

Finding Complete Impossible Differential Attacks on AndRX Ciphers and Efficient Distinguishers for ARX Designs

Debasmita Chakraborty² , Hosein Hadipour¹ , Phuong Hoa Nguyen³ 
and Maria Eichlseder¹ 

¹ Graz University of Technology, Graz, Austria

hsn.hadipour@gmail.com, maria.eichlseder@iaik.tugraz.at

² Indian Statistical Institute, Kolkata, India

debasmitachakraborty1@gmail.com

³ Univ Rennes, Inria, Centre National de la Recherche Scientifique, Institut de Recherche en Informatique et Systèmes Aléatoires, Rennes, France

phuong-hoa.nguyen@irisa.fr

Abstract. The impossible differential (ID) attack is one of the most important cryptanalytic techniques for block ciphers. There are two phases to finding an ID attack: searching for the distinguisher and building a key recovery upon it. Previous works only focused on automated distinguisher discovery, leaving key recovery as a manual post-processing task, which may lead to a suboptimal final complexity. At EUROCRYPT 2023, Hadipour et al. introduced a unified constraint programming (CP) approach based on satisfiability for finding optimal complete ID attacks in strongly aligned ciphers. While this approach was extended to weakly-aligned designs like PRESENT at ToSC 2024, its application to ARX and AndRX ciphers remained as future work. Moreover, this method only exploited ID distinguishers with direct contradictions at the junction of two deterministic transitions. In contrast, some ID distinguishers, particularly for ARX and AndRX designs, may not be detectable by checking only the existence of direct contradictions.

This paper fills these gaps by extending Hadipour et al.’s method to handle indirect contradictions and adapting it for ARX and AndRX designs. We also present a similar method for identifying zero-correlation (ZC) distinguishers. Moreover, we extend our new model for finding ID distinguishers to a unified optimization problem that includes both the distinguisher and the key recovery for AndRX designs. Our method improves ID attacks and introduces new distinguishers for several ciphers, such as SIMON, SPECK, Simeck, ChaCha, Chaskey, LEA, and SipHash. For example, we achieve a one-round improvement in ID attacks against SIMON-64-96, SIMON-64-128, SIMON-128-128, SIMON-128-256 and a two-round improvement against SIMON-128-192. These results significantly contribute to our understanding of the effectiveness of automated tools in the cryptanalysis of different design paradigms.

Keywords: Cryptanalysis · Impossible differentials · Key recovery · CP · ARX · AndRX · SIMON · SPECK · Simeck · ChaCha · Chaskey · LEA · SipHash

1 Introduction

Impossible differential (ID) cryptanalysis is one of the most powerful cryptanalytic techniques for block ciphers, independently introduced by Biham et al. [BBS99] and Knudsen [Knu98]. Its core idea is first using impossible differentials, which are differential transitions with probability zero, to distinguish the block cipher from a random permutation. Once an

ID distinguisher is obtained, we extend it by a few rounds, possibly at both ends. A guessed value for the involved key bits that partially encrypts/decrypts a given pair to the impossible differential is undoubtedly wrong, and we should discard it. Having access to a certain number of pairs, the goal is to discard as many wrong candidates for the involved keys as possible (referred to as the *guess-and-filter* step). Finally, we brute-force the remaining candidates to uniquely identify the correct key (referred to as the *exhaustive search step*). We can adjust the attack parameters, for example, the number of pairs, to make a trade-off between the complexity of the guess-and-filter and the exhaustive search phases. The ID attack has been successfully applied to many block ciphers. For example, ID attacks are the best cryptanalysis results for CAMELLIA [LLG⁺12, BNPS14]. As another example, ID attacks were the first attack on 7 rounds of AES [ZWF07, LDKK08, MDRMH10], and that remained one of the best attacks for a long time. The dual of the ID attack in the context of linear cryptanalysis is the zero-correlation (ZC) attack that was introduced by Bogdanov and Rijmen [BR14]. While the ID attack exploits differential transitions with probability zero, the ZC attack exploits linear hulls with correlation zero as a distinguishing property.

Building an ID attack, similar to many other statistical attacks on block ciphers, includes two main phases: finding a distinguisher and extending it for key recovery. Together with introducing the ID attack, Biham et al. [BBS99] also introduced a method to find ID distinguishers: the *miss-in-the-middle* approach. The core idea of the miss-in-the-middle approach is to find input and output differences such that if the difference propagations are deterministic through the cipher forward and backward, respectively, they contradict each other somewhere in the middle. A similar method applies to the ZC attack as well. While the miss-in-the-middle method provides a systematic way to check whether a given input/output difference results in an ID distinguisher, it does not offer a systematic way to choose input and output differences that efficiently result in an ID distinguisher. In practice, one should try several input/output differences (typically with very few active words or bits) and propagate them halfway with probability one to see if they contradict each other. While this approach might be easy to apply at the word level for strongly aligned and symmetric designs like AES [DR99] and CLEFIA [SSA⁺07], it is not straightforward to apply it to designs like SKINNY [BJK⁺16] with its slower and less regular diffusion properties. Its application to bit-oriented designs like SIMON, SPECK [BSS⁺15], and Simeck [YZS⁺15] is even more complicated.

Finding ID distinguishers requires tracking the differential transitions through the building blocks of block ciphers at the level of words (nibbles or bytes) or sometimes bits. Regarding the key recovery phase, the attacker has to extend the distinguisher, possibly at two ends, considering more cryptographic properties. This includes identifying the internal states and, subsequently, the key bits whose values are needed to determine the input/output difference of the ID distinguisher. The distinguisher and key recovery parameters, like the number of pairs, should be chosen to minimize the total time complexity of the attack. Overall, building the ID attack is a combinatorial optimization problem that can be daunting and prone to human error if done manually. Therefore, several efforts have been made to automate ID attacks. There are two common approaches to automating the cryptanalytic techniques in symmetric-key cryptography: the first relies on dedicated algorithms, and the second relies on general-purpose solvers. In the second approach, the cryptanalyst models the cryptanalytic problem as a constraint satisfaction problem (CSP) or a constraint optimization problem (COP) and then uses state-of-the-art general-purpose constraint programming (CP) solvers to solve it. Note that CP solvers include many solvers, such as Satisfiability Modulo Theories (SMT), Satisfiability (SAT), and MILP solvers.

The first few efforts to automate the ID attack relied on dedicated algorithms and only focused on finding the distinguishers. Examples include the \mathcal{U} -method [KHS⁺03] and the UID-method [LLWG14]. Another tool based on a dedicated algorithm is the tool provided

by Derbez and Fouque at CRYPTO 2016 [DF16] for finding \mathcal{DS} -MITM attacks that is also applicable for finding ID attacks. The main downside of the dedicated algorithms is that they are typically designed for a specific type of design, and modifying them for a new design may require substantial effort. Additionally, providing dedicated, efficient algorithms for solving cryptanalytic problems is typically a challenging task.

As one of the pioneering methods in the category of general-purpose solvers, at EUROCRYPT 2017, Sasaki and Todo [ST17] proposed a method based on Mixed-Integer Linear Programming (MILP) to find ID distinguishers. Cui et al. [CCJ⁺21] also independently introduced this method almost simultaneously (the paper [CCJ⁺21] was published in 2021, but its eprint version was online since 2016 [CJF⁺16]) and applied it to finding ZC distinguishers. The main advantage of the CP-based method by Sasaki and Todo (and Cui et al.) is that the attacker does not have to predict the contradiction mechanism, and this tool can find more complicated contradictions that are not simply detectable by a naive miss-in-the-middle approach. However, the main disadvantage of this approach is that the input/output difference (or linear mask) should be fixed on each try, checking whether the resulting model is unsatisfiable. If so, the input/output difference (resp. linear mask) yields an impossible differential (resp. zero-correlation linear hull). Thus, the attacker needs to try many input/output differences before finding the distinguisher, with non-negligible complexity per try. As a result, the search space for the input/output differences is typically limited to the input/output differences with very few active words or bits. More importantly, the CP-based method by Sasaki and Todo [ST17] and Cui et al. [CCJ⁺21] is based on the unsatisfiability of the CP/MILP model and thus cannot be extended to a unified constraint optimization model for key recovery. This limitation restricts the usage of this CP-based method to only finding the distinguishers.

At EUROCRYPT 2023, Hadipour et al. [HSE23] introduced a CP-based model that converts the search for impossible differentials into a satisfiability problem, extendable to a unified constraint optimization model for key recovery. This method also applies to finding ZC and integral attacks. However, the CP model in [HSE23] was word-oriented, suitable for strongly/moderately aligned designs like SKINNY. At ToSC 2024, Hadipour et al. [HGSE24] enhanced this method by extending it to a bit-wise model, considering the internal structure of S-boxes. As a result, they provided a bit-wise CP model based on satisfiability for finding ID/ZC distinguishers for weakly aligned designs such as Ascon [DEMS21]. While the methods in [HSE23, HGSE24] improved the best-known ID/ZC and integral attacks on several SPN ciphers, their application to an essential category of block ciphers, i.e., Addition-Rotation-XOR (ARX) and And-Rotation-XOR (AndRX) designs, was left for future work. Additionally, the method introduced in [HSE23, HGSE24] identifies contradictions at the junction of two deterministic propagations (referred to as *direct* contradictions). However, for some ID distinguishers, the contradiction is more complicated and cannot be identified solely based on checking the existence of direct contradictions [SB18]. We refer to these contradictions as *indirect* contradictions. Therefore, extending the method in [HSE23, HGSE24] to identify indirect contradictions while maintaining the model based on satisfiability is an open problem.

Our contributions. This paper extends the methods proposed at EUROCRYPT 2023 and ToSC 2024 for finding ID attacks [HSE23, HGSE24] from different aspects. First, we provide a CP-based model based on satisfiability to find ID distinguishers for ARX and AndRX ciphers. Then, as the main contribution regarding the distinguisher part, we propose a CP-based model based on satisfiability capable of identifying particular indirect contradictions for the first time. The application of our CP model for identifying indirect contradictions is not limited to ARX and AndRX ciphers; it applies to other categories of block ciphers like SPN and Feistel ciphers. We also show the applicability of our new CP models for finding ZC distinguishers. Next, we show how to extend the CP-model for key

recovery in [HSE23, HGSE24] for bit-wise designs, particularly AndRX designs. Lastly, we put our new models for distinguisher and key-recovery parts into a unified CP model for finding the complete ID attacks, including the key recovery evaluations. To show the usefulness of our methods, we apply them to find ID distinguishers/attacks on several ARX and AndRX ciphers and improve the best previous results. Table 1 and Table 2 provide a summary of ID distinguishers of ARX ciphers, and a summary of the complete ID attack on AndRX ciphers, respectively. Additionally, Table 11 (Subsection D.3 in Appendix D) and Table 12 (Subsection D.4 in Appendix D) present an overview of existing attacks (excluding ID attacks) on SIMON, and Simeck, respectively.

- We provide ID distinguishers for ChaCha [Ber08], Siphash [AB12], SPECK-96, and SPECK-128 [BSS⁺15] for the first time.
- We provide several new ID distinguishers for Chaskey [MMH⁺14] and SPECK with truncated input/output differences.
- We improve ID attacks on SIMON-64-96, SIMON-64-128, SIMON-128-128, and SIMON-128-256 by one round, and SIMON-128-192 by two rounds.
- We provide improved attacks on various variants of SIMON and Simeck: While many previous attacks required the full code-book, we provide ID attacks for the same number of rounds with a lower data complexity than the full code-book.

Table 1: ID Distinguishers on ARX ciphers. #R: Length of the distinguisher. #Dist. : Number of distinguishers found using our tool.

Cipher	Contradiction	#R	#Dist.	Ref.
SPECK-32	Direct	6	3	[RC19]
	Direct	6	2^4	F
SPECK-48	Direct	6	20	[RC19]
	Direct	6	2^{17}	F
SPECK-64	Direct	6	157	[LKH ⁺ 16, RC19]
	Direct	6	2^{33}	F
SPECK-96	Direct	6	2^{65}	5.1.1
SPECK-128	Direct	6	2^{97}	5.1.1
LEA	Direct	10	-	[CCJ ⁺ 21]
	Direct	10	2^2	5.1
ChaCha	Direct	5	2^{80}	5.1
SipHash	Direct	4	2^{14}	5.1
Chaskey	Direct	4	15	[SBS21]
	Direct	4	2^7	5.1

Performance. Unlike previous tools based on unsatisfiability, our tool efficiently identifies a group of ID/ZC distinguishers (or truncated distinguishers) in just one execution, without fixing input/output differences, and terminates within minutes to a few hours on a laptop (Intel Core i5-8250U CPU $1.6GHz \times 8$ and 8GB of memory). MiniZinc [NSB⁺07] is used to model and solve CSP problems. The source code of our tool is available at: <https://github.com/Debasmita-isi/zeroplusplus>.

Outline. We start with an overview of ID attacks and recall Hadipour et al.’s model in Section 2. Next, in Section 3, we describe our new approach for identifying ID/ZC distinguishers with indirect contradictions for ARX and AndRX designs. Section 4 extends

Table 2: Summary of our ID attacks. Dist. = Length of the distinguisher. #R = Number of rounds attacked. † : Distinguisher based on indirect contradiction.

Cipher	Dist.	#R	Time	Data	Mem.	Ref.
SIMON-32-64	11	19	$2^{62.56}$	2^{32}	2^{44}	[BNPS14]
	11	19	$2^{58.919}$	2^{32}	$2^{49.674}$	[CWW15]
	11	20	$2^{62.8}$	2^{32}	$2^{43.5}$	[DF16]
	11	19/20	$2^{59} / 2^{62}$	$2^{30.79} / 2^{30.47}$	$2^{49.8} / 2^{51.5}$	F
SIMON-48-72	12	20	$2^{70.69}$	2^{48}	2^{58}	[BNPS14]
	12	20	$2^{71.278}$	2^{48}	$2^{63.393}$	[CWW15]
	12	20	2^{67}	$2^{46.79}$	$2^{64.8}$	F
SIMON-48-96	12	21	$2^{94.73}$	2^{48}	2^{70}	[BNPS14]
	12	21	$2^{94.556}$	2^{48}	$2^{86.447}$	[CWW15]
	12	21	$2^{86.79}$	$2^{47.7}$	$2^{77.8}$	F
SIMON-64-96	13	21	$2^{94.56}$	2^{64}	2^{60}	[BNPS14]
	13	21	$2^{95.279}$	2^{64}	$2^{72.469}$	[CWW15]
	13	21/22	$2^{70.28} / 2^{91}$	$2^{59.27} / 2^{62.79}$	$2^{69.3} / 2^{83.8}$	F
SIMON-64-128	13	22	$2^{126.56}$	2^{64}	2^{75}	[BNPS14]
	13	22	$2^{125.115}$	2^{64}	$2^{98.773}$	[CWW15]
	13	22/23	$2^{98} / 2^{123}$	$2^{59.4} / 2^{61.27}$	$2^{85.37} / 2^{96.3}$	F
SIMON-96-96	16	24	$2^{94.62}$	2^{94}	2^{61}	[BNPS14]
	16	24	2^{88}	$2^{83.47}$	$2^{69.5}$	F
SIMON-96-144	16	25	$2^{142.59}$	2^{96}	2^{77}	[BNPS14]
	16	25	2^{122}	$2^{94.93}$	$2^{87.9}$	F
SIMON-128-128	19	27	$2^{126.6}$	2^{94}	2^{61}	[BNPS14]
	19	27/28	$2^{95.79} / 2^{112.64}$	$2^{97.79} / 2^{112.6}$	$2^{66.8} / 2^{84.7}$	5.2.1
SIMON-128-192	19	28	$2^{190.56}$	2^{128}	2^{77}	[BNPS14]
	19	29/30	$2^{162} / 2^{185.47}$	$2^{127.37} / 2^{127.5}$	$2^{107.4} / 2^{111.47}$	F
SIMON-128-256	19	30	$2^{254.68}$	2^{128}	2^{111}	[BNPS14]
	19	30/31	$2^{226} / 2^{247}$	$2^{125.37} / 2^{127.64}$	$2^{120.4} / 2^{127.7}$	F
Simeck-32	11	20	$2^{61.11}$	2^{32}	2^{51}	[ZLW+23]
	11	20	$2^{57.27}$	$2^{27.28}$	$2^{47.3}$	F
Simeck-48	15 [†]	25	$2^{94.23}$	2^{46}	2^{67}	[ZLW+23]
	15 [†]	25	$2^{93.05}$	$2^{47.05}$	$2^{68.12}$	F
Simeck-64	17 [†]	27	$2^{126.56}$	2^{63}	2^{68}	[ZLW+23]
	17 [†]	27	2^{126}	$2^{63.47}$	$2^{68.45}$	F

our improved model to key-recovery ID attacks on AndRX ciphers. Then we discuss the application of our methods in Section 5. Finally, Section 6 concludes the paper.

2 Background

Here, we briefly review the key recovery and complexity analysis of ID attacks. We also recall the bit-wise CP model in [HGSE24] for identifying ID/ZC distinguishers.

2.1 Key Recovery and Complexity Analysis in ID Attacks

Consider a block cipher E with an n -bit block size and κ -bit key size. We consider pairs (X, X') with $X, X' \in \mathbb{F}_2^n$ and denote their difference by $\Delta = X \oplus X'$. Then, $\Pr(\Delta_U \rightarrow \Delta_L)$ denotes the expected differential probability that an input pair (X, X') with $X \oplus X' = \Delta_U$

yields an output pair with $E(X) \oplus E(X') = \Delta_L$, where E is implicit from the context. More generally, we use the same notation to also denote bitwise truncated differences $\Delta \subseteq \mathbb{F}_2^n$ and the corresponding probabilities averaged over all input differences in the set. Given a (truncated) input difference Δ_U , we refer to the minimal truncated output difference Δ_L such that $\Pr(\Delta_U \rightarrow \Delta_L) = 1$ as propagation with probability 1 or deterministic propagation. If $\Pr(\Delta_U \rightarrow \Delta_L) = 0$, we write $\Delta_U \not\rightarrow \Delta_L$ and call this an impossible differential.

Suppose there exists an impossible differential $\Delta_U \not\rightarrow \Delta_L$ for r_D rounds of E (denoted as E_D). To perform a key recovery, as illustrated in Figure 1, we extend the distinguisher by a few rounds at both ends. Let E_B and E_F represent the rounds added before and after E_D , respectively, with r_B and r_F denoting their respective numbers, such that $E = E_F \circ E_D \circ E_B$. Subsequently, we propagate the difference Δ_U (and Δ_L) through E_B^{-1} (and E_F) with probability one to obtain the truncated difference Δ_B (and Δ_F). Here, $|\Delta_B|$ and $|\Delta_F|$ denote the number of non-fixed bit differences in Δ_B and Δ_F , respectively. Assume that $\Pr(\Delta_B \rightarrow \Delta_U) = 2^{-c_B}$ and $\Pr(\Delta_L \leftarrow \Delta_F) = 2^{-c_F}$. In the context of (impossible) differential key recovery, c_B and c_F are typically referred to as the number of bit filters that should be satisfied for differential transitions $\Delta_B \rightarrow \Delta_U$ and $\Delta_L \leftarrow \Delta_F$, respectively. As illustrated in Figure 1, assume that the key bits $k_B \cup k_F$ are involved in deriving the difference Δ_U and Δ_L from Δ_B and Δ_F , respectively. With these parameters established, we divide the key recovery of an ID attack into three steps:

- **Pair Generation.** In this step, we generate N plaintext pairs (P, P') such that $P \oplus P' \in \Delta_B$ and $E(P) \oplus E(P') \in \Delta_F$. The problem of finding such pairs is known as the *limited birthday problem*. The complexity of this step is (see [BNPS14]) $T_0 := \max \{ \min_{\Delta \in \{\Delta_B, \Delta_F\}} \{ \sqrt{N2^{n+1-|\Delta|}}, N2^{n+1-|\Delta_B|-|\Delta_F|} \}$.
- **Guess-and-Filter.** In this step, we eliminate the incorrect candidates for $k_B \cup k_F$ by checking whether a candidate for involved key bits yields the impossible differential for at least one of the N pairs. We typically use the *early abort* technique to perform this step [LKKD08]: we split $k_B \cup k_F$ into several subsets and guess them one by one. At each step, we check some new bit filters and discard a portion of the pairs that do not satisfy the bit filters. The correct key guess never suggests an impossible differential for any pairs. Thus, we perform N partial encryptions/decryptions for the correct key guess. However, a wrong key guess may suggest an impossible differential for some pairs. The more pairs we have, the more likely a wrong key guess suggests an impossible differential for at least one of the pairs. A lower bound for the complexity of this step is (see [BNPS14]) $T_1 + T_2 = N + 2^{|k_B \cup k_F|} \frac{N}{2^{c_B+c_F}}$ partial encryptions.
- **Exhaustive Search.** The probability that a wrong key passes the guess-and-filter step is $P = (1 - 2^{-(c_B+c_F)})^N$, which means the expected number of wrong keys that pass the guess-and-filter step is $P \cdot 2^{|k_B \cup k_F|}$. Considering that $\kappa - |k_B \cup k_F|$ key bits are not involved in the guess-and-filter step, we should brute-force a key space of size $T_3 = 2^{\kappa - |k_B \cup k_F|} \cdot P \cdot 2^{|k_B \cup k_F|} = P \cdot 2^\kappa$ to uniquely retrieve the correct key.

If we assume that C_E represents the cost of executing E , and $C_{E'}$ represents the proportion of the cost for executing E_B and E_F compared to complete encryption, the total time complexity of the ID key recovery is: $T_{\text{tot}} = (T_0 + (T_1 + T_2) C_{E'} + T_3) C_E$. If we consider the number of encryption queries as the data complexity, then T_0 represents the data complexity. To keep the data complexity less than the full code-book, we should have $T_0 < 2^n$, and to keep the time complexity less than brute force, we require $T_{\text{tot}} < 2^\kappa$.

Given that the complexity formula of ID attacks includes some exponential terms and also the square root of some attack parameters, following the approach in [HSE23], we reformulate them as follows to be able to incorporate them in our CP model: Let g denote the number of key bits that we retrieve in the guess-and-filter step, i.e., $P = 2^{-g}$. Assuming that $P < \frac{1}{2}$, we have $1 < g \leq |k_B \cup k_F|$. Also assume that $(1 - 2^{-(c_B+c_F)})^N \approx e^{-N \cdot 2^{-(c_B+c_F)}}$.

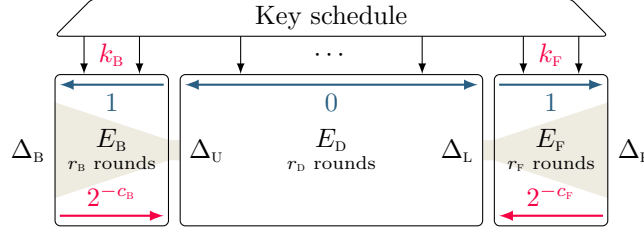


Figure 1: Overview and parameters of impossible differential attacks.

Thus, we have $N = 2^{c_B+c_F+\log_2(g)-0.53}$. Moreover, suppose that $\text{LG}(g) = \log_2(g) - 0.53$. Therefore, we can reformulate the complexity analysis of the ID attack as follows:

$$\begin{aligned}
 T_0 &= \max \left\{ \min_{\Delta \in \{\Delta_B, \Delta_F\}} \left\{ 2^{\frac{c_B+c_F+n+1-|\Delta|+\text{LG}(g)}{2}} \right\}, \right. & T_0 < 2^n \\
 & \left. 2^{c_B+c_F+n+1-|\Delta_B|-|\Delta_F|+\text{LG}(g)} \right\}, \\
 T_1 &= 2^{c_B+c_F+\text{LG}(g)}, & T_2 &= 2^{|k_B \cup k_F|+\text{LG}(g)}, & T_3 &= 2^{k-g} \\
 T_{\text{tot}} &= (T_0 + (T_1 + T_2) C_{E'} + T_3) C_E, & T_{\text{tot}} &< 2^k \\
 M_{\text{tot}} &= \min \{ 2^{c_B+c_F+\text{LG}(g)}, 2^{|k_B \cup k_F|} \}, & M_{\text{tot}} &< 2^k.
 \end{aligned} \tag{1}$$

2.2 Bit-wise CP Model for Deterministic Trails

Here, we recall the bit-wise CP model in [HGSE24, HDE24] to encode the propagation of deterministic differential/linear trails. We explain the model for differential trails, but a similar approach can be used for linear trails. The idea is to encode the difference at each bit position via an integer variable with a $\{-1, 0, 1\}$ domain. The integer values “0” and “1” represent the fixed difference value of “0” and “1”, and “-1” means the difference value is either “0” or “1” (i.e., unknown). Then, the propagation of deterministic differential trails through XOR, Branching, and S-boxes can be encoded as follows.

Proposition 1 (Branching [HGSE24, HDE24]). For $f : \mathbb{F}_2 \rightarrow \mathbb{F}_2^n$, $f(x) = (y_0, y_1, \dots, y_{n-1})$ where $y_0 = y_1 = \dots = x$, the valid transitions for deterministic differential trails satisfy

$$\text{Branch}(\mathbf{x}, y_0, \dots, y_{n-1}) := \bigwedge_{i=0}^{n-1} (y_i = \mathbf{x}),$$

where the integer variables $\mathbf{x}, y_i \in \{-1, 0, 1\}$ encode the difference in x, y_i for $0 \leq i \leq n-1$.

Proposition 2 (XOR [HGSE24, HDE24]). For $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$, $f(x_0, x_1, \dots, x_{n-1}) = y$, where $y = x_0 \oplus x_1 \oplus \dots \oplus x_{n-1}$, the valid deterministic differential transitions satisfy

$$\text{XOR}(\mathbf{y}, \mathbf{x}_0, \dots, \mathbf{x}_{n-1}) := \begin{cases} \text{if } \bigvee_{i=0}^{n-1} (\mathbf{x}_i = -1) \text{ then } \mathbf{y} = -1 \\ \text{else } \mathbf{y} = \mathbf{x}_0 + \dots + \mathbf{x}_{n-1} \pmod{2} \end{cases}$$

The propagation of deterministic differential/linear trails through S-boxes can be explained as follows. We can model this by using the Difference Distribution Table (DDT) of the S-box, through which we can identify the differential transitions that have a known output difference in at least one bit position. These transitions are known as bit-wise deterministic differential transitions. To be more precise, let’s assume that the S-box is an $m \times n$ S-box. We examine all input activeness patterns in $\{-1, 0, 1\}^m$, where for each input activeness pattern, we check whether at least one bit of the output difference is known to be “0” or “1” with certainty. Next, we model all deterministic bit-wise differential transitions through the S-box using CP constraints. For details, we refer to [HGSE24, Sec

3.2]. A similar method works for modeling the bit-wise deterministic linear trails of S-boxes, using the *Linear Approximation Table* (LAT) to identify the bit-wise deterministic linear transitions. The CP constraints for bit-wise deterministic differential/linear propagation through S-boxes can be automatically derived with an extended version [HGSE24, HDE24] of the S-box Analyzer tool [HNE22]. In Subsection 3.1, we extend this method to model the building blocks of AndRX and ARX ciphers, particularly the modular addition.

2.3 CP Model for Finding ID/ZC Distinguishers

The CP model for finding ID/ZC distinguishers in [HSE23, HGSE24] is based on the *miss-in-the-middle* [BBS99] technique. According to this technique, we propagate a given input and output differences (resp. linear masks) through the block cipher with certainty forward and backward, respectively. If the two propagations contradict each other somewhere in the middle, then we can prove that the given input difference (resp. linear mask) never propagates to the given output difference (resp. linear mask). As a result, we have an ID (resp. ZC) distinguisher. The idea of Hadipour et al. is to model the deterministic differential (resp. linear) transitions through the block cipher in two opposite directions using CP constraints. The CP model is then extended by including some contradiction checker constraints for each bit position to guarantee the contradiction between the two deterministic propagations in at least one bit position. This way, any feasible solutions of the CP model are an impossible differential (resp. zero-correlation) distinguisher. The main advantage is that there are no constraints for the input/output differences (resp. linear masks), and the CP model is based on satisfiability. For more details, refer to [HSE23, HGSE24].

This method only identifies ID/ZC distinguishers relying on *direct* contradictions, i.e., contradictions that happen at the junction of two deterministic differential (or linear) trails propagated in two opposite directions. However, some ID distinguishers [SB18] are not detectable by only checking the existence of direct contradictions. The contradictions in these distinguishers are more complicated, and we refer to them as *indirect* contradictions. To address this gap, in Subsection 3.2, we provide a new CP model based on satisfiability, which is capable of identifying a particular type of indirect contradiction.

2.4 Unified CP Model for Finding Complete ID Attacks

Once we have a CP model based on satisfiability, we can extend it to find an optimal complete ID key recovery attack. We briefly recall the general view of the first CP model for finding complete ID attacks in [HSE23]. As visualized in Figure 1, assume that we split the block ciphers E into three sub-ciphers $E = E_F \circ E_D \circ E_B$, such that the distinguisher covers E_D , E_B , and E_F denote the extension of the distinguisher backwards and forwards for key recovery, respectively. Also, assume that CSP_D represents the Constraint Satisfaction Problem (CSP) modeling the distinguisher part. The idea is to extend CSP_D by additional CP variables/constraints that aim at modeling the key recovery procedure, as well as the complexity analysis of ID key recovery.

The key recovery process involves first propagating the input/output difference of the ID distinguisher backward/forward, then identifying the filters, and finally, pinpointing the cell/bit positions within the internal state or sub-keys whose difference or value is necessary for the guess-and-filter step. Once we have this information, referring to Equation 1, we can provide a rough estimation of the time, memory, and data complexity of the ID attack, along with a sketch of the key recovery procedure. For this purpose, according to [HSE23], one can define four types of binary CP variables for the extended parts E_B and E_F . The first type of binary variable encodes whether the difference in a particular position through E_B or E_F is zero. The second type of variable encodes whether a particular position acts as a filter. The third variable type encodes whether the difference of a certain cell within the

internal state or sub-keys should be known, and the fourth variable type encodes whether the value pair at a certain cell should be known. Next, we can define some constraints on these variables to model the guess-and-filter procedure and the complexity formula of ID key recovery. Finally, one can integrate the CP models for key-bridging [HE22] into this model to consider the relation between the involved keys $k_B \cup k_F$ (see Figure 1) to identify the actual size of $k_B \cup k_F$, which is a critical parameter in the time complexity of ID key recovery.

3 Modeling the Distinguishers

In this section, we expand the bit-wise CP model presented in [HGSE24] for identifying ID/ZC distinguishers in two key aspects. First, we introduce a rule to encode AND and modular addition operations within the bit-wise CP model. This enables us to create a CP model based on satisfiability to find ID/ZC distinguishers for ARX and AndRX ciphers. Subsequently, and of greater significance, we extend the bit-wise model to detect ID/ZC distinguishers with more intricate contradictions beyond direct ones. This adaptation empowers our new model to identify the longest existing ID/ZC distinguishers of Simeck that are not detectable by the models in [HSE23, HGSE24]. The versatility of our new model in identifying complex contradictions is not restricted to ARX and AndRX ciphers; it can also be applied to other designs, such as SPN ciphers. Similar to the CP models in [HSE23, HGSE24], the primary advantage of our new model is its extensibility to a unified optimization problem for discovering a complete ID attack.

3.1 Modeling the Distinguishers for ARX and AndRX Ciphers

Here, we propose some rules to model deterministic differential (linear) propagation through AND and modular addition operations. We elaborate on our modeling of deterministic differential trails, noting that the same approach applies to linear trails.

Proposition 3 (AND). *Let $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ be such that $y = f(x_0, x_1, \dots, x_{n-1}) = \bigwedge_{i=0}^{n-1} x_i$. Let \mathbf{x}_i and y be the corresponding integer variables with domain $\{-1, 0, 1\}$ to encode the difference in x_i and y . Then, the valid transitions for deterministic differential trails satisfy*

$$\text{AND}(\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1}, y) := \begin{cases} \text{if} & \mathbf{x}_0 = \mathbf{x}_1 = \dots = \mathbf{x}_{n-1} = 0 \quad \text{then} \quad y = 0 \\ \text{else} & y = -1 \end{cases}$$

Suppose that $f : \mathbb{F}_2^n \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ is such that $z = f(x, y) = x \boxplus y$, where \boxplus denotes addition modulo 2^n . Assume that we represent x as a bit-vector $x_0 || x_1 || \dots || x_{n-1}$, where $x_i \in \mathbb{F}_2$ for $0 \leq i \leq n-1$, and x_0 is the Most Significant Bit (MSB). As visualized in Figure 2, we decompose the modular addition into n smaller Boolean functions (a.k.a. full-/half-adders). Assuming that c_i for $0 \leq i \leq n-1$ are binary variables to represent the carry bits, we define $(z_i, c_i) = f(x_i, y_i, c_{i+1}) := (x_i \oplus y_i \oplus c_i, x_{i+1} \cdot y_{i+1} \oplus c_{i+1} \cdot (x_{i+1} \oplus y_{i+1}))$ for $1 \leq i \leq n-1$, where $c_{n-1} = 0$. Additionally, we define $z_0 = g(x_0, y_0, c_0) := x_0 \oplus y_0 \oplus c_0$. Next, we model the propagation of deterministic differential trails through f and g by CP constraints. For g , we can use the rules for modeling XOR in Proposition 2. Regarding f , following the same approach as [HGSE24], we consider it as an S-box: referring to its *Differential Distribution Table* (DDT), we identify differential transitions in which the difference is known with certainty in at least one output bit (referred to as deterministic bit-wise differential transitions), and express them as CP constraints. Proposition 4 and Proposition 5 briefly describe our CP constraints to model modular addition.

Proposition 4 (Full adder). *Assume that $(z, c') = f(x, y, c) = (x \oplus y \oplus c, x \cdot y \oplus c \cdot (x \oplus y))$, and let $\mathbf{x}, \mathbf{y}, \mathbf{c}, \mathbf{z}, \mathbf{c}'$ denote the integer variables with domain $\{-1, 0, 1\}$ to encode the*

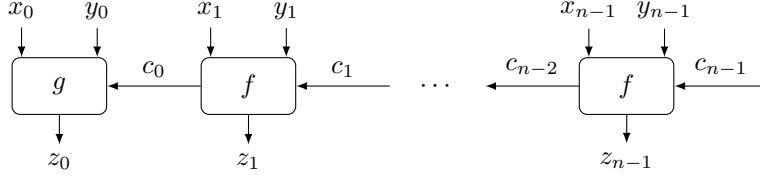


Figure 2: Representing the modular addition $X \boxplus Y$ using full-adders f and a half-adder g .

corresponding differences. Then, the following CP constraints model all valid bit-wise deterministic differential transitions through the full adder:

$$\text{FA}(x, y, c, z, c') := \begin{cases} \text{if } (x = 0 \wedge y = 0 \wedge c = 0) & \text{then } (z = 0 \wedge c' = 0) \\ \text{elseif } (x = 0 \wedge y = 0 \wedge c = 1) & \text{then } (z = 1 \wedge c' = -1) \\ \text{elseif } (x = 0 \wedge y = 1 \wedge c = 0) & \text{then } (z = 1 \wedge c' = -1) \\ \text{elseif } (x = 0 \wedge y = 1 \wedge c = 1) & \text{then } (z = 0 \wedge c' = -1) \\ \text{elseif } (x = 1 \wedge y = 0 \wedge c = 0) & \text{then } (z = 1 \wedge c' = -1) \\ \text{elseif } (x = 1 \wedge y = 0 \wedge c = 1) & \text{then } (z = 0 \wedge c' = -1) \\ \text{elseif } (x = 1 \wedge y = 1 \wedge c = 0) & \text{then } (z = 0 \wedge c' = -1) \\ \text{elseif } (x = 1 \wedge y = 1 \wedge c = 1) & \text{then } (z = 1 \wedge c' = 1) \\ \text{else} & (z = -1 \wedge c' = -1) \end{cases}$$

Proposition 5 (Modular Addition). Assume we express the modular addition $z = x \boxplus y$ as a composition of $n - 1$ full adders f along with a half adder g as explained before, and let x_i, y_i, z_i, c_i denote the integer variables with the domain $\{-1, 0, 1\}$ for the corresponding difference at bit positions x_i, y_i, z_i, c_i . Then the following constraints model the bit-wise deterministic differential transitions through modular addition:

$$\text{ModAdd} := \left(\bigwedge_{i=1}^{n-1} \text{FA}(x_i, y_i, c_i, z_i, c_{i-1}) \right) \wedge \text{XOR}(z_0, x_0, y_0, c_0) \wedge (c_{n-1} = 0). \quad (2)$$

To model bit-wise deterministic linear transitions, we follow a similar approach. In this case, for the vectorial Boolean function f, g , we refer to its *Linear Approximation Table* (LAT).

3.2 New CP Model to Identify Indirect Contradictions

We now provide a CP model based on satisfiability, which can identify both direct and indirect contradictions. In particular, we focus on the indirect contradictions first described in [SB18]. We first give the intuition for our approach. Assume no direct contradiction exists between the two deterministic differential trails propagated in opposite directions. However, what if we merge the information from the two deterministic propagations at a particular round and propagate this new information with probability one in both directions such that further propagation of this new information contradicts one of the original forward and backward propagations? If so, based on the proof by contradiction, we can conclude that the original two deterministic differential trails cannot exist simultaneously. To include these cases in our CP model, we extend it with new CP constraints that merge the information from both deterministic differential trails and then propagate the latest information with probability one in both directions. We equip this CP model with extra contradiction checkers between the original and new differential trails. Lastly, we include a constraint to ensure that at least one of the (direct or indirect) contradiction checkers is activated.

Modeling indirect contradictions. Suppose our goal is to find an ID or ZC distinguisher for r_D rounds of a block cipher E denoted by E_D . We first split E_D into two parts: an upper part E_U that covers r_M rounds, and a lower part E_L covering the remaining $(r_D - r_M)$ rounds. Hereafter, we refer to the trails discovered for E_U (E_L) as the upper (lower) trail. The internal state of E_U after r rounds is denoted by xu_r , where $0 \leq r \leq r_M$. Likewise, we denote the internal state of E_L after r rounds by xl_{r_D-r} , where $0 \leq r \leq (r_D - r_M)$. Therefore, xu_{r_M} and xl_{r_M} correspond to the same internal state at the junction of the two sub-ciphers.

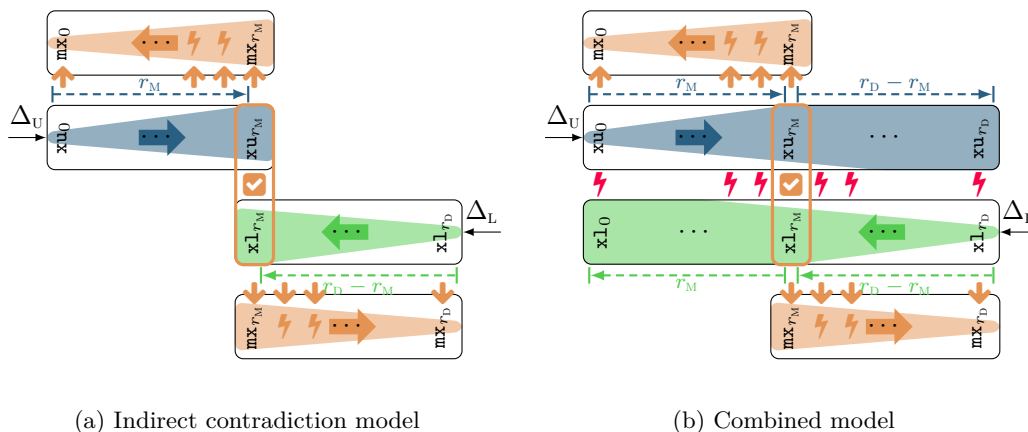


Figure 3: Model for impossible-differential distinguishers with indirect contradiction.

Let xu_r and xl_r represent the difference patterns of the state variables xu_r and xl_r , respectively, as illustrated in Figure 3. In particular, $xu_r[i]$ (or $xl_r[i]$) is an integer variable with a domain of $\{-1, 0, 1\}$, depicting the difference pattern in the i -th bit of xu_r (or xl_r). We model the propagation of the deterministic truncated differential trail through E_U and E_L in the encryption (forward) and decryption (backward) directions as separate CSP models. For this purpose, we utilize the propagation rules from [HGSE24, HSE23] along with our new rules from Subsection 3.1. We denote the model for the propagation of deterministic truncated trails through E_U and E_L^{-1} as $\text{CSP}_U(xu_0, \dots, xu_{r_M})$, and $\text{CSP}_L(xl_{r_M}, \dots, xl_{r_D})$, respectively. Also, let f denote the round function of block cipher E . We represent the CP constraints for the propagation of deterministic truncated trails over f (resp. f^{-1}) as $\mathbf{f}_U(\mathbf{x}, \mathbf{y})$ (resp. $\mathbf{f}_L(\mathbf{y}, \mathbf{x})$), where \mathbf{x} (resp. \mathbf{y}) denotes the activeness pattern at the input (resp. output) of f .

Now, we explain how we model the merging of information from the upper and lower trails and identify the indirect contradictions by defining some new CP variables and constraints. In round r_M , we need to merge the information from the upper and lower trails at the junction of E_U and E_L . After merging the information in round r_M , we need to propagate this new information forward and backward. After each round of propagation of the new information, we check whether the new activeness pattern is consistent with the activeness pattern at the corresponding state from the original propagation. Moreover, to determine the activeness pattern at each round, we must merge the information from the previous round in the new propagation with the information from the corresponding state in the original propagation.

To this end, for the internal state at each round, we define three new types of integer variables \mathbf{mx}_r , \mathbf{mx}'_r , and \mathbf{mc}_r with a domain $\{-1, 0, 1\}$. The integer variable \mathbf{mx}'_r encodes the information propagated from the previous round, \mathbf{mx}_r encodes the result of merging the information from the previous round with the information from the corresponding state at the original propagation, and \mathbf{mc}_r checks if there is a contradiction between the new propagation and the original one. A more detailed explanation regarding these variables is

as follows.

To merge the information at each round of propagation, we first define the predicate $\text{merge}(\mathbf{xu}_{r_M}[i], \mathbf{xL}_{r_M}[i], \mathbf{mx}_{r_M}[i], \mathbf{mc}_{r_M}[i])$:

$$\begin{cases} \text{if } (\mathbf{xu}_{r_M}[i] = -1) & \text{then } (\mathbf{mx}_{r_M}[i] = \mathbf{xL}_{r_M}[i] \wedge \mathbf{mc}_{r_M}[i] = 0) \\ \text{elseif } (\mathbf{xL}_{r_M}[i] = -1) & \text{then } (\mathbf{mx}_{r_M}[i] = \mathbf{xu}_{r_M}[i] \wedge \mathbf{mc}_{r_M}[i] = 0) \\ \text{elseif } (\mathbf{xu}_{r_M}[i] = \mathbf{xL}_{r_M}[i]) & \text{then } (\mathbf{mx}_{r_M}[i] = \mathbf{xu}_{r_M}[i] \wedge \mathbf{mc}_{r_M}[i] = 0) \\ \text{else} & (\mathbf{mc}_{r_M}[i] = 1) \end{cases}$$

Using this predicate, we start from the meeting point of E_U and E_L and merge the activeness patterns from the upper and lower trails at the junction of E_U and E_L :

$$\text{CSP}_M(\mathbf{xu}_{r_M}, \mathbf{xL}_{r_M}, \mathbf{mx}_{r_M}, \mathbf{mc}_{r_M}) := \bigwedge_{i=0}^{n-1} \text{merge}(\mathbf{xu}_{r_M}[i], \mathbf{xL}_{r_M}[i], \mathbf{mx}_{r_M}[i], \mathbf{mc}_{r_M}[i]) \quad (3)$$

Now, we must propagate the result of merging, namely \mathbf{mx}_{r_M} , to both encryption and decryption direction with certainty. We first explain the backward propagation. Assume that we aim to determine the activeness pattern \mathbf{mx}_{r-1} at round $r-1$ based on the activeness pattern \mathbf{mx}_r at round r in the new propagation and also the activeness pattern \mathbf{xu}_{r-1} at round $r-1$ in the original propagation. For this purpose, using $\mathbf{f}_L(\mathbf{mx}_r, \mathbf{mx}'_{r-1})$ we first propagate \mathbf{mx}_r into \mathbf{mx}'_{r-1} backward through f . Next, we merge the activeness patterns at \mathbf{mx}'_{r-1} and \mathbf{xu}_{r-1} into \mathbf{mx}_{r-1} , and finally, we check if there is a contradiction between \mathbf{mx}'_{r-1} and \mathbf{xu}_{r-1} using \mathbf{mc}_{r-1} . We use the following constraint for this purpose:

$$\begin{aligned} \text{CSP}_B(\mathbf{xu}_0, \dots, \mathbf{xu}_{r_M-1}, \mathbf{mx}_0, \dots, \mathbf{mx}_{r_M}, \mathbf{mc}_0, \dots, \mathbf{mc}_{r_M-1}) := & \quad (4) \\ \bigwedge_{r=1}^{r_M} \left(\mathbf{f}_L(\mathbf{mx}_r, \mathbf{mx}'_{r-1}) \wedge \left(\bigwedge_{i=0}^{n-1} \text{merge}_i(\mathbf{xu}_{r-1}[i], \mathbf{mx}'_{r-1}[i], \mathbf{mx}_{r-1}[i], \mathbf{mc}_{r-1}[i]) \right) \right) \end{aligned}$$

Similarly, we model the merging and propagation in the forward direction. For each round r , where $r_M \leq r < r_D$, we first use $\mathbf{f}_U(\mathbf{mx}_r, \mathbf{mx}'_{r+1})$ to propagate \mathbf{mx}_r into \mathbf{mx}'_{r+1} . Next, we merge \mathbf{mx}'_{r+1} with \mathbf{xL}_{r+1} into \mathbf{mx}_{r+1} and check if there is a contradiction between \mathbf{mx}_{r+1} and \mathbf{xL}_{r+1} using \mathbf{mc}_{r+1} . To this end, we use the following constraint:

$$\begin{aligned} \text{CSP}_F(\mathbf{xL}_{r_M+1}, \dots, \mathbf{xL}_{r_D}, \mathbf{mx}_{r_M}, \dots, \mathbf{mx}_{r_D}, \mathbf{mc}_{r_M+1}, \dots, \mathbf{mc}_{r_D}) := & \quad (5) \\ \bigwedge_{r=r_M}^{r_D} \left(\mathbf{f}_U(\mathbf{mx}_r, \mathbf{mx}'_{r+1}) \wedge \left(\bigwedge_{i=0}^{n-1} \text{merge}_i(\mathbf{xL}_{r+1}[i], \mathbf{mx}'_{r+1}[i], \mathbf{mx}_{r+1}[i], \mathbf{mc}_{r+1}[i]) \right) \right) \end{aligned}$$

We must ensure that at least one of the contradiction checker constraints is met. To achieve this, we introduce the following constraint. It ensures that there is a mismatch between the new backward propagation over E_U^{-1} and the original forward propagation over E_U , or the new forward propagation over E_L , and the original backward propagation over E_L^{-1} in at least one bit across the entire distinguisher.

$$\text{CSP}_C(\mathbf{mc}_0, \mathbf{mc}_1, \dots, \mathbf{mc}_{r_D}) := \bigvee_{r=0}^{r_D-1} \left(\bigvee_{i=0}^{n-1} (\mathbf{mc}_r[i] = 1) \right) \quad (6)$$

The conjunction of the CSP models above, denoted by CSP_D , creates a unified CP model based on satisfiability whose feasible solutions are impossible differential distinguishers:

$$\text{CSP}_D := \text{CSP}_U \wedge \text{CSP}_L \wedge \text{CSP}_M \wedge \text{CSP}_F \wedge \text{CSP}_B \wedge \text{CSP}_C$$

Hence, when provided r_D and r_M , this model yields a distinguisher for r_D rounds of the block cipher where either we find a contradiction in r_M round (in case of direct contradiction),

or we can find some indirect contradiction where r_M is the round where first time merging happens ($r_M = 8$ in Figure 9 in Section 5). We explain our model for ID distinguishers, but the same approach applies to ZC distinguishers. We provide a more detailed analysis of the attack model to identify indirect contradictions, as explained in Figure 9 (Section 5) in the subsequent section.

Combined model of indirect and direct contradictions. We can extend our idea to construct a combined CSP model capable of identifying both direct and indirect contradictions. Let $\text{CSP}_U(\mathbf{xu}_0, \mathbf{xu}_1, \dots, \mathbf{xu}_{r_D})$ and $\text{CSP}_L(\mathbf{x1}_0, \mathbf{x1}_1, \dots, \mathbf{x1}_{r_D})$ denote the CSP models for the forward and backward propagations through E_D , and E_D^{-1} , respectively. We extend the models to the full r_D rounds. Next, following a similar approach, we can construct the CSP models CSP_M , CSP_B , and CSP_F as described in Equation 3, Equation 4, and Equation 5, respectively. Finally, we add the following constraints to ensure the inconsistency between the four deterministic propagations:

$$\text{CSP}_C(\mathbf{xu}_0, \dots, \mathbf{xu}_{r_D}, \mathbf{x1}_0, \dots, \mathbf{x1}_{r_D}, \mathbf{mc}_0, \mathbf{mc}_1, \dots, \mathbf{mc}_{r_D}) := \bigvee_{r=0}^{r_D-1} \left(\bigvee_{i=0}^{n-1} (\mathbf{mc}_r[i] = 1) \vee (\mathbf{xu}_r[i] + \mathbf{x1}_r[i] = 1) \right)$$

The conjunction of the CSP models above, denoted by CSP_D , creates a unified CP model based on satisfiability that can successfully detect ID distinguishers based on direct and indirect contradictions:

$$\text{CSP}_D := \text{CSP}_U \wedge \text{CSP}_L \wedge \text{CSP}_M \wedge \text{CSP}_F \wedge \text{CSP}_B \wedge \text{CSP}_C$$

We apply the above idea to several ARX and AndRX ciphers and discover several new distinguishers. Section 5 elaborates on the details of our applications.

3.3 Modeling ZC Distinguishers

Both ID and ZC distinguishers primarily leverage the miss-in-the-middle technique in their construction. In Subsection 3.2, we provided a CP model based on satisfiability to identify direct and indirect contradictions when searching for ID distinguishers. A similar approach applies to finding ZC distinguishers. However, we encountered specific challenges when developing the CP model for ZC distinguishers based on satisfiability for ARX ciphers. This section delves into these challenges and presents our approach to overcome them partially. First, we provide some basic rules to model the propagation of deterministic bit-wise linear trails through XOR and Branching operations.

Proposition 6 (XOR_L). *Suppose $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ is such that $f(x_0, x_1, \dots, x_{n-1}) = y$ where $y = x_0 \oplus x_1 \oplus \dots \oplus x_{n-1}$, the valid deterministic linear trails satisfy*

$$\text{XOR}_L(\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1}, y) := \bigwedge_{i=0}^{n-1} (\mathbf{x}_i = y)$$

where \mathbf{x}_i , and y are integer variables with the domain $\{-1, 0, 1\}$, for all $0 \leq i \leq (n-1)$, representing the activeness pattern of linear mask in x_i and y , respectively.

For non-deterministic linear transitions, the propagation rule for branching is the same as the rule for non-deterministic differential transitions through XOR. However, we cannot directly apply this duality between an XOR's differential behavior and a branching point's linear behavior when dealing with deterministic transitions. Consider a branching point $f(x) = (y_0, y_1)$, where $y_0 = y_1 = x$. Let \mathbf{x} , y_0 , and y_1 be integer variables with a domain of

$\{-1, 0, 1\}$ representing the activeness pattern of linear masks for x , y_0 , and y_1 , respectively. Suppose $\mathbf{x} = 1$ with certainty. Then, the linear mask of (y_0, y_1) can take either $(1, 0)$ or $(0, 1)$. Therefore, $y_0 = y_1 = -1$. The same is true if $\mathbf{x} = 0$. Additionally, if $\mathbf{x} = -1$, then $y_0 = y_1 = -1$. This example demonstrates that if we limit ourselves to using a 3-digit encoding (i.e., $\{-1, 0, 1\}$) for modeling the propagation of deterministic linear trails through a branching point, we quickly lose information, and the entire state becomes “-1” (unknown) very quickly. However, suppose that, due to the location of the branching point within the round function (e.g., Feistel structure), the activeness pattern of linear masks at x and y_0 can be derived based on information from the previous round. In that case, we can utilize Proposition 7 to model the propagation of deterministic linear transitions through the branching point.

Proposition 7 (Branching_L). *Suppose $f : \mathbb{F}_2 \rightarrow \mathbb{F}_2^n$ is such that $f(x) = (y_0, y_1, \dots, y_{n-1})$ where $y_0 = y_1 = \dots = y_{n-1}$. Also assume that the linear masks of x and y_0, y_1, \dots, y_{n-2} are determined in advance. Then, the valid transitions for deterministic linear trails through the branching point satisfy*

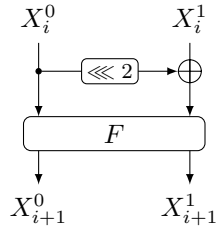
$$\text{Branch}_L(\mathbf{x}, y_0, \dots, y_{n-1}) := \begin{cases} \text{if} & \left(\bigvee_{i=0}^{n-2} (y_i = -1) \right) \vee (\mathbf{x} = -1) \quad \text{then } y_{n-1} = -1 \\ \text{else} & y_{n-1} = \mathbf{x} + y_0 + \dots + y_{n-2} \pmod{2} \end{cases}$$

where \mathbf{x} and y_i are integer variables with the domain $\{-1, 0, 1\}$ for all $0 \leq i \leq (n-1)$, representing the activeness pattern of the linear mask in x and y_i , respectively.

In addressing the challenge of modeling the propagation of deterministic linear trails through the branching point, our initial focus is on AndRX ciphers, with particular attention to two prominent ones: SIMON [BSS⁺15] and SIMECK [YZS⁺15]. We outline our strategy for modeling ZC distinguishers for SIMON, but the same approach applies to SIMECK. The idea is to rearrange the state array such that we can model the round function as several consecutive S-boxes along with some branching points, such that the branching points satisfy the requirements of Proposition 7. Then we use the rules for propagation of deterministic linear trails through the S-boxes in [HGSE24], together with our rule for modeling particular branching points (Proposition 7) to model the whole round function.

In what follows, we explain the details of our workaround for SIMON. Let X_r^0 , and X_r^1 represent the two n -bit input words to the r -th round function of SIMON. The output of the r -th round X_{r+1}^0 , X_{r+1}^1 is computed as:

$$\begin{aligned} X_{r+1}^1 &= X_r^0 \\ X_{r+1}^0 &= ((X_r^0 \lll 8) \odot (X_r^0 \lll 1)) \oplus ((X_r^0 \lll 2) \oplus X_r^1) \oplus K_r \end{aligned}$$



(a) One round of SIMON

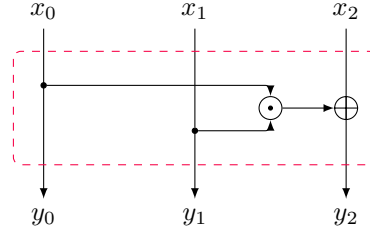
(b) S_{SIMON} , the core operation of F

Figure 4: Round function structure of SIMON, where F is defined by $(y^0, y^1) = F(x^0, x^1) = (x^1 \oplus (x^0 \lll 8) \odot (x^0 \lll 1), x^0)$ and can be expressed in terms of the 3-bit function S_{SIMON} .

Now, we can express the round function of SIMON as illustrated in Figure 4. In Figure 4, we represent the core operation of the function F : a Toffoli gate (Figure 4b). We treat the red dotted box as a 3×3 S-box, referred to as S_{SIMON} hereafter. As illustrated in Figure 4b, we represent the input and output of S_{SIMON} by $x = (x_0, x_1, x_2)$, $y = (y_0, y_1, y_2)$, respectively. The algebraic normal form (ANF) of this S-box is as follows:

$$y_0 = x_0, \quad y_1 = x_1, \quad y_2 = (x_0 \odot x_1) \oplus x_2$$

We can represent the function F as a sequence of these S-boxes. Next, we use Proposition 8 to encode the propagation of deterministic bit-wise linear transitions through S_{SIMON} .

Proposition 8 (Modeling deterministic linear behavior of S_{SIMON}). *Assume that $\mathbf{x} = (x_0, x_1, x_2)$ and $\mathbf{y} = (y_0, y_1, y_2)$ are integer variables with domain $\{-1, 0, 1\}$ to encode the activeness pattern of linear masks at the input and output of S_{SIMON} , respectively. The CP constraints to describe all valid deterministic bit-wise linear transitions through S_{SIMON} can be derived from its LAT and are summarized in Appendix A.*

Let $\mathbf{x}^{(r)} = (x_0^r, x_1^r)$ and $\mathbf{x}^{(r+1)} = (x_0^{r+1}, x_1^{r+1})$ denote the activeness pattern of deterministic linear masks at the input and output of the r -th round of SIMON, respectively. We also assume that y_0^r and z_0^r represent the activeness pattern for the linear masks of the two branches of x_0^r (z_0^r is one of the branches of x_0^r which further proceeds into the function F). Since the activeness patterns x_0^r and x_1^r are derived from the preceding round, using the rule to model a deterministic linear transition through XOR operation, y_0^r is actually derived from x_1^r . Then, as depicted in Figure 4a, it is evident that this branch fulfills the conditions outlined in Proposition 7. As a result, according to Proposition 7, we can use the following rule to model this branching point:

$$\text{Branch}_L(x_0^r[i], y_0^r[i], z_0^r[i]) := \begin{cases} \text{if} & (x_0^r[i] = -1 \vee y_0^r[i] = -1) \text{ then } z_0^r[i] = -1 \\ \text{else} & z_0^r[i] = x_0^r[i] + y_0^r[i] \bmod 2, \end{cases}$$

where $x_0^r[i]$, $y_0^r[i]$, and $z_0^r[i]$ are integer variables with domain $\{-1, 0, 1\}$, for all $0 \leq i \leq (n-1)$. It is important to note that, for SIMON, and Simeck, the AND gates in one round share certain input bits. Specifically, For SIMON, the AND function operates on X_r^0 as follows:





$$Y_r^0 = (X_r^0 \lll 8) \odot (X_r^0 \lll 1)$$

where X_r^0 represent the left n -bit input words to the r -th round function of SIMON. Therefore, AND gates within a round share certain input bits, e.g., $Y_r^0[0] = X_r^0[8] \odot X_r^0[1]$, and $Y_r^0[7] = X_r^0[15] \odot X_r^0[8]$, which implies these two AND operations share the input bit $X_r^0[8]$. Our tool accounts for this shared input bit situation while modeling ID/ZC distinguishers or ID key recovery, including for Simeck.

3.4 Application of Our Distinguisher Modeling.

To demonstrate the utility of our improved model for finding ID distinguishers, we applied it to several AndRX (SIMON, Simeck) and ARX (SPECK, LEA, ChaCha, SipHash, Chaskey) ciphers. Additionally, we successfully utilized our bit-wise modeling to find ZC distinguishers of AndRX ciphers (SIMON, Simeck). While searching for ID and ZC distinguishers, we did not fix any input/output differences. Each bit can take one of three values: 0, 1, or -1 (indicating it can be either 0 or 1). Therefore, having more bits with the value -1 at the input/output of the distinguisher leads to a larger set of distinguishers. Solving one instance of the model returns one solution with several -1s at the input and output, essentially providing a truncated ID/ZC distinguisher. Since bit positions with -1 can take either 0 or 1, the returned solution represents a group or cluster

of distinguishers. If there are n input/output bits with -1 , we have 2^n distinguishers. Additionally, when searching for ID/ZC distinguishers, we include the objective function $\min. (\sum_{i=0}^{n-1} \mathbf{xu}_0[i] + \sum_{i=0}^{n-1} \mathbf{x1}_{r_D}[i])$ to maximize the number of differentially active bits at the input and output. The number of distinguishers increases with the number of unknown bits at the input and output of the distinguishers.

While analyzing AndRX and ARX ciphers, we found that ID and ZC distinguishers for Simeck rely on indirect contradiction, unlike other ciphers where the best distinguishers use direct contradiction. This technique reveals new trails for Simeck, likely due to its weaker diffusion properties. Figure 6 (Section 5), Figure 15 (Appendix F), Figure 40 (Appendix F), and Figure 9 (Section 5) illustrate some of the ID/ZC distinguishers discovered by our tool. The unknown bits (difference or linear mask) in the forward and backward propagations are depicted by , and , respectively. Also, the bit difference (linear mask) 1 (this means active bit) is illustrated by  and  in the forward and backward propagations, respectively. According to our modeling, for all $0 \leq r \leq r_D$, the CP variables \mathbf{xu}_r and $\mathbf{x1}_r$ are represented in the upper triangle and lower triangle of the r th state, $L_r||R_r$, respectively. For instance, Figure 6 in Section 5 shows the ID distinguishers for 6-round SPECK-96, Figure 15 in Appendix F for 6-round SPECK-128, and Figure 40 in Appendix F for 19-round SIMON-128. In Figure 6 (Section 5), the 42-th bit of R_2 has difference values of 1 and 0 in the forward and backward propagation, respectively, indicating a 0-1 contradiction in 6-round SPECK-96.

Here, Figure 9 (Section 5) illustrates the ZC distinguisher of 15-round Simeck, demonstrating the use of indirect contradiction. According to our model, the CP variable \mathbf{xu}_r is represented in the upper triangle of the r -th state, $L_r||R_r$ (on the left side column of Figure 9 (Section 5)) for all $0 \leq r \leq r_M = 8$. Similarly, the CP variable $\mathbf{x1}_r$ is represented in the lower triangle of the r -th state, $L_r||R_r$ (on the left side column of Figure 9 (Section 5)) for all $8 = r_M \leq r \leq r_D = 15$.

Following our model to find an indirect contradiction, the variable \mathbf{mx}_{r_M} (represented in both the upper and lower triangle of $L_M||R_M$, on the right side column of Figure 9 (Section 5) with $r_M = 8$) is the result of merging of two variables \mathbf{xu}_{r_M} (upper triangle of $L_M||R_M$, on the left side column of Figure 9 (Section 5) with $r_M = 8$), and $\mathbf{x1}_{r_M}$ (lower triangle of $L_M||R_M$, on the left side column of Figure 9 (Section 5) with $r_M = 8$). Now, for all $0 \leq r \leq r_M$, we first propagate \mathbf{mx}_r (lower triangle of the state $L_r||R_r$ on the right side column of Figure 9 (Section 5)) to \mathbf{mx}'_{r-1} (lower triangle of the state $L'_{r-1}||R'_{r-1}$ on the right side column of Figure 9 (Section 5)) backward through one round function. Then, we merge activeness pattern of \mathbf{mx}'_{r-1} , and \mathbf{xu}_{r-1} into \mathbf{mx}_{r-1} , and check whether there is a contradiction between \mathbf{mx}'_{r-1} , and \mathbf{xu}_{r-1} . For example, we can see in Figure 9 (Section 5), the value of $\mathbf{xu}_2[0]$ (depicted in the upper triangle of the 0-th bit of L_2 in the left column) is 1, while the value of $\mathbf{mx}'_2[0]$ (depicted in the lower triangle of the 0-th bit of L'_2 in the right column) is 0. This implies an indirect contradiction occurs in 15-round Simeck-48. Similarly, for all $r_M \leq r \leq r_D$, we propagate \mathbf{mx}_r (upper triangle of the state $L_r||R_r$ on the right side column of Figure 9 (Section 5)) into \mathbf{mx}'_{r+1} (upper triangle of the state $L'_{r+1}||R'_{r+1}$ on the right side column of Figure 9 (Section 5)), and merge \mathbf{mx}'_{r+1} with $\mathbf{x1}_{r+1}$ into \mathbf{mx}_{r+1} and check if there is a contradiction between \mathbf{mx}_{r+1} and $\mathbf{x1}_{r+1}$. This type of contradiction (represented in Figure 9 (Section 5)) cannot be detected by the previous methods.

Unlike previous tools based on unsatisfiability [ST17, CCJ⁺21], which require multiple executions by fixing the input and output of the distinguisher in each run, our tool efficiently identifies a group of ID/ZC distinguishers (or truncated distinguishers) in just one execution, without the need to fix input/output differences (as detailed in Table 4, Table 5, Table 6, and Table 7 in Section 5). A single execution of our tool terminates within a few seconds (or minutes) for our models based on direct contradiction (or indirect contradiction) when running on a regular laptop.

3.5 Comparison of Our Distinguisher Modeling to Prior Methods.

Following the approach in [HSE23, HGSE24], our primary objective with our CP models for distinguishers is to create satisfiability-based models that can extend to a unified COP for finding complete ID and ZC attacks. In contrast to earlier tools such as [CCJ⁺21, ST17, DF16], which depend on unsatisfiability and necessitate fixing input/output differences or linear masks to identify distinguishers, our attack model is based on satisfiability. This approach eliminates the need to fix input/output differences or linear masks. Moreover, our bit-wise CP model for distinguishers demonstrates a novel capability by identifying ID and ZC distinguishers based on indirect contradictions, improving on previous ones [HSE23, HGSE24], which only handled direct contradictions. We tested our model on various ciphers. Interestingly, only Simeck yielded longer distinguishers based on indirect contradiction compared to the direct contradiction approach. In other AndRX and ARX applications, the improved distinguishers we found, along with the longest existing ones, could be explained or found through direct contradiction. Still, we believe this doesn't lessen our contribution with indirect contradiction, as our model is overall more complete than its predecessor [HSE23, HGSE24], since the previous models in [HSE23, HGSE24] cannot find the longest existing ID and ZC distinguishers of Simeck as we did. Furthermore, our enhanced bit-wise model, which utilizes satisfiability for identifying ID/ZC distinguishers, can be extended into a unified COP model to uncover full ID attacks on both SIMON and Simeck. Consequently, our model enables the discovery of improved ID attacks on all versions of Simeck, a feat beyond the capabilities of previous tools [HSE23, HGSE24].

Although our primary objective in modeling distinguishers is not to develop tools for proving the non-existence of ID/ZC distinguishers, it is worth discussing whether our model could also fulfill this purpose. We cannot claim that our tool can capture the longest possible ID/ZC distinguisher or that our tools can be used to prove the non-existence of ID/ZC distinguishers because we rely on the assumptions of round independence and subkey independence. However, this limitation is not exclusive to our approach; it applies to all existing tools for finding ID/ZC distinguishers. Even the tool developed in [ST17], which captures complex contradictions, requires checking every possible input/output combination for non-existence, which is impractical. Developing a tool for proving non-existence remains an exciting future direction. However, our current focus is on creating a satisfiability-based model for ID/ZC distinguishers that can be extended for key recovery attacks. Nonetheless, our tools have demonstrated their effectiveness in identifying the longest existing distinguishers, generating numerous new trails, and revisiting old ones within a few minutes on a regular laptop. These applications include SIMON, Simeck, SPECK, ChaCha, LEA, SipHash, and Chaskey.

Concerning cross-round dependencies, akin to previous tools for searching for ID/ZC, we assume that consecutive non-linear operations (rounds) are statistically independent. Consequently, like previous tools, any ID/ZC distinguisher we identify remains valid. However, it is worth noting that, similar to previous tools, we may overlook some ID/ZC distinguishers that are detectable only by considering cross-round dependencies. Nevertheless, it is important to acknowledge that addressing cross-round dependencies has been an open problem in the context of ID/ZC distinguishers so far. While we are aware of recent works that consider cross-round dependencies for differential characteristics (e.g., [PT22]), these works primarily address dependency issues for differential characteristics (single trail) and are not applicable to differentials (i.e., differential hulls) and ID/ZC distinguishers. Therefore, we have chosen to keep our model simple, efficient, and based on satisfiability to facilitate its extension for key recovery, which has been the main motivation of this work and the underlying methods [HSE23, HGSE24]. Regarding key recovery, our approach aligns with previous works, wherein we utilize deterministic properties and refrain from exploiting any properties that may conflict with the fact that certain non-linear operations may have dependencies.

In summary, our new bit-wise models bridge the gap in developing satisfiability-based models for distinguishers applicable to ARX and AndRX ciphers. Along with that, they upgrade the distinguisher models by handling the detection of both direct and indirect contradictions. In Section 4, we show how to extend our CP models for distinguishers to a unified CP model for finding complete ID attacks.

4 Modeling the Key-Recovery for Impossible Differentials

This section presents a framework to extend the distinguisher model into a unified model for discovering complete ID attacks, including the key recovery for AndRX ciphers. Our framework takes four integer parameters (r_B , r_D , r_M , r_F) that represent the lengths of specific parts in Figure 1, where r_M specifies the merging point throughout the distinguisher part as explained in Subsection 3.2 and generates an optimum ID attack for $r = r_B + r_D + r_F$ rounds. When searching for the full ID attack, our objective is to minimize the overall time complexity while also ensuring that memory and data complexity stay below specified limits.

Following the discussion on the complexity formulas described in Subsection 2.1, c_B , c_F , $|\Delta_B|$, $|\Delta_F|$, and $|k_B \cup k_F|$ are the critical parameters which directly affect the overall complexity of the ID attack. To determine (c_B, Δ_B) , we need to model the propagation of truncated differential trails through E_B taking the probability of all transitions (truncated difference \rightarrow fixed difference called as probabilistic transitions) into account. To determine k_B , we need to detect the state bits that require their difference or data values through partial encryption over E_B . The same applies for partial decryption over E_F^{-1} to determine c_F , $|\Delta_F|$, and k_F . Moreover, to determine the actual size of $k_B \cup k_F$, we can consider the idea of the *equivalent sub-key technique* for Feistel ciphers or the *key bridging technique*.

4.1 Brief Overview of the COP model

Our bit-wise key recovery model consists of 4 sub-models as follows:

- **Modeling the distinguisher.** We model the distinguisher part according to the method explained in Section 3.
- **Modeling the difference propagation through E_B and E_F .** In this part, we model truncated differential propagation $\Delta_B \xleftarrow{E_B^{-1}} \Delta_U$ and truncated differential propagation $\Delta_L \xrightarrow{E_F} \Delta_F$. There is a similarity between this modeling and the modeling of distinguishers: Both use integer variables of domain $\{-1, 0, 1\}$ for each state bit. The deterministic forward and backward trails are modeled using the simple rules for deterministic propagation through the building block operations, e.g., XOR, AND, Branch, etc. We also model the number of filters c_B and c_F using new binary variables and constraints to encode the probability of $\Delta_B \xrightarrow{E_B} \Delta_U$ and $\Delta_L \xleftarrow{E_F^{-1}} \Delta_L$.
- **Modeling guess-and-determine in outer parts.** In this part, we model the constraints over E_B and E_F to detect the state bits whose difference or data values must be known to verify the differences Δ_U starting from Δ_B through E_B , and Δ_L starting from Δ_F through E_F^{-1} . We can find the key bits involved in the key recovery attack by utilizing this information.
- **Modeling the complexity formula.** In this component, we model the complexity formulas described in Subsection 2.1, and finally, we set the objective function as

Minimize T , where T is the total time complexity (in our CP model, we minimize the maximal term in time complexity).

All the variables in our model are either integer or binary variables with domain $\{-1, 0, 1\}$, whereas the variables related to the complexity formula are real numbers. Our tool exclusively relies on four integer inputs, delineating the lengths of E_B (r_B), E_D (r_D), E_U (r_M), and E_F (r_F). Users have the flexibility to experiment with various length configurations for these components to discover an optimal ID attack strategy. Additionally, the model's objective function can be adapted to minimize data or memory complexities under constraints such as time or other parameters.

4.2 Detailed Description of Bit-Wise Key Recovery Model and Application to SIMON

In this section, we describe our bit-wise model for a full ID attack on AndRX ciphers in more detail. To this end, we construct the COP model for finding a full ID attack on SIMON as an example. Given four integer numbers r_B , r_D , r_M , r_F , we model the full ID attack on $r = r_B + r_D + r_F$ rounds of SIMON, where r_D is the length of the distinguisher, and r_M is the round where one can find a contradiction (in case of direct contradiction), or one can find some indirect contradiction where r_M is the round where the first time merging operation happens ($r_M = 8$ in Figure 9 in Section 5). To perform key recovery (as shown in Figure 1), we must extend the distinguisher by a few rounds at both ends. r_B and r_F are the lengths of extended backward and forward parts, respectively.

4.2.1 Modeling the Distinguisher

First, we want to model the difference propagation through the round function of SIMON. For the detailed structure of the round function SIMON, please refer to Subsection D.1. Here, we define xu_r^0 and xu_r^1 to be the CP variables corresponding to the two n -bit input words to the r -th round function of SIMON, where the block size of SIMON is $2n$. In more detail, for all $0 \leq i \leq (n-1)$, $xu_r^0[i]$, and $xu_r^1[i]$ are integer variables with domain $\{-1, 0, 1\}$. In the data path of SIMON, the operations like AND and XOR can change the difference pattern of the state, and the rotation (ROT) can change the position of the difference pattern of the state while propagating the deterministic differences. We described the rules for deterministic differential propagation through these basic operations in Subsection 3.1. Then, we construct the model CSP_U for the upper part as described in Algorithm 1 in Appendix B. Similarly, we can construct CSP_L . Along with this, we also build the CSP_M , CSP_B , CSP_F , and CSP_C according to Equation 3, Equation 4, Equation 5, and Equation 6, respectively. The combined CSP model is $CSP_D := CSP_U \wedge CSP_L \wedge CSP_M \wedge CSP_F \wedge CSP_B \wedge CSP_C$. Therefore, any feasible solution of CSP_D corresponds to an ID distinguisher for SIMON with block size $2n$.

4.2.2 Modeling the Difference Propagation in Outer Parts

To model the deterministic difference propagations $\Delta_B \xleftarrow{E_B^{-1}} \Delta_U$ and $\Delta_L \xrightarrow{E_F} \Delta_F$, we define an integer variable of domain $\{-1, 0, 1\}$ for each state bit to indicate whether its difference value is 0, 1, or unknown. Hence, utilizing the propagation rules for deterministic differential propagation through basic operations (AND, XOR, ROT), we can determine the deterministic difference backward propagation starting from Δ_U and the deterministic difference forward propagation starting from Δ_L .

To model the probability of difference propagations $\Delta_B \xrightarrow{E_B} \Delta_U$ and $\Delta_L \xleftarrow{E_F^{-1}} \Delta_F$, we should identify probabilistic transitions. The probabilistic transitions in our modeling are the truncated difference \rightarrow fixed difference throughout XOR operations. For

example, if at least one of the two input differences of the XOR operation is truncated (unknown) but its output difference is fixed, then we have a **truncated difference** \rightarrow **fixed difference** transition. We call such a transition probabilistic. Given that the primary source of probabilistic transitions through $\Delta_B \rightarrow \Delta_U$ and $\Delta_L \leftarrow \Delta_F$ in our modeling are probabilistic transitions through the XOR operations, we provide a basic rule to identify the probabilistic transitions for XOR. Let $z = x \oplus y$, where $x, y, z \in \mathbb{F}_2$. Additionally, $dx, dy, dz \in \{-1, 0, 1\}$ are integer variables to encode the difference at the input and output of the XOR operation. We define a binary variable **cb** to indicate whether there is a probabilistic transition over the corresponding XOR operation. According to our definition, the binary variable **cb** for each bit: **cb** = 0 (no probabilistic transition) means that the differential propagation happens with probability one; **cb** = 1 (**truncated difference** \rightarrow **fixed difference**) means probability 1/2. The constraints for deterministic propagation already avoid cases with probability zero. Hence, the probability of difference propagations from $\Delta_B \rightarrow \Delta_U$ is $2^{-|\mathcal{F}|}$, where \mathcal{F} is the set of all bits where **cb** = 1. In our model, we look at this in terms of bit-conditions, not probability. Then, we use the following constraint to determine the value of **cb**:

$$\text{XOR}_{dp}(dx, dy, dz, cb) := \text{if } (dz \geq 0 \wedge (dx = -1 \vee dy = -1)) \text{ then } cb = 1 \text{ else } cb = 0$$

We use this constraint for each XOR throughout E_B and E_F . For this purpose, we define a binary variable $cb_r^j[i]$ ($cf_r^j[i]$) for each XOR operation in the r -th round of E_B (E_F), where $0 \leq i \leq (n-1)$, and $1 \leq j \leq t$ such that t is the total number of XOR operations in one round function. For SIMON, there are two XOR operations in one round. Algorithm 2 in Appendix B describes our model for difference propagation over E_B . Similarly, we can construct our model for difference propagation over E_F . Finally, we combine CSP_B^{dp} and CSP_L^{dp} into $\text{CSP}_{DP} := \text{CSP}_B^{dp} \wedge \text{CSP}_F^{dp}$ to model the difference propagation through the outer parts.

4.2.3 Modeling Guess-and-Determine in Outer Parts

In this component, we detect the state bits whose difference or value or both are needed for checking the bit conditions in $\Delta_B \xrightarrow{E_B} \Delta_U$ and $\Delta_L \xleftarrow{E_F^{-1}} \Delta_F$. We first discuss detecting the state bits whose differences are needed. The value of the difference in a state bit is needed if the corresponding state bit contributes to a bit condition or equivalently probabilistic transition. In our model, we propagate Δ_U to Δ_B (resp. Δ_L to Δ_F) with probability one, whereas for key recovery, we process the data in the opposite direction. Therefore, considering the Feistel structure of SIMON and Simeck, when processing data from Δ_B to Δ_U and (resp. from Δ_F to Δ_L), probabilistic transitions (**truncated difference** \rightarrow **fixed difference**) only occur through XOR operations, while transitions through AND operations are deterministic.

For example, there are two bits x , and y such that $z = x \oplus y$. dx , dy , and dz be integer variables with domain $\{-1, 0, 1\}$ indicating the deterministic difference pattern of x , y , and z , and **cb** be the binary variable depicting whether there is a probabilistic transition through XOR. Moreover, **kdx**, **kdy**, and **kdz** are binary variables indicating whether the values of the differences of x , y , and z are needed or not. Our goal is to predict the values of **kdx** and **kdy** given the values of dx , dy , **kdz**, and **cb**. We analyze all possible cases:

- When **kdz** = **cb** = 0, it means that the value of the difference in the z position is not needed, and there is no probabilistic transition through the XOR operation. Consequently, the differences in x and y positions are also not needed, implying that **kdx** = **kdy** = 0.
- If either **kdz** or **cb** equals 1, then the values of **kdx** and **kdy** depend on the values of dx and dy . For instance, if dx is 0 or 1, this indicates that the value of the difference

at position x is already known, which means $\text{kdx} = 0$. Conversely, if $\text{dx} = -1$, it signifies that the value of the difference at position x is unknown, leading to $\text{kdx} = 1$. A similar analysis applies to dy .

Therefore, based on the aforementioned concept, we state the following proposition.

Proposition 9 (XOR_1^{gd}). *Suppose $f : \mathbb{F}_2^2 \rightarrow \mathbb{F}_2$ is such that $z = f(x, y) = x \oplus y$. Let dx , dy , and dz be integer variables with domain $\{-1, 0, 1\}$ indicating the deterministic difference pattern of x, y , and z , and cb be the binary variable depicting whether there is a probabilistic transition through XOR. Then, the valid transitions for detecting state bits whose values of the differences are needed through the XOR operation should satisfy the following constraints:*

$$\text{XOR}_1^{gd}(\text{dx}, \text{dy}, \text{kdx}, \text{kdy}) := \begin{cases} \text{if } (\text{kdx} = 0 \wedge \text{kdy} = 0) & \text{then } (\text{kdx} = 0 \wedge \text{kdy} = 0) \\ \text{elseif } ((\text{kdx} + \text{kdy}) \geq 1 \wedge \text{dx} = -1 \wedge \text{dy} \geq 0) & \text{then } (\text{kdx} = 1 \wedge \text{kdy} = 0) \\ \text{elseif } ((\text{kdx} + \text{kdy}) \geq 1 \wedge \text{dx} \geq 0 \wedge \text{dy} = -1) & \text{then } (\text{kdx} = 0 \wedge \text{kdy} = 1) \\ \text{elseif } ((\text{kdx} + \text{kdy}) \geq 1 \wedge \text{dx} = -1 \wedge \text{dy} = -1) & \text{then } (\text{kdx} = 1 \wedge \text{kdy} = 1) \\ \text{else} & (\text{kdx} = 0 \wedge \text{kdy} = 0) \end{cases}$$

where kdx , kdy , and kdz are binary variables indicating whether the values of the differences of x , y , and z are needed or not.

Additionally, we can construct the necessary constraints to propagate the state bits with the required difference value through the AND operation similarly. In this context, we analyze the possible cases in the following way:

- If $\text{kdz} = 0$, it means that the value of the difference in the z position is not needed. Therefore, the differences in the x , and y position also not needed, implying that $\text{kdx} = \text{kdy} = 0$.
- If $\text{kdz} = 1$, then the values of kdx and kdy depend on the values of dx , and dy .

Hence, we state the following proposition:

Proposition 10 (AND_1^{gd}). *Suppose $f : \mathbb{F}_2^2 \rightarrow \mathbb{F}_2$ is such that $z = f(x, y) = x \cdot y$. Let dx , dy , and dz be integer variables with domain $\{-1, 0, 1\}$ indicating the deterministic difference pattern of x, y , and z , and kdx , kdy , and kdz binary variables indicating whether the values of the differences of x, y , and z are needed or not. Then, the valid transitions for detecting state bits whose values of the differences are needed through the AND operation satisfy the following constraints:*

$$\text{AND}_1^{gd}(\text{dx}, \text{dy}, \text{kdx}, \text{kdy}) := \begin{cases} \text{if } (\text{kdx} = 0) & \text{then } (\text{kdx} = 0 \wedge \text{kdy} = 0) \\ \text{elseif } (\text{dx} = -1 \wedge \text{dy} \geq 0 \wedge \text{kdx} = 1) & \text{then } (\text{kdx} = 1 \wedge \text{kdy} = 0) \\ \text{elseif } (\text{dx} \geq 0 \wedge \text{dy} = -1 \wedge \text{kdx} = 1) & \text{then } (\text{kdx} = 0 \wedge \text{kdy} = 1) \\ \text{elseif } (\text{dx} = -1 \wedge \text{dy} = -1 \wedge \text{kdx} = 1) & \text{then } (\text{kdx} = 1 \wedge \text{kdy} = 1) \\ \text{else} & (\text{kdx} = 0 \wedge \text{kdy} = 0) \end{cases}$$

Therefore, we define binary variables for each state bit through E_B and E_F to indicate whether the difference value of each state bit over E_B and E_F is needed, and using Proposition 9 and Proposition 10, we encode the propagation of state bits whose difference value is needed. In addition, we also define a new constraint to link the beginning of E_U to the end of E_B , and the end of E_L to the beginning of E_F .

When considering the determination of data values, the nonlinear operation **AND** becomes relevant. We describe this determination over the functions in E_B , although a similar model can be applied to E_F . Let's suppose $z = x \cdot y$, where $x, y, z \in \mathbb{F}_2$. Assuming δx , δy , and δz represent the values of the differences in x , y , and z , respectively, we have: $\delta z = (x \cdot y) \oplus ((x \oplus \delta x) \cdot (y \oplus \delta y))$. Now, let's assume the values of the differences δx and δy are known, and we aim to determine the value of δz , for instance, to check a filter. Additionally, assume that we do not need to know the value of z . Consequently, whether we need the value of x or y to determine δz depends on δx and δy . For instance, if $\delta y = 0$, we do not need to know the value of x to derive δz . Similarly, if $\delta x = 0$, we do not need to know the value of y to determine δz . Thus, it is crucial to consider the value of differences when modeling the **AND** operation in guess-and-determine. Proposition 11 outlines how to model the **AND** operation in guess-and-determine.

Proposition 11 (AND_2^{gd}). *Suppose $f : \mathbb{F}_2^2 \rightarrow \mathbb{F}_2$ is such that $z = f(x, y) = x \cdot y$. Let \mathbf{dx} , \mathbf{dy} , and \mathbf{dz} be integer variables with domain $\{-1, 0, 1\}$ indicating the deterministic difference pattern of x, y , and z , and \mathbf{kz} binary variable indicating whether the value of the differences of z are needed. Then, the valid transitions for detecting state bits whose values are needed through the **AND** operation satisfy the following constraints:*

$$\text{AND}_2^{gd}(\mathbf{kz}, \mathbf{kz}, \mathbf{dx}, \mathbf{dy}, \mathbf{kx}, \mathbf{ky}) := \begin{cases} \text{if } (\mathbf{kz} = 0 \wedge \mathbf{kz} = 0) & \text{then } (\mathbf{kx} = 0 \wedge \mathbf{ky} = 0) \\ \text{elseif } (\mathbf{kz} = 1 \wedge \mathbf{kz} = 0 \wedge \mathbf{dx} = 0 \wedge \mathbf{dy} \neq 0) & \text{then } (\mathbf{kx} = 1 \wedge \mathbf{ky} = 0) \\ \text{elseif } (\mathbf{kz} = 1 \wedge \mathbf{kz} = 0 \wedge \mathbf{dx} \neq 0 \wedge \mathbf{dy} = 0) & \text{then } (\mathbf{kx} = 0 \wedge \mathbf{ky} = 1) \\ \text{else} & (\mathbf{kx} = 1 \wedge \mathbf{ky} = 1) \end{cases}$$

where \mathbf{kx} , \mathbf{ky} , and \mathbf{kz} are binary variables indicating whether the values of x , y , and z are needed.

Proposition 12 describes how to model the **XOR** operation in guess-and-determine.

Proposition 12 (XOR_2^{gd}). *Suppose $f : \mathbb{F}_2^2 \rightarrow \mathbb{F}_2$ is such that $z = f(x, y) = x \oplus y$. Let \mathbf{kx} , \mathbf{ky} , and \mathbf{kz} be binary variables indicating whether the values of x, y , and z are needed. Then, the valid transitions for detecting state bits whose values are needed through the **XOR** operation satisfy the following constraints:*

$$\text{XOR}_2^{gd}(\mathbf{kz}, \mathbf{kx}, \mathbf{ky}, \mathbf{dy}) := \begin{cases} \text{if } (\mathbf{kz} = 0) & \text{then } (\mathbf{kx} = 0 \wedge \mathbf{ky} = 0) \\ \text{else} & (\mathbf{kx} = 1 \wedge \mathbf{ky} = 1) \end{cases}$$

Finally, we now explain how to detect the key bits that are involved in the determination of data values. Let $\mathbf{ikb}_r[i] \in \{0, 1\}$, where $0 \leq i \leq (n - 1)$, $0 \leq r \leq (r_B - 1)$, be a binary variable that indicates whether the i th bit of the subkey in the r th round of E_B is involved. Let $\mathbf{kxu}_r^0[i]$ and $\mathbf{kxu}_r^1[i]$ be binary variables indicating whether the value of the i -th bit of the input of the r -th round function of E_B needs to be determined. Then, we can conclude that $\mathbf{ikb}_r[i] = 1$ if and only if $\mathbf{kxu}_{r+1}^0[i] = 1$. Otherwise $\mathbf{ikb}_r[i] = 0$. Similarly, we define binary variables $\mathbf{ikf}_r[i]$ to encode the involved subkey in E_F . Algorithm 3 in Appendix B describes our CSP model for guess-and-determine through E_B . We refer to $\text{CSP}_{\text{GD}} := \text{CSP}_B^{gd} \wedge \text{CSP}_F^{gd}$ as our CSP models for guess-and-determine through E_B and E_F .

4.2.4 Modeling Equivalent Subkey Technique.

The equivalent subkey technique has been widely used in various key-recovery attacks. This technique aims to reduce the number of guessed subkey bits by replacing the original subkeys with the equivalent subkeys. This technique was initially introduced by Isobe et al. [IS13] to investigate generic key recovery attacks on the Feistel scheme. Subsequently, it

was adapted for zero correlation attacks on SIMON [SFW15], and for impossible differential and zero correlation attacks on Simeck [SB18, ZGHL18].

To reduce the number of guessed subkey bits in the key recovery process, one can move the subkey K_i of the i -th round to the $(i+1)$ -th round where $0 \leq i \leq (r_B - 1)$, to get the equivalent subkey K_{i+1}^e (please refer to Figure 8 in Section 5). Similarly, one can move the subkey K_i of the i -th round to the $(i-1)$ -th round, for $(r_B + r_D) \leq i \leq (r_B + r_D + r_F - 1) = (r - 1)$, to get the equivalent subkey K_{i-1}^e (please refer to Figure 8 in Section 5). We can incorporate this idea into our key recovery model to reduce the number of involved keys. For example, we can write the expressions of equivalent subkeys (Figure 5) as follows:

$$\left\{ \begin{array}{l} K_1^e = K_0 \\ K_2^e = (K_1^e \lll 2) \oplus K_1 \\ K_3^e = K_1^e \oplus (K_2^e \lll 2) \oplus K_2 \\ K_4^e = K_2^e \oplus (K_3^e \lll 2) \oplus K_3 \\ \vdots \\ K_{r_B-1}^e = K_{r_B-3}^e \oplus (K_{r_B-2}^e \lll 2) \oplus K_{r_B-2} \\ K_{r_B}^e = K_{r_B-2}^e \oplus (K_{r_B-1}^e \lll 2) \oplus K_{r_B-1} \end{array} \right. \quad \left\{ \begin{array}{l} K_{r-2}^e = K_{r-1} \\ K_{r-3}^e = (K_{r-2}^e \lll 2) \oplus K_{r-2} \\ K_{r-4}^e = K_{r-2}^e \oplus (K_{r-3}^e \lll 2) \oplus K_{r-3} \\ K_{r-5}^e = K_{r-3}^e \oplus (K_{r-4}^e \lll 2) \oplus K_{r-4} \\ \vdots \\ K_{r_B+r_D}^e = K_{r_B+r_D+2}^e \oplus (K_{r_B+r_D+1}^e \lll 2) \oplus K_{r_B+r_D+1} \\ K_{r_B+r_D-1}^e = K_{r_B+r_D+1}^e \oplus (K_{r_B+r_D}^e \lll 2) \oplus K_{r_B+r_D} \end{array} \right.$$

For the r -th round of E_B , where $1 \leq r \leq r_B - 1$, let the two state variables where the equivalent subkey is xored be yu_r and zu_r . Furthermore, we define binary variables $\text{kyu}_r^0[i]$ and $\text{kzu}_r^0[i]$ to indicate whether the values of $yu_r[i]$ and $zu_r[i]$ are needed, and $\text{ikb}_r[i]$ to indicate whether the i th bit of the equivalent subkey in the r -th round of E_B is involved. Finally, we use the following constraints to model the idea of the equivalent subkey technique:

$$\text{CSP}^{ESK} := \begin{cases} \text{if } (\text{kyu}_r^0[i] = 1 \vee \text{kzu}_r^0[i] = 1) \text{ then} & (\text{ikb}_r[i] = 1) \text{ for } 0 \leq i \leq (n-1) \\ \text{else} & (\text{ikb}_r[i] = 0) \text{ for } 0 \leq i \leq (n-1) \end{cases}$$

where the variables kyu_r^0 and kzu_r^0 indicate the state after the α -bit right-rotation on $\text{kyu}^0[r]$ and after the β -bit rotation on $\text{kzu}^0[r]$, respectively (where $(\alpha, \beta) = (8, 1)$ for SIMON, and $(0, 5)$ for Simeck).

Important Observation. We can further reduce the number of involved subkey bits by considering a simple observation. Let $z = x \cdot y$, where x, y, z are binary variables. Also assume that $\text{kx}, \text{ky}, \text{kz}$ are binary variables to indicate whether the values of x, y, z are needed, and $\text{dx}, \text{dy}, \text{dz}$ are binary variables to indicate the truncated difference pattern in x, y , and z . Furthermore, assume x, y each involve some key bit information; for example, $x = x_1 \oplus k$ and $y = y_1 \oplus k'$, where k, k' are two key bits.

Consider the case $\text{dx} = \text{dy} = 0$ and $\text{kx} = \text{ky} = 1$. This implies we need to know the value of the state variables x, y , which normally means we must guess two key bits k and k' in the basic model. However, this can be reduced to one bit of key information as follows:

- If we guess the key bit k such that $x = 0$, then $z = 0$, and we do not need to guess the value of k' as we don't need to know the value of y in this case.
- If we guess the key bit k such that $x = 1$, then $z = y$, and the key bit k' in y goes into z linearly. Then, it can usually be merged with the next key addition (to z) and does not need to be guessed separately.

This implies that we can reduce the number of key bits involved using the aforementioned idea. To optimize the complexity, we can choose whether guessing k or k' is the better choice overall. We have included this technique in our automated model, which is easily integrated into the model with additional decision variables to optimize the choice of whether to guess k or k' .

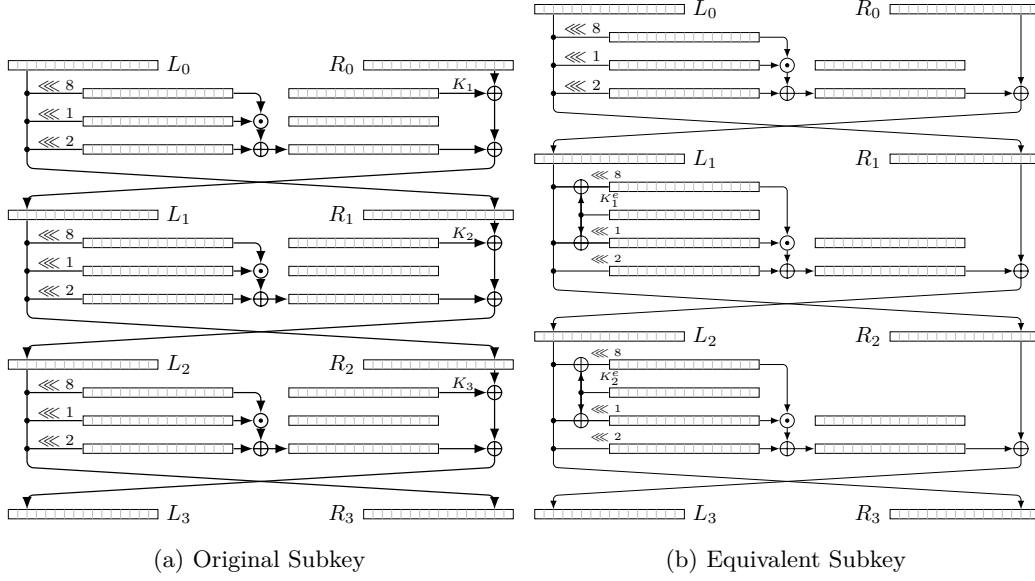


Figure 5: Original Subkey vs Equivalent Subkey

4.2.5 Modeling the Complexity Formula

In this component, we discuss how to combine all CSP models and finally model the complexity formula. The complexity formula of the ID attack is described in Equation 1. Therefore, we can model the complexity formulas in Equation 1 by the following constraints:

$$\begin{cases} d_0 := \min_{\Delta \in \{\Delta_B, \Delta_F\}} \frac{1}{2}(c_B + c_F + \text{block} - 1 - |\Delta| + \text{LG}(g)), \\ d_1 := c_B + c_F + \text{block} - 1 - |\Delta_B| - |\Delta_F| + \text{LG}(g), \\ t_0 := \max\{d_0, d_1\}, \quad t_0 < n \\ t_1 := c_B + c_F + \text{LG}(g) \\ t_2 := |k_B \cup k_F| + \text{LG}(g) \\ t_3 := k - g, \\ T := \max\{t_0, t_1, t_2, t_3\} \end{cases}$$

Lastly, we set the objective function to **Minimize T**. All the variables in our model are binary or integer variables with a limited domain except for d_0, d_1 , and t_i for $i \in \{0, 1, 2, 3\}$, which are real numbers. As a result, we can model all the critical parameters of the ID attack, and then combine all CSP models into a unified model and define an objective function to minimize the time complexity of the ID attack. We use the CP solver Or-Tools to find optimized ID attacks.

4.3 Results

We applied our unified COP model to find a complete ID attack on all versions of SIMON and Simeck. Table 2 summarizes the enhanced ID attacks achieved through our novel approach. As our analysis approach remains consistent across all the versions of SIMON and Simeck, we will detail an ID distinguisher and the corresponding key-recovery attack on SIMON-64-128 found using our model. The attack parameters for the other versions will be detailed in the following section.

13-round ID distinguisher of SIMON-64-128. Our unified CP model, as previously discussed, requires four integer parameters (r_B, r_D, r_M, r_F) which denote the length of the extended backward direction, the length of the distinguisher, the round of merging (in case of indirect contradiction), and the length of the extended forward direction. Using this framework, we discovered a 13-round ID distinguisher (based on direct contradiction) used to construct the optimized 23-round full ID attack on SIMON-64-128, as illustrated in Figure 7 in Section 5. In Figure 7 (Section 5), we notice that the truncated input difference Δ_U (illustrated in the upper triangle of $L_5||R_5$) where the upper triangle of the bits $L_5[0 - 18, 20, 22 - 31]$ and all bits of R_5 have a constant difference (either 0 \square or 1 \blacktriangle), and the upper triangle of the bit $L_5[19, 21]$ has an unknown difference (\blacksquare we do not know whether there is a difference or not). Similarly, in the truncated output difference Δ_L (illustrated in the lower triangle of $L_{18}||R_{18}$), where the lower triangle of the bits $L_{18}[0 - 19, 21 - 31]$ and all the bits R_{18} have zero difference, and the lower triangle of the bit $L_{18}[20]$ has an active difference (\blacktriangle). The 0-1 contradiction can be found in the bit $L_{10}[21]$ (equivalently $R_{11}[21]$).

It can be seen that this impossible differential is placed between rounds 5 and 18 and extended by $r_B = 5$ and $r_F = 5$ rounds in both directions. In this way, the first $23 = 5 + 13 + 5$ rounds of SIMON-64-128 were attacked.

23-round full ID attack on SIMON-64-128. For the key recovery, the attack is illustrated in Figure 8 in Section 5. As discussed before, first we propagate the difference Δ_U (and Δ_L) through E_B^{-1} (and E_F) with probability one to obtain the truncated difference Δ_B (and Δ_F), where E_B and E_F represent the $r_B = 5$ rounds added before and $r_F = 5$ rounds after E_D , respectively. In this context, the bit difference one (active bit) and unknown bit difference are represented by \blacksquare and \square . It can be seen that the truncated difference Δ_B is such that the bits $L_0[0, 5 - 6, 12, 21, 28, 30 - 31]$ and the bits $R_0[4, 29 - 30]$ have a constant difference (either \square 0 or 1 \blacksquare), which implies $|\Delta_B| = 53$ (the total number of non-fixed bit differences in Δ_B , illustrated by \blacksquare). Similarly, the truncated difference Δ_F is such that the bits $L_{23}[20 - 23, 27, 29]$ and the bits $R_{23}[3, 5, 12, 19, 21 - 25, 28 - 31]$ have constant difference (either \square 0 or 1 \blacksquare). This means $|\Delta_F| = 45$. More precisely, by seeing Δ_U and Δ_L , we can state that there exist one output patterns that give the longest impossible differential for fixed input patterns, and as $|\Delta_U| = 4$, then there are four possible input patterns. Using the idea of multiple differentials¹ discussed in [BNPS14], we can update $|\Delta_B| = 55$ and $|\Delta_F| = 45$. In the context, c_B and c_F are typically referred to as the number of bit filters (bit conditions, or we can say probabilistic transitions) that should be satisfied for differential transitions $\Delta_B \rightarrow \Delta_U$ and $\Delta_L \leftarrow \Delta_F$, respectively. In Figure 8 (Section 5), the position of bit conditions is represented by \square (which means in our model, the values of the variable cb corresponding to those positions are 1). Therefore, it can be seen that the number of filters that should be satisfied for differential transition $\Delta_B \rightarrow \Delta_U$ is $c_B = 49$, and the number of filters that should be satisfied for differential transition $\Delta_L \leftarrow \Delta_F$ is $c_F = 45$.

As discussed before, we have to detect the state bits whose difference or value, or both are needed for checking bit conditions in $\Delta_B \xrightarrow{E_B} \Delta_U$ and $\Delta_L \xleftarrow{E_F^{-1}} \Delta_F$. In Figure 8 (Section 5), the bit position where the value is needed and the difference is required for checking bit conditions are represented by \blacktriangle (which means in our model, the value of the variable kx corresponding to that bit position is 1) and \blacksquare (which means in our model, the value of the variable kdx corresponding to that bit position is 1), respectively. Finally, we

¹In [BNPS14], the authors introduced a technique to attack SIMON using multiple impossible differentials simultaneously, thereby reducing data complexity. Our key recovery attacks handle multiple impossible differentials, as described in [BNPS14]. This increases the value of $|\Delta_B|$ (resp. $|\Delta_F|$) by $\log\{\text{the number of multiple impossible input patterns}\}$ (resp. $\log\{\text{the number of multiple impossible output patterns}\}$), thereby reducing the data complexity of our attack directly following the complexity formula.

know that the key bits $k_B \cup k_F$ are involved in deriving the difference Δ_U and Δ_L from Δ_B and Δ_F , respectively. In Figure 8 (Section 5), it can be seen that $k_B = 66$ and $k_F = 55$, which means the total number of subkey bits involved is 121.

Finally, we can have the complexities of this attack by using the complexity formula discussed in Equation 1, which can be modeled in our framework by the constraints discussed in Subsubsection 4.2.5. Therefore, the complexities of our attack are:

$$\begin{cases} \text{Time Complexity} : 2^{123} \\ \text{Data Complexity} : 2^{61.27} \\ \text{Memory Complexity} : 2^{96.3} \end{cases}$$

The previous best ID attack is found on 22-round SIMON-64-128 [BNPS14, DF16].

Discussion. The gap between the actual time complexity and the output of Boura et al.’s formula [BNPS14] is well known. This formula provides a lower bound for the complexity of the guess-and-filter step, which is typically close to the actual value. The reason that [HSE23, HGSE24] and we are using this formula to build a unified CP model has been discussed in [HGSE24] (see the last paragraph of Section 2.1 of [HGSE24]). Briefly, there are two approaches: using this formula to estimate the complexity (as done in most of the previous works) and keeping the model very easy to solve, or incorporating all the details of the step-by-step early-abort technique into the CP model that is more accurate but makes the model very hard to solve. The first approach is relatively fast and mostly returns an optimum attack quickly. The second one can theoretically find the exact time complexity, but it is tough to solve because the early-abort technique is a multistep guess-and-determine procedure; implementing it into the CP model makes it very hard to solve. So, we chose the first one to find a nearly optimum attack and then check whether the actual time complexity matches the formula’s output discussed in [BNPS14]. In this regard, we found that sometimes the complexities deduced from Boura et al.’s formula [BNPS14] are lower (but relatively close) than those inferred from the step-by-step early abort technique. For example, in Simon-64-128, we performed the step-by-step early abort technique and got the actual complexities as follows:

$$\begin{cases} \text{Time Complexity} : 2^{123.76} \\ \text{Data Complexity} : 2^{61.27} \\ \text{Memory Complexity} : 2^{96.28} \end{cases}$$

5 Applications

This section discusses the application of our method to both ARX and AndRX ciphers. ARX ciphers are based on three primary operations: modular addition ($x \boxplus y$), bitwise rotation ($x \lll n$), and XOR ($x \oplus y$). We analyze different ARX constructions used in block ciphers (*e.g.*, LEA and SPECK), stream ciphers (*e.g.*, ChaCha [Ber08]) and MAC algorithms (*e.g.*, SipHash and Chaskey). Furthermore, we expand our analysis to AndRX ciphers, which replace modular addition with bitwise AND ($x \odot y$) operations, as utilized in ciphers such as SIMON and Simeck. Refer to Appendix C and Appendix D for brief specifications of these ciphers.

5.1 Application to ARX Ciphers

We have successfully applied our bit-wise CP-based model to find ID distinguishers based on satisfiability for several ARX ciphers: all versions of SPECK, LEA, ChaCha, Chaskey, and SipHash. Table 1 presents the ID distinguishers identified using our tool and compares

them with previous ID distinguishers. Additionally, Table 3 outlines our attack parameters. The subsequent sections provide detailed explanations of the attacks on each ARX cipher.

Table 3: Summary of our ID distinguishers for ARX ciphers

Cipher	(r_D, r_M)	Contradiction (round/type)
SPECK-32	(6, 2)	2/Direct
SPECK-48	(6, 2)	2/Direct
SPECK-64	(6, 2)	2/Direct
SPECK-96	(6, 2)	2/Direct
SPECK-128	(6, 2)	2/Direct
LEA	(10, 6)	6/Direct
ChaCha	(5, 2)	1.5/Direct
Chaskey	(4, 2)	2/Direct
SipHash	(4, 2)	2/Direct

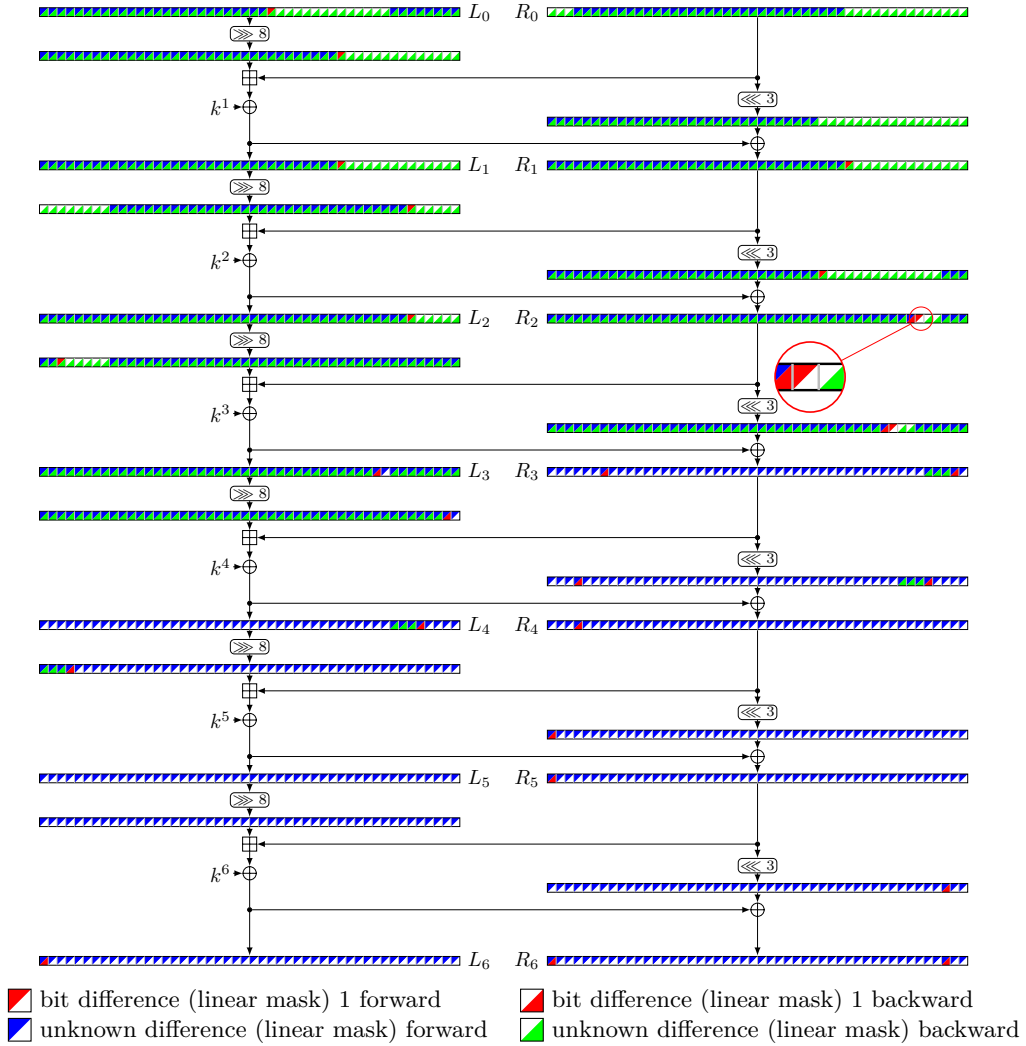
5.1.1 Application To SPECK

SPECK is a family of ARX ciphers with a generalized Feistel structure, known for its excellent performance in both hardware and software implementations. For more details on the specification, please refer to Appendix C. Since its publication, numerous cryptanalyses have been conducted [LLJW21, WW22, BdST⁺23]. In this context, we discuss the impact of impossible differential cryptanalysis on SPECK.

We applied our method to search for ID distinguishers for all versions of SPECK. Our findings include 6-round ID distinguishers for SPECK-32, SPECK-48, and SPECK-64, matching the previous best-known ID distinguishers. Specifically, the authors of [RC19] analyzed the differential properties of SPECK’s round function and transformed these properties into Boolean expressions to construct a SAT model for searching impossible differentials. They identified 3, 20, and 157 6-round IDs for SPECK-32, SPECK-48, and SPECK-64, respectively, with Hamming weight one in both input and output differences. In contrast, our tool found clusters of 2^4 , 2^{17} , and 2^{33} ID distinguishers for these versions with Hamming weights of at least one in both input and output differences (Figure 13a, Figure 13b, Figure 14 in Appendix F). These clusters were discovered in a single run of our tool, demonstrating the advantage of our framework over previous methods, where finding such large clusters would be difficult or infeasible. Additionally, we report 6-round ID distinguishers for SPECK-96 and SPECK-128 for the first time, discovering clusters of 2^{65} and 2^{97} IDs, respectively (Figure 6, Figure 15 in Appendix F).

5.1.2 Application to ChaCha

We applied our method to search for an ID distinguisher for ChaCha256. For a brief specification of ChaCha, please refer to Appendix C. The most popular cryptanalysis of ChaCha relies on differential cryptanalysis using probabilistic neutral bits (PNBs). Since 2008 [AFK⁺08], there have been several advancements in PNB-based attacks on ChaCha, resulting in improved attacks. Researchers have traditionally focused on incorporating single-bit differences at the beginning of differential-linear distinguishers for key-recovery attacks on ChaCha. Recently, [BGG⁺23] introduced an innovative approach with a 5-round differential-linear distinguisher considering 2-bit differences at the beginning. This 5-round

Figure 6: Cluster of 2^{65} ID distinguishers for 6-round SPECK-96.

distinguisher, integrated with the PNB framework, led to an enhanced key-recovery attack specifically tailored for a 7-round ChaCha cipher.

To the best of our knowledge, no cryptanalytic results have been reported regarding ID distinguishers for ChaCha. Using our model, we have detected a cluster of 2^{80} ID distinguishers for 5-round ChaCha for the first time. For detailed results, please refer to Table 4.

5.1.3 Application to Chaskey

Chaskey is a permutation-based MAC algorithm presented by Mouha *et al.* in 2014 [MMH⁺14]. Appendix C briefly describes the specification of Chaskey. Several attacks have been performed on Chaskey such as rotational cryptanalysis [KAR20], differential-linear cryptanalysis on 7-round Chaskey [Leu16], and impossible differential cryptanalysis [SBS21]. More precisely, the authors of [SBS21] used the MILP model proposed by [CCJ⁺21] to search for impossible differentials by fixing the input/output differential. This approach led to the discovery of 15 4-round ID distinguishers for Chaskey.

We applied our advanced bit-wise CP model to find ID distinguishers of Chaskey. Our model detected a cluster of 2^7 ID distinguishers for the 4-round Chaskey in a single run, aligning with the best previous results in terms of the number of rounds. For detailed results, please refer to Table 5.

5.1.4 Application to LEA

LEA (Lightweight Encryption Algorithm) [HLK⁺13] is a block cipher developed by the Korea Internet & Security Agency (KISA) to provide lightweight encryption in resource-constrained environments. Please refer to Appendix C for a concise specification of LEA. The designers of LEA [HLK⁺13], conducted differential cryptanalysis on the LEA cipher and identified a differential path extending up to 11 rounds with a probability of 2^{-98} . Additionally, they performed differential-linear, impossible differential, and linear cryptanalysis, presenting paths for 14, 10, and 11 rounds, respectively. In [ZGH16], the authors proposed 9-round ZC distinguishers for LEA. Additionally, in [CCJ⁺21], the authors applied their MILP model to search for ID distinguishers of LEA and reported a 10-round ID distinguisher.

We employed our advanced model to search for an ID distinguisher for LEA and identified a cluster of 2^2 ID distinguishers for the 10-round LEA in a single run, matching the best previous results in terms of the number of rounds. For detailed results, please see Table 6.

5.1.5 Applications to SipHash

SipHash is a family of pseudorandom functions introduced by Aumasson and Bernstein at Indocrypt 2012 [AB12], designed for short message inputs. Please refer to Appendix C for a brief specification of SipHash. In [DMS14], the authors generalized the concepts to calculate the probability of ARX functions, which results in a characteristic for SipHash-2-4 with a probability of $2^{-236.3}$ and a distinguisher for the Finalization of SipHash-2-4 with practical complexity. Several other improved cryptanalyses on SipHash can be found [XLSL19, HY24]. We applied our advanced model to search for an ID distinguisher for SipHash. To the best of our knowledge, we are the first to detect a cluster of 2^{14} ID distinguishers of 4-round SipHash using our model. Please refer to Table 7 for the detailed result.

In summary, within a few minutes, our bit-wise CP model can produce the best-known ID distinguishers for the targeted ARX ciphers with large block sizes. It is important to note that, for all the ARX ciphers used in our paper, our distinguishers for the same number of rounds as in previous works are actually supersets of those reported earlier.

5.2 Application to AndRX Ciphers

In this paper, we first introduce the most advanced bit-wise CP models for detecting ID and ZC distinguishers for AndRX ciphers (SIMON, Simeck). We also extend the model for ID distinguishers to construct a unified COP model to generate a full ID attack on AndRX ciphers. Please refer to Appendix D for a brief overview of the specification of SIMON and Simeck. Moreover, we demonstrated most of the existing cryptanalytic attacks on SIMON and Simeck in Table 11 (Subsection D.3 in Appendix D) and Table 12 (Subsection D.4 in Appendix D), respectively. It is important to note that the best existing cryptanalysis of both ciphers is linear cryptanalysis. Thus, we do not claim that ID attacks are the best cryptanalytic results on SIMON and Simeck. Instead, our contribution lies in developing an efficient tool to find full ID attacks on these ciphers. Our tool provides a novel solution that eliminates the need for manual contradiction discovery or input-output fixing, as

required by previous automated methods. The details of our ID and ZC attacks on SIMON, and Simeck are as follows.

Table 8: Summary of our ZC distinguishers for AndRX ciphers

Cipher	(r_D, r_M)	Contradiction (round/type)	Ref.
SIMON-32	(11, -)	6/Direct	[SFW15]
	(11, 6)	6/Direct	This Paper
SIMON-48	(12, -)	7/Direct	[SFW15]
	(12, 7)	7/Direct	This paper
SIMON-64	(13, -)	8/Direct	[SFW15]
	(13, 8)	8/Direct	This paper
SIMON-96	(16, -)	10/Direct	[SFW15]
	(16, 10)	10/Direct	This paper
SIMON-128	(19, -)	12/Direct	[SFW15]
	(19, 12)	12/Direct	This paper
Simeck-32	(11, -)	-/Direct	[ZWL ⁺ 23]
	(11, 5)	5/Direct	This paper
Simeck-48	(15, -)	2/Indirect	[SB18]
	(15, 8)	2/Indirect	This paper
Simeck-64	(17, -)	2/Indirect	[SB18]
	(17, 8)	2/Indirect	This paper

5.2.1 Applications to SIMON

Distinguisher. First, we applied our improved bit-wise CP models for ID distinguishers on all variants of SIMON. The results include clusters of 2^4 , 2^7 , 2^{20} , 2^{33} and 2^{46} ID distinguishers for 11-round SIMON-32, 12-round SIMON-48, 13-round SIMON-64, 16-round SIMON-96 and 19-round SIMON-128, respectively. These results match the previously best-known ID distinguishers [BNPS14] in terms of number of rounds. In [BNPS14], the authors used a miss-in-the-middle approach to find ID distinguishers, relying on manual methods to identify contradictions and fix the input/output differences. In contrast, our tool does not fix input/output differences, allowing us to discover a large cluster of IDs in a single run. Moreover, an important distinction is that most of the ID distinguishers previously reported in [BNPS14] have a Hamming weight of one in both input and output differences. Our tool, however, identifies ID distinguishers with Hamming weights greater than one in both input and output differences. This capability underscores the significant contribution of our approach, offering more comprehensive and efficient discovery of ID distinguishers. For instance, our distinguishers for SIMON are supersets of those identified in [BNPS14].

Furthermore, we applied our bit-wise model for ZC distinguishers to all versions of SIMON and produced ZC distinguishers of 11-round, 12-round, 13-round, 16-round, and 19-round SIMON-32, SIMON-48, SIMON-64, SIMON-96, and SIMON-128, respectively. Although no new results were found for SIMON regarding ID and ZC distinguishers, our automated tool can identify several ID and ZC distinguishers in just one run within a few minutes. Please refer to Table 8 for more details of the attack parameters for ZC distinguishers of SIMON, and Table 9 for ID distinguishers of SIMON.

Key-Recovery. We applied our unified COP model to develop a complete ID attack on all variants of SIMON in single key settings. The parameters of our ID attack are detailed in

Table 9: Summary of our ID attack parameters for ANDRX ciphers. r_D : The length of the distinguisher. r_B : The length of the extended backward direction. r_F : The length of the extended forward direction. r_M : The position of merging

Cipher	(r_B, r_D, r_F, r_M)	Contradiction (round/type)	(c_B, c_F)	(Δ_B , Δ_F)	$ k_B \cup k_F $	g
SIMON-32-64	(3, 11, 5, 5)	5/ Direct	(19, 29)	(23, 30)	53	5
SIMON-32-64	(4, 11, 5, 5)	5/ Direct	(22, 29)	(24, 30)	61	2
SIMON-48-72	(4, 12, 4, 5)	5/ Direct	(34, 29)	(38, 29)	64	5
SIMON-48-96	(4, 12, 5, 5)	5/ Direct	(34, 41)	(38, 41)	84	10
SIMON-64-96	(3, 13, 5, 5)	5/ Direct	(20, 45)	(30, 45)	66	28
SIMON-64-96	(4, 13, 5, 5)	5/ Direct	(37, 45)	(41, 45)	88	5
SIMON-64-128	(4, 13, 5, 5)	5/ Direct	(36, 45)	(46, 45)	93	30
SIMON-64-128	(5, 13, 5, 6)	6/ Direct	(49, 45)	(55, 45)	121	7
SIMON-96-96	(4, 16, 4, 9)	9/ Direct	(30, 37)	(30, 53)	82	8
SIMON-96-144	(4, 16, 5, 9)	9/ Direct	(30, 54)	(30, 60)	110	22
SIMON-128-128	(4, 19, 4, 11)	11/ Direct	(30, 32)	(30, 68)	91	40
SIMON-128-128	(4, 19, 5, 7)	7/ Direct	(34, 47)	(54, 47)	109	18
SIMON-128-192	(5, 19, 5, 7)	7/ Direct	(56, 47)	(62, 47)	145	30
SIMON-128-192	(5, 19, 6, 7)	7/ Direct	(45, 64)	(49, 64)	183	8
SIMON-128-256	(5, 19, 6, 7)	7/ Direct	(60, 64)	(52, 64)	194	30
SIMON-128-256	(6, 19, 6, 7)	7/ Direct	(61, 64)	(65, 64)	240	9
Simeck-32-64	(5, 11, 4, 4)	4/ Direct	(23, 22)	(31, 22)	55	7
Simeck-48-96	(5, 15, 5, 7)	13/ indirect	(33, 33)	(35, 35)	91	3
Simeck-64-128	(5, 17, 5, 9)	1/ indirect	(34, 34)	(35, 35)	101	2

Table 9, including input parameters (r_B, r_D, r_F, r_M) and output parameters $(c_B, c_F, |\Delta_B|, |\Delta_F|)$, the total involved keys as $|k_B \cup k_F|$, g , crucial for generating the complexities of the ID attacks, as shown in Equation 1. As detailed in Table 2, our ID attack for SIMON demonstrated significant improvements:

- A 22-round complete ID attack on SIMON-64-96 (Figure 31 in Appendix F)
- A 23-round complete ID attack on SIMON-64-128 (Figure 8 in Section 5)
- A 28-round complete ID attack on SIMON-128-128 (Figure 41 in Appendix F)
- A 31-round complete ID attack on SIMON-128-256 (Figure 49 in Appendix F)

Each of these attacks extends the number of rounds by one compared to the previous best ID attacks on the respective versions. Furthermore, we achieved a 30-round full ID attack on SIMON-128-192, which surpasses the previous best ID attacks by two rounds. For a detailed comparison of our attacks with the previous best attacks, please refer to Table 2. While we did not improve the number of rounds for other versions of SIMON, we significantly improved the time complexity of the attacks. Moreover, it is noteworthy that while most of the previous attacks on SIMON were full codebook attacks, our COP model enabled us to devise ID attacks that are not full codebook attacks for most SIMON variants. This is a notable achievement of our COP model. Most previous full ID attacks on SIMON relied on manual approaches. In contrast, [DF16] introduced an automated tool to find ID key recovery attacks on SIMON, though their tool required several days to produce results for larger versions. In comparison, our unified COP model offers a more efficient and streamlined approach. A detailed comparison of our method and the previous approaches is presented in the subsequent section, highlighting the advancements and efficiency of our contributions.

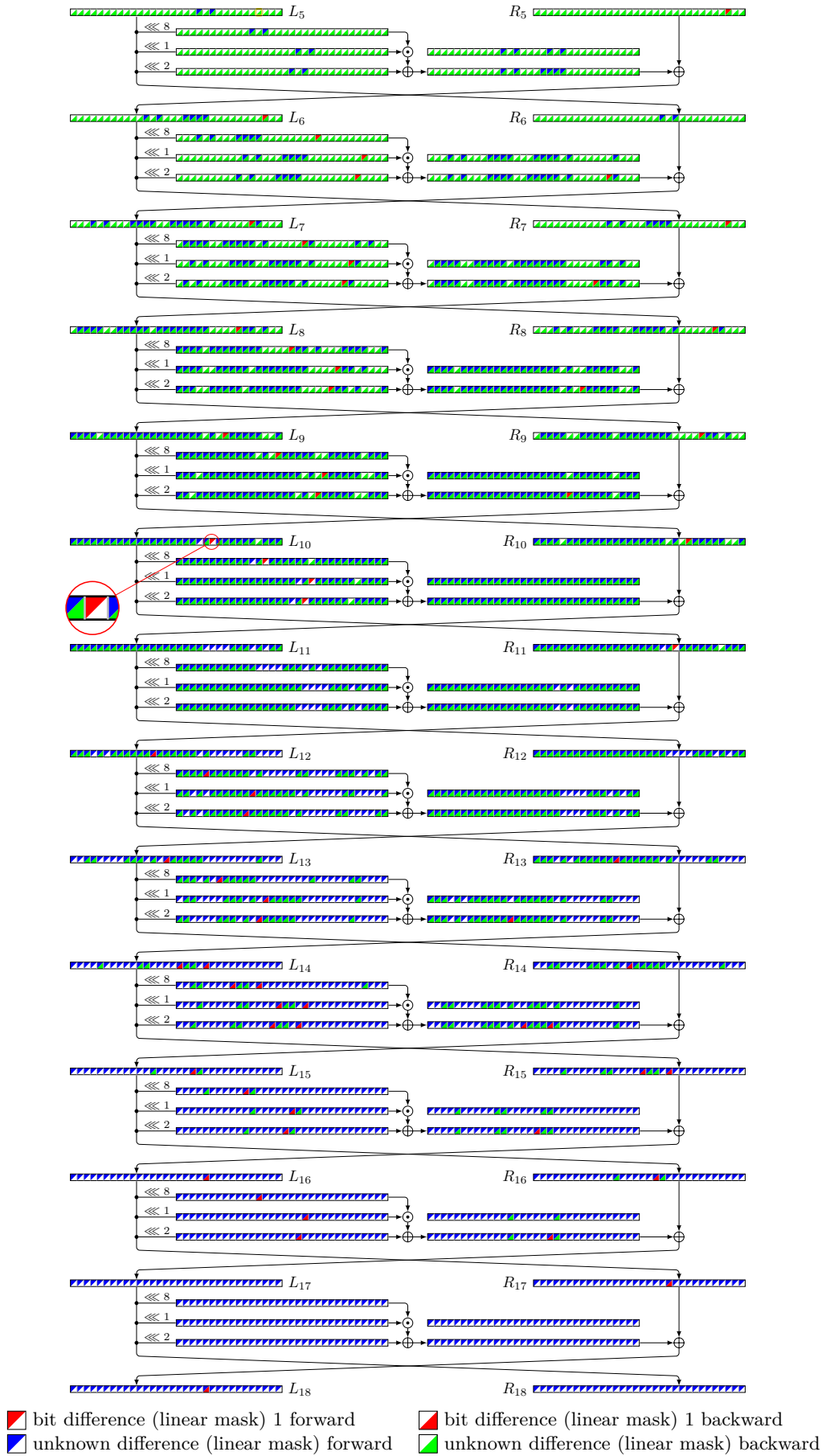


Figure 7: 13-round ID distinguisher for attack on 23-round SIMON64-128.

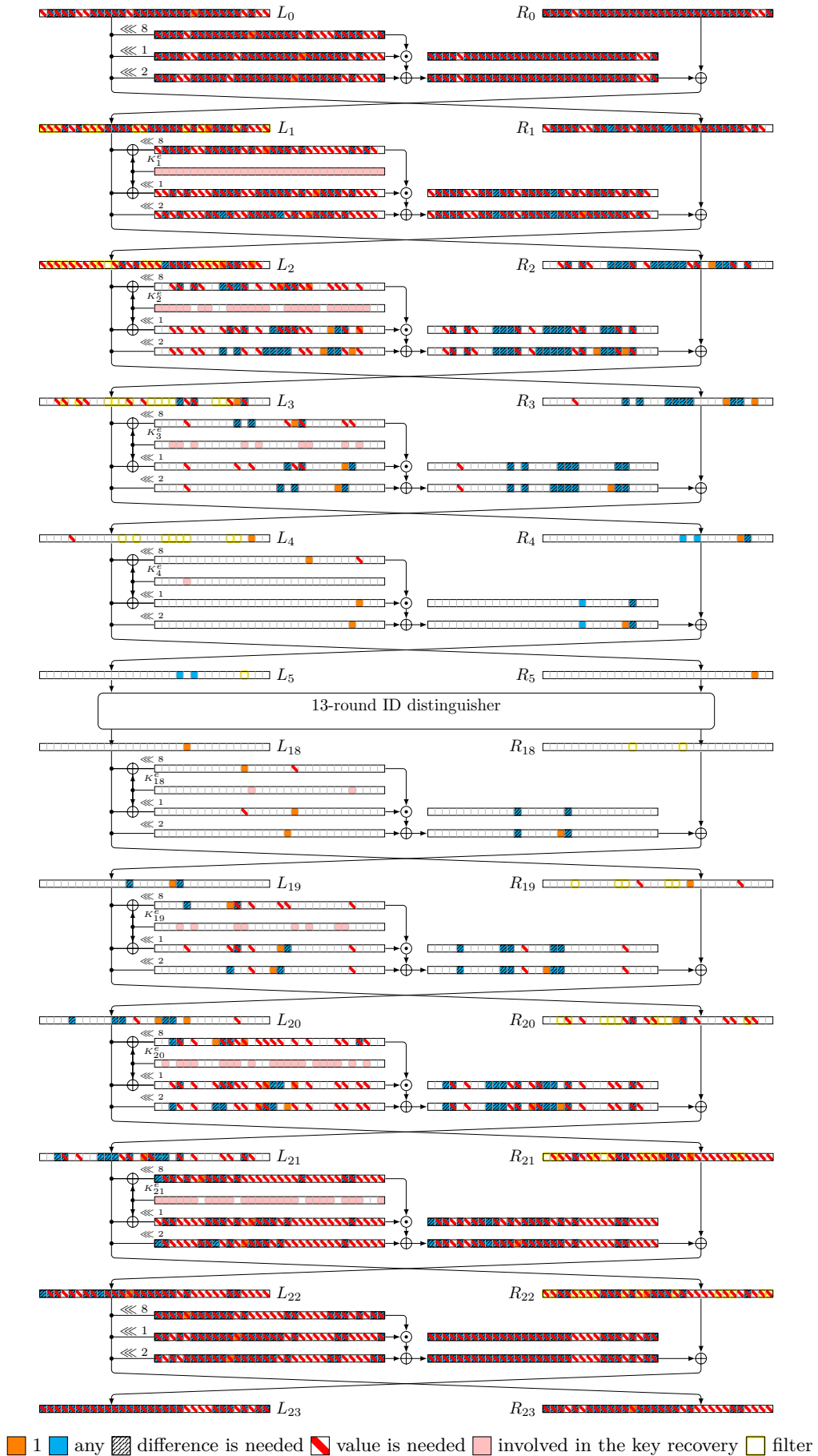


Figure 8: Key recovery of the attack on 23-round SIMON64-128.

5.2.2 Applications to Simeck

Distinguisher. Initially, we employed our bit-wise CP models for ID and ZC distinguishers, across all variants of *Simeck*. As a result of this, we construct 11-, 15- (indirect), and 17-round (indirect) ID and ZC distinguishers of *Simeck*-32, *Simeck*-48, *Simeck*-64, respectively. All the distinguishers are in accordance with the best previous ID and ZC distinguishers of *Simeck*. It is important to note that, for *Simeck*, our tool can recover the same distinguishers as found in [SB18].

Key Recovery. We applied our unified COP model to find a full ID attack on all variants of *Simeck* in single key settings. Please refer to Table 9 for details of our ID attack parameters on all versions of *Simeck*. The authors of [ZLW⁺23] proposed 20-, 25-, and 27-round ID attacks on *Simeck*-32, *Simeck*-48, and *Simeck*-64 with time complexities of $2^{61.11}$, $2^{94.23}$, and $2^{126.56}$, respectively. The authors in [ZLW⁺23] presented their attack, but did not explain anything regarding their attack algorithm. However, we found 20-, 25-, and 27-round ID attacks for the corresponding variants of *Simeck* with time complexities of $2^{57.27}$, $2^{93.05}$, and 2^{126} , respectively. Interestingly, we discovered a 20-round ID attack on *Simeck*-32 with data complexity $2^{27.28}$, while the previous best attack on *Simeck*-32 was a full codebook attack. For a more detailed comparison of our ID attacks with previous works, please refer to Table 2.

5.2.3 Discussion

Table 11 (Subsection D.3 in Appendix D) and Table 12 (Subsection D.4 in Appendix D) provide an overview of the most popular existing cryptanalytic results on *SIMON* and *Simeck*. Table 2 summarizes all ID attacks constructed using our tool and compares them with the existing ID attacks on *SIMON* and *Simeck*. When looking for an attack, usually our model picks those distinguishers with many zeroes, some ones, and some unknown (-1) bits in both input and output differences (e.g., Figure 8 and other key-recovery figures in the appendix). It is important to note that, for any distinguishers when the output (or input) difference is either -1 (unknown) or 0 (inactive) for each bit, then there should be at least one active difference (this condition is included in our model, which actually depicts the accuracy of our tool). This is because the input or output of an ID distinguisher cannot be entirely inactive. In [BNPS14], the authors followed a manual approach to find ID attacks on all versions of *SIMON*, whereas in [DF16], the authors introduced an automatic tool to find ID attacks on various ciphers, including *SIMON*. We strongly believe that the performance of our tool significantly surpasses the tool presented in [DF16]. While the tool in [DF16] requires several days to generate attacks on variants of *SIMON* with the largest block size, our tool achieves enhanced results within hours, even when executed on a regular laptop. Additionally, the authors of [DF16] did not provide detailed attack descriptions for larger variants of *SIMON*. Our tool offers several other advantages, such as its applicability for zero-correlation attacks and its ability to leverage any state-of-the-art CP/SMT/MILP solvers, unlike the tool in [DF16]. Another key advantage of our tool lies in its foundation on a unified optimization problem that considers all attack parameters, in contrast to the approach in [DF16], which involves enumerating numerous ID distinguishers within a limited search space and then selecting the optimal one. This fundamental difference contributes significantly to the speed of our tool, allowing us to readily apply it to large variants of *SIMON* and *Simeck*. It is important to recognize the challenges encountered when constructing a unified CP model to identify a complete ID attack on ARX ciphers. The primary difficulty arises in modeling the guess-and-determine process in the outer sections, particularly when dealing with state bits whose differences, values, or both are essential for verifying bit conditions through modular additions. Consequently, we have decided to leave this aspect for future exploration.

6 Conclusion and Future Work

This paper enhanced the CP model for discovering complete ID attacks in [HSE23, HGSE24] from three significant perspectives. Firstly, we expanded the bit-wise CP model of [HGSE24] to encompass ARX and AndRX designs. Additionally, we demonstrated how to extend this model to detect ZC distinguishers. Moreover, we introduced a novel CP model for finding ID distinguishers based on direct and indirect contradictions. This model is generic and not limited to ARX and AndRX designs. Subsequently, we extended the CP model for the key recovery of ID attacks in [HSE23] to encompass bit-oriented designs, particularly AndRX designs. Finally, we integrated our new CP model for detecting ID distinguishers with the CP model for key recovery, proposing a unified CP model for identifying complete ID attacks.

To demonstrate the utility of our new methods, we applied them to several ARX and AndRX designs, including SIMON, Simeck, ChaCha, Siphash, and other ciphers. Notably, we improved the ID attack on SIMON-64-96, SIMON-64-128, SIMON-128-128, SIMON-128-256 by one round and SIMON-128-192 by two rounds. Additionally, we introduced several new ID and ZC distinguishers for our targeted ciphers. Our work also prompts intriguing questions for future research. While we presented a workaround to apply CP models based on satisfiability for detecting ZC distinguishers on AndRX designs, it would be interesting for future work to develop a similar CP model for ARX designs. Another interesting work is applying our CP model to find complete ID attacks to other bit-oriented ciphers beyond ARX and AndRX designs. Finally, our key-recovery models may be a useful starting point for developing and evaluating further optimization techniques for key recovery.

Acknowledgments. This work has been supported in part by the Austrian Science Fund (FWF SFB project SPyCoDe), partially by the French Agence Nationale de la Recherche through the OREO project under Contract ANR-22-CE39-0015 and by the France 2030 program under grant agreement No. ANR-22-PECY-0010. Debasmita Chakraborty was a visiting research scholar at TU Graz, funded by the OeAD Ernst Mach-Stipendium, weltweit, while carrying out this work. We sincerely thank Sadegh Sadeghi for insightful discussions on impossible differential attacks, especially for his elucidation of their method for finding ID distinguishers as outlined in [SB18]. We would also like to thank the anonymous reviewers for their valuable comments and suggestions.

References

- [AB12] Jean-Philippe Aumasson and Daniel J. Bernstein. SipHash: A fast short-input PRF. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 489–508. Springer, 2012. doi:10.1007/978-3-642-34931-7_28.
- [AFK⁺08] Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, and Christian Rechberger. New features of latin dances: Analysis of salsa, chacha, and rumba. In Kaisa Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 470–488. Springer, 2008. doi:10.1007/978-3-540-71039-4_30.
- [BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In *EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 12–23. Springer, 1999. doi:10.1007/3-540-48910-X_2.
- [BdST⁺23] Alex Biryukov, Luan Cardoso dos Santos, Je Sen Teh, Aleksei Udovenko, and Vesselin Velichkov. Meet-in-the-filter and dynamic counting with applications

- to speck. In *ACNS 2023*, volume 13905 of *LNCS*, pages 149–177. Springer, 2023. doi:10.1007/978-3-031-33488-7_6.
- [Ber08] Daniel J Bernstein. Chacha, a variant of salsa20, 2008. URL: <https://cr.yp.to/chacha.html>.
- [BGG⁺23] Emanuele Bellini, David Gérardt, Juan Grados, Rusydi H. Makarim, and Thomas Peyrin. Boosting differential-linear cryptanalysis of chacha7 with MILP. *IACR Trans. Symmetric Cryptol.*, 2023(2):189–223, 2023. doi:10.46586/TOSC.V2023.I2.189-223.
- [BJK⁺16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *CRYPTO 2016*, pages 123–153. Springer, 2016. doi:10.1007/978-3-662-53008-5_5.
- [BNPS14] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and improving impossible differential attacks: applications to CLEFIA, Camellia, LBlock and Simon. In *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 179–199. Springer, 2014. doi:10.1007/978-3-662-45611-8_10.
- [BR14] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *Des. Codes Cryptogr.*, 70(3):369–383, 2014. doi:10.1007/s10623-012-9697-z.
- [BSS⁺15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *DAC 2015*, pages 175:1–175:6. ACM, 2015. doi:10.1145/2744769.2747946.
- [CCJ⁺21] Tingting Cui, Shiyao Chen, Keting Jia, Kai Fu, and Meiqin Wang. New automatic search tool for impossible differentials and zero-correlation linear approximations. *Sci. China Inf. Sci.*, 64(2), 2021. See also <https://eprint.iacr.org/2016/689>. doi:10.1007/S11432-018-1506-4.
- [CCW⁺18] Zhihui Chu, Huaifeng Chen, Xiaoyun Wang, Xiaoyang Dong, and Lu Li. Improved integral attacks on SIMON32 and SIMON48 with dynamic key-guessing techniques. *Secur. Commun. Networks*, 2018:5160237:1–5160237:11, 2018. doi:10.1155/2018/5160237.
- [CJF⁺16] Tingting Cui, Keting Jia, Kai Fu, Shiyao Chen, and Meiqin Wang. New automatic search tool for impossible differentials and zero-correlation linear approximations. *IACR Cryptol. ePrint Arch.*, page 689, 2016. URL: <http://eprint.iacr.org/2016/689>.
- [CW16] Huaifeng Chen and Xiaoyun Wang. Improved linear hull attack on round-reduced simon with dynamic key-guessing techniques. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 428–449. Springer, 2016. doi:10.1007/978-3-662-52993-5_22.
- [CWW15] Zhan Chen, Ning Wang, and Xiaoyun Wang. Impossible differential cryptanalysis of reduced round SIMON. *IACR Cryptol. ePrint Arch.*, page 286, 2015. URL: <http://eprint.iacr.org/2015/286>.
- [DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläpfer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, 2021. doi:10.1007/s00145-021-09398-9.

- [DF16] Patrick Derbez and Pierre-Alain Fouque. Automatic search of meet-in-the-middle and impossible differential attacks. In *CRYPTO 2016*, volume 9815 of *LNCS*, pages 157–184. Springer, 2016. doi:10.1007/978-3-662-53008-5_6.
- [DMS14] Christoph Dobraunig, Florian Mendel, and Martin Schl affer. Differential cryptanalysis of SipHash. In Antoine Joux and Amr M. Youssef, editors, *SAC 2014*, volume 8781 of *LNCS*, pages 165–182. Springer, 2014. doi:10.1007/978-3-319-13051-4_10.
- [DR99] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. 1999.
- [HDE24] Hosein Hadipour, Patrick Derbez, and Maria Eichlseder. Revisiting differential-linear attacks via a boomerang perspective with application to AES, ascon, CLEFIA, SKINNY, PRESENT, KNOT, TWINE, WARP, LBlock, simeck, and SERPENT. *IACR Cryptology ePrint Archive*, Report 2016/689, 2024. URL: <https://eprint.iacr.org/2024/255>.
- [HE22] Hosein Hadipour and Maria Eichlseder. Autoguess: A tool for finding guess-and-determine attacks and key bridges. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 2022*, volume 13269 of *LNCS*, pages 230–250. Springer, 2022. doi:10.1007/978-3-031-09234-3_12.
- [HGSE24] Hosein Hadipour, Simon Gerhalter, Sadegh Sadeghi, and Maria Eichlseder. Improved search for integral, impossible differential and zero-correlation attacks application to ascon, forkskinny, skinny, mantis, PRESENT and qarmav2. *IACR Trans. Symmetric Cryptol.*, 2024(1):234–325, 2024. doi:10.46586/TOSC.V2024.I1.234-325.
- [HLK⁺13] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Donggeon Lee. LEA: A 128-bit block cipher for fast encryption on common processors. In *WISA 2013*, volume 8267 of *LNCS*, pages 3–27. Springer, 2013. doi:10.1007/978-3-319-05149-9_1.
- [HNE22] Hosein Hadipour, Marcel Nageler, and Maria Eichlseder. Throwing boomerangs into Feistel structures: Application to CLEFIA, WARP, LBlock, LBlock-s and TWINE. *IACR Trans. Symmetric Cryptol.*, 2022(3):271–302, 2022. doi:10.46586/tosc.v2022.i3.271-302.
- [HSE23] Hosein Hadipour, Sadegh Sadeghi, and Maria Eichlseder. Finding the impossible: Automated search for full impossible-differential, zero-correlation, and integral attacks. In *EUROCRYPT 2023*, volume 14007 of *LNCS*, pages 128–157. Springer, 2023. doi:10.1007/978-3-031-30634-1_5.
- [HY24] Le He and Hongbo Yu. Cryptanalysis of reduced-round SipHash. *Comput. J.*, 67(3):875–883, 2024. doi:10.1093/COMJNL/BXAD026.
- [IS13] Takanori Isobe and Kyoji Shibutani. Generic key recovery attack on feistel scheme. 8269:464–485, 2013. doi:10.1007/978-3-642-42033-7_24.
- [KAR20] Liliya Krалеva, Tomer Ashur, and Vincent Rijmen. Rotational cryptanalysis on MAC algorithm chaskey. In *ACNS 2020*, volume 12146 of *LNCS*, pages 153–168. Springer, 2020. doi:10.1007/978-3-030-57808-4_8.
- [KHS⁺03] Jongsung Kim, Seokhie Hong, Jaechul Sung, Changhoon Lee, and Sangjin Lee. Impossible differential cryptanalysis for block cipher structures. In Thomas Johansson and Subhamoy Maitra, editors, *INDOCRYPT*

- 2003, volume 2904 of *LNCS*, pages 82–96. Springer, 2003. doi:10.1007/978-3-540-24582-7_6.
- [Knu98] Lars Knudsen. DEAL – a 128-bit block cipher. *Complexity*, 258(2):216, 1998.
- [LDKK08] Jiqiang Lu, Orr Dunkelman, Nathan Keller, and Jongsung Kim. New impossible differential attacks on AES. In *INDOCRYPT 2008*, volume 5365 of *LNCS*, pages 279–293. Springer, 2008. doi:10.1007/978-3-540-89754-5_22.
- [Leu16] Gaëtan Leurent. Improved differential-linear cryptanalysis of 7-round chaskey with partitioning. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016*, volume 9665 of *LNCS*, pages 344–371. Springer, 2016. doi:10.1007/978-3-662-49890-3_14.
- [LKH⁺16] Hochang Lee, HyungChul Kang, Deukjo Hong, Jaechul Sung, and Seokhie Hong. New impossible differential characteristic of SPECK64 using MILP. *IACR Cryptol. ePrint Arch.*, page 1137, 2016. URL: <http://eprint.iacr.org/2016/1137>.
- [LKKD08] Jiqiang Lu, Jongsung Kim, Nathan Keller, and Orr Dunkelman. Improving the efficiency of impossible differential cryptanalysis of reduced Camellia and MISTY1. In *CT-RSA 2008*, volume 4964 of *LNCS*, pages 370–386. Springer, 2008. doi:10.1007/978-3-540-79263-5_24.
- [LLG⁺12] Ya Liu, Leibo Li, Dawu Gu, Xiaoyun Wang, Zhiqiang Liu, Jiazhe Chen, and Wei Li. New observations on impossible differential cryptanalysis of reduced-round camellia. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 90–109. Springer, 2012. doi:10.1007/978-3-642-34047-5_6.
- [LLJW21] Zhengbin Liu, Yongqiang Li, Lin Jiao, and Mingsheng Wang. A new method for searching optimal differential and linear trails in ARX ciphers. *IEEE Trans. Inf. Theory*, 67(2):1054–1068, 2021. doi:10.1109/TIT.2020.3040543.
- [LLWG14] Yiyuan Luo, Xuejia Lai, Zhongming Wu, and Guang Gong. A unified method for finding impossible differentials of block cipher structures. *Inf. Sci.*, 263:211–220, 2014. See also <http://eprint.iacr.org/2009/627>. doi:10.1016/J.INS.2013.08.051.
- [LPS21] Gaëtan Leurent, Clara Pernot, and André Schrottenloher. Clustering effect in simon and simeck. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021*, volume 13090 of *LNCS*, pages 272–302. Springer, 2021. doi:10.1007/978-3-030-92062-3_10.
- [LRC19] Hang Li, Jiongjiong Ren, and Shaozhen Chen. Improved integral attack on reduced-round simeck. *IEEE Access*, 7:118806–118814, 2019. doi:10.1109/ACCESS.2019.2936834.
- [LSG⁺23] Yin Lv, Danping Shi, Yi Guo, Qiu Chen, Lei Hu, and Zihui Guo. Automatic Demirci-Selçuk meet-in-the-middle attack on SIMON. *Comput. J.*, 66(12):3052–3068, 2023. doi:10.1093/COMJNL/BXAC149.
- [MDRMH10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. Improved impossible differential cryptanalysis of 7-round AES-128. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT*

- 2010, volume 6498 of *LNCS*, pages 282–291. Springer, 2010. doi:[10.1007/978-3-642-17401-8_20](https://doi.org/10.1007/978-3-642-17401-8_20).
- [MMH⁺14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In Antoine Joux and Amr M. Youssef, editors, *SAC 2014*, volume 8781 of *LNCS*, pages 306–323. Springer, 2014. doi:[10.1007/978-3-319-13051-4_19](https://doi.org/10.1007/978-3-319-13051-4_19).
- [NSB⁺07] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *CP 2007*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007. doi:[10.1007/978-3-540-74970-7_38](https://doi.org/10.1007/978-3-540-74970-7_38).
- [PT22] Thomas Peyrin and Quan Quan Tan. Mind your path: On (key) dependencies in differential characteristics. *IACR Trans. Symmetric Cryptol.*, 2022(4):179–207, 2022. doi:[10.46586/TOSC.V2022.I4.179-207](https://doi.org/10.46586/TOSC.V2022.I4.179-207).
- [QCW16] Lingyue Qin, Huaifeng Chen, and Xiaoyun Wang. Linear hull attack on round-reduced simeck with dynamic key-guessing techniques. In *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Proceedings, Part II*, volume 9723 of *LNCS*, pages 409–424. Springer, 2016. doi:[10.1007/978-3-319-40367-0_26](https://doi.org/10.1007/978-3-319-40367-0_26).
- [QHS16] Kexin Qiao, Lei Hu, and Siwei Sun. Differential analysis on simeck and SIMON with dynamic key-guessing techniques. In Olivier Camp, Steven Furnell, and Paolo Mori, editors, *ICISSP 2016*, volume 691 of *Communications in Computer and Information Science*, pages 64–85. Springer, 2016. doi:[10.1007/978-3-319-54433-5_5](https://doi.org/10.1007/978-3-319-54433-5_5).
- [RC19] Jiongjiong Ren and Shaozhen Chen. Cryptanalysis of reduced-round speck. *IEEE Access*, 7:63045–63056, 2019. doi:[10.1109/ACCESS.2019.2917015](https://doi.org/10.1109/ACCESS.2019.2917015).
- [SB18] Sadegh Sadeghi and Nasour Bagheri. Improved zero-correlation and impossible differential cryptanalysis of reduced-round SIMECK block cipher. *IET Inf. Secur.*, 12(4):314–325, 2018. doi:[10.1049/IET-IFS.2016.0590](https://doi.org/10.1049/IET-IFS.2016.0590).
- [SBS21] Mahshid Saberi, Nasour Bagheri, and Sadegh Sadeghi. Impossible differential and zero-correlation linear cryptanalysis of marx, chaskey and speck32. In *ICCKE 2021*, pages 48–54, 2021. doi:[10.1109/ICCKE54056.2021.9721479](https://doi.org/10.1109/ICCKE54056.2021.9721479).
- [SFW15] Ling Sun, Kai Fu, and Meiqin Wang. Improved zero-correlation cryptanalysis on SIMON. In *Inscrypt*, volume 9589 of *LNCS*, pages 125–143. Springer, 2015.
- [SHMS14] Ling Song, Lei Hu, Bingke Ma, and Danping Shi. Match box meet-in-the-middle attacks on the SIMON family of block ciphers. In Thomas Eisenbarth and Erdiñç Öztürk, editors, *LightSec 2014*, volume 8898 of *LNCS*, pages 140–151. Springer, 2014. doi:[10.1007/978-3-319-16363-5_9](https://doi.org/10.1007/978-3-319-16363-5_9).
- [SSA⁺07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In *FSE 2007*, volume 4593 of *LNCS*, pages 181–195. Springer, 2007. doi:[10.1007/978-3-540-74619-5_12](https://doi.org/10.1007/978-3-540-74619-5_12).

- [ST17] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects. In *EUROCRYPT 2017*, pages 185–215, Cham, 2017. Springer. doi:10.1007/978-3-319-56617-7_7.
- [WW22] Feifan Wang and Gaoli Wang. Improved differential-linear attack with application to round-reduced speck32/64. In *ACNS 2022*, volume 13269 of *LNCS*, pages 792–808. Springer, 2022. doi:10.1007/978-3-031-09234-3_39.
- [WWJZ18] Ning Wang, Xiaoyun Wang, Keting Jia, and Jingyuan Zhao. Differential attacks on reduced SIMON versions with dynamic key-guessing techniques. *Sci. China Inf. Sci.*, 61(9):098103:1–098103:3, 2018. doi:10.1007/S11432-017-9231-5.
- [XLSL19] Wenqian Xin, Yunwen Liu, Bing Sun, and Chao Li. Improved cryptanalysis on SipHash. In Yi Mu, Robert H. Deng, and Xinyi Huang, editors, *CANS 2019*, volume 11829 of *LNCS*, pages 61–79. Springer, 2019. doi:10.1007/978-3-030-31578-8_4.
- [YZS⁺15] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D. Aagaard, and Guang Gong. The simeck family of lightweight block ciphers. In *CHES 2015*, volume 9293 of *LNCS*, pages 307–329. Springer, 2015. doi:10.1007/978-3-662-48324-4_16.
- [ZGH16] Kai Zhang, Jie Guan, and Bin Hu. Zero correlation linear cryptanalysis on LEA family ciphers. *J. Commun.*, 11(7):677–685, 2016. doi:10.12720/JCM.11.7.677-685.
- [ZGHL18] Kai Zhang, Jie Guan, Bin Hu, and Dongdai Lin. Security evaluation on simeck against zero-correlation linear cryptanalysis. *IET Inf. Secur.*, 12(1):87–93, 2018. doi:10.1049/IET-IFS.2016.0503.
- [ZLW⁺23] Kai Zhang, Xuejia Lai, Lei Wang, Jie Guan, and Bin Hu. A revisited security evaluation of simeck family ciphers against impossible differential cryptanalysis. *Sci. China Inf. Sci.*, 66(3), 2023. doi:10.1007/S11432-022-3466-X.
- [ZWF07] Wentao Zhang, Wenling Wu, and Dengguo Feng. New results on impossible differential cryptanalysis of reduced AES. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *ICISC 2007*, volume 4817 of *LNCS*, pages 239–250. Springer, 2007. doi:10.1007/978-3-540-76788-6_19.
- [ZWL⁺23] Kai Zhang, Senpeng Wang, Xuejia Lai, Lei Wang, Jie Guan, Bin Hu, and Tairong Shi. Impossible differential cryptanalysis and a security evaluation framework for and-rx ciphers. *IEEE Transactions on Information Theory*, pages 1–1, 2023. doi:10.1109/TIT.2023.3292241.

Appendix

A Constraints to Model the Toffoli Gate for SIMON

The constraints derived in [Proposition 8](#) for the linear propagation through the Toffoli gate S_{SIMON} are as follows:

if	$(x_0 = 0 \wedge x_1 = 0 \wedge x_2 = 0)$	then	$(y_0 = 0 \wedge y_1 = 0 \wedge y_2 = 0)$
elseif	$(x_0 = 0 \wedge x_1 = 0 \wedge x_2 = 1)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 1)$
elseif	$(x_0 = 0 \wedge x_1 = 1 \wedge x_2 = 0)$	then	$(y_0 = 0 \wedge y_1 = 1 \wedge y_2 = 0)$
elseif	$(x_0 = 0 \wedge x_1 = 1 \wedge x_2 = 1)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 1)$
elseif	$(x_0 = 0 \wedge x_1 = -1 \wedge x_2 = 0)$	then	$(y_0 = 0 \wedge y_1 = -1 \wedge y_2 = 0)$
elseif	$(x_0 = 0 \wedge x_1 = -1 \wedge x_2 = 1)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 1)$
elseif	$(x_0 = 1 \wedge x_1 = 0 \wedge x_2 = 0)$	then	$(y_0 = 1 \wedge y_1 = 0 \wedge y_2 = 0)$
elseif	$(x_0 = 1 \wedge x_1 = 0 \wedge x_2 = 1)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 1)$
elseif	$(x_0 = 1 \wedge x_1 = 1 \wedge x_2 = 0)$	then	$(y_0 = 1 \wedge y_1 = 1 \wedge y_2 = 0)$
elseif	$(x_0 = 1 \wedge x_1 = 1 \wedge x_2 = 1)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 1)$
elseif	$(x_0 = 1 \wedge x_1 = -1 \wedge x_2 = 0)$	then	$(y_0 = 1 \wedge y_1 = -1 \wedge y_2 = 0)$
elseif	$(x_0 = 1 \wedge x_1 = -1 \wedge x_2 = 1)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 1)$
elseif	$(x_0 = -1 \wedge x_1 = 0 \wedge x_2 = 0)$	then	$(y_0 = -1 \wedge y_1 = 0 \wedge y_2 = 0)$
elseif	$(x_0 = -1 \wedge x_1 = 0 \wedge x_2 = 1)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 1)$
elseif	$(x_0 = -1 \wedge x_1 = 0 \wedge x_2 = 1)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 1)$
elseif	$(x_0 = -1 \wedge x_1 = 1 \wedge x_2 = 0)$	then	$(y_0 = -1 \wedge y_1 = 1 \wedge y_2 = 0)$
elseif	$(x_0 = -1 \wedge x_1 = 1 \wedge x_2 = 1)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 1)$
elseif	$(x_0 = -1 \wedge x_1 = -1 \wedge x_2 = 0)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 0)$
elseif	$(x_0 = -1 \wedge x_1 = -1 \wedge x_2 = 1)$	then	$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = 1)$
else			$(y_0 = -1 \wedge y_1 = -1 \wedge y_2 = -1)$

B Algorithms for the Full SIMON Model

Algorithms 1, 2, and 3 provide our detailed model for SIMON as introduced in [Section 4](#).

Algorithm 1: CSP_U model of difference propagation through E_D for SIMON

Input: The integer number r_D
Output: CSP_U

- 1 Declare an empty CSP model \mathcal{M} ;
- 2 $\mathcal{M}.var \leftarrow \{xu_r^0[i] \in \{-1, 0, 1\}, xu_r^1[i] \in \{-1, 0, 1\} : 0 \leq r \leq r_D, 0 \leq i \leq (n-1)\}$;
- 3 $\mathcal{M}.var \leftarrow \{yu_r^0[i] \in \{-1, 0, 1\} : 0 \leq r \leq r_D - 1, 0 \leq i \leq (n-1)\}$;
- 4 $\mathcal{M}.var \leftarrow \{zu_r^0[i] \in \{-1, 0, 1\} : 0 \leq r \leq r_D - 1, 0 \leq i \leq (n-1)\}$;
- 5 $\mathcal{M}.var \leftarrow \{wu_r^0[i] \in \{-1, 0, 1\} : 0 \leq r \leq r_D - 1, 0 \leq i \leq (n-1)\}$;
- 6 $\mathcal{M}.var \leftarrow \{pu_r^0[i] \in \{-1, 0, 1\}, qu_r^0[i] \in \{-1, 0, 1\} : 0 \leq r \leq r_D - 1, 0 \leq i \leq (n-1)\}$;
- 7 **for** $r = 0, \dots, r_D - 1$ **do**
- 8 $\mathcal{M}.con \leftarrow yu_r^0 = xu_r^0 \lll 8$;
- 9 $\mathcal{M}.con \leftarrow zu_r^0 = xu_r^0 \lll 1$;
- 10 $\mathcal{M}.con \leftarrow wu_r^0 = xu_r^0 \lll 2$;
- 11 **for** $r = 0, \dots, r_D - 1, i = 0, \dots, (n-1)$ **do**
- 12 $\mathcal{M}.con \leftarrow \text{AND}(yu_r^0[i], zu_r^0[i], pu_r^0[i])$;
- 13 **for** $r = 0, \dots, r_D - 1, i = 0, \dots, (n-1)$ **do**
- 14 $\mathcal{M}.con \leftarrow \text{XOR}(pu_r^0[i], wu_r^0[i], qu_r^0[i])$;
- 15 **for** $r = 0, \dots, r_D - 1, i = 0, \dots, (n-1)$ **do**
- 16 $\mathcal{M}.con \leftarrow \text{XOR}(qu_r^0[i], xu_r^1[i], xu_{r+1}^0[i])$;
- 17 **for** $r = 0, \dots, r_D - 1, i = 0, \dots, (n-1)$ **do**
- 18 $\mathcal{M}.con \leftarrow (xu_r^0[i] = xu_{r+1}^1[i])$;
- 19 **return** \mathcal{M} ;

Algorithm 2: CSP_B^{dp} model of difference propagation through E_B for SIMON

Input: CSP_U.var, and the integer number r_B
Output: CSP_B^{dp}

- 1 Declare an empty CSP model \mathcal{M} ;
- 2 $\mathcal{M}.var \leftarrow \{dxu_r^0[i] \in \{-1, 0, 1\}, dxu_r^1[i] \in \{-1, 0, 1\} : 0 \leq r \leq r_B, 0 \leq i \leq (n-1)\}$;
- 3 $\mathcal{M}.var \leftarrow \{dyu_r^0[i] \in \{-1, 0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 4 $\mathcal{M}.var \leftarrow \{dzu_r^0[i] \in \{-1, 0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 5 $\mathcal{M}.var \leftarrow \{dwu_r^0[i] \in \{-1, 0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 6 $\mathcal{M}.var \leftarrow \{dpu_r^0[i] \in \{-1, 0, 1\}, dqu_r^0[i] \in \{-1, 0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 7 $\mathcal{M}.var \leftarrow \{cb_r^1[i] \in \{0, 1\}, cb_r^2[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 8 **for** $i = 0, \dots, (n-1)$ **do**
- 9 $\mathcal{M}.con \leftarrow (dxu_{r_B}^0[i] = xu_0^0[i])$;
- 10 **for** $i = 0, \dots, (n-1)$ **do**
- 11 $\mathcal{M}.con \leftarrow (dxu_{r_B}^1[i] = xu_0^1[i])$;
- 12 **for** $r = 0, \dots, r_B - 1$ **do**
- 13 $\mathcal{M}.con \leftarrow dyu_r^0 = dxu_{r+1}^1 \lll 8$;
- 14 $\mathcal{M}.con \leftarrow dzu_r^0 = dxu_{r+1}^1 \lll 1$;
- 15 $\mathcal{M}.con \leftarrow dwu_r^0 = dxu_{r+1}^1 \lll 2$;
- 16 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 17 $\mathcal{M}.con \leftarrow \text{AND}(dyu_r^0[i], dzu_r^0[i], dpu_r^0[i])$;
- 18 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 19 $\mathcal{M}.con \leftarrow \text{XOR}(dpu_r^0[i], dwu_r^0[i], dqu_r^0[i])$;
- 20 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 21 $\mathcal{M}.con \leftarrow \text{XOR}(dqu_r^0[i], dxu_{r+1}^0[i], dxu_{r+1}^1[i])$;
- 22 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 23 $\mathcal{M}.con \leftarrow (dxu_r^0[i] = dxu_{r+1}^1[i])$;
- 24 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 25 $\mathcal{M}.con \leftarrow \text{XOR}_{dp}(dpu_r^0[i], dwu_r^0[i], dqu_r^0[i], cb_r^1[i])$;
- 26 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 27 $\mathcal{M}.con \leftarrow \text{XOR}_{dp}(dqu_r^0[i], dxu_r^1[i], dxu_{r+1}^0[i], cb_r^2[i])$;
- 28 **return** \mathcal{M} ;

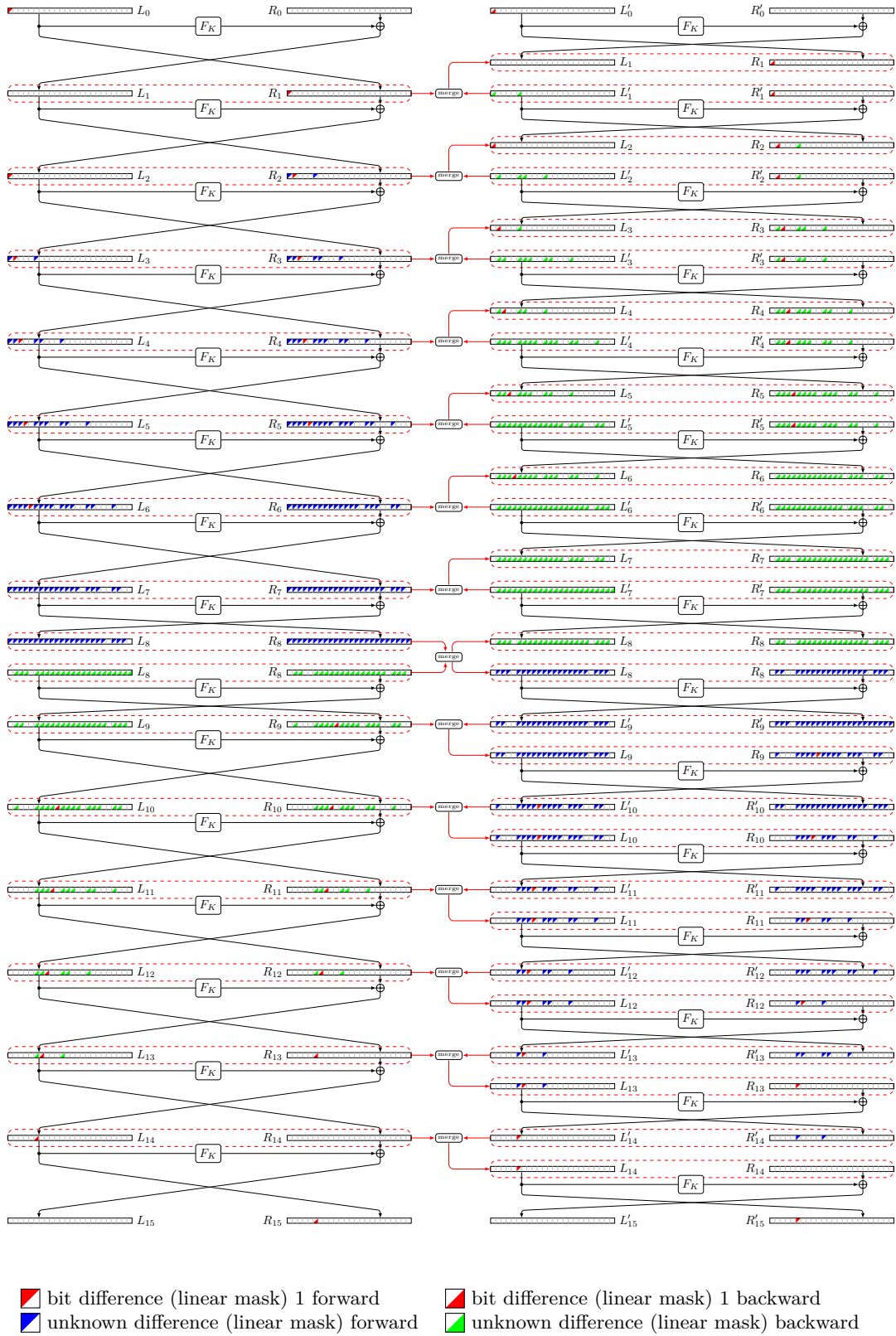


Figure 9: 15-round (indirect) ZC distinguisher for Simeck48. In this case, the bit difference in the upper triangle of $L_2[0]$ (in the left-hand column) is 1, whereas the bit difference in the lower triangle of $L'_2[0]$ is 0. This leads to a contradiction occurring in the second round.

Algorithm 3: CSP_B^{gd} model for guess-and-determine through E_B for SIMON

Input: CSP_U.var, CSP_B^{dp}, and the integer number r_B
Output: CSP_B^{gd}

- 1 Declare an empty CSP model \mathcal{M} ;
- 2 $\mathcal{M}.var \leftarrow \{\text{kdxu}_r^0[i] \in \{0, 1\}, \text{kdxu}_r^1[i] \in \{0, 1\} : 0 \leq r \leq r_B, 0 \leq i \leq (n-1)\}$;
- 3 $\mathcal{M}.var \leftarrow \{\text{kdyu}_r^0[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 4 $\mathcal{M}.var \leftarrow \{\text{kzdu}_r^0[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 5 $\mathcal{M}.var \leftarrow \{\text{kdwu}_r^0[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 6 $\mathcal{M}.var \leftarrow \{\text{kdpu}_r^0[i] \in \{0, 1\}, \text{kdqu}_r^0[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 7 $\mathcal{M}.var \leftarrow \{\text{kdyu}_r^1[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 8 $\mathcal{M}.var \leftarrow \{\text{kzdu}_r^1[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 9 $\mathcal{M}.var \leftarrow \{\text{kdwu}_r^1[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 10 $\mathcal{M}.var \leftarrow \{\text{kxu}_r^0[i] \in \{0, 1\}, \text{kxu}_r^1[i] \in \{0, 1\} : 0 \leq r \leq r_B, 0 \leq i \leq (n-1)\}$;
- 11 $\mathcal{M}.var \leftarrow \{\text{kyu}_r^0[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 12 $\mathcal{M}.var \leftarrow \{\text{kzu}_r^0[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 13 $\mathcal{M}.var \leftarrow \{\text{kwu}_r^0[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 14 $\mathcal{M}.var \leftarrow \{\text{kpu}_r^0[i] \in \{0, 1\}, \text{kqu}_r^0[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 15 $\mathcal{M}.var \leftarrow \{\text{kyu}_r^1[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 16 $\mathcal{M}.var \leftarrow \{\text{kzu}_r^1[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 17 $\mathcal{M}.var \leftarrow \{\text{kwu}_r^1[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 18 $\mathcal{M}.var \leftarrow \{\text{ikb}_r[i] \in \{0, 1\} : 0 \leq r \leq r_B - 1, 0 \leq i \leq (n-1)\}$;
- 19 **for** $i = 0, \dots, (n-1)$ **do**
- 20 $\mathcal{M}.con \leftarrow (\text{if } \text{dxu}_{r_B}^0[i] \geq 0 \text{ then } \text{kdxu}_{r_B}^0[i] = 0 \text{ else true});$
- 21 **for** $i = 0, \dots, (n-1)$ **do**
- 22 $\mathcal{M}.con \leftarrow (\text{if } \text{dxu}_{r_B}^1[i] \geq 0 \text{ then } \text{kdxu}_{r_B}^1[i] \text{ else true});$
- 23 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 24 $\mathcal{M}.con \leftarrow \text{XOR}_1^{\text{gd}}(\text{dxu}_r^1[i], \text{dqu}_r^0[i], \text{kdxu}_{r+1}^0[i], \text{cb}_r^2[i], \text{kdxu}_r^1[i], \text{kdqu}_r^0[i]);$
- 25 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 26 $\mathcal{M}.con \leftarrow \text{XOR}_1^{\text{gd}}(\text{dqu}_r^0[i], \text{dwu}_r^0[i], \text{kdqu}_r^0[i], \text{cb}_r^1[i], \text{kdpu}_r^0[i], \text{kdwu}_r^0[i]);$
- 27 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 28 $\mathcal{M}.con \leftarrow \text{AND}_1^{\text{gd}}(\text{dyu}_r^0[i], \text{dzu}_r^0[i], \text{kdpu}_r^0[i], \text{kdyu}_r^0[i], \text{kzdu}_r^0[i]);$
- 29 **for** $r = 0, \dots, r_B - 1$ **do**
- 30 $\mathcal{M}.con \leftarrow \text{kdyu}_r^0 = \text{kdyu}_r^0 \ggg 8;$
- 31 $\mathcal{M}.con \leftarrow \text{kzdu}_r^0 = \text{kzdu}_r^0 \ggg 1;$
- 32 $\mathcal{M}.con \leftarrow \text{kdwu}_r^0 = \text{kdwu}_r^0 \ggg 2;$
- 33 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 34 $\mathcal{M}.con \leftarrow (\text{if } \text{kdyu}_r^0[i] = \text{kzdu}_r^0[i] = \text{kdwu}_r^0[i] = \text{kdxu}_{r+1}^1[i] = 0 \text{ then } \text{kdxu}_r^0[i] =$
 $0 \text{ else } \text{kdxu}_r^0[i] = 1);$
- 35 **for** $i = 0, \dots, (n-1)$ **do**
- 36 $\mathcal{M}.con \leftarrow (\text{kxu}_{r_B}^0[i] = 0);$
- 37 **for** $i = 0, \dots, (n-1)$ **do**
- 38 $\mathcal{M}.con \leftarrow (\text{kxu}_{r_B}^1[i] = 0);$
- 39 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 40 $\mathcal{M}.con \leftarrow \text{XOR}_2^{\text{gd}}(\text{kxu}_{r+1}^0[i], \text{kxu}_r^1[i], \text{kqu}_r^0[i]);$
- 41 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 42 $\mathcal{M}.con \leftarrow \text{XOR}_2^{\text{gd}}(\text{kqu}_r^0[i], \text{kpu}_r^0[i], \text{kwu}_r^0[i]);$
- 43 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 44 $\mathcal{M}.con \leftarrow \text{AND}_2^{\text{gd}}(\text{kdpu}_r^0[i], \text{kpu}_r^0[i], \text{dyu}_r^0[i], \text{dzu}_r^0[i], \text{kyu}_r^0[i], \text{kyu}_r^0[i]);$
- 45 **for** $r = 0, \dots, r_B - 1$ **do**
- 46 $\mathcal{M}.con \leftarrow \text{kyu}_r^0 = \text{kyu}_r^0 \ggg 8;$
- 47 $\mathcal{M}.con \leftarrow \text{kzu}_r^0 = \text{kzu}_r^0 \ggg 1;$
- 48 $\mathcal{M}.con \leftarrow \text{kwu}_r^0 = \text{kwu}_r^0 \ggg 2;$
- 49 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 50 $\mathcal{M}.con \leftarrow (\text{if } \text{kyu}_r^0[i] = \text{kzu}_r^0[i] = \text{kwu}_r^0[i] = \text{kxu}_{r+1}^1[i] = 0 \text{ then } \text{kxu}_r^0[i] = 0 \text{ else } \text{kxu}_r^0[i] =$
 $1);$
- 51 **for** $r = 0, \dots, r_B - 1, i = 0, \dots, (n-1)$ **do**
- 52 $\mathcal{M}.con \leftarrow (\text{if } \text{kxu}_{r+1}^0[i] = 1 \text{ then } \text{ikb}_r[i] = 1 \text{ else } \text{ikb}_r[i] = 0);$

C Brief Specification of ARX Ciphers

C.1 LEA

LEA (Lightweight Encryption Algorithm) [HLK⁺13] is a block cipher developed by the Korea Internet & Security Agency (KISA) to provide lightweight encryption in resource-constrained environments. LEA encrypts data 1.5–2 times faster than AES, the most popular block cipher. Operating on 128-bit blocks, LEA supports key lengths of 128, 192, and 256 bits. The round's numbers are 24, 28, and 32 for 128-, 192- and 256-bit keys, respectively.

The encryption algorithm of LEA divides a plaintext of four 32-bit words $(x_0^0, x_0^1, x_0^2, x_0^3)$ into a ciphertext $(x_i^0, x_i^1, x_i^2, x_i^3)$, where r represents the number of rounds. The round function for round $r, r = 0, \dots, n - 1$ is defined as follows:

$$\begin{aligned} x_{i+1}^0 &\leftarrow ((x_i^0 \oplus k_i^0) \boxplus (x_i^1 \oplus k_i^1)) \lll 9 \\ x_{i+1}^1 &\leftarrow ((x_i^1 \oplus k_i^2) \boxplus (x_i^2 \oplus k_i^3)) \ggg 5 \\ x_{i+1}^2 &\leftarrow ((x_i^2 \oplus k_i^4) \boxplus (x_i^3 \oplus k_i^5)) \ggg 3 \\ x_{i+1}^3 &\leftarrow x_i^0 \end{aligned}$$

where $k_i = k_i^0, k_i^1, k_i^2, k_i^3, k_i^4, k_i^5$ is the round key generated by the key schedule. One round of LEA can be seen in Figure 10a.

C.2 SPECK

The lightweight block cipher SPECK [BSS⁺15] was announced by the NSA in 2013. SPECK2n/mn has a block size of $2n$ bits and a key size of mn bits, where n can be 16, 24, 32, 48, and 64, and m can be 2, 3, or 4.

The round function of SPECK is defined in Figure 10b with the rotation parameter $(\alpha, \beta) = (7, 2)$ if block size is 32, and $(8, 3)$ otherwise. Let (x_i^0, x_i^1) be the input of the i -th round and (x_{i+1}^0, x_{i+1}^1) be the output. In each round, the state is updated as follows:

$$(x_{i+1}^0, x_{i+1}^1) = R_{k^i}(x_i^0, x_i^1) = (((x_i^0 \ggg \alpha) \boxplus x_i^1) \oplus k^i, (x_i^1 \lll \beta) \oplus (((x_i^0 \ggg \alpha) \boxplus x_i^1) \oplus k^i))$$

where k^i is the round key. The key schedule reuses the round function to generate round keys.

C.3 ChaCha

Chacha [Ber08] is a stream cipher designed by Daniel J. Bernstein. It belongs to the family of ciphers known as Salsa20. ChaCha operates on 512-bit blocks, which is divided into 16 words, and supports key lengths of 128 or 256 bits. The state of ChaCha can be presented as a 4×4 matrix:

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & t_1 & v_0 & v_1 \end{pmatrix}$$

In the initial matrix, the first row contains 4 constant words: c_0, c_1, c_2, c_3 . Following are two rows, each comprising 8 key words k_0, k_1, \dots, k_7 . The final row consists of the block counter first, followed by the nonce.

The rows and columns are updated by an operation called quarter-round, which transforms a 4-word vector (a, b, c, d) into (a'', b'', c'', d'') via an intermediate vector (a', b', c', d') :

$$\begin{aligned}
 a' &= a \boxplus b, \\
 d' &= (d \oplus a') \lll 16, \\
 c' &= c \boxplus d, \\
 b' &= (b \oplus c') \lll 12, \\
 a'' &= a' \boxplus b', \\
 d'' &= (d' \oplus a'') \lll 8, \\
 c'' &= c' \boxplus d'', \\
 b'' &= (b' \oplus c'') \lll 7.
 \end{aligned}$$

Figure 10c illustrates the round function of ChaCha.

C.4 SipHash

SipHash is a family of pseudorandom functions introduced by Aumasson and Bernstein at Indocrypt 2012 [AB12], designed specifically for short message inputs. SipHash has an internal state size of 256 bits, uses a 128-bit key, and produces a 64-bit tag. SipHash variants are denoted as SipHash-c-d where c is the number of *Compression* rounds processing each message block and d is the number of *Finalization* rounds. The 64-bit tag is computed as follows:

- *Initialization*: Four 64-bit words of internal state v_0, v_1, v_2 and v_3 with the 128-bit key $K = k_1 || k_0$ are initialized as

$$\begin{aligned}
 v_0 &= k_0 \oplus 736f6d6570736575 \\
 v_1 &= k_1 \oplus 646f72616e646f6d \\
 v_2 &= k_0 \oplus 6c7967656e657261 \\
 v_3 &= k_1 \oplus 7465646279746573
 \end{aligned}$$

- *Compression*: SipHash-c-d processes the b -byte string m by parsing it into 64-bit little-endian words. Each word is processed iteratively, first with $v_3 \oplus = m_i$, then through c iterations of SipRound, and finally with $v_0 \oplus = m_i$.
- *Finalization*:: Once all message blocks are processed, the constant **ff** is xored with v_2 . Then d iterations of SipRound are executed, and SipHash-c-d yields the 64-bit value: $v_0 \oplus v_1 \oplus v_2 \oplus v_3$

The round function of the SipHash is shown in Figure 10d.

C.5 Chaskey

Chaskey is a permutation-based MAC algorithm presented by Mouha *et al.* in 2014 [MMH⁺14], inspired by Siphash. Chaskey processes an arbitrary-sized message M and a 128-bit key K . The message M is divided into blocks m_1, m_2, \dots, m_k of 128 bits each. In case the last block is incomplete, padding is applied. It generates a t -bit tag τ (where $t \leq n$) to authenticate the message M . The core function is a permutation constructed using the ARX design as depicted in Figure 10e.

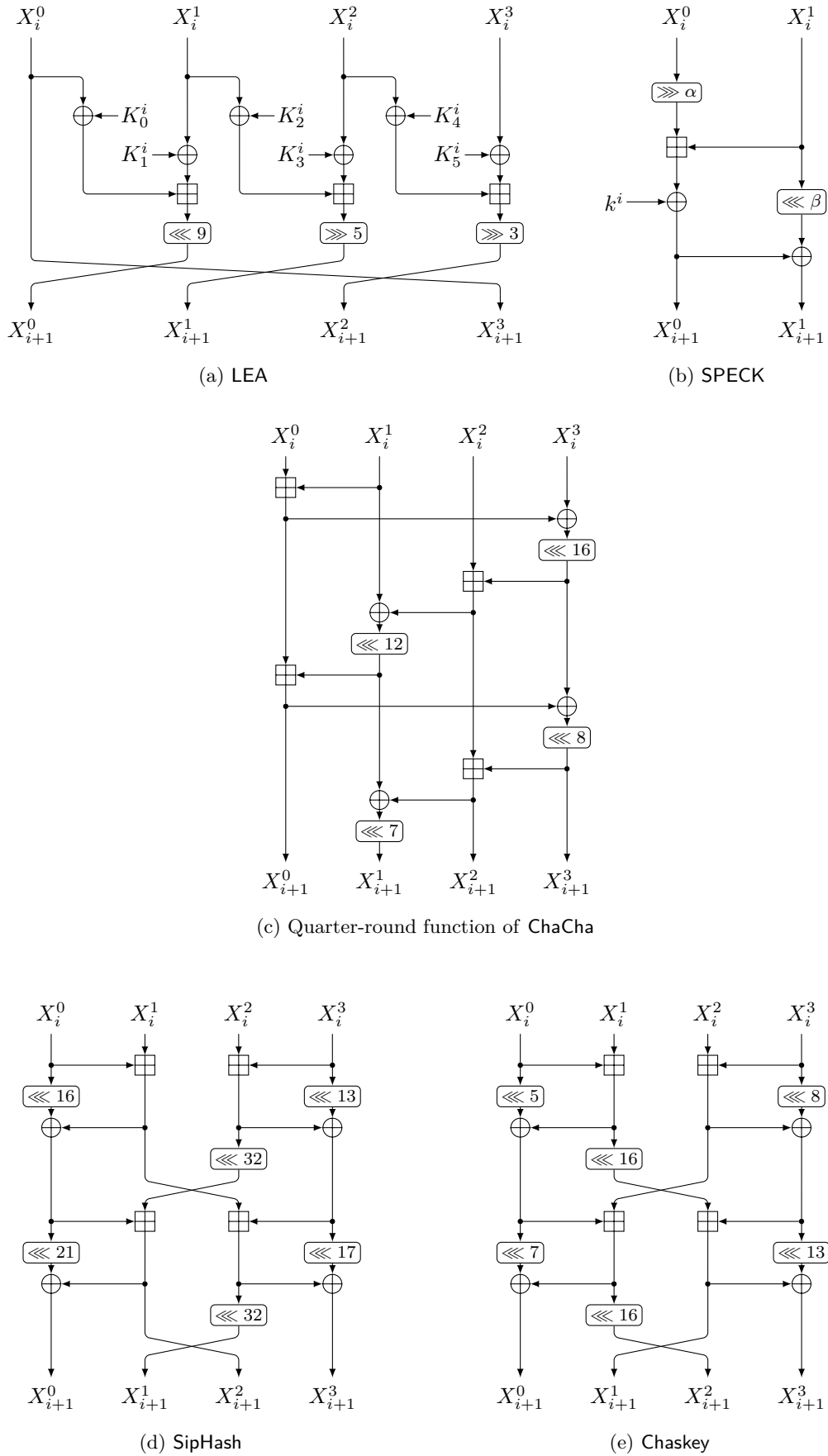


Figure 10: Round functions of various ARX designs.

D Brief Specification and Existing Cryptanalysis of AndRX Ciphers

D.1 Specification of SIMON

SIMON is a family of lightweight ciphers designed by the National Security Agency (NSA) in 2013 [BSS⁺15] to provide high security and efficiency for use in constrained environments. It is based on a typical Feistel design, where each block is divided into two halves and consists of bitwise AND, rotation, and XOR operations. SIMON has several variants on n -bit words and the block size is $2n$ -bit for $n \in \{16, 24, 32, 48, 64\}$. The key size is a multiple of n by m , for $m \in \{2, 3, 4\}$. Each variant can be denoted as SIMON $2n/mn$. The number of rounds depends on the block size and key size, according to Table 10.

Table 10: SIMON parameters

Variant	Block size $2n$	Key size mn	Word size n	Key words m	Round T
SIMON32	32	64	16	4	32
SIMON48	48	72	24	3	36
		96		4	36
SIMON64	64	96	32	3	42
		128		4	44
SIMON96	96	96	48	2	52
		144		3	54
SIMON128	128	128	64	2	68
		192		3	69
		256		4	72

Figure 11 illustrates the operations of the round function. Let L_i, R_i represent the left and right n -bit input words to the i -th round of SIMON, the output of i -th round L_{i+1}, R_{i+1} is computed as:

$$R_{i+1} = L_i$$

$$L_{i+1} = R_i \oplus K_i \oplus ((L_i \lll 8) \odot (L_i \lll 1)) \oplus (L_i \lll 2)$$

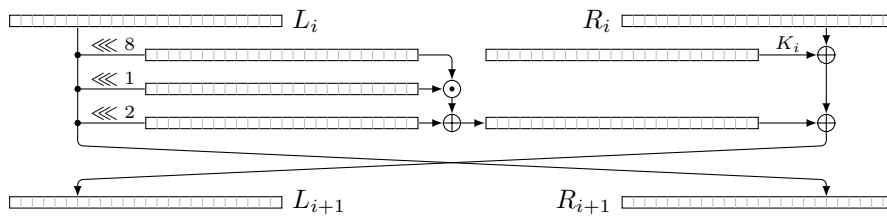


Figure 11: Round function of SIMON-48

The key schedule function of SIMON is linear. Depending on the size of the master key, the subkeys are derived on mn -bit word. A more detailed specification can be found in [BSS⁺15].

D.2 Specification of Simeck

Simeck is also a family of lightweight Feistel block ciphers introduced in CHES 2015 [YZS⁺15] that combines the most advantageous features from both SIMON and SPECK.

The Simeck block cipher with a $2n$ -bit block is denoted Simeck $2n/4n$, where $n \in \{16, 24, 32\}$, with a $4n$ -bit key. More specifically, there are 3 variants of Simeck, namely Simeck32/64, Simeck48/96 and Simeck64/128. The corresponding numbers of rounds for these variants are 32, 36, and 44, respectively. As shown in Figure 12, the round function of Simeck is also composed of three operations: AND, rotation, and XOR, slightly modified from SIMON.

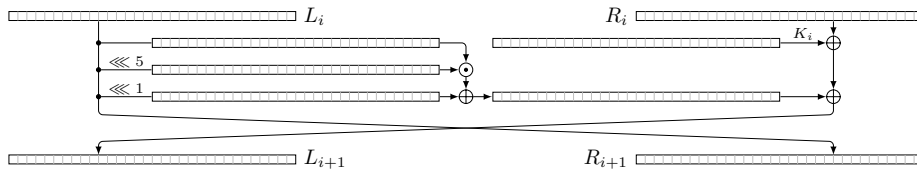


Figure 12: Round function of Simeck-64

Let L_i, R_i represent the left and right n -bit input words to the i -th round of Simeck, the output of i -th round L_{i+1}, R_{i+1} is computed as:

$$\begin{aligned} R_{i+1} &= L_i \\ L_{i+1} &= R_i \oplus K_i \oplus (L_i \odot (L_i \lll 1)) \oplus (L_i \lll 2) \end{aligned}$$

The key schedule of Simeck uses an LFSR procedure. A given master key generates the subkeys. For more details of Simeck key scheduling, we refer the reader to [YZS⁺15].

D.3 Cryptanalysis of SIMON

Table 11: Cryptanalysis of SIMON

Cipher	Attack	#R	Time	Data	Mem.	Ref.
SIMON-32-64	Differential	22	$2^{58.76}$	2^{32}	-	[QHS16]
	Linear	23	$2^{61.84}$	$2^{31.19}$	-	[CW16]
	MITM	18	$2^{62.57}$	2^3	-	[SHMS14]
	DS-MITM	16	$2^{56.29}$	2^{30}	-	[LSG ⁺ 23]
	Zero-correlation	21	$2^{59.4}$	2^{32}	2^{31}	[SFW15]
	Integral	24	2^{63}	2^{32}	$2^{33.64}$	[CCW ⁺ 18]
SIMON-48-72	Differential	23	$2^{63.25}$	2^{47}	-	[WWJZ18]
	Linear	24	$2^{67.89}$	$2^{47.92}$	-	[CW16]
	MITM	17	$2^{71.75}$	2^3	-	[SHMS14]
	DS-MITM	16	$2^{63.24}$	2^{44}	-	[LSG ⁺ 23]
	Zero-correlation	21	$2^{59.4}$	2^{48}	2^{43}	[SFW15]
	Integral	24	2^{71}	2^{48}	2^{50}	[CCW ⁺ 18]
SIMON-48-96	Differential	24	$2^{78.99}$	2^{48}	-	[WWJZ18]
	Linear	25	$2^{47.92}$	$2^{89.89}$	-	[CW16]
	MITM	19	$2^{95.26}$	2^3	-	[SHMS14]
	DS-MITM	18	$2^{91.62}$	2^{45}	-	[LSG ⁺ 23]
	Zero-correlation	22	$2^{80.5}$	2^{48}	2^{43}	[SFW15]
	Integral	25	2^{95}	2^{48}	2^{50}	[CCW ⁺ 18]
SIMON-64-96	Differential	30	2^{88}	$2^{63.3}$	-	[WWJZ18]
	Linear	30	$2^{93.62}$	$2^{63.53}$	-	[CW16]
	MITM	17	$2^{94.05}$	2^3	-	[SHMS14]
	DS-MITM	18	$2^{95.94}$	2^{58}	-	[LSG ⁺ 23]
	Zero-correlation	23	$2^{90.4}$	2^{64}	2^{54}	[SFW15]
	SIMON-64-128	Differential	31	2^{120}	$2^{63.3}$	-
Linear		31	2^{120}	$2^{63.53}$	-	[CW16]
MITM		19	$2^{126.01}$	2^3	-	[SHMS14]
DS-MITM		19	$2^{100.94}$	2^{58}	-	[LSG ⁺ 23]
Zero-correlation		24	$2^{116.8}$	2^{64}	2^{54}	[SFW15]
SIMON-96-96		Differential	37	$2^{87.17}$	2^{95}	-
	Linear	43	$2^{89.6}$	2^{94}	-	[LPS21]
	MITM	-	-	-	-	-
	DS-MITM	18	$2^{77.92}$	2^{36}	-	[LSG ⁺ 23]
	Zero-correlation	-	-	-	-	-
	SIMON-96-144	Differential	37	$2^{130.75}$	2^{95}	-
Linear		45	$2^{136.5}$	2^{95}	-	[LPS21]
MITM		21	$2^{141.27}$	2^4	-	[SHMS14]
DS-MITM		20	$2^{111.90}$	2^{36}	-	[LSG ⁺ 23]
Zero-correlation		28	2^{141}	2^{96}	2^{85}	[SFW15]
SIMON-128-128		Differential	50	$2^{119.19}$	2^{127}	-
	Linear	53	2^{121}	2^{127}	-	[LPS21]
	MITM	-	-	-	-	-
	DS-MITM	21	$2^{106.98}$	2^{47}	-	[LSG ⁺ 23]
	Zero-correlation	-	-	-	-	-
	SIMON-128-192	Differential	51	$2^{183.17}$	2^{127}	-
Linear		55	$2^{185.2}$	2^{127}	-	[LPS21]
MITM		25	$2^{190.60}$	2^3	-	[SHMS14]
DS-MITM		23	$2^{150.46}$	2^{62}	-	[LSG ⁺ 23]
Zero-correlation		32	$2^{156.8}$	2^{128}	2^{117}	[SFW15]
SIMON-128-256		Differential	51	$2^{247.17}$	2^{127}	-
	Linear	56	2^{249}	2^{126}	-	[LPS21]
	MITM	25	$2^{253.94}$	2^3	-	[SHMS14]
	DS-MITM	26	$2^{250.08}$	2^{53}	-	[LSG ⁺ 23]
	Zero-correlation	34	$2^{255.6}$	2^{128}	2^{117}	[SFW15]

D.4 Cryptanalysis of Simeck

Table 12: Cryptanalysis of Simeck

Cipher	Attack	#R	Time	Data	Mem.	Ref.
Simeck-32-64	Differential	22	$2^{57.9}$	2^{32}	-	[QHS16]
	Linear	23	$2^{61.78}$	$2^{31.91}$	-	[QCW16]
	Zero-correlation	21	$2^{58.78}$	2^{32}	2^{31}	[ZGHL18]
	Integral	22	2^{63}	2^{31}	$2^{55.88}$	[LRC19]
Simeck-48-96	Differential	28	$2^{68.3}$	2^{46}	-	[QHS16]
	Linear	32	$2^{90.9}$	2^{47}	-	[LPS21]
	Zero-correlation	27	$2^{85.67}$	2^{48}	$2^{58.78}$	[SB18]
	Integral	26	2^{95}	2^{47}	$2^{82.52}$	[LRC19]
Simeck-64-128	Differential	35	$2^{116.3}$	2^{63}	-	[QHS16]
	Linear	42	$2^{123.9}$	$2^{63.5}$	-	[LPS21]
	Zero-correlation	31	$2^{124.08}$	2^{64}	$2^{89.35}$	[SB18]
	Integral	30	$2^{127.3}$	2^{63}	$2^{109.02}$	[LRC19]

E Encoding S-boxes

The S-box Analyzer [HNE22] is an open-source tool developed to encode the differential, linear, differential-linear, and integral properties of S-boxes using MILP, SMT/SAT, and CP models efficiently. This tool has been used in several works [HNE22, HE22, HGSE24, HDE24], and is publicly accessible at:

<https://github.com/hadipourh/sboxanalyzer>

In our paper, the CP constraints for bit-wise deterministic differential/linear propagation through S-boxes can be automatically derived with an extended version of the S-box Analyzer tool. The corresponding functions in S-box Analyzer for this task are as follows. “-1” means the difference/linear mask can be either 0 or 1.

```

1 sage: from sboxanalyzer import *
2 sage: from sage.crypto.sboxes import CRAFT as sb
3 sage: sa = SboxAnalyzer(sb)
4 sage: ddp = sa.encode_deterministic_differential_behavior(); ddp
5 {(0, 0, 0, 0): [0, 0, 0, 0],
6  (0, 0, 1, 0): [-1, -1, 0, -1],
7  (0, 0, -1, 0): [-1, -1, 0, -1],
8  (1, 1, 0, 1): [-1, -1, 1, -1],
9  (1, 1, 1, 1): [-1, -1, 1, -1],
10 (1, 1, -1, 1): [-1, -1, 1, -1]}
11 sage: cddp = sa.generate_cp_constraints(ddp); print(cddp)
12 Input: a0||a1||a2||a3; a0: msb
13 Output: b0||b1||b2||b3; b0: msb
14 if (a0 == 0 /\ a1 == 0 /\ a2 == 0 /\ a3 == 0) then (b0 = 0 /\ b1 = 0 /\ b2 = 0 /\ b3 = 0)
15 elseif (a0 == 0 /\ a1 == 0 /\ a2 == 1 /\ a3 == 0) then (b0 = -1 /\ b1 = -1 /\ b2 = 0 /\ b3 = -1)
16 elseif (a0 == 0 /\ a1 == 0 /\ a2 == -1 /\ a3 == 0) then (b0 = -1 /\ b1 = -1 /\ b2 = 0 /\ b3 = -1)
17 elseif (a0 == 1 /\ a1 == 1 /\ a2 == 0 /\ a3 == 1) then (b0 = -1 /\ b1 = -1 /\ b2 = 1 /\ b3 = -1)
18 elseif (a0 == 1 /\ a1 == 1 /\ a2 == 1 /\ a3 == 1) then (b0 = -1 /\ b1 = -1 /\ b2 = 1 /\ b3 = -1)
19 elseif (a0 == 1 /\ a1 == 1 /\ a2 == -1 /\ a3 == 1) then (b0 = -1 /\ b1 = -1 /\ b2 = 1 /\ b3 = -1)
20 else (b0 = -1 /\ b1 = -1 /\ b2 = -1 /\ b3 = -1)
21 endif
22 sage: dlp = sa.encode_deterministic_linear_behavior()
23 {(0, 0, 0, 0): [0, 0, 0, 0],
24  (0, 0, 1, 0): [-1, -1, 0, -1],
25  (0, 0, -1, 0): [-1, -1, 0, -1],
26  (0, 1, 0, 1): [1, -1, 1, -1],
27  (0, 1, 1, 1): [-1, -1, 1, -1],
28  (0, 1, -1, 1): [-1, -1, 1, -1]}
29 sage: cdlp = sa.generate_cp_constraints(dlp); print(cdlp)
30 Input: a0||a1||a2||a3; a0: msb
31 Output: b0||b1||b2||b3; b0: msb
32 if (a0 == 0 /\ a1 == 0 /\ a2 == 0 /\ a3 == 0) then (b0 = 0 /\ b1 = 0 /\ b2 = 0 /\ b3 = 0)
33 elseif (a0 == 0 /\ a1 == 0 /\ a2 == 1 /\ a3 == 0) then (b0 = -1 /\ b1 = -1 /\ b2 = 0 /\ b3 = -1)
34 elseif (a0 == 0 /\ a1 == 0 /\ a2 == -1 /\ a3 == 0) then (b0 = -1 /\ b1 = -1 /\ b2 = 0 /\ b3 = -1)
35 elseif (a0 == 0 /\ a1 == 1 /\ a2 == 0 /\ a3 == 1) then (b0 = 1 /\ b1 = -1 /\ b2 = 1 /\ b3 = -1)
36 elseif (a0 == 0 /\ a1 == 1 /\ a2 == 1 /\ a3 == 1) then (b0 = -1 /\ b1 = -1 /\ b2 = 1 /\ b3 = -1)

```

```

37 elseif (a0 == 0 /\ a1 == 1 /\ a2 == -1 /\ a3 == 1) then (b0 = -1 /\ b1 = -1 /\ b2 = 1 /\ b3 = -1)
38 else (b0 = -1 /\ b1 = -1 /\ b2 = -1 /\ b3 = -1)
39 endif

```

Listing 1: Encoding deterministic behaviour of S-boxes in Sbox Analyzer

F Figures Related to ID, ZC distinguisher, and Full ID Attack on AndRX

List of Figures

1	Overview and parameters of impossible differential attacks.	7
2	Representing the modular addition $X \boxplus Y$ using full-adders f and a half-adder g	10
3	Model for impossible-differential distinguishers with indirect contradiction.	11
4	Round function structure of SIMON, where F is defined by $(y^0, y^1) = F(x^0, x^1) = (x^1 \oplus (x^0 \lll 8) \odot (x^0 \lll 1), x^0)$ and can be expressed in terms of the 3-bit function S_{SIMON}	14
5	Original Subkey vs Equivalent Subkey	24
6	Cluster of 2^{65} ID distinguishers for 6-round SPECK-96.	28
7	13-round ID distinguisher for attack on 23-round SIMON64-128.	33
8	Key recovery of the attack on 23-round SIMON64-128.	34
9	15-round (indirect) ZC distinguisher for Simeck48. In this case, the bit difference in the upper triangle of $L_2[0]$ (in the left-hand column) is 1, whereas the bit difference in the lower triangle of $L'_2[0]$ is 0. This leads to a contradiction occurring in the second round.	44
10	Round functions of various ARX designs.	48
11	Round function of SIMON-48	49
12	Round function of Simeck-64	50
13	6-round ID distinguisher for Speck-32 and Speck-48.	55
14	6-round ID distinguisher for Speck-64.	56
15	Cluster of 2^{97} ID distinguishers for 6-round SPECK-128.	57
16	11-round ID distinguisher for attack on 20-round Simeck32-64.	58
17	Key recovery of the attack on 20-round Simeck32-64	59
18	Key recovery of the attack on 25-round Simeck48-96	60
19	Key recovery of the attack on 27-round Simeck64-128	61
20	11-round ID distinguisher for attack on 19-round SIMON32-64.	62
21	Key recovery of the attack on 19-round SIMON32-64.	63
22	11-round ID distinguisher for attack on 20-round SIMON32-64.	64
23	Key recovery of the attack on 20-round SIMON32-64.	65
24	12-round ID distinguisher for attack on 20-round SIMON48-72.	66
25	Key recovery of the attack on 20-round SIMON48-72.	67
26	12-round ID distinguisher for attack on 21-round SIMON48-96.	68
27	Key recovery of the attack on 21-round SIMON48-96.	69
28	13-round ID distinguisher for attack on 21-round SIMON64-96.	70
29	Key recovery of the attack on 21-round SIMON64-96.	71
30	13-round ID distinguisher for attack on 22-round SIMON64-96.	72
31	Key recovery of the attack on 22-round SIMON64-96.	73
32	13-round ID distinguisher for attack on 22-round SIMON64-128.	74
33	Key recovery of the attack on 22-round SIMON64-128.	75
34	16-round ID distinguisher for attack on 24-round SIMON96-96.	76
35	Key recovery of the attack on 24-round SIMON96-96.	77

36	16-round ID distinguisher for attack on 25-round SIMON96-144.	78
37	Key recovery of the attack on 25-round SIMON96-144.	79
38	19-round ID distinguisher for attack on 27-round SIMON128-128.	80
39	Key recovery of the attack on 27-round SIMON128-128.	81
40	19-round ID distinguisher for attack on 28-round SIMON128-128.	82
41	Key recovery of the attack on 28-round SIMON128-128.	83
42	19-round ID distinguisher for attack on 29-round SIMON128-192.	84
43	Key recovery of the attack on 29-round SIMON128-192.	85
44	19-round ID distinguisher for attack on 30-round SIMON128-192.	86
45	Key recovery of the attack on 30-round SIMON128-192.	87
46	19-round ID distinguisher for attack on 30-round SIMON128-256.	88
47	Key recovery of the attack on 30-round SIMON128-256.	89
48	19-round ID distinguisher for attack on 31-round SIMON128-256.	90
49	Key recovery of the attack on 31-round SIMON128-256.	91
50	11-round ZC distinguisher for Simeck32.	92
51	17-round ZC distinguisher for Simeck64. In this case, the bit difference in the upper triangle of $L_2[0]$ (in the left-hand column) is 1, whereas the bit difference in the lower triangle of $L'_2[0]$ is 0. This leads to a contradiction occurring in the second round.	93

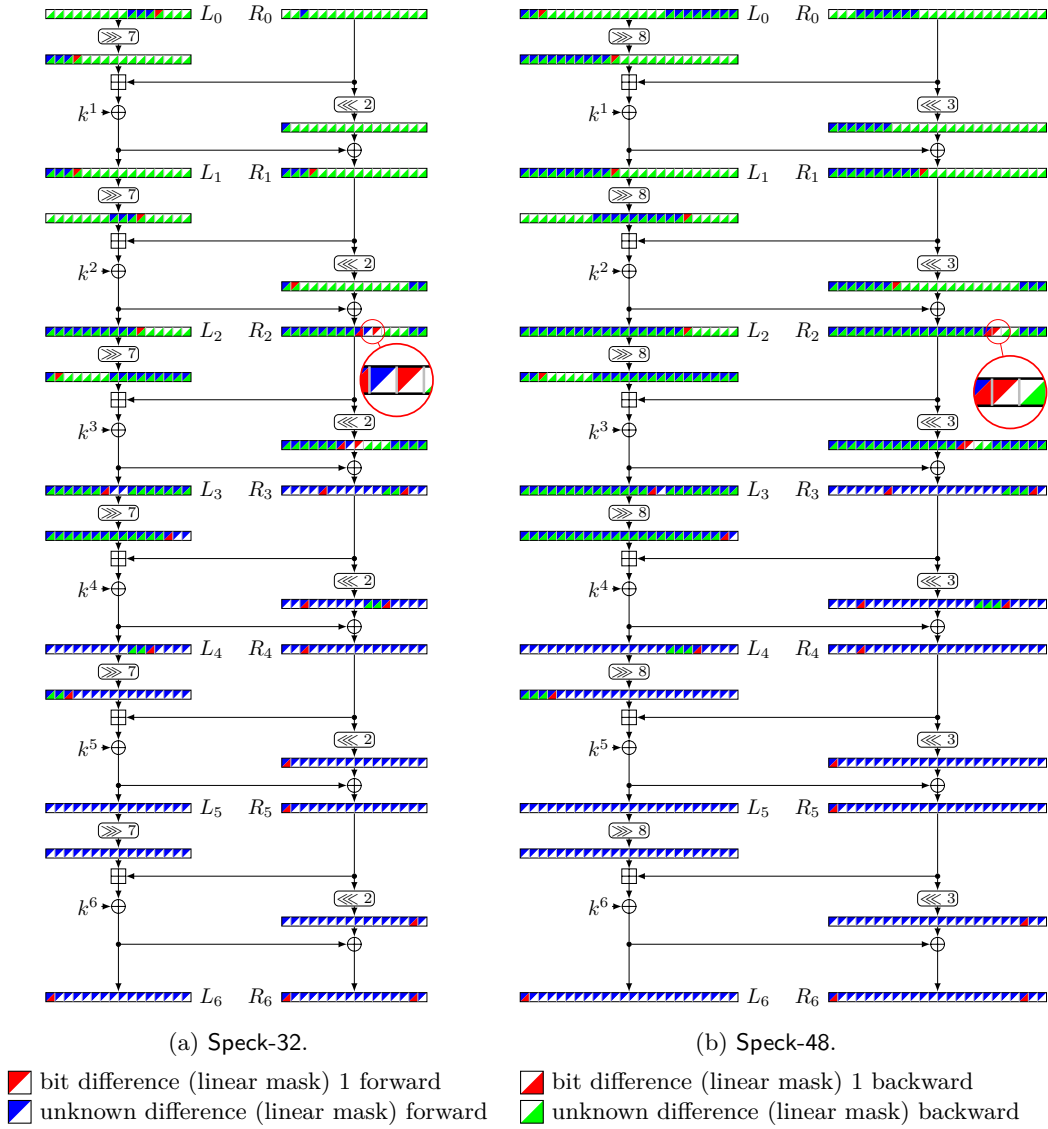


Figure 13: 6-round ID distinguisher for Speck-32 and Speck-48.

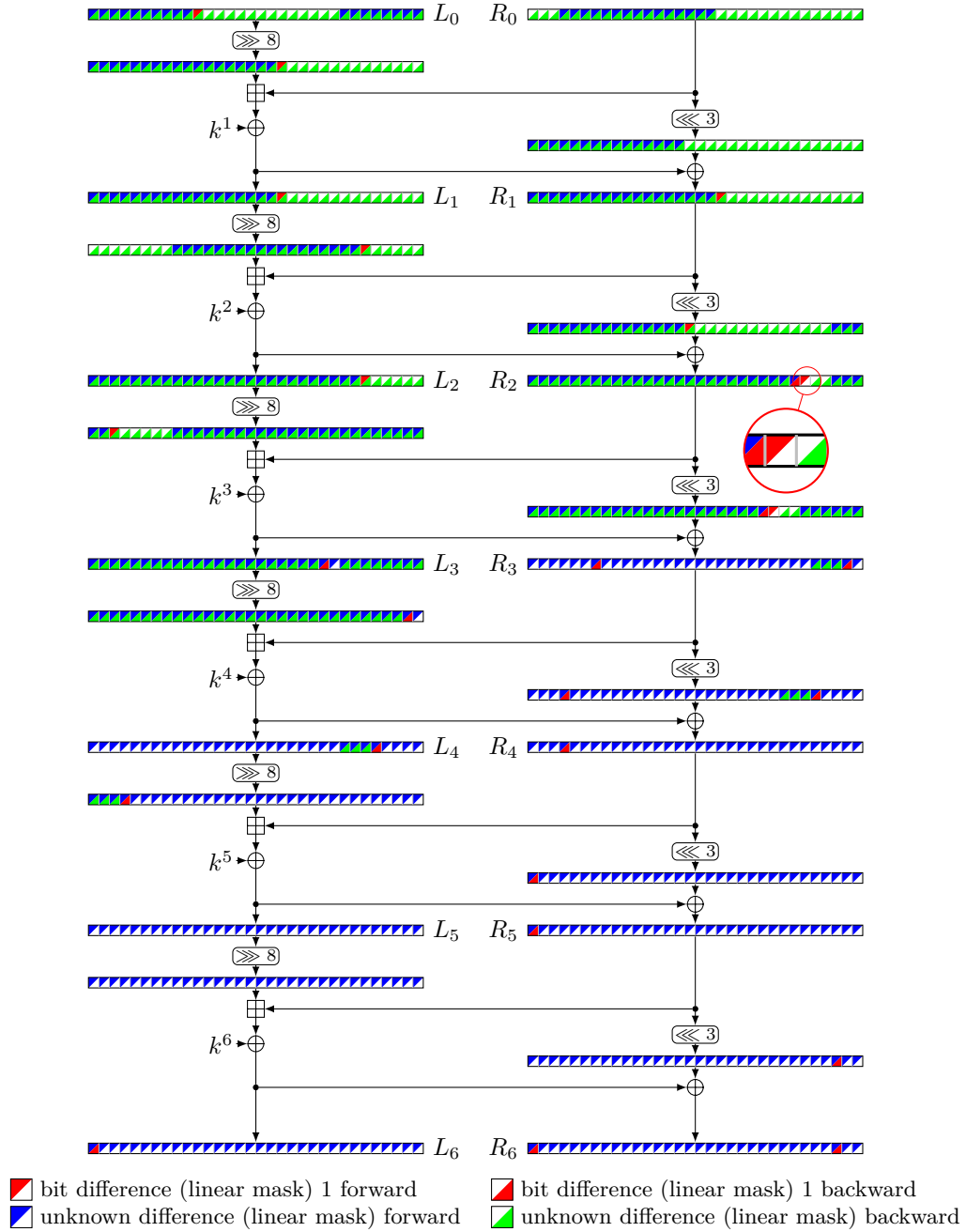


Figure 14: 6-round ID distinguisher for Speck-64.

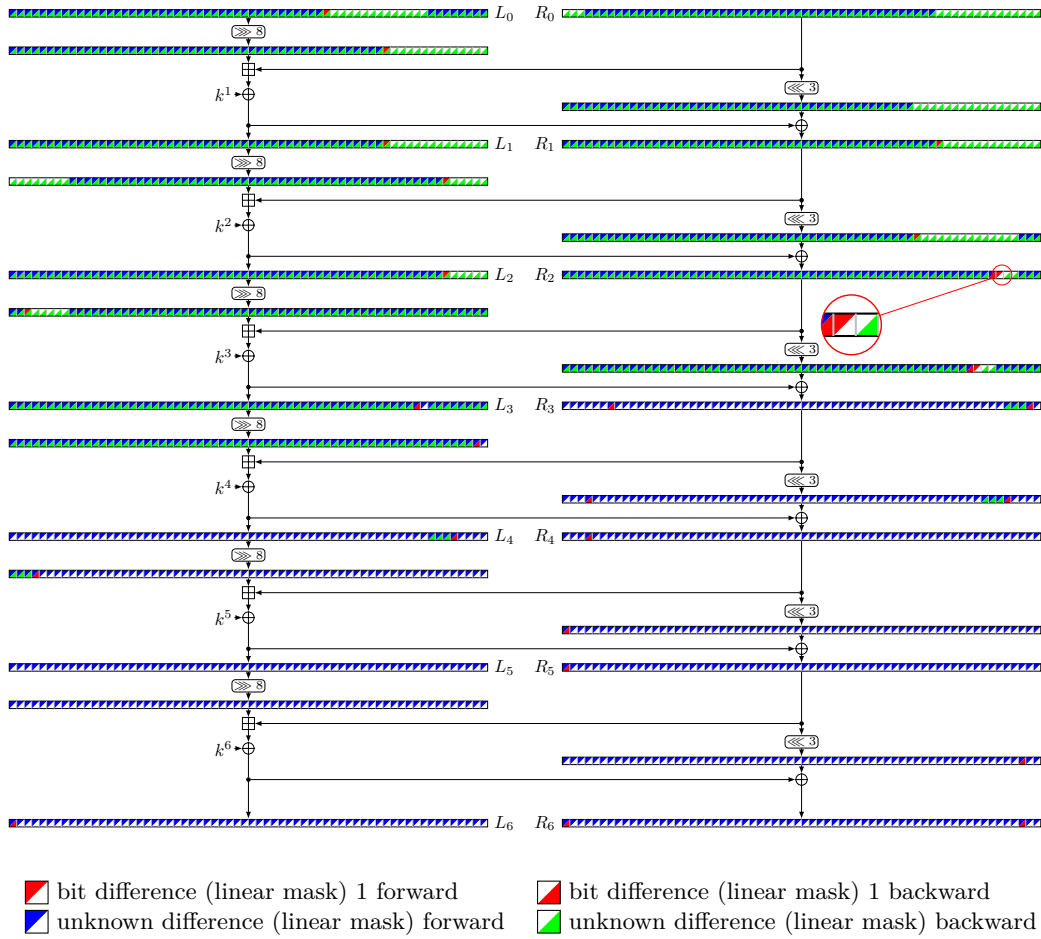


Figure 15: Cluster of 2^{97} ID distinguishers for 6-round SPECK-128.

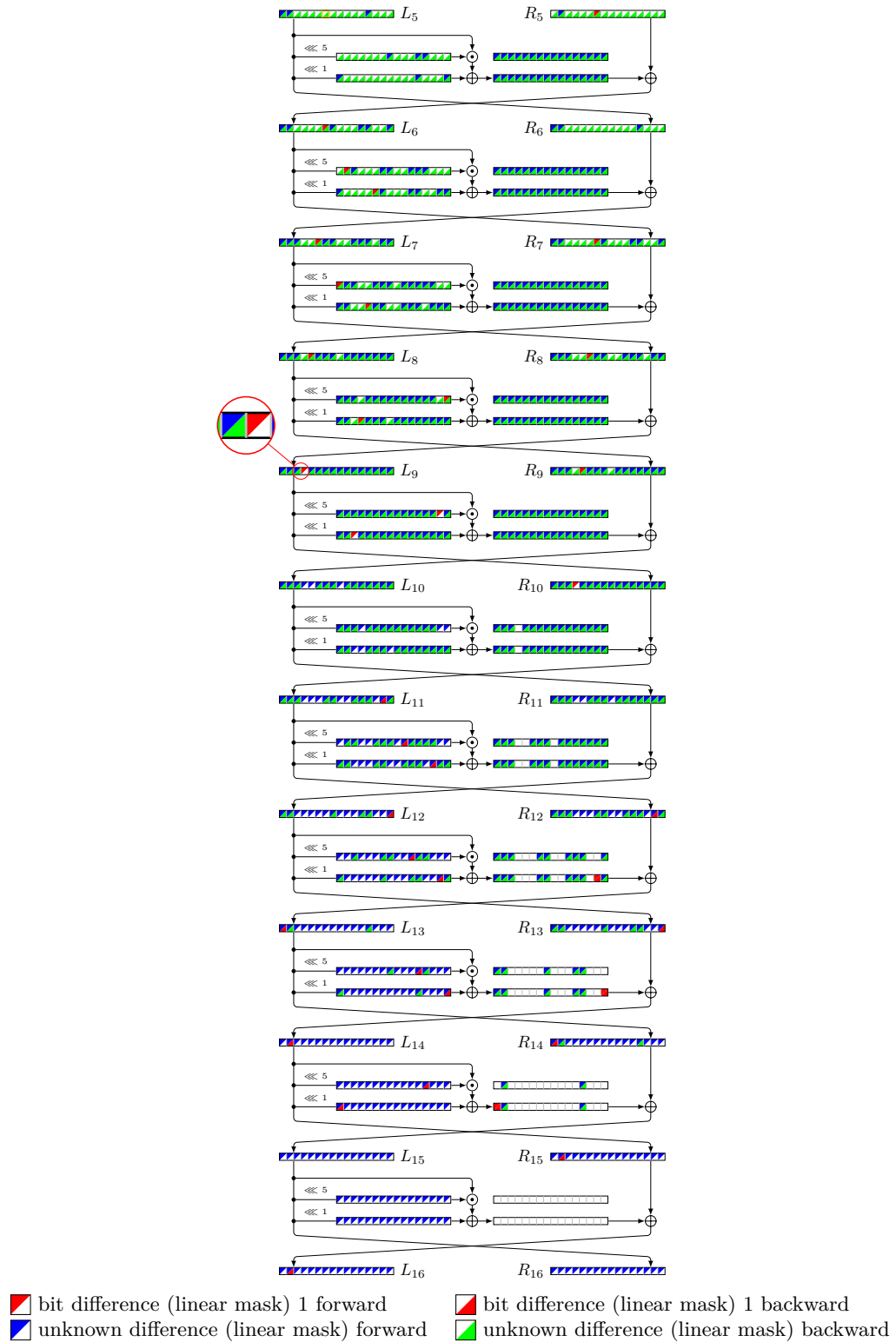


Figure 16: 11-round ID distinguisher for attack on 20-round Simeck32-64.

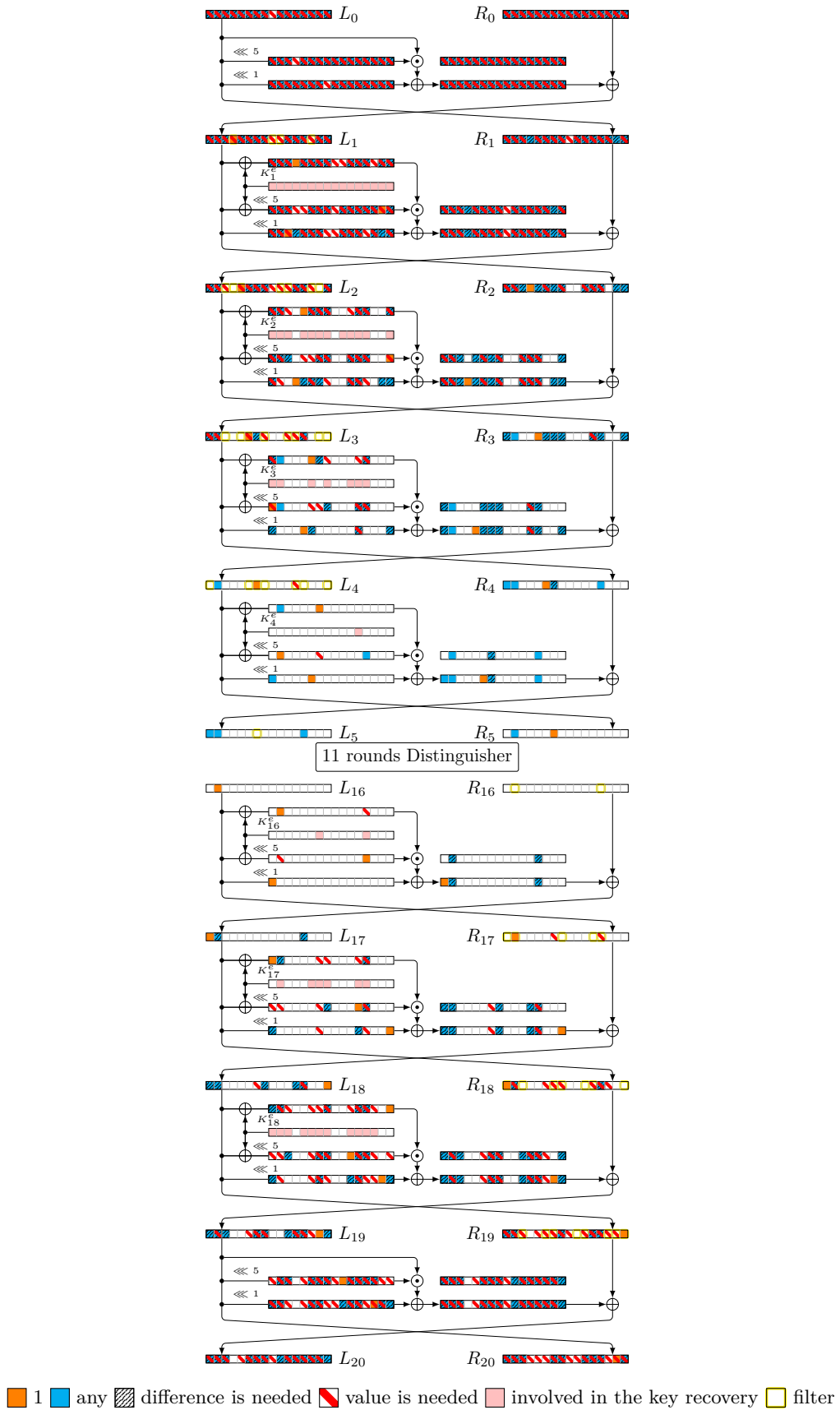


Figure 17: Key recovery of the attack on 20-round Simeck32-64

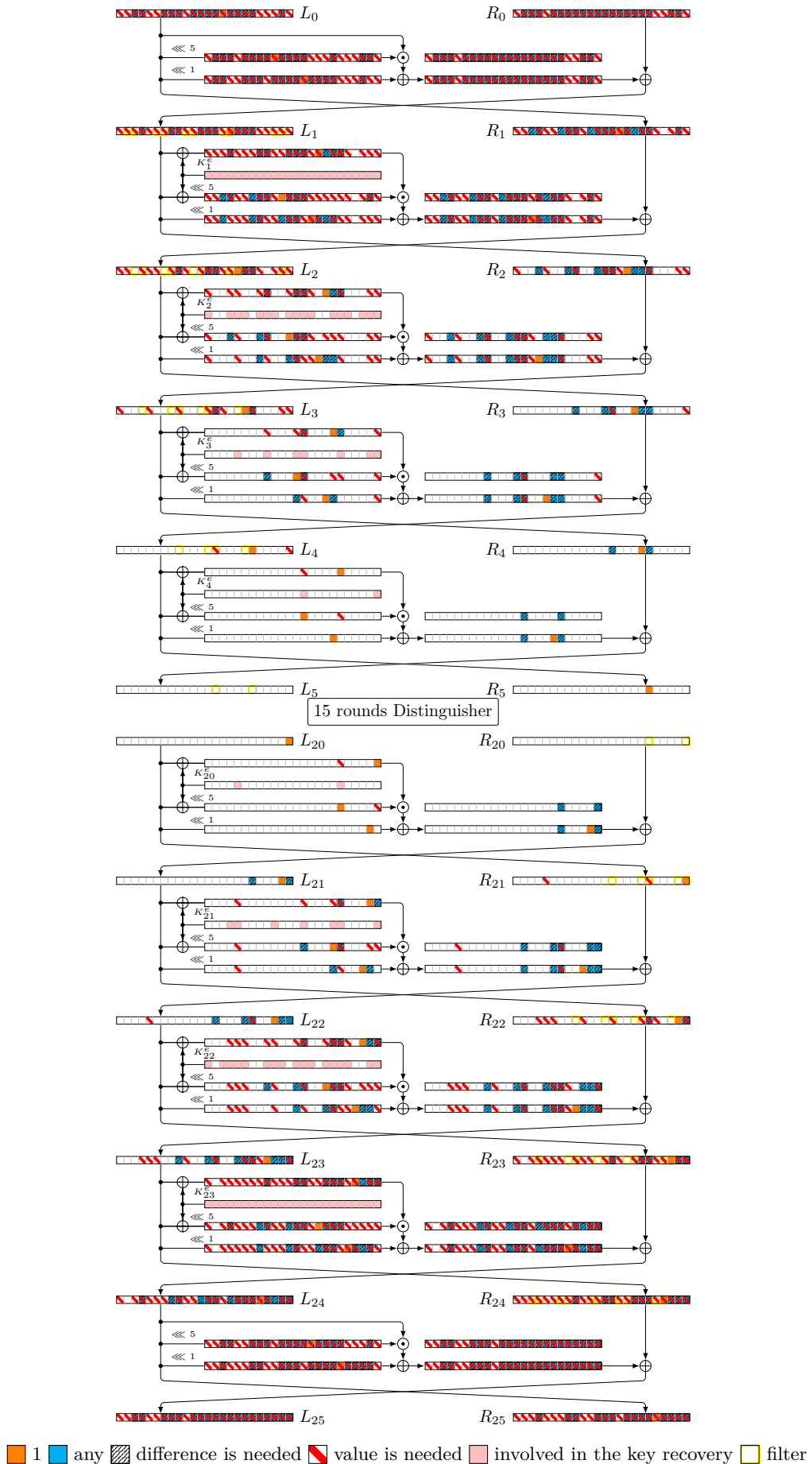


Figure 18: Key recovery of the attack on 25-round Simeck48-96

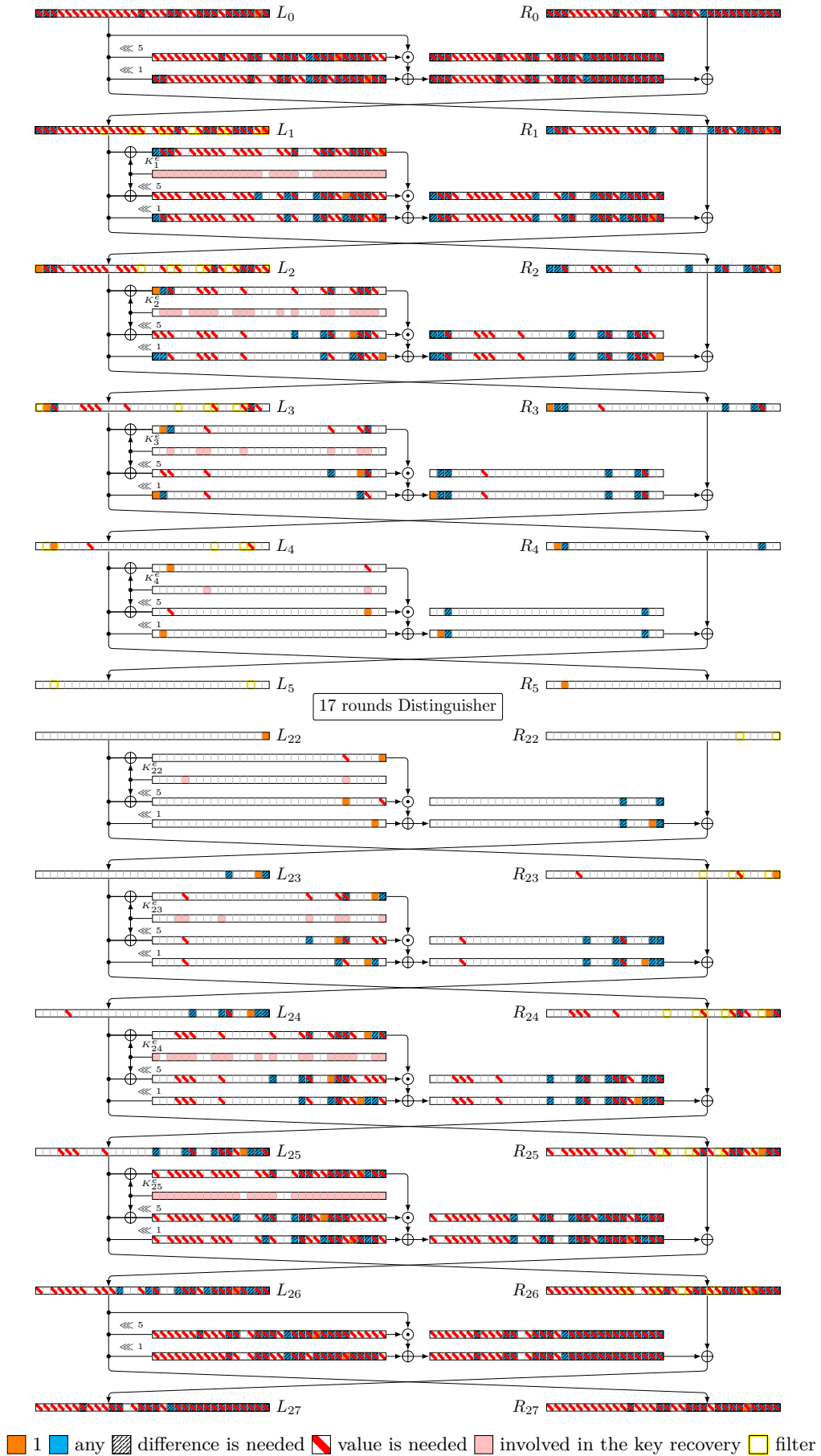


Figure 19: Key recovery of the attack on 27-round Simeck64-128

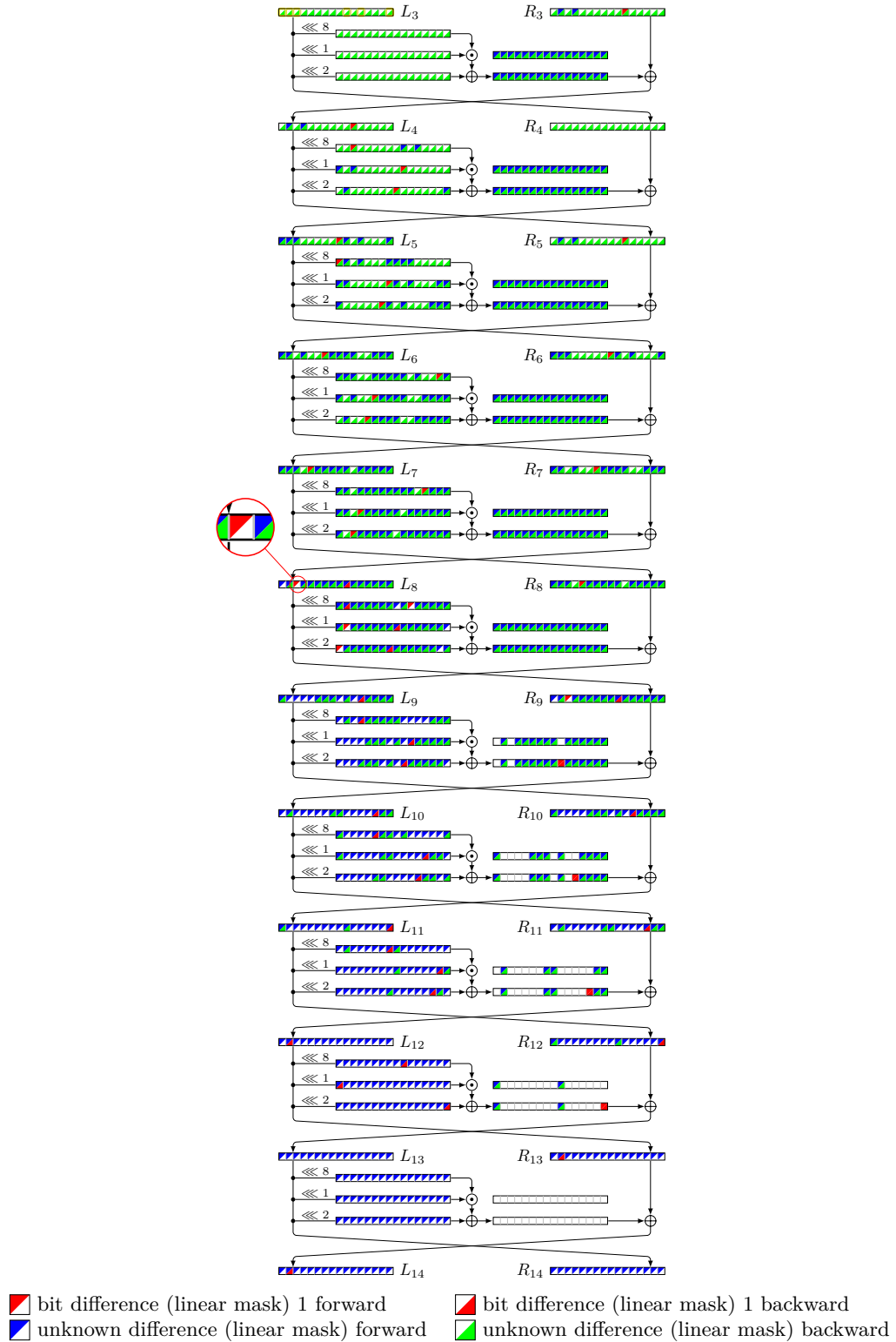


Figure 20: 11-round ID distinguisher for attack on 19-round SIMON32-64.

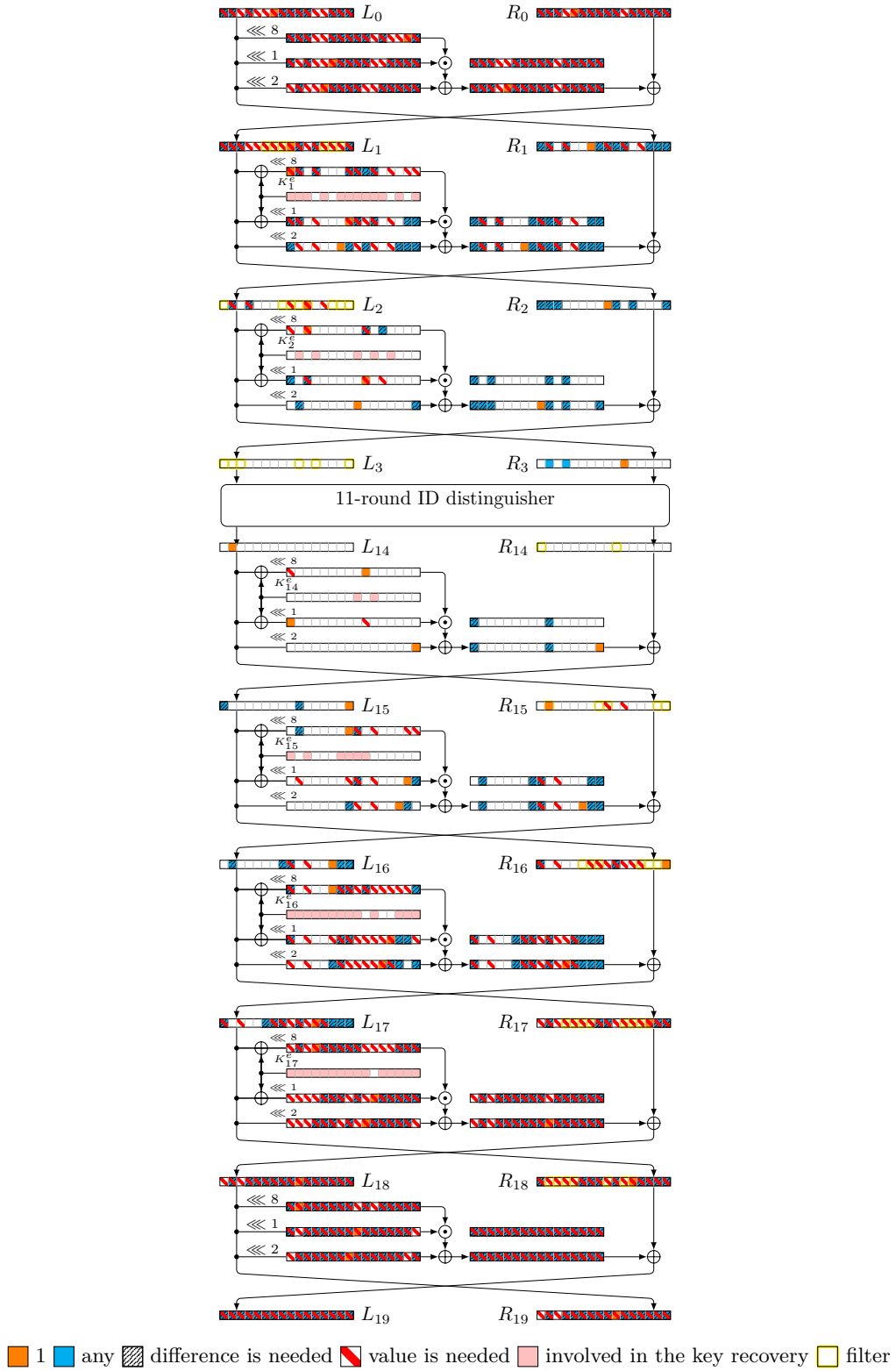


Figure 21: Key recovery of the attack on 19-round SIMON32-64.

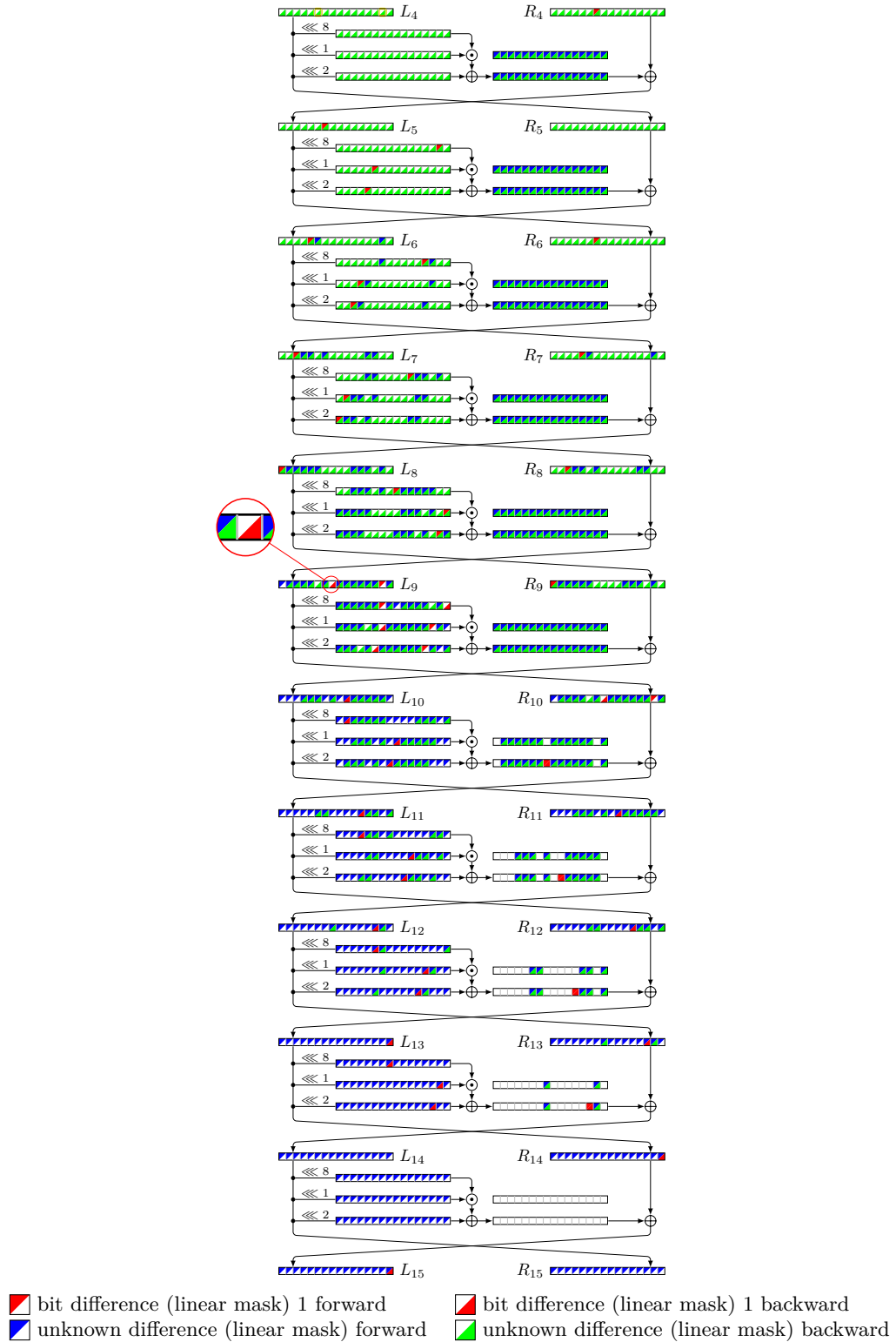


Figure 22: 11-round ID distinguisher for attack on 20-round SIMON32-64.

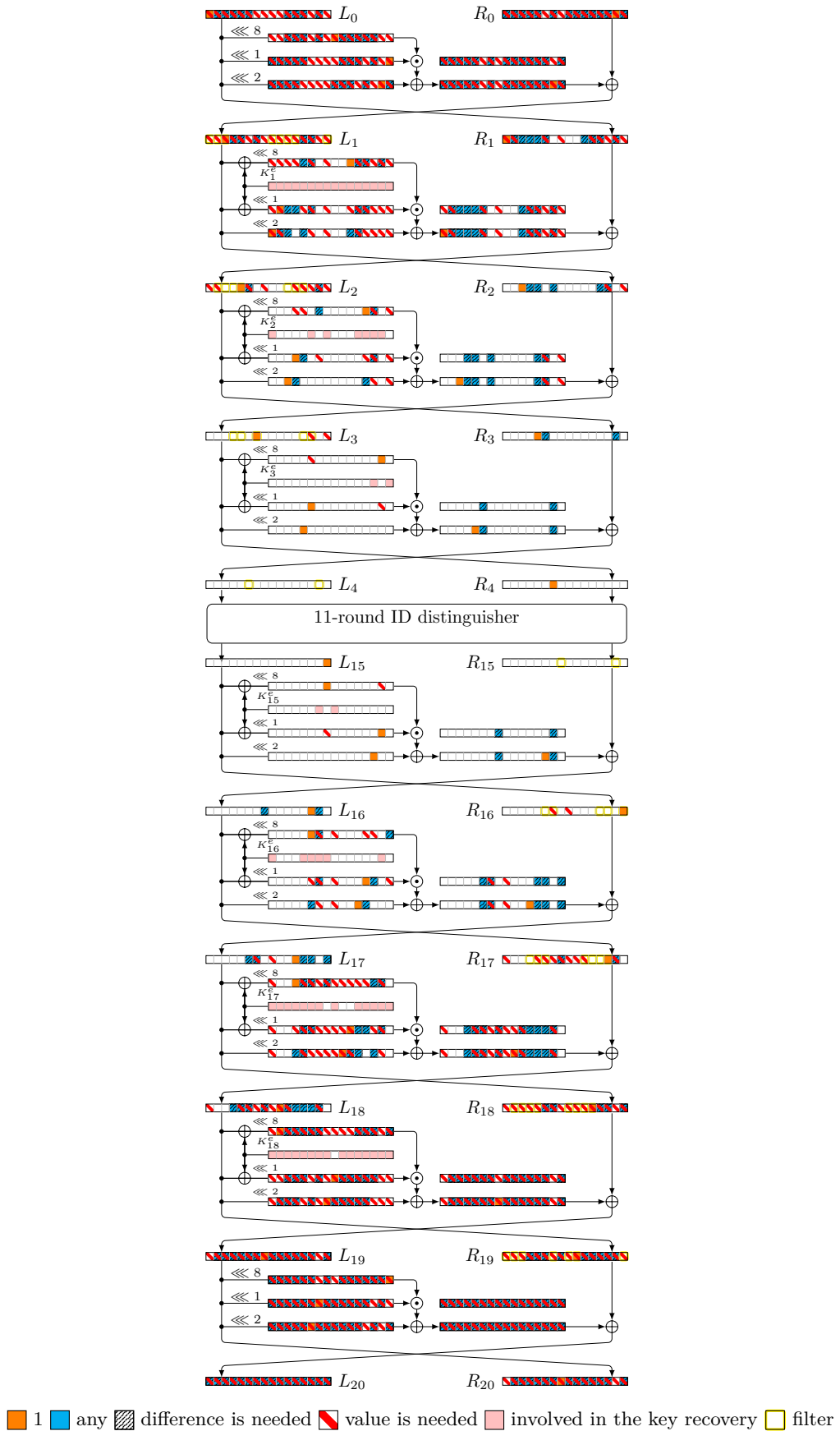


Figure 23: Key recovery of the attack on 20-round SIMON32-64.

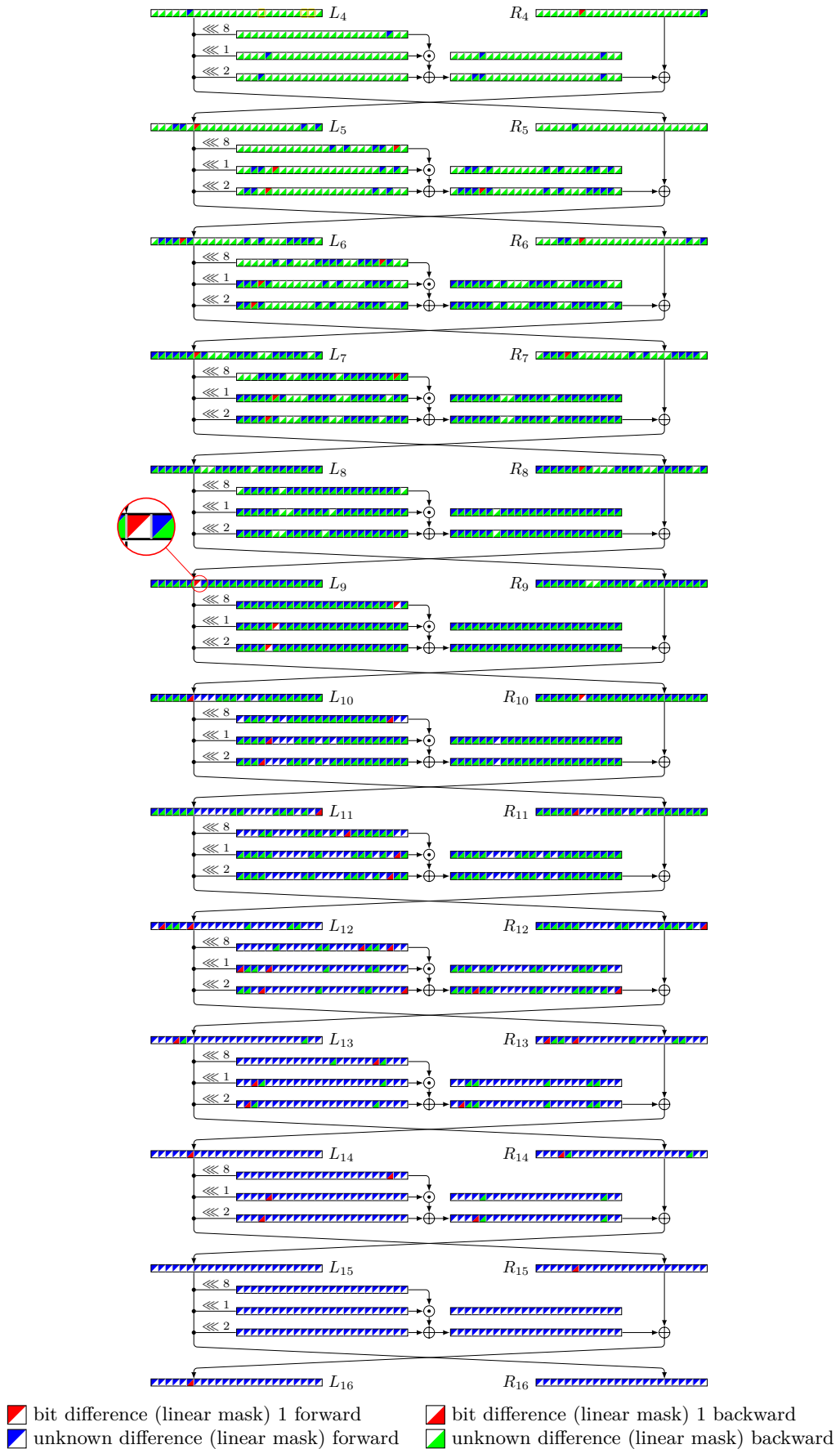
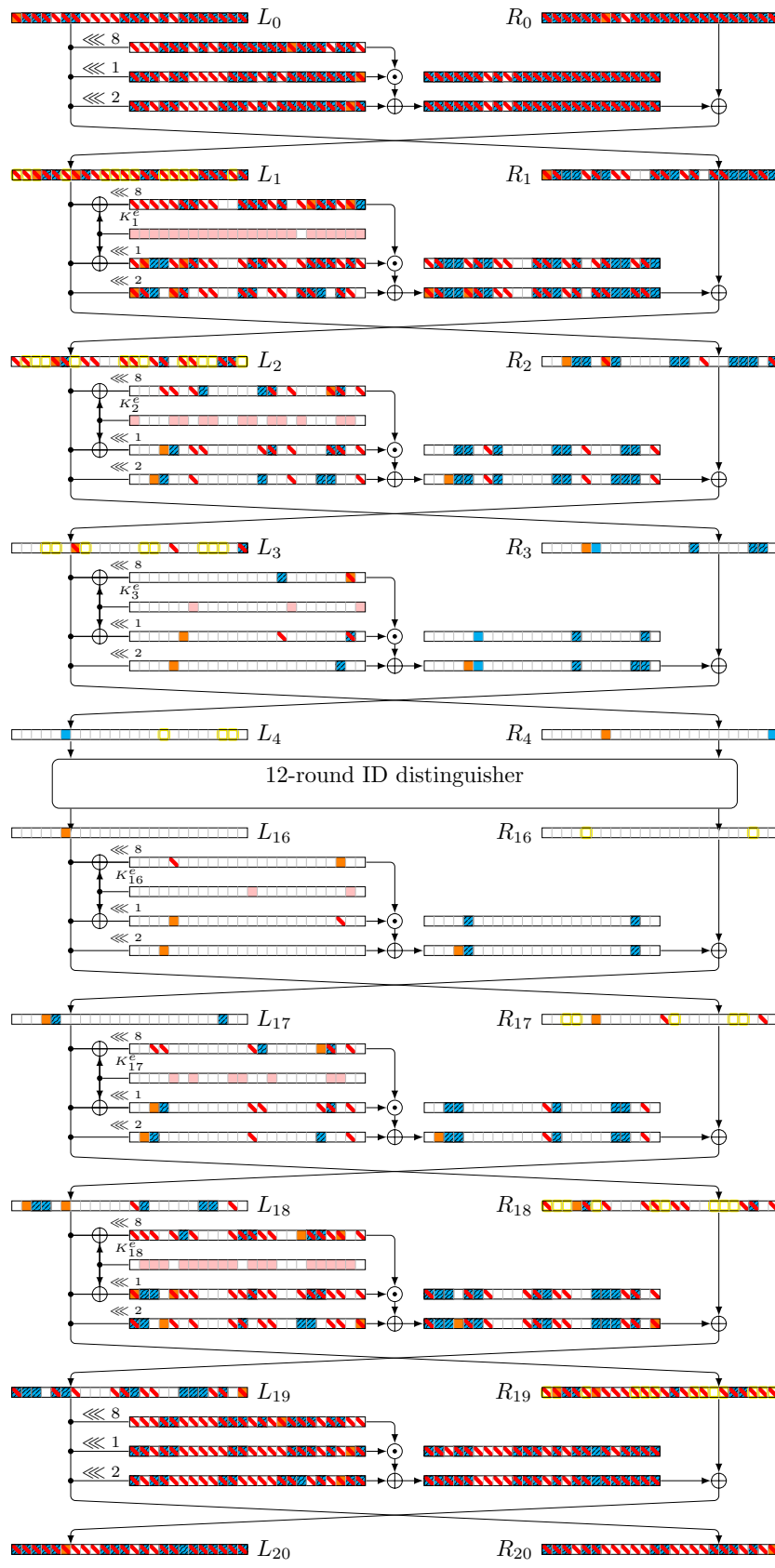


Figure 24: 12-round ID distinguisher for attack on 20-round SIMON48-72.



■ 1
 ■ any
 / difference is needed
 ■ value is needed
 involved in the key recovery
 filter

Figure 25: Key recovery of the attack on 20-round SIMON48-72.

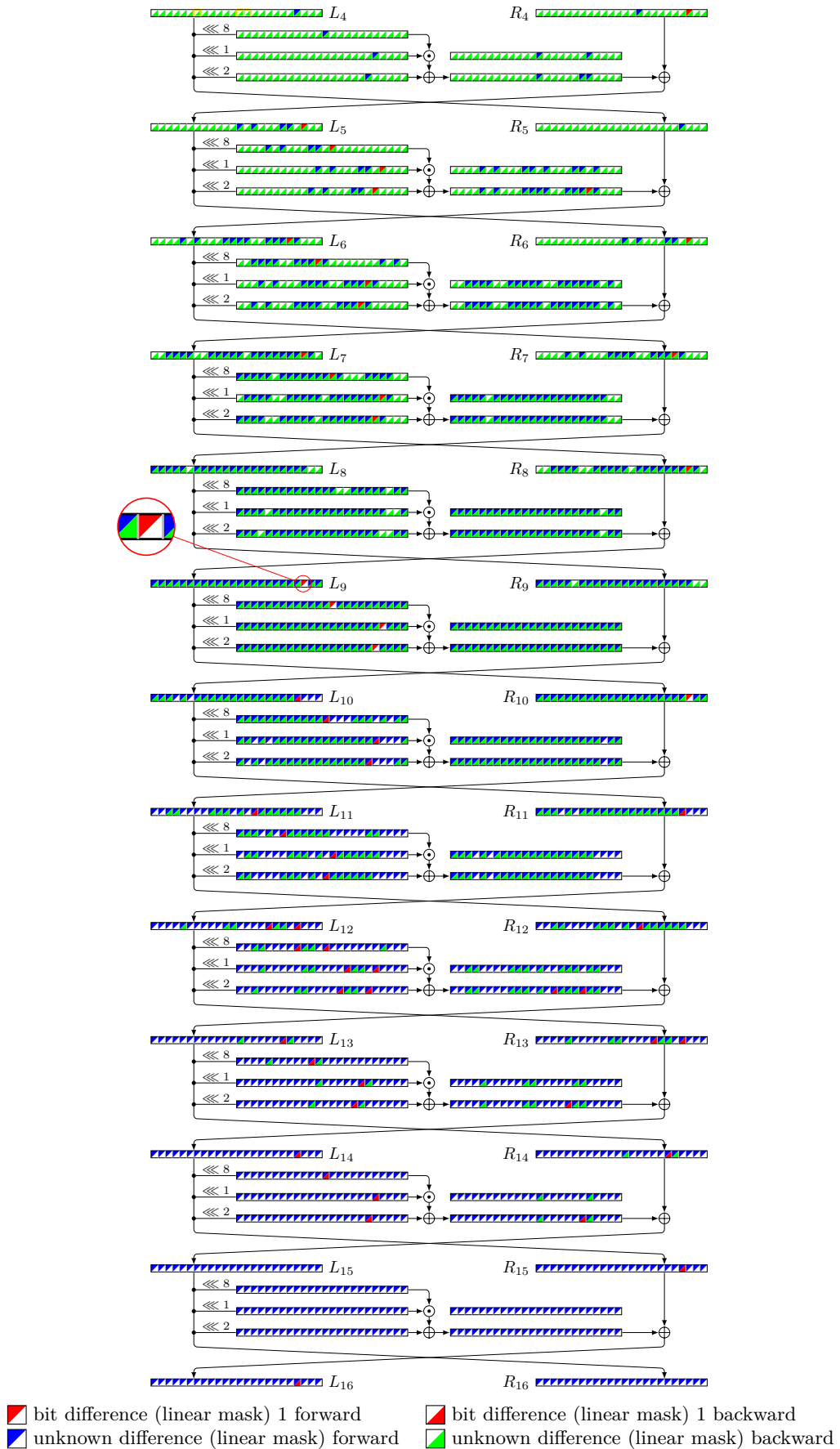


Figure 26: 12-round ID distinguisher for attack on 21-round SIMON48-96.

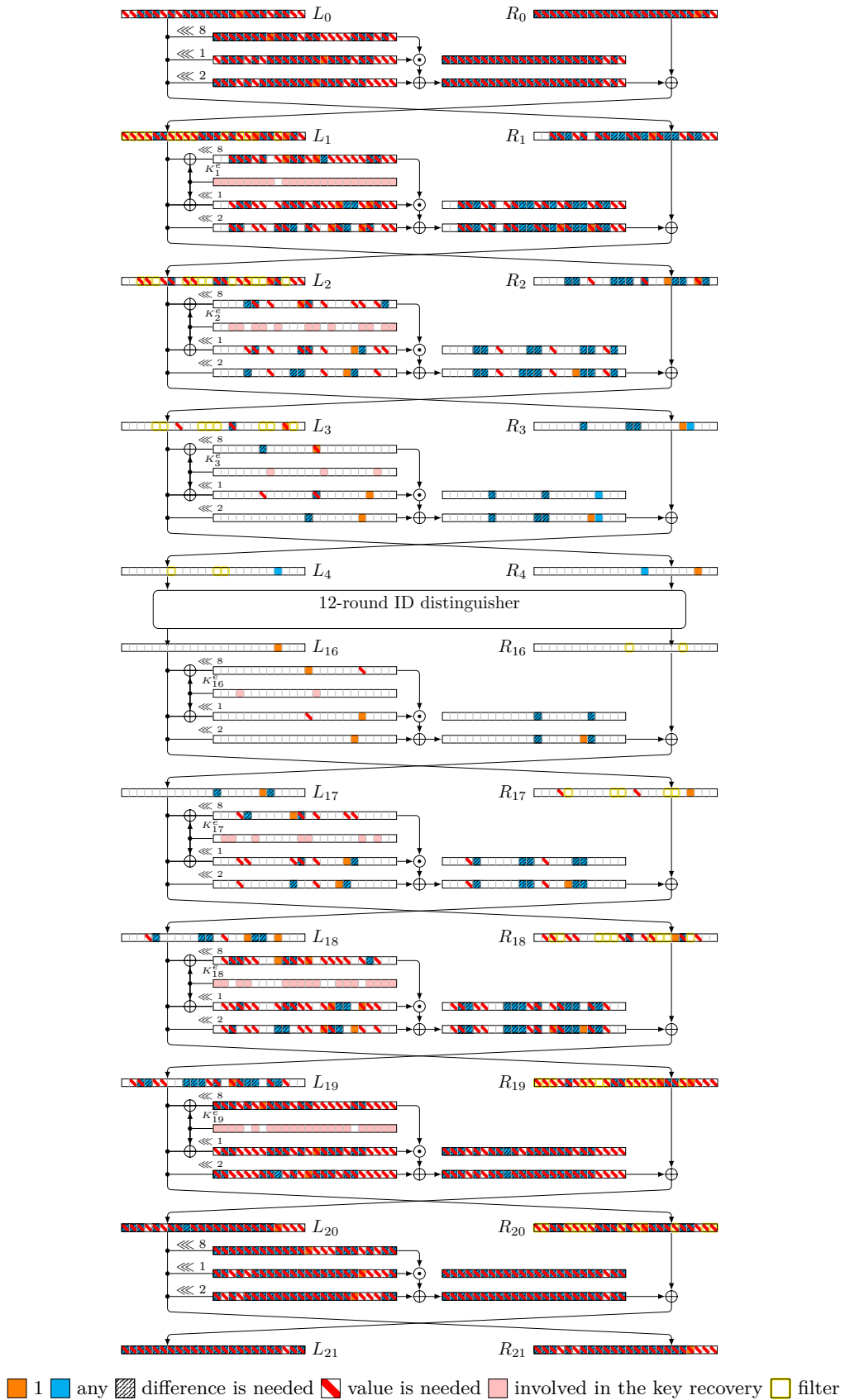


Figure 27: Key recovery of the attack on 21-round SIMON48-96.

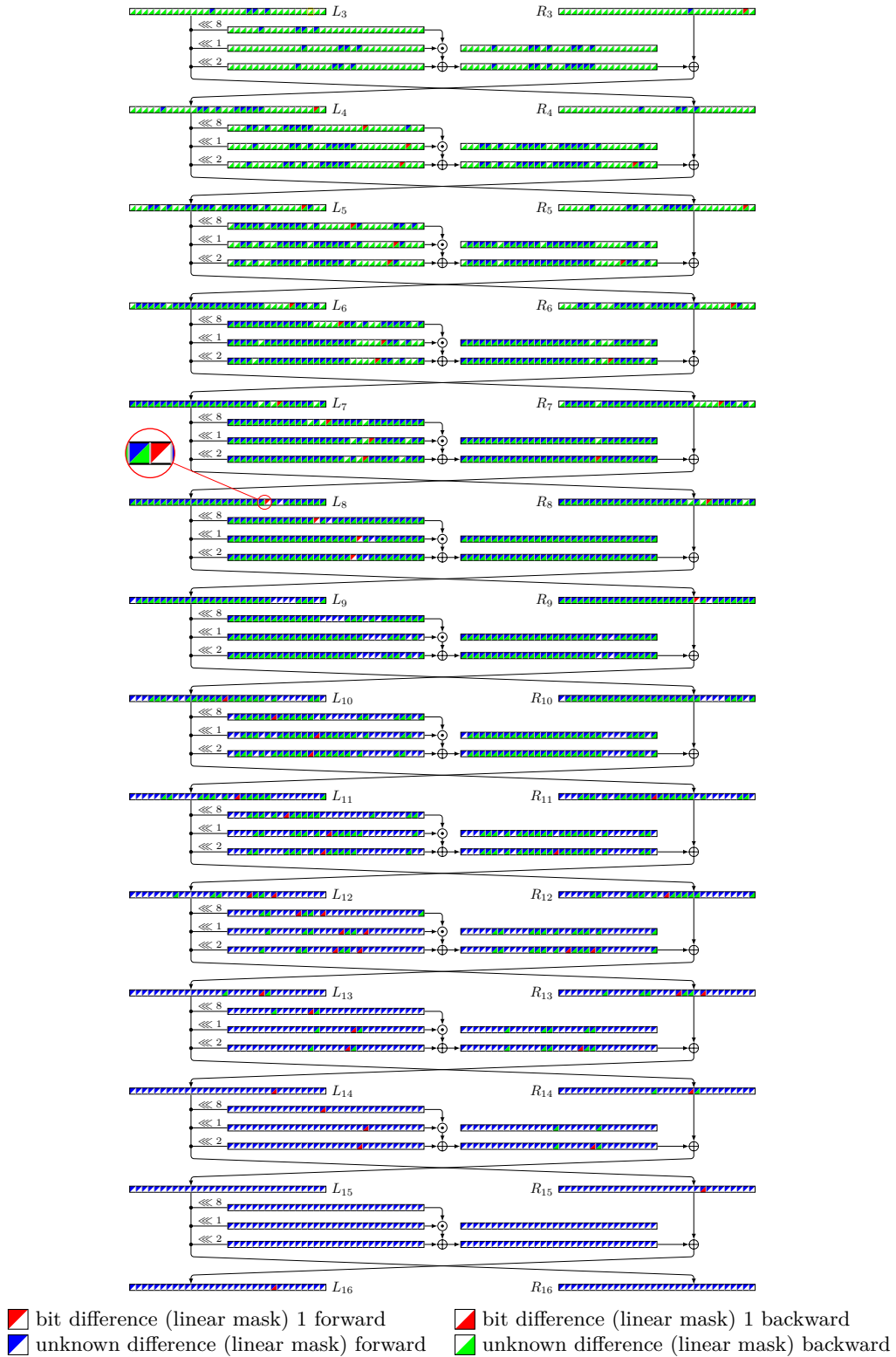


Figure 28: 13-round ID distinguisher for attack on 21-round SIMON64-96.

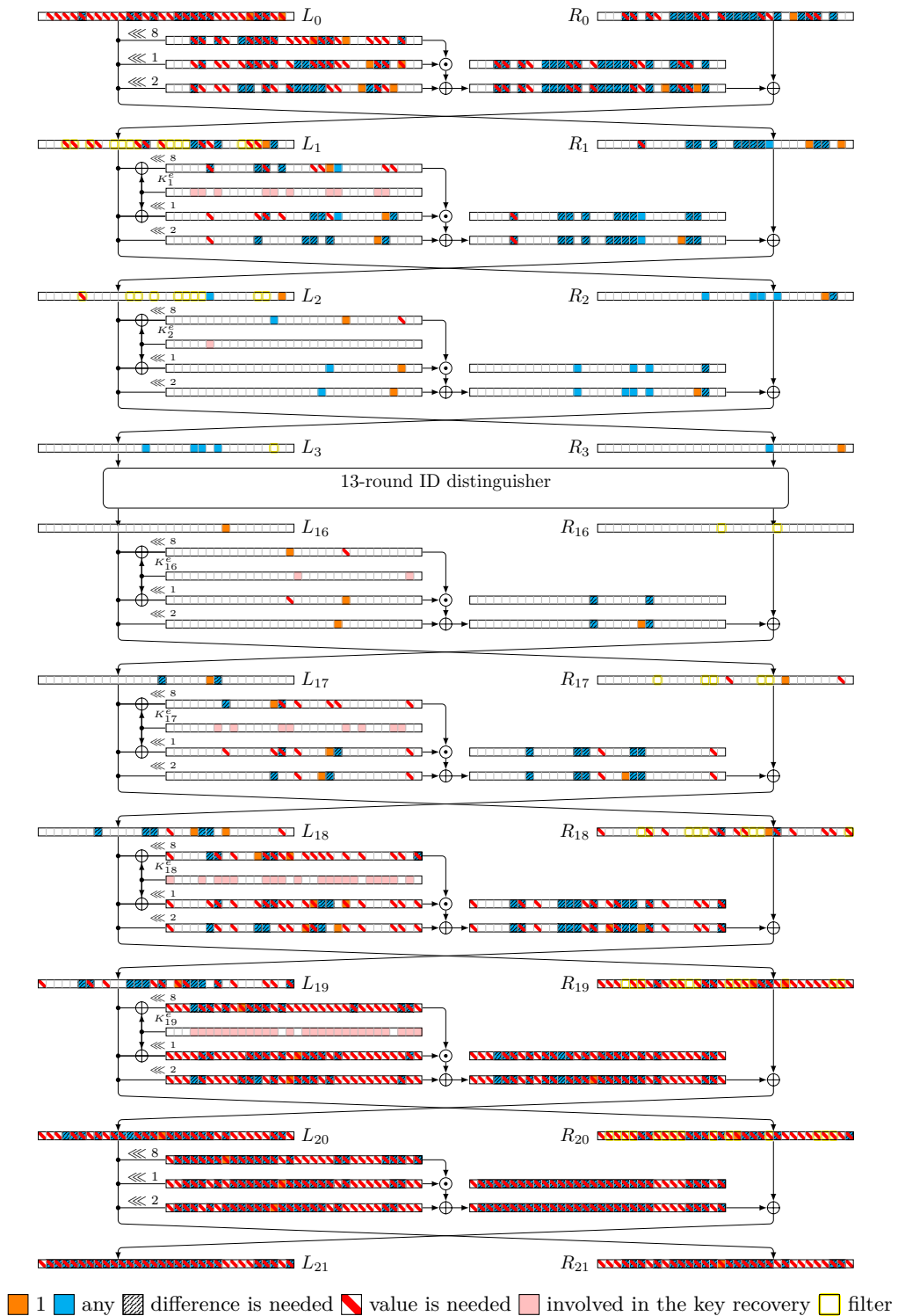


Figure 29: Key recovery of the attack on 21-round SIMON64-96.

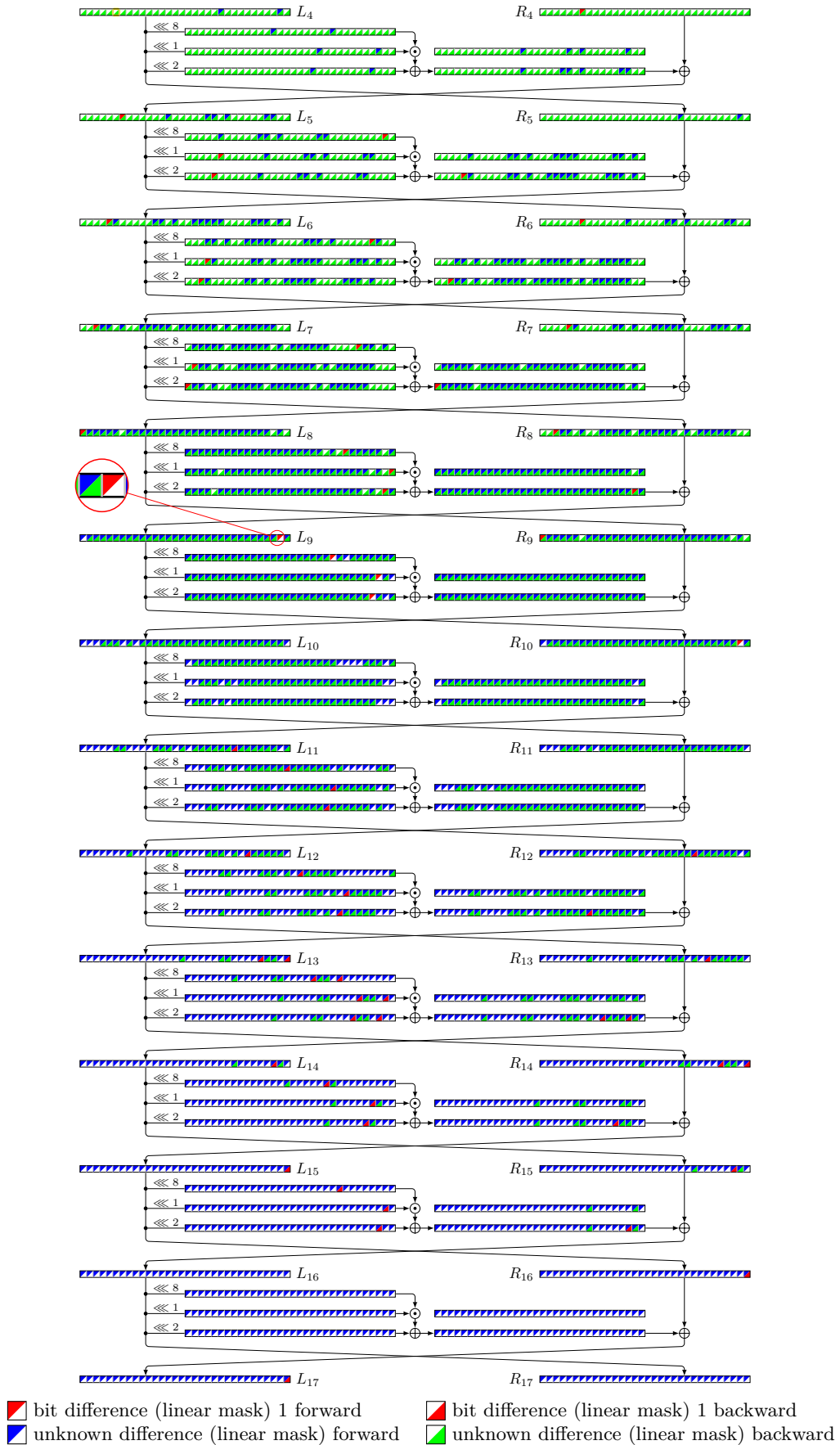


Figure 30: 13-round ID distinguisher for attack on 22-round SIMON64-96.

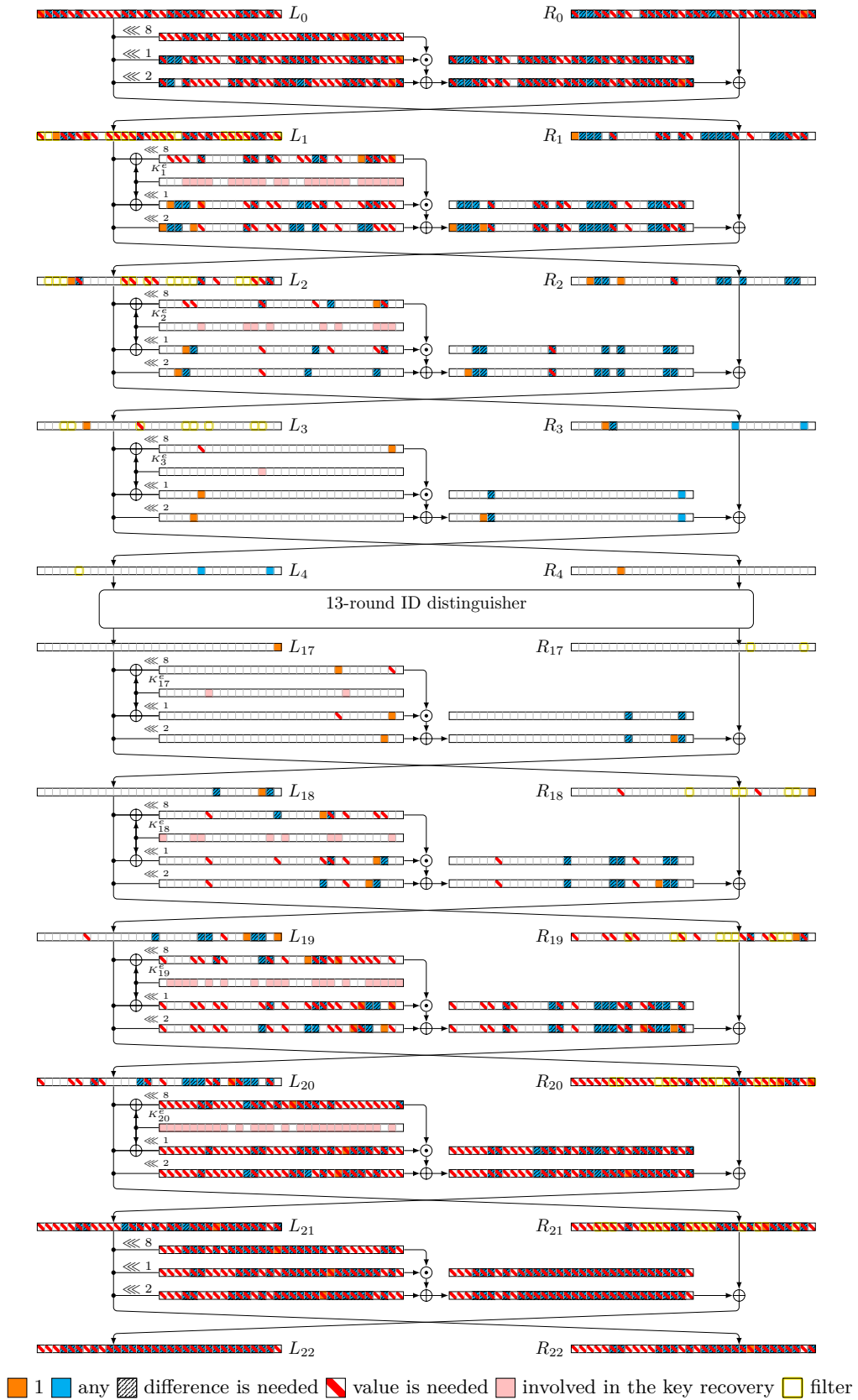


Figure 31: Key recovery of the attack on 22-round SIMON64-96.

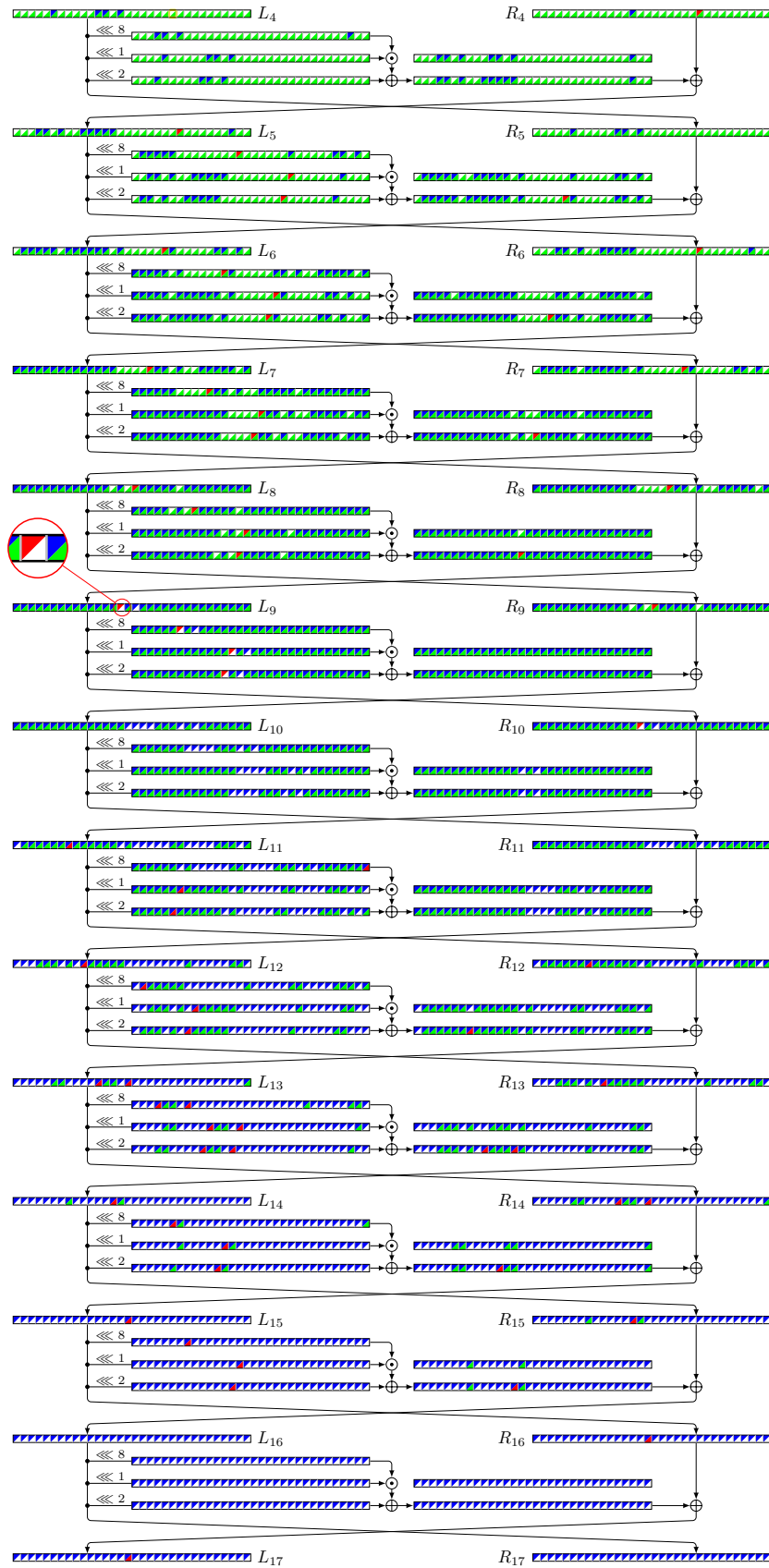


Figure 32: 13-round ID distinguisher for attack on 22-round SIMON64-128.

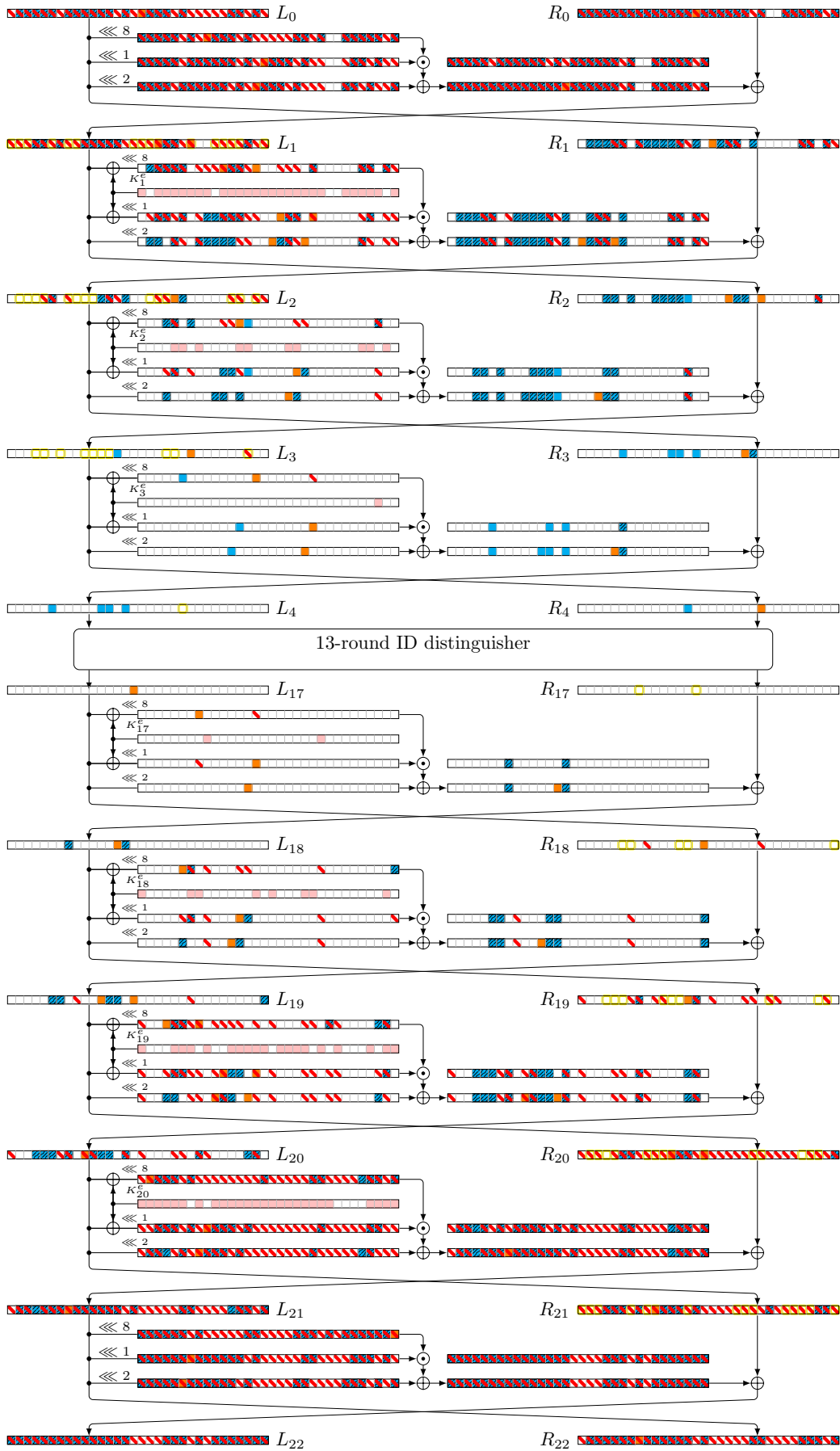


Figure 33: Key recovery of the attack on 22-round SIMON64-128.

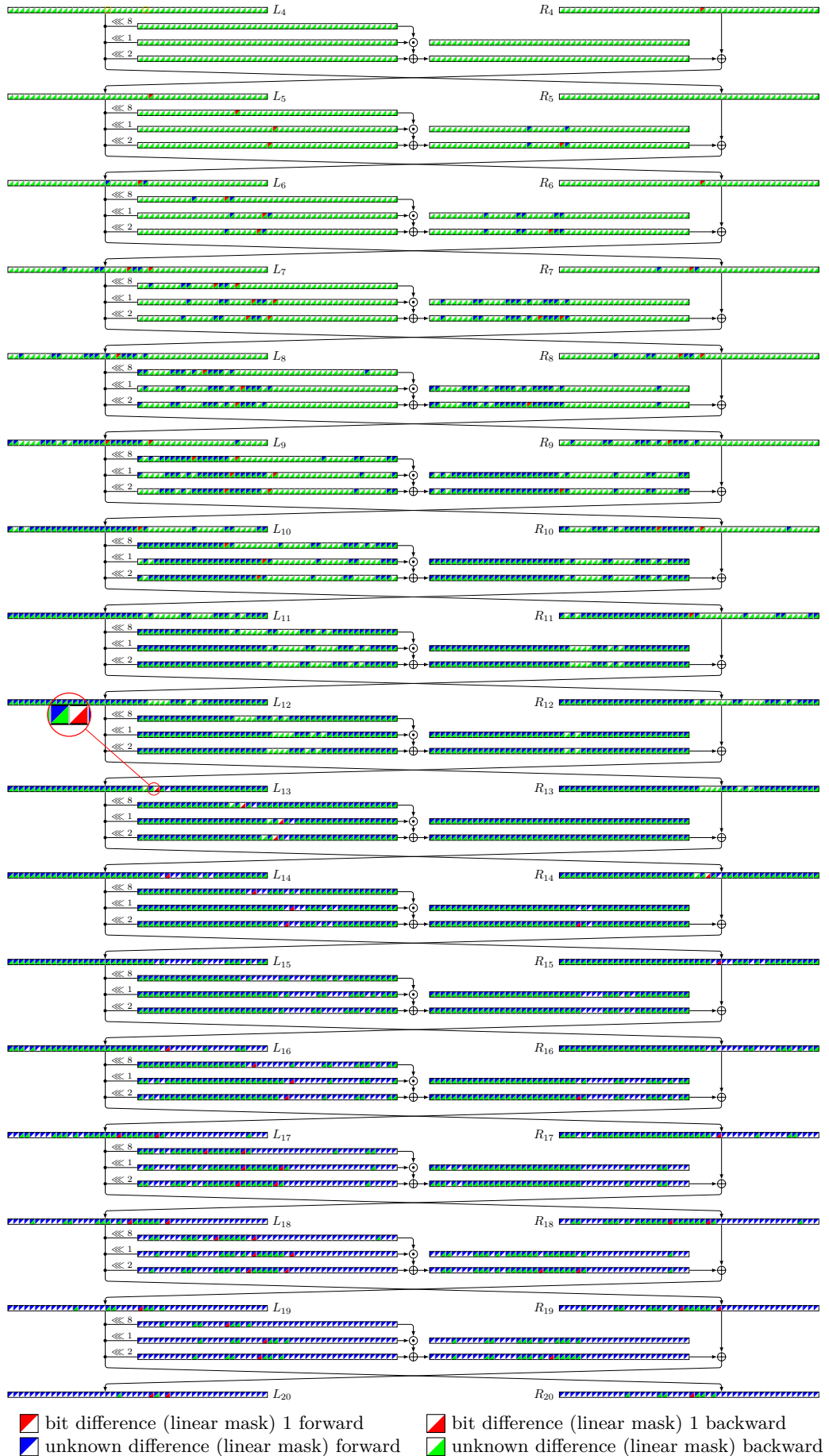


Figure 34: 16-round ID distinguisher for attack on 24-round SIMON96-96.

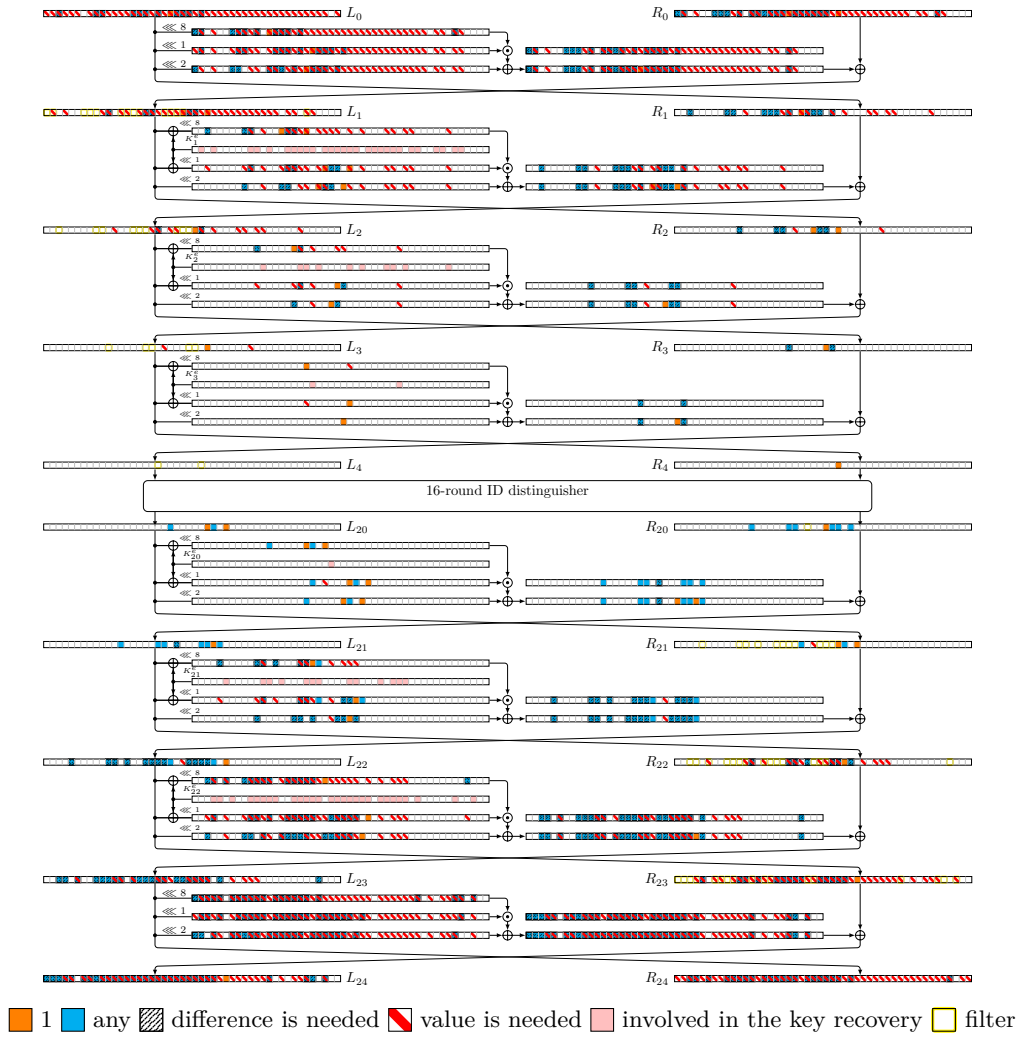


Figure 35: Key recovery of the attack on 24-round SIMON96-96.

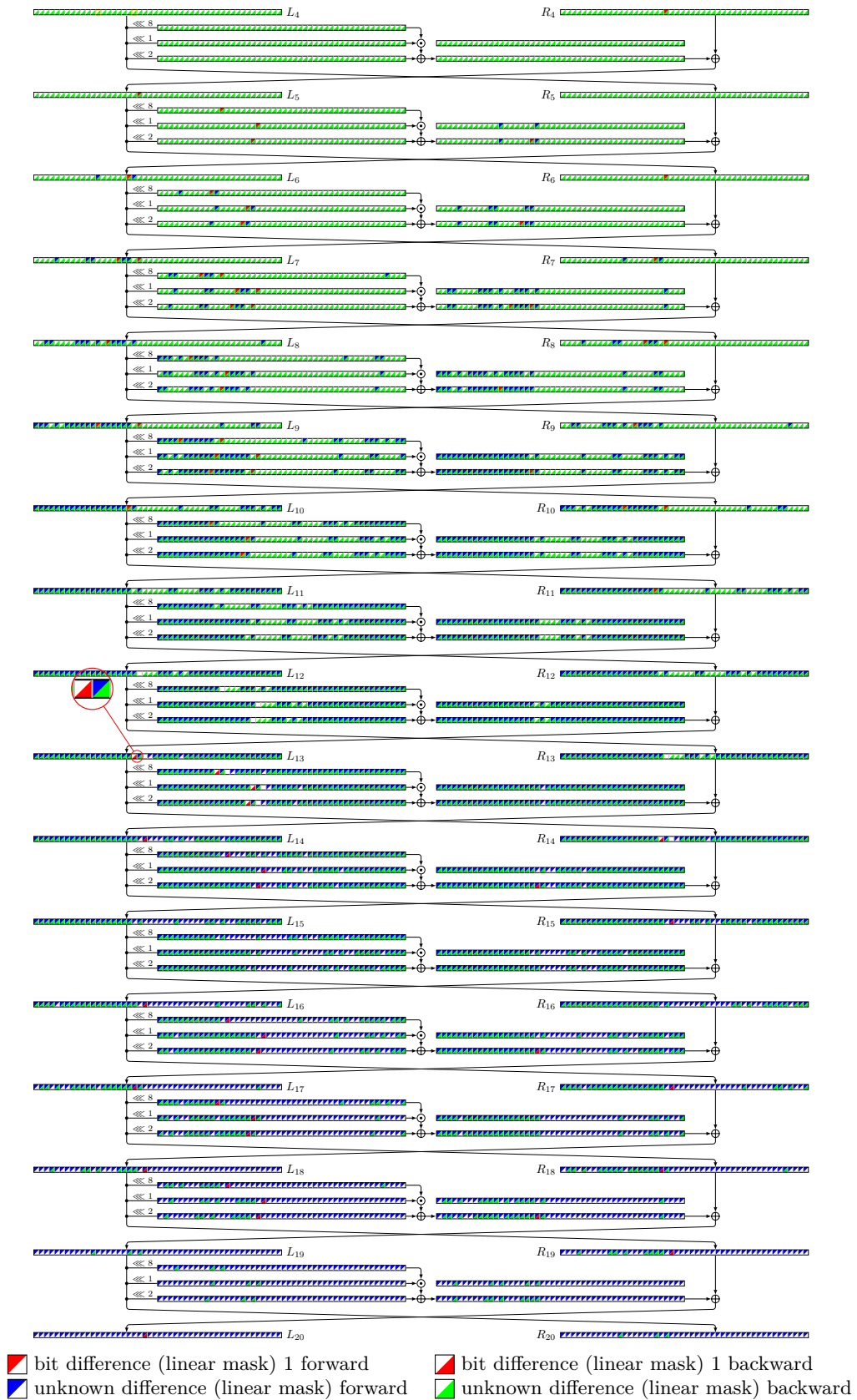


Figure 36: 16-round ID distinguisher for attack on 25-round SIMON96-144.

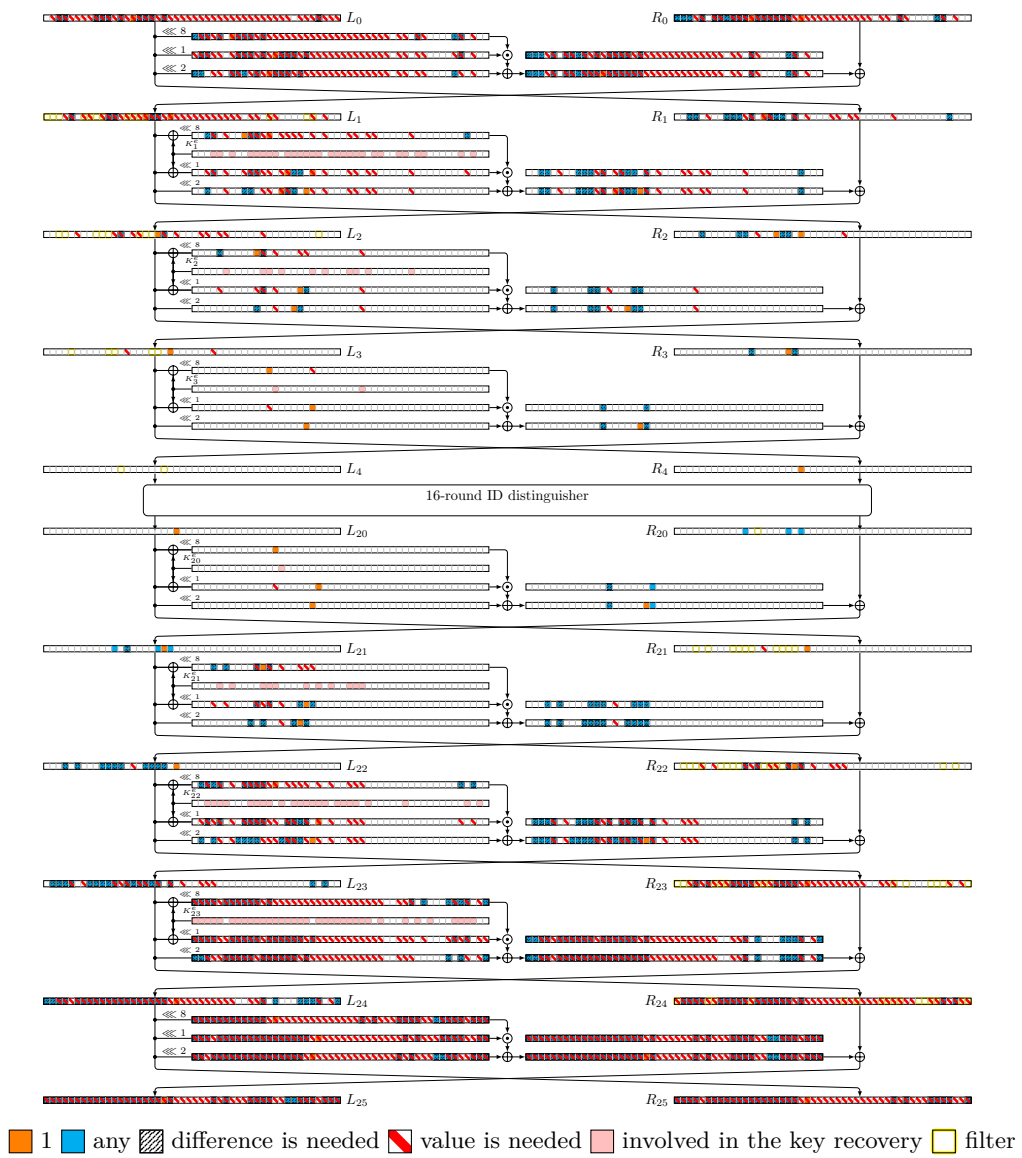
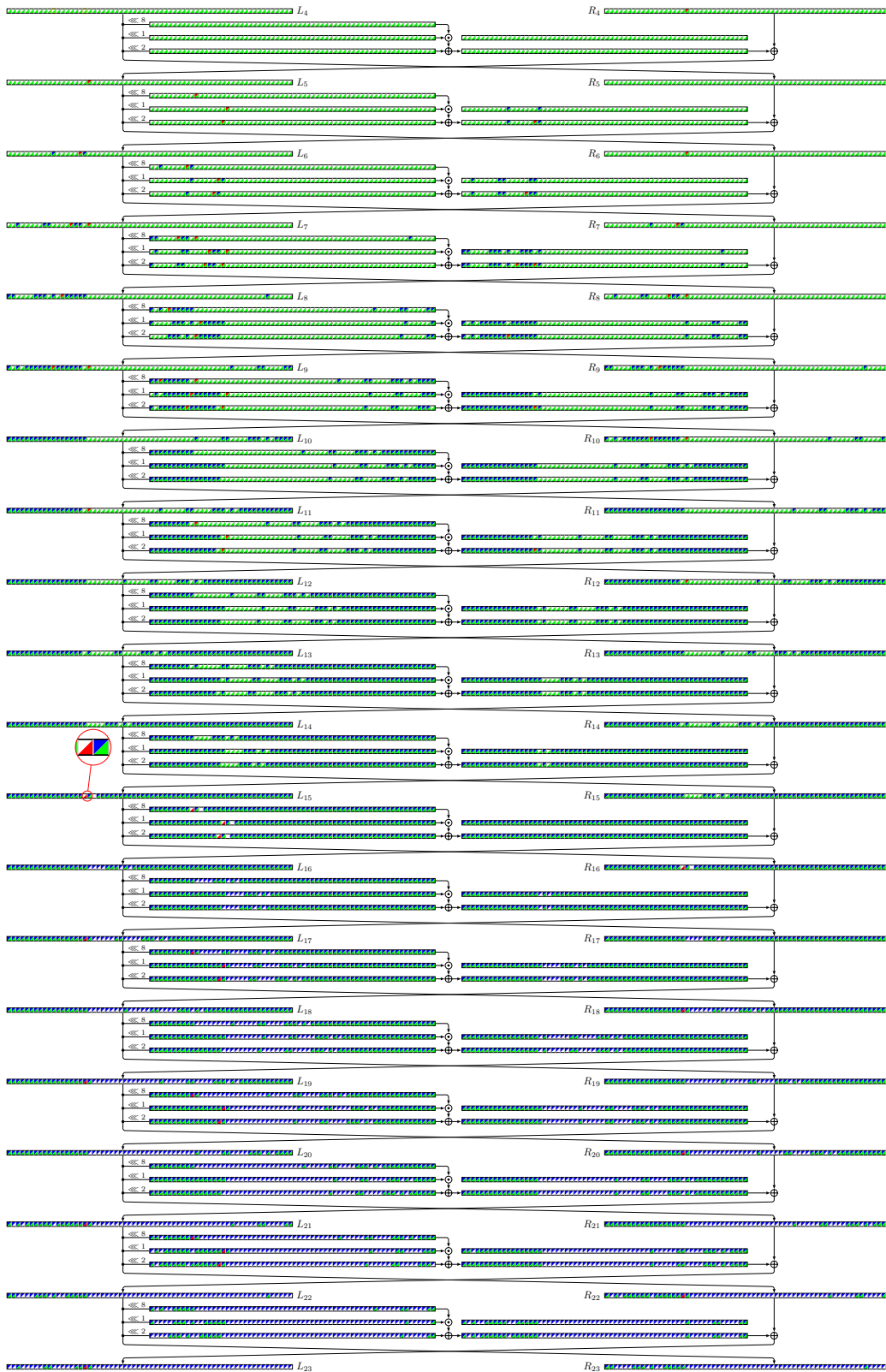


Figure 37: Key recovery of the attack on 25-round SIMON96-144.



■ bit difference (linear mask) 1 forward ■ bit difference (linear mask) 1 backward
■ unknown difference (linear mask) forward ■ unknown difference (linear mask) backward

Figure 38: 19-round ID distinguisher for attack on 27-round SIMON128-128.

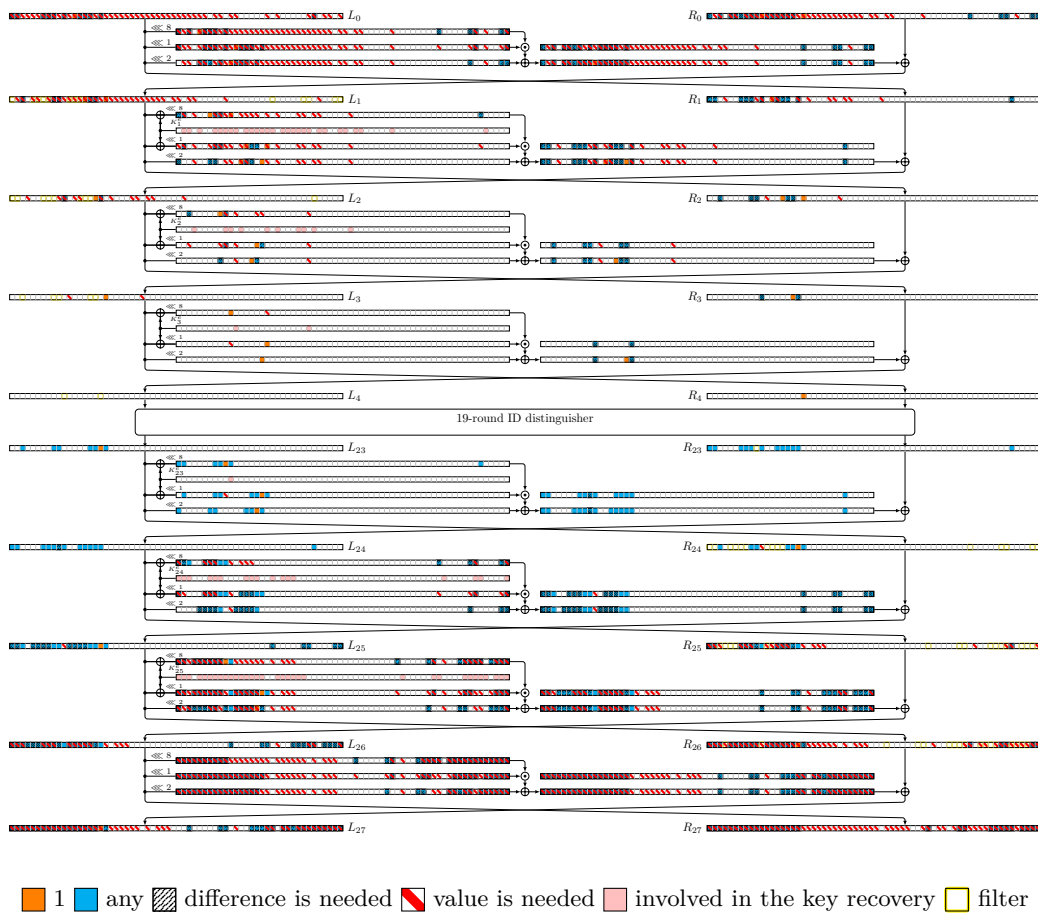


Figure 39: Key recovery of the attack on 27-round SIMON128-128.

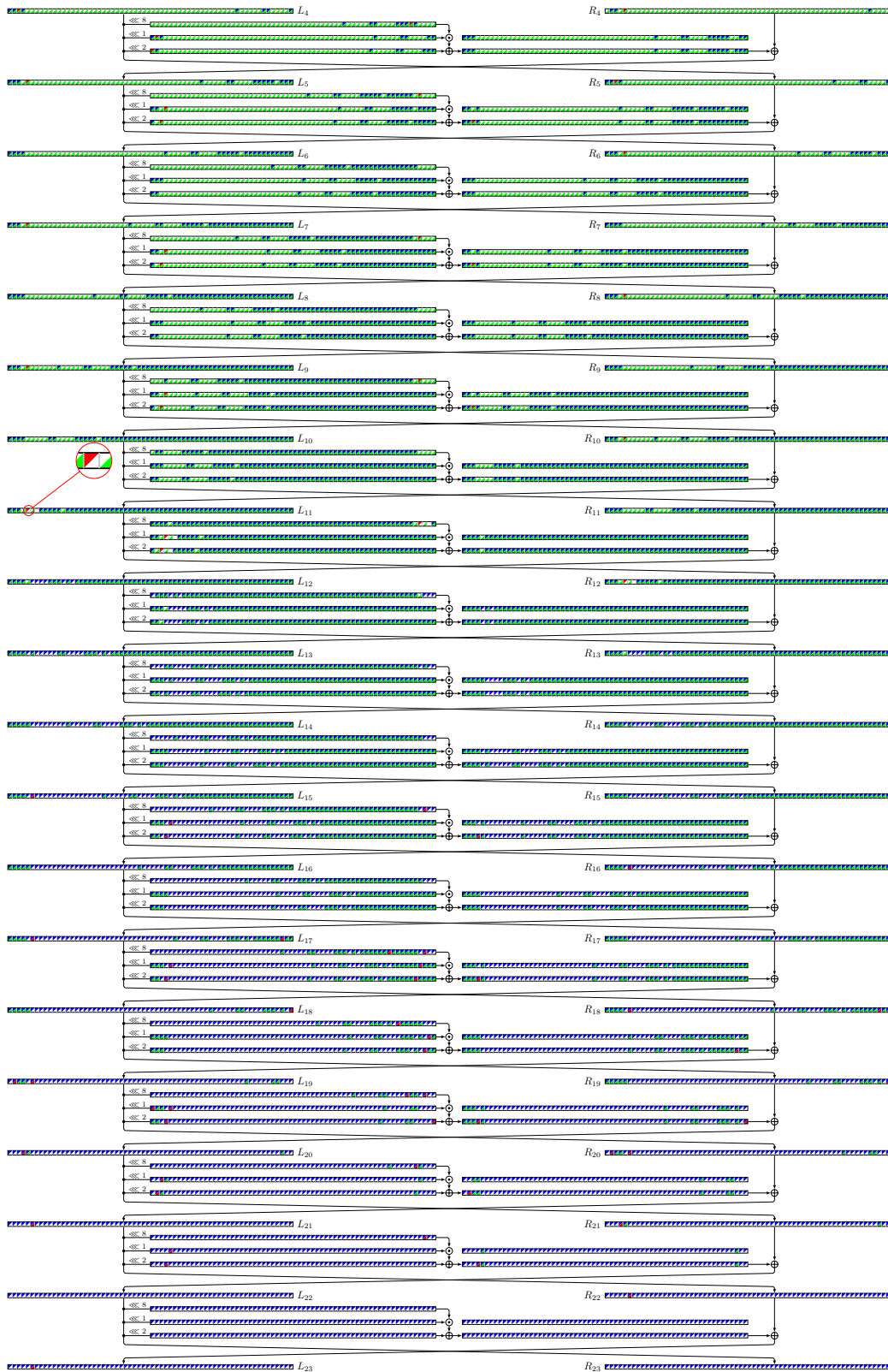


Figure 40: 19-round ID distinguisher for attack on 28-round SIMON128-128.

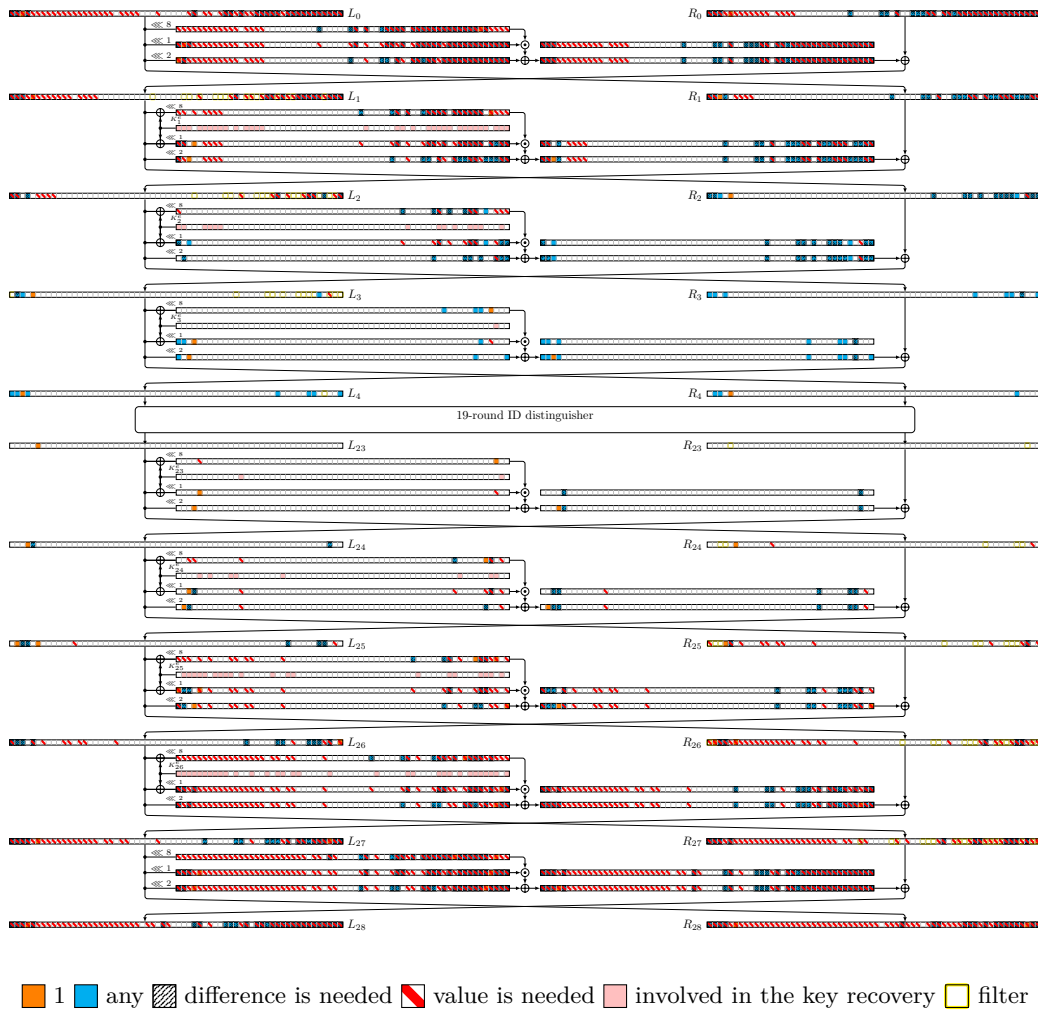
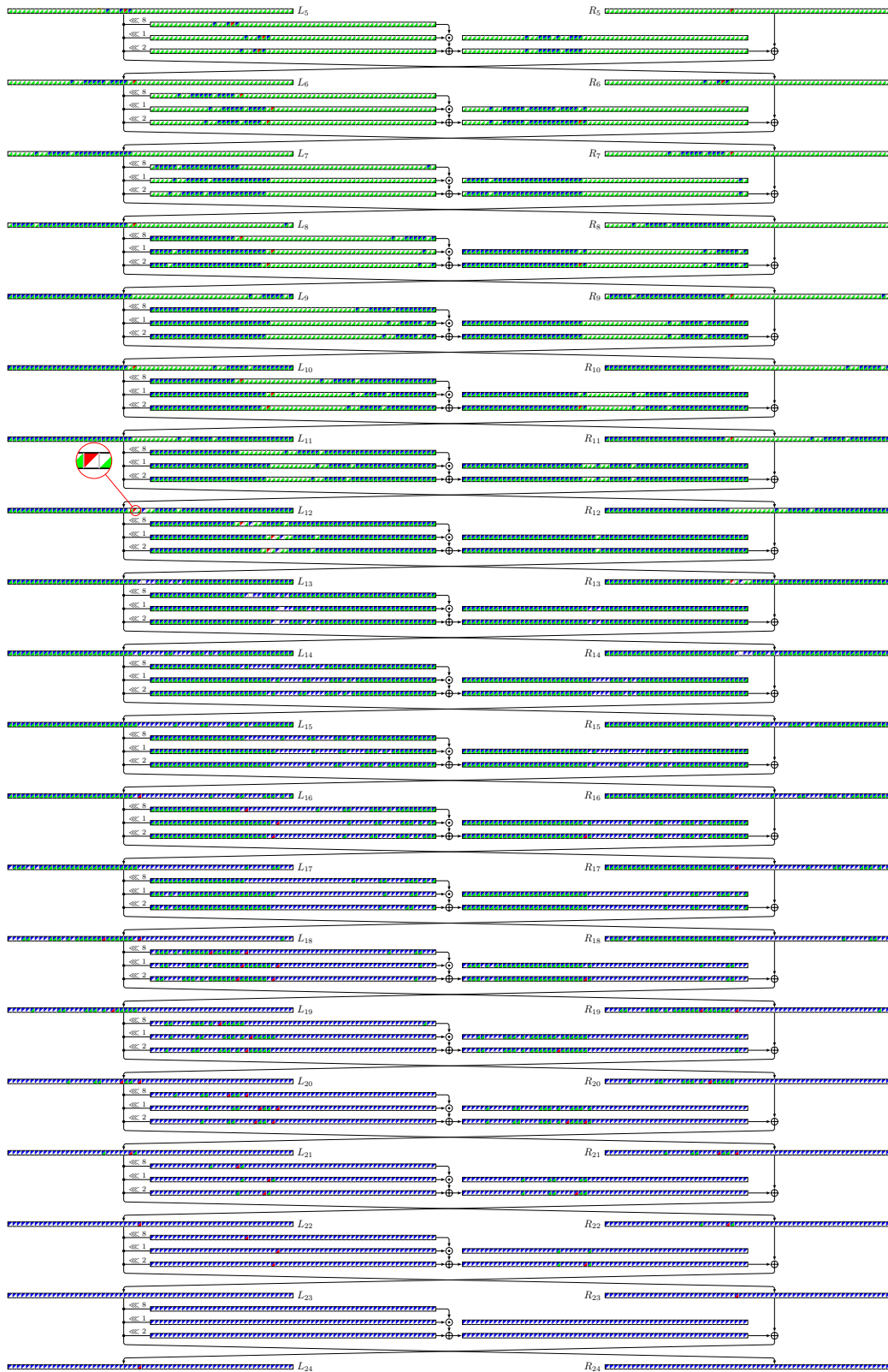


Figure 41: Key recovery of the attack on 28-round SIMON128-128.



bit difference (linear mask) 1 forward
 bit difference (linear mask) 1 backward
 unknown difference (linear mask) forward
 unknown difference (linear mask) backward

Figure 42: 19-round ID distinguisher for attack on 29-round SIMON128-192.

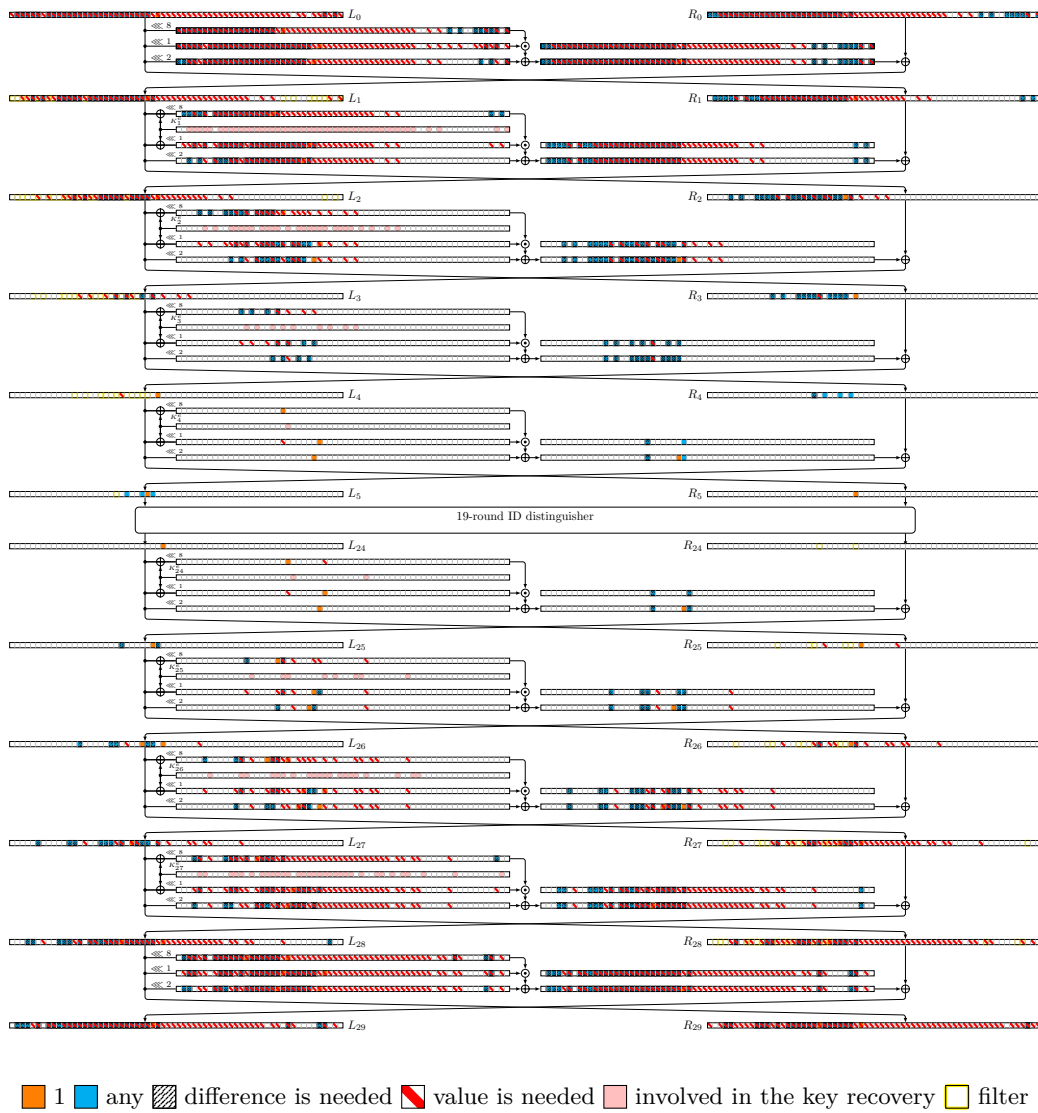


Figure 43: Key recovery of the attack on 29-round SIMON128-192.

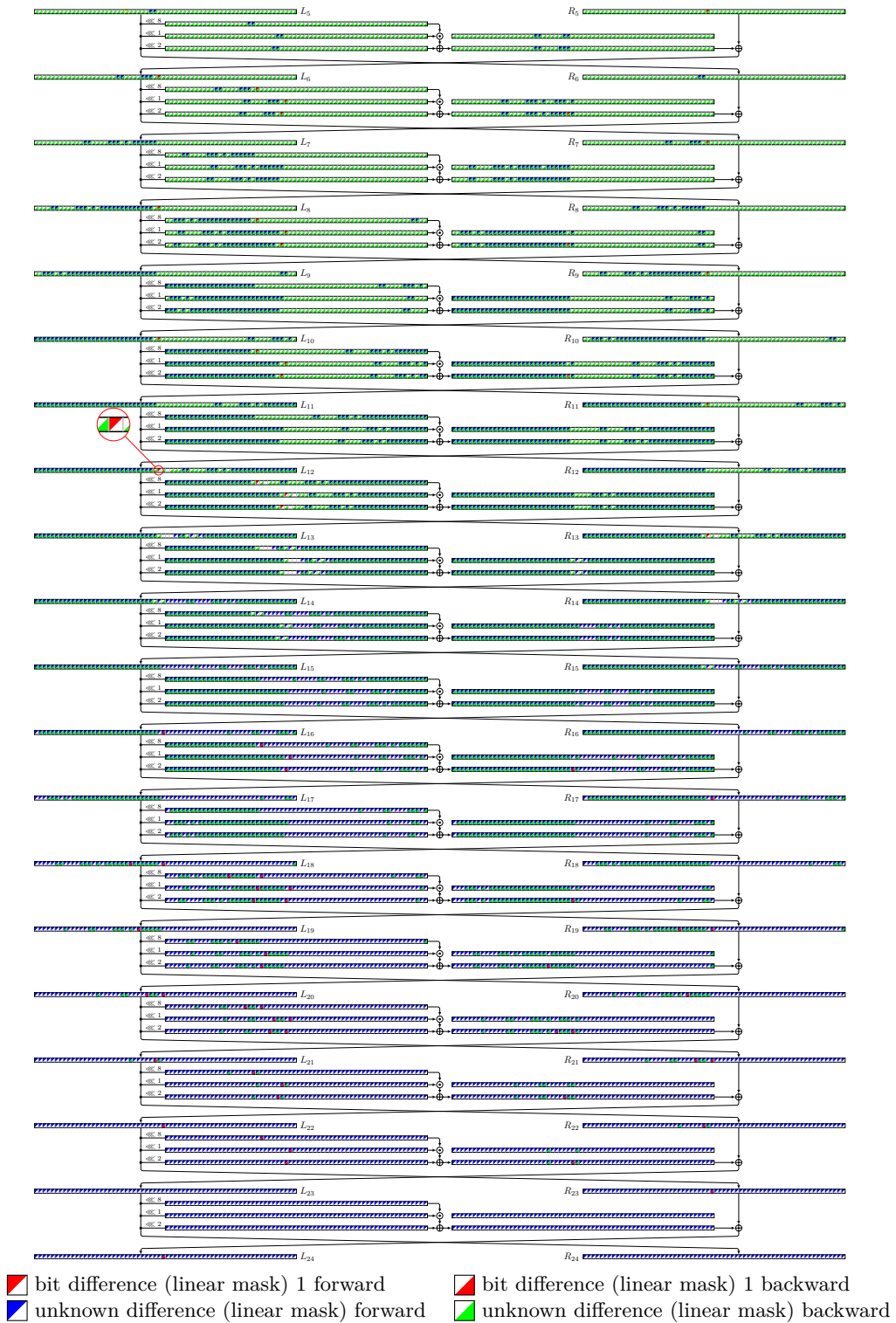


Figure 44: 19-round ID distinguisher for attack on 30-round SIMON128-192.

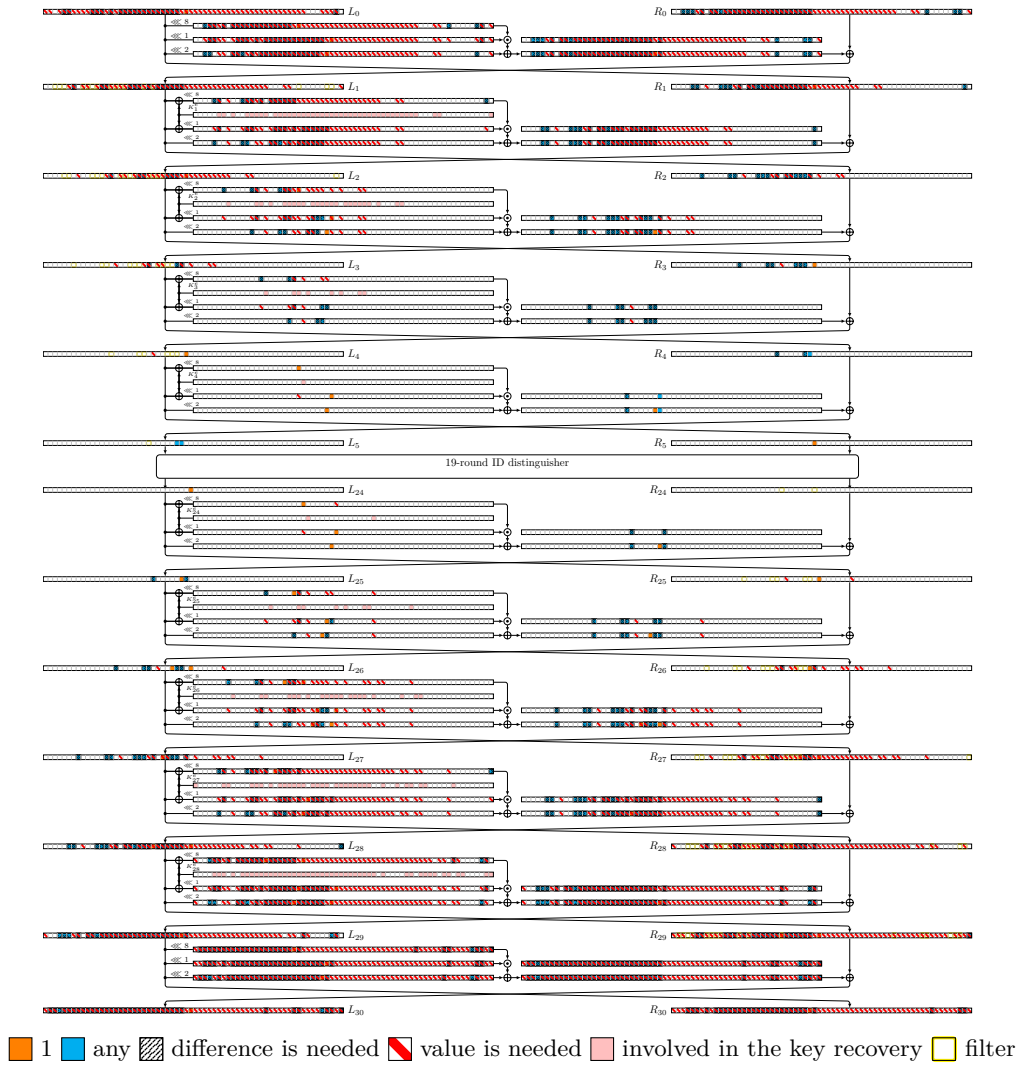


Figure 45: Key recovery of the attack on 30-round SIMON128-192.

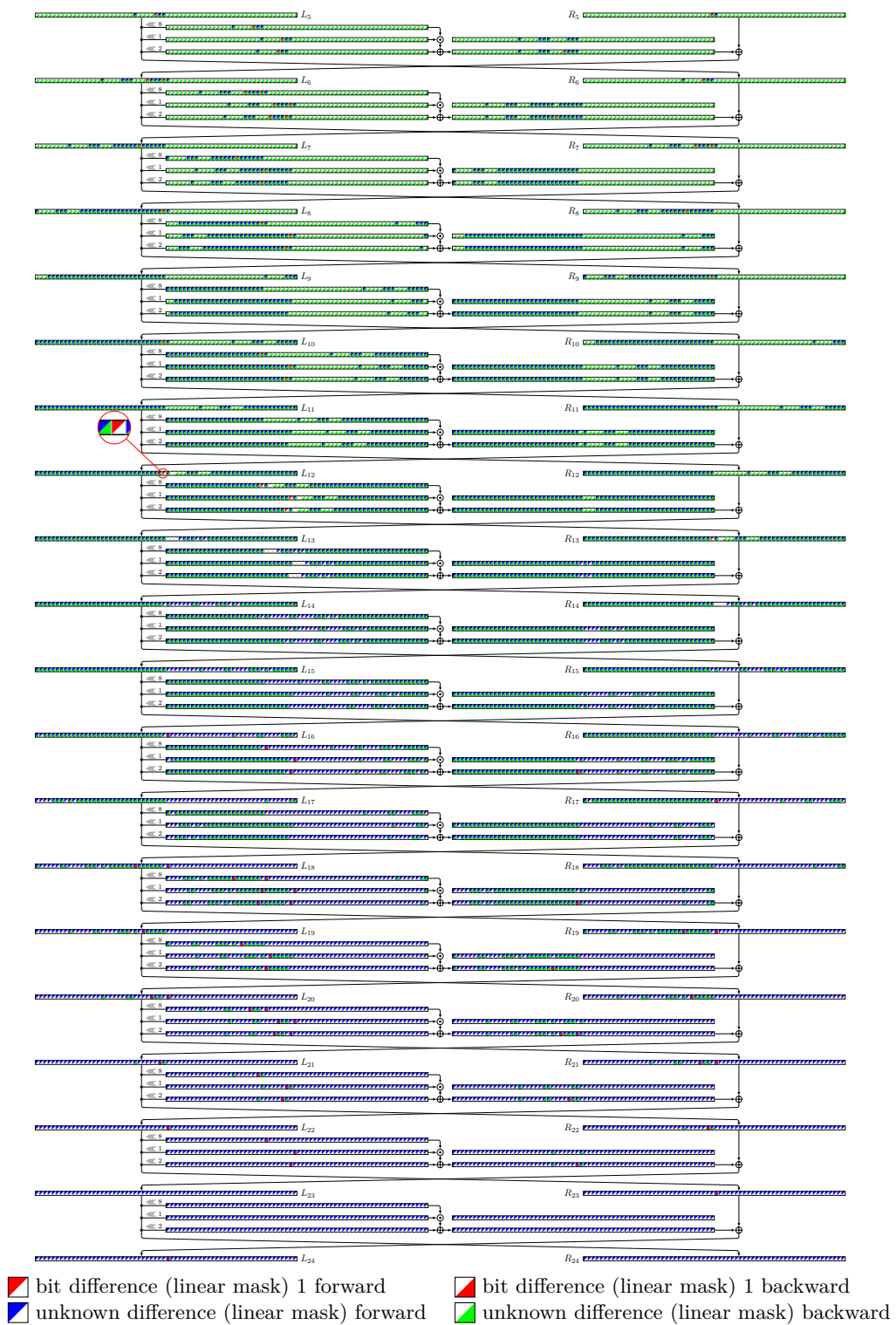
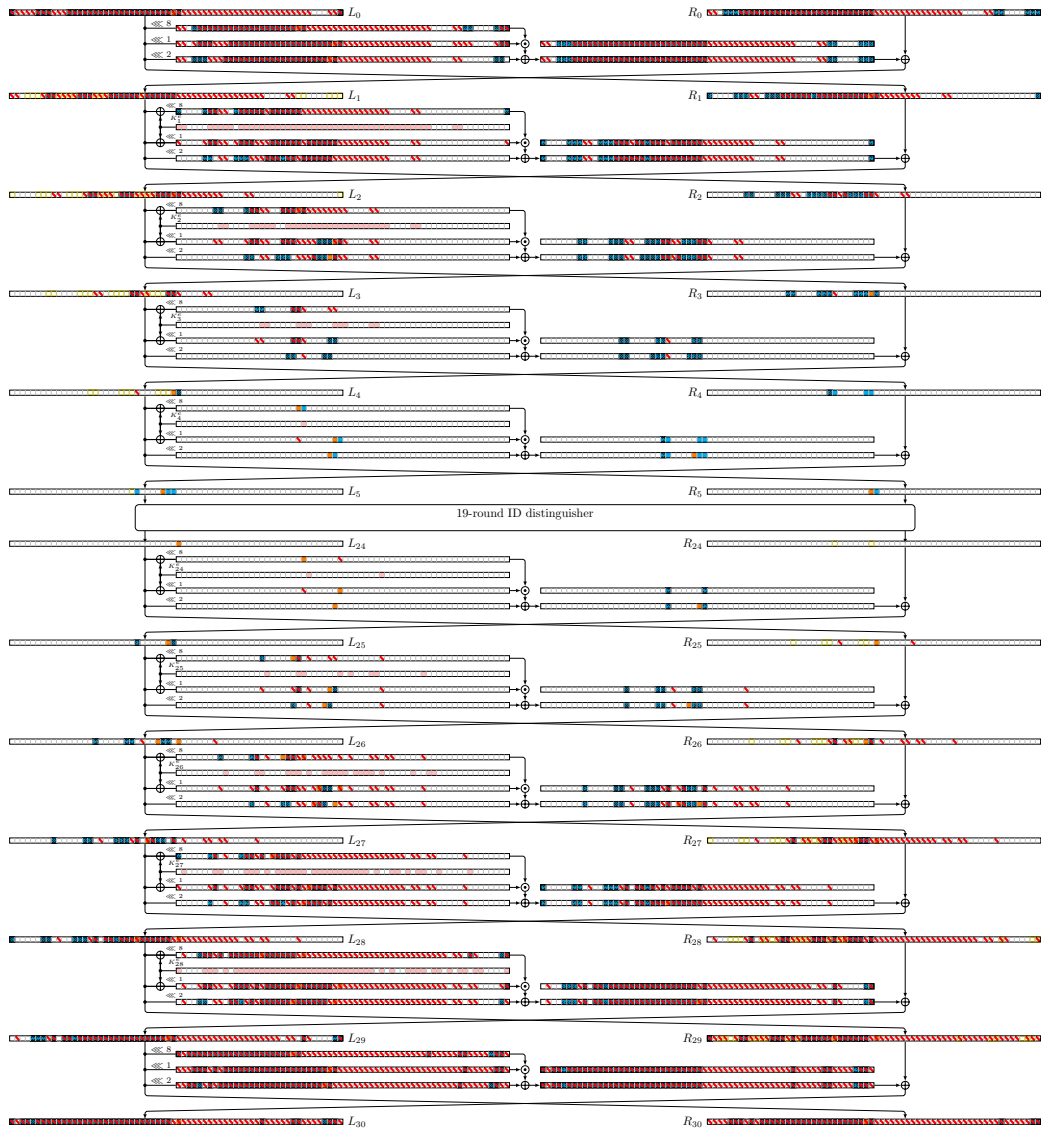


Figure 46: 19-round ID distinguisher for attack on 30-round SIMON128-256.



■ 1
 ■ any
 difference is needed
 ■ value is needed
 involved in the key recovery
 filter

Figure 47: Key recovery of the attack on 30-round SIMON128-256.

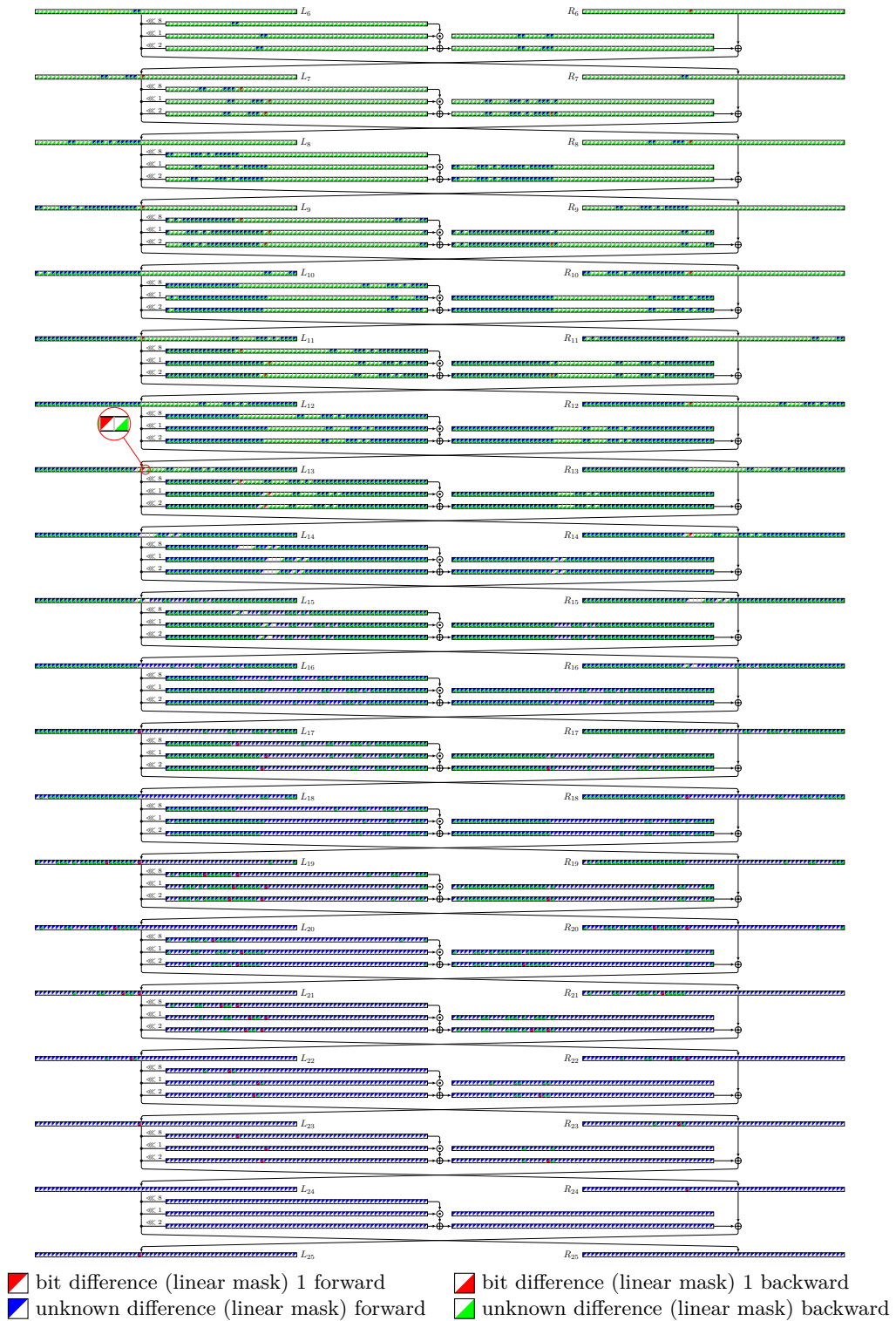
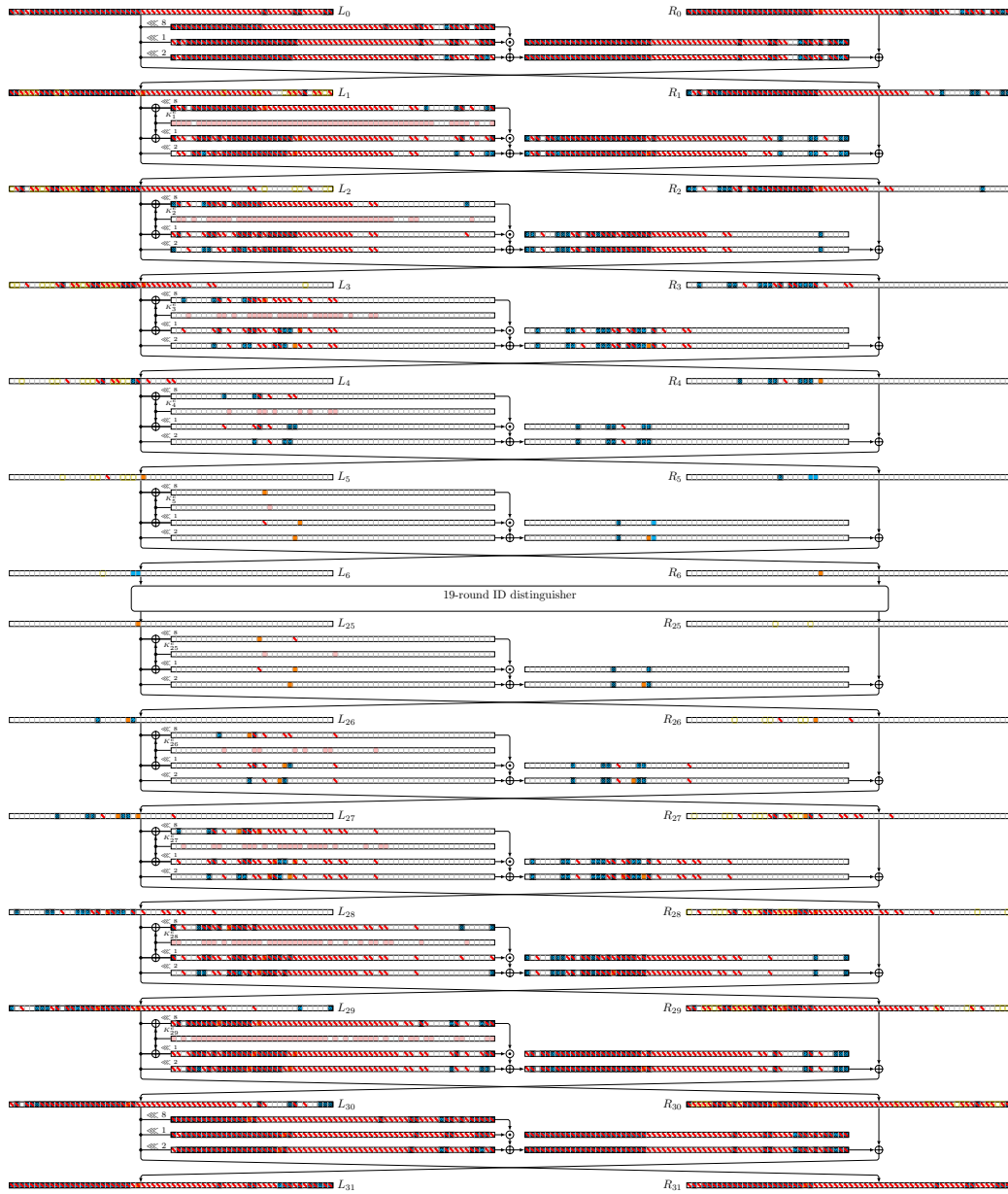


Figure 48: 19-round ID distinguisher for attack on 31-round SIMON128-256.



1
 any
 difference is needed
 value is needed
 involved in the key recovery
 filter

Figure 49: Key recovery of the attack on 31-round SIMON128-256.

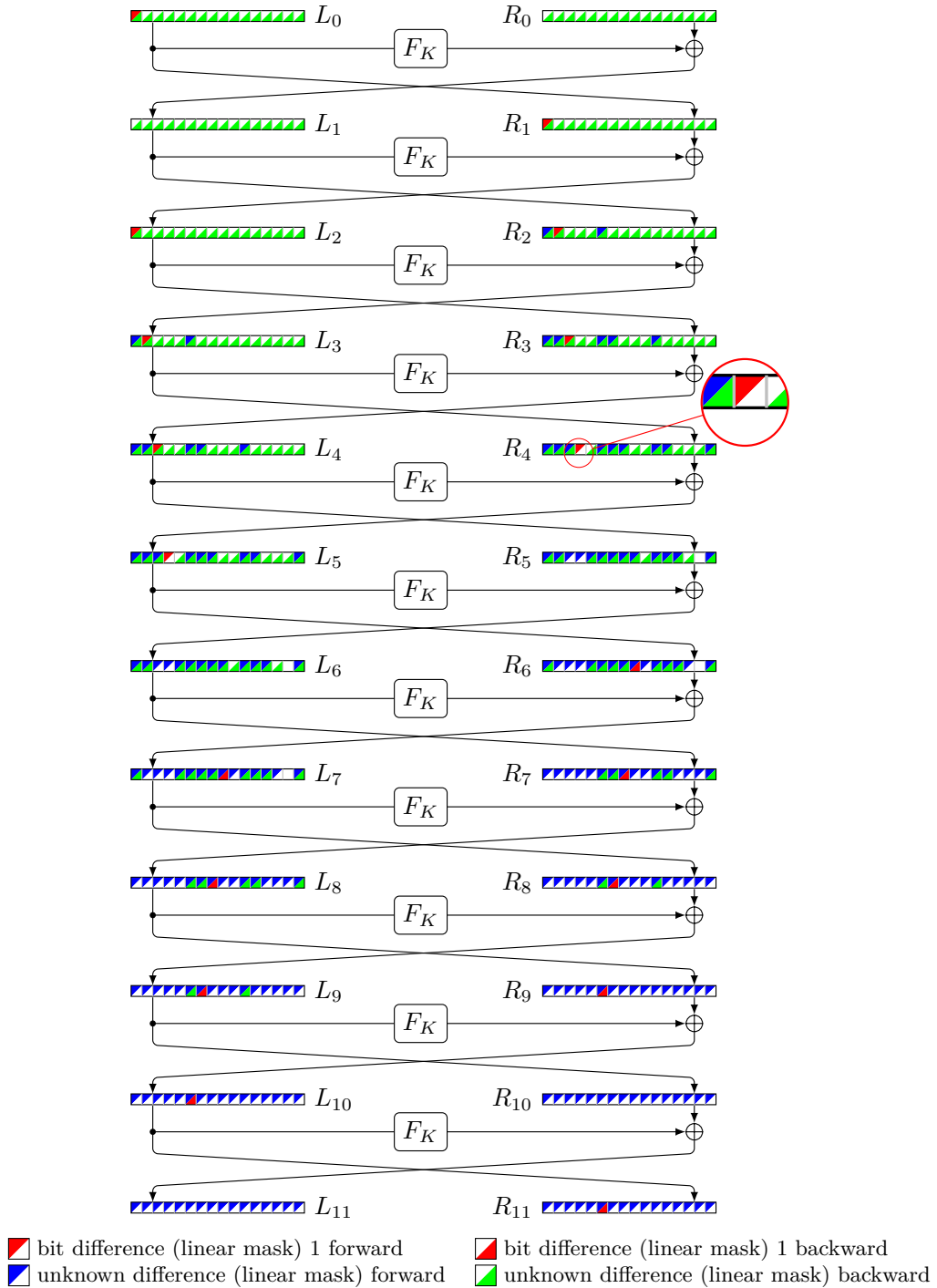


Figure 50: 11-round ZC distinguisher for Simeck32.

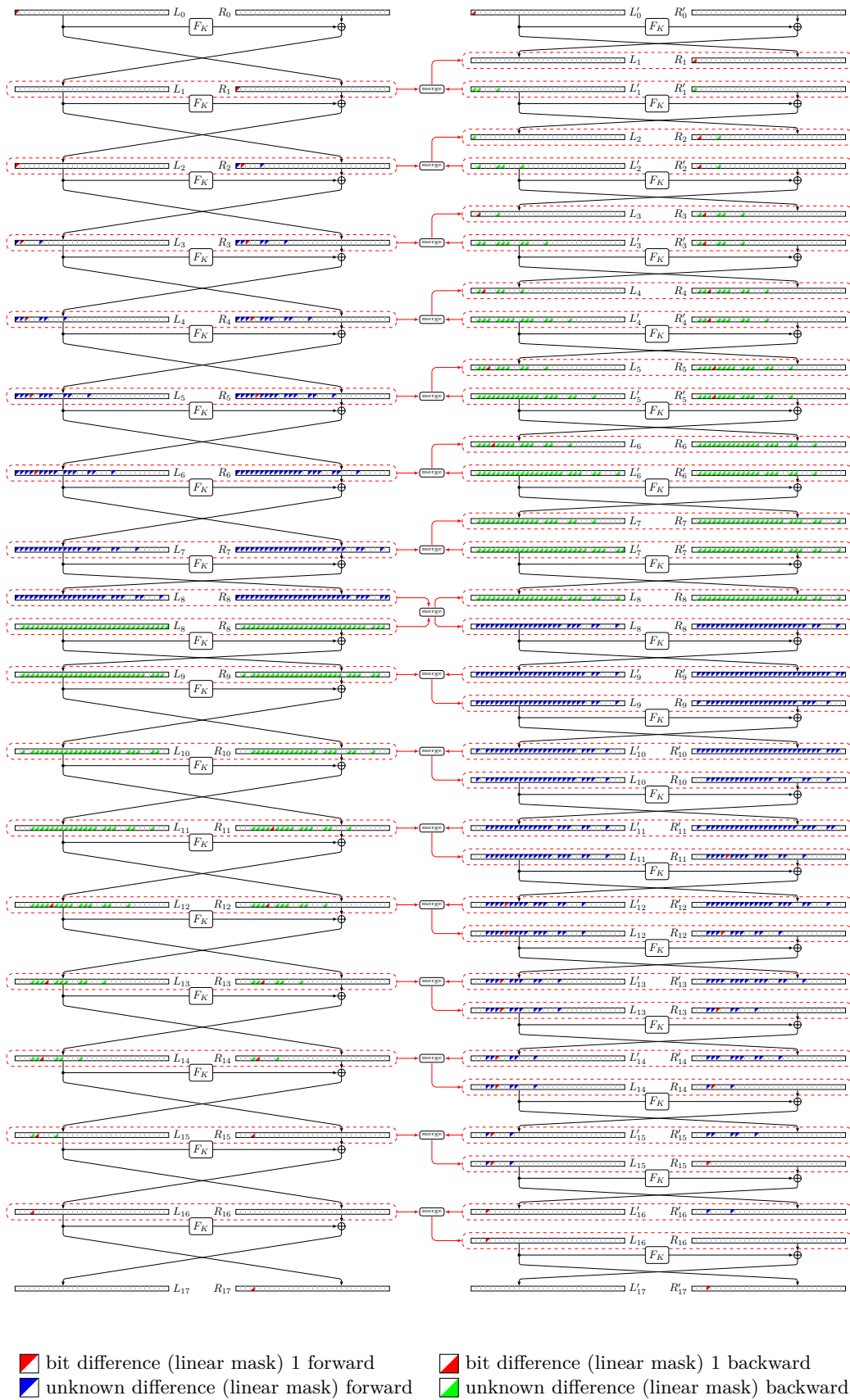


Figure 51: 17-round ZC distinguisher for Simeck64. In this case, the bit difference in the upper triangle of $L_2[0]$ (in the left-hand column) is 1, whereas the bit difference in the lower triangle of $L_2'[0]$ is 0. This leads to a contradiction occurring in the second round.