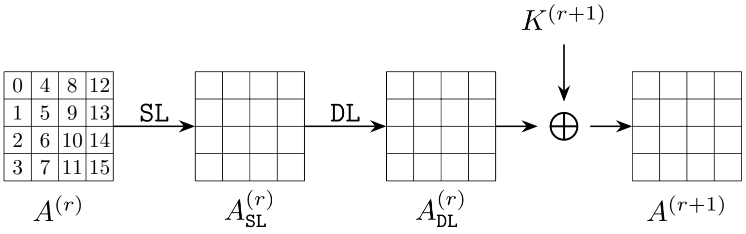


Fig. 16: One full round function of Aria

C Figure and algorithms for Midori64 and Aria

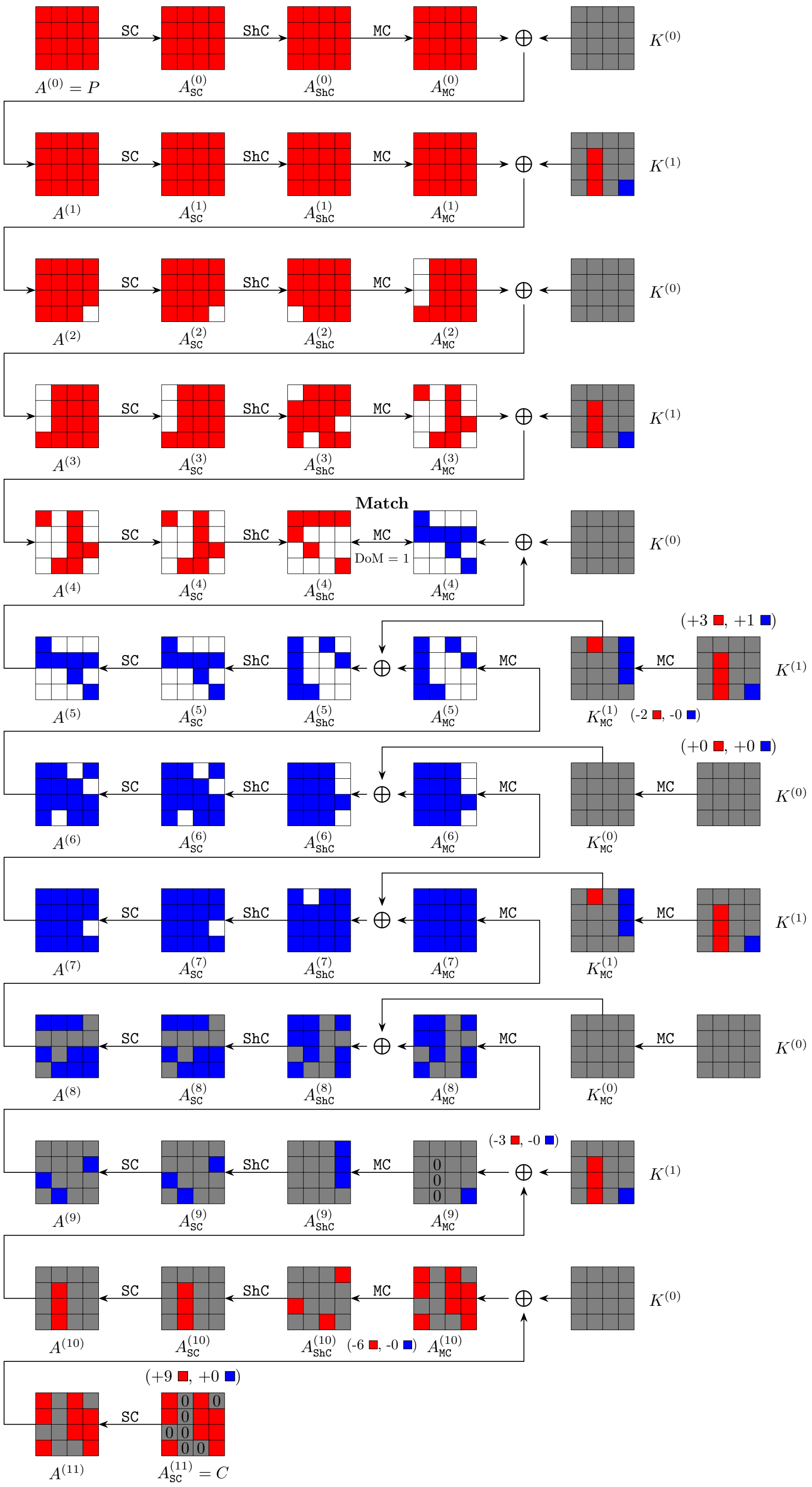


Fig. 17: Meet-in-the-Middle key recovery attack on 12-round weakened `Midori64`, optimized for data complexity

Algorithm 4: MitM Key Recovery Attack on 12-round weakened `Midori64`, , optimized for data complexity

```
$C[1,3,5,8,9,13,14] \leftarrow 0$, $A^{(9)}_{\texttt{MC}}[5,9,13] \leftarrow 0$
$A^{(10)}_{\texttt{MC}}[0] \oplus A^{(10)}_{\texttt{MC}}[4] \leftarrow 0$, $A^{(10)}_{\texttt{MC}}[0] \oplus A^{(10)}_{\texttt{MC}}[12] \leftarrow 0$, $A^{(10)}_{\texttt{MC}}[2] \oplus A^{(10)}_{\texttt{MC}}[6] \leftarrow 0$,
  $A^{(10)}_{\texttt{MC}}[2] \oplus A^{(10)}_{\texttt{MC}}[10] \leftarrow 0$, $A^{(10)}_{\texttt{MC}}[7] \oplus A^{(10)}_{\texttt{MC}}[11] \leftarrow 0$, $A^{(10)}_{\texttt{MC}}[7] \oplus A^{(10)}_{\texttt{MC}}[15] \leftarrow 0$
Collecting plaintext-ciphertext pairs by traversing the non-constant $16 - 7 = 9$
  cells in $C$, and storing them in table $H$
for all possible values of the ■ cells in $K^{(0)}$ and $K^{(1)}$ do
    for $(\mathfrak{c}_{\mathcal{R},1}, \mathfrak{c}_{\mathcal{R},2}) \in \mathbb{F}_2^{2\times 4}$ do
        Derive the solution space $\mathcal{S}_{\mathcal{R}}$ of 🟥 cells by
            $$\begin{cases} K^{(0)}[5] \oplus K^{(0)}[9] = \mathfrak{c}_{\mathcal{R},1} \\ K^{(0)}[5] \oplus K^{(0)}[13] = \mathfrak{c}_{\mathcal{R},2} \end{cases}$$
        $L \leftarrow [\,]$
        for $v_{\mathcal{R}} \in \mathcal{S}_{\mathcal{R}}$ do
            Compute $A^{(4)}_{\texttt{ShC}}[0,4]$ along the forward computation path:
            $A^{(9)}_{\texttt{MC}} \rightarrow C \rightarrow Dec_K(C) \rightarrow A^{(4)}_{\texttt{ShC}}$ by accessing $H$
            $L[A^{(4)}_{\texttt{ShC}}[0] \oplus A^{(4)}_{\texttt{ShC}}[4]] \leftarrow v_{\mathcal{R}}$
        end
        for $2^4$ possible values of $K^{(1)}[15]$ do
            Compute $A^{(4)}_{\texttt{MC}}[0,4]$ along the backward computation path:
              $C \rightarrow A^{(4)}_{\texttt{MC}}$
            for Candidate keys in $L[A^{(4)}_{\mathit{MC}}[0] \oplus A^{(4)}_{\mathit{MC}}[4]]$ do
                Test the guessed key with several plaintext-ciphertext pairs
            end
        end
    end
end
```

Algorithm 5: MitM Pseudo-Preimage Attack on 6-round `Aria-DM`

```
for $2^x$ possible values of ■ in $A^{(1)}$    /* $x + 104 = 120 - 8$, i.e., $x = 8$ */
do
    $V \leftarrow [\,]$;
    for $v_{\mathcal{R}} \in \mathbb{F}_2^{8\times 14}$ in $A^{(1)}$ do
        Compute backward to to get the values of the 🟥 cells in $A^{(0)}_{\texttt{DL}}$,
            $\mathfrak{c}_{\mathcal{R}}[0] \leftarrow A^{(0)}_{\texttt{DL}}[0] \oplus A^{(0)}_{\texttt{DL}}[6] \oplus A^{(0)}_{\texttt{DL}}[7] \oplus A^{(0)}_{\texttt{DL}}[8] \oplus A^{(0)}_{\texttt{DL}}[10] \oplus A^{(0)}_{\texttt{DL}}[13]$,
            $\mathfrak{c}_{\mathcal{R}}[1] \leftarrow A^{(0)}_{\texttt{DL}}[0] \oplus A^{(0)}_{\texttt{DL}}[4] \oplus A^{(0)}_{\texttt{DL}}[5] \oplus A^{(0)}_{\texttt{DL}}[9] \oplus A^{(0)}_{\texttt{DL}}[11] \oplus A^{(0)}_{\texttt{DL}}[14]$.
        Compute forward to the 🟥 cells in $A^{(1)}_{\texttt{SL}}$ and $A^{(2)}_{\texttt{SL}}$,
            $\mathfrak{c}_{\mathcal{R}}[2] \leftarrow A^{(1)}_{\texttt{SL}}[4] \oplus A^{(1)}_{\texttt{SL}}[6] \oplus A^{(1)}_{\texttt{SL}}[8] \oplus A^{(1)}_{\texttt{SL}}[9] \oplus A^{(1)}_{\texttt{SL}}[13] \oplus A^{(1)}_{\texttt{SL}}[14]$,
            $\mathfrak{c}_{\mathcal{R}}[3] \leftarrow A^{(1)}_{\texttt{SL}}[4] \oplus A^{(1)}_{\texttt{SL}}[9] \oplus A^{(1)}_{\texttt{SL}}[10] \oplus A^{(1)}_{\texttt{SL}}[14] \oplus A^{(1)}_{\texttt{SL}}[15]$,
            $\mathfrak{c}_{\mathcal{R}}[4] \leftarrow A^{(1)}_{\texttt{SL}}[2] \oplus A^{(1)}_{\texttt{SL}}[5] \oplus A^{(1)}_{\texttt{SL}}[6] \oplus A^{(1)}_{\texttt{SL}}[8] \oplus A^{(1)}_{\texttt{SL}}[13] \oplus A^{(1)}_{\texttt{SL}}[15]$,
            $\mathfrak{c}_{\mathcal{R}}[5] \leftarrow A^{(1)}_{\texttt{SL}}[0] \oplus A^{(1)}_{\texttt{SL}}[6] \oplus A^{(1)}_{\texttt{SL}}[7] \oplus A^{(1)}_{\texttt{SL}}[8] \oplus A^{(1)}_{\texttt{SL}}[10] \oplus A^{(1)}_{\texttt{SL}}[13]$,
            $\mathfrak{c}_{\mathcal{R}}[6] \leftarrow A^{(1)}_{\texttt{SL}}[5] \oplus A^{(1)}_{\texttt{SL}}[7] \oplus A^{(1)}_{\texttt{SL}}[10] \oplus A^{(1)}_{\texttt{SL}}[11]$,
            $\mathfrak{c}_{\mathcal{R}}[7] \leftarrow A^{(1)}_{\texttt{SL}}[10] \oplus A^{(1)}_{\texttt{SL}}[11] \oplus A^{(1)}_{\texttt{SL}}[12] \oplus A^{(1)}_{\texttt{SL}}[15]$.
            $\mathfrak{c}_{\mathcal{R}}[8] \leftarrow A^{(2)}_{\texttt{SL}}[2] \oplus A^{(2)}_{\texttt{SL}}[8] \oplus A^{(2)}_{\texttt{SL}}[15]$,
            $\mathfrak{c}_{\mathcal{R}}[9] \leftarrow A^{(2)}_{\texttt{SL}}[1] \oplus A^{(2)}_{\texttt{SL}}[4] \oplus A^{(2)}_{\texttt{SL}}[15]$,
            $\mathfrak{c}_{\mathcal{R}}[10] \leftarrow A^{(2)}_{\texttt{SL}}[3] \oplus A^{(2)}_{\texttt{SL}}[6] \oplus A^{(2)}_{\texttt{SL}}[8]$,
            $\mathfrak{c}_{\mathcal{R}}[11] \leftarrow A^{(2)}_{\texttt{SL}}[4] \oplus A^{(2)}_{\texttt{SL}}[6] \oplus A^{(2)}_{\texttt{SL}}[12] \oplus A^{(2)}_{\texttt{SL}}[15]$,
            $\mathfrak{c}_{\mathcal{R}}[12] \leftarrow A^{(2)}_{\texttt{SL}}[8] \oplus A^{(2)}_{\texttt{SL}}[9] \oplus A^{(2)}_{\texttt{SL}}[12] \oplus A^{(2)}_{\texttt{SL}}[15]$.
        $V[\mathfrak{c}_{\mathcal{R}}] \leftarrow v_{\mathcal{R}}$; /* There are $2^8$ elements in $V[\mathfrak{c}_{\mathcal{R}}]$ for each $\mathfrak{c}_{\mathcal{R}}$ */
    end
    for $\mathfrak{c}_{\mathcal{R}} \in \mathbb{F}_2^{8\times 13}$ do
        $L \leftarrow [\,]$
        for $v_{\mathcal{R}} \in V[\mathfrak{c}_{\mathcal{R}}]$ do
            Compute to the 🟥 cells in $A^{(4)}_{\texttt{DL}}$, and one-byte $End_{\mathcal{R}}$ for matching is
              derived by
                $End_{\mathcal{R}} \leftarrow \left(A^{(4)}_{\texttt{DL}}[3] \oplus A^{(4)}_{\texttt{DL}}[4] \oplus A^{(4)}_{\texttt{DL}}[6] \oplus A^{(4)}_{\texttt{DL}}[8] \oplus A^{(4)}_{\texttt{DL}}[9]\right)$
            $L[End_{\mathcal{R}}] \leftarrow v_{\mathcal{R}}$
        end
        for $2^8$ possible values of $A^{(1)}[3]$ do
            Compute to the 🟦 cells in $A^{(4)}_{\texttt{DL}}$ and $A^{(4)}_{\texttt{SL}}$, derive one-byte $End_{\mathcal{B}}$ for
              matching by
                $End_{\mathcal{B}} \leftarrow \left(A^{(4)}_{\texttt{SL}}[0] \oplus A^{(4)}_{\texttt{DL}}[13] \oplus A^{(4)}_{\texttt{DL}}[14]\right)$
            for $v_{\mathcal{R}} \in L[End_{\mathcal{B}}]$ do
                Reconstruct the (candidate) message $X$
                if $X$ is a preimage then
                    Output $X$ and stop
                end
            end
        end
    end
end
```