

Improved Search for Integral, Impossible-Differential and Zero-Correlation Attacks

Application to Ascon, ForkSKINNY, SKINNY, MANTIS, PRESENT
and QARMAv2

Hosein Hadipour¹(✉), Simon Gerhalter¹, Sadegh Sadeghi² and Maria
Eichlseder¹

¹ Graz University of Technology, Graz, Austria

hossein.hadipour@iaik.tugraz.at

² Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran

Abstract. Integral, impossible-differential (ID), and zero-correlation (ZC) attacks are three of the most important attacks on block ciphers. However, manually finding these attacks can be a daunting task, which is why automated methods are becoming increasingly important. Most automatic tools regarding integral, ZC, and ID attacks have focused only on finding distinguishers rather than complete attacks. At EUROCRYPT 2023, Hadipour et al. proposed a generic and efficient constraint programming (CP) model based on satisfiability for finding ID, ZC, and integral distinguishers. This new model can be extended to a unified CP model for finding full key recovery attacks. However, it has limitations, including determining the contradiction location beforehand and a cell-wise model unsuitable for weakly aligned ciphers like Ascon and PRESENT. They also deferred developing a CP model for the partial-sum technique in key recovery as future work.

In this paper, we enhance Hadipour et al.'s method in several ways. First, we remove the limitation of determining the contradiction location in advance. Second, we show how to extend the distinguisher model to a bit-wise model, considering the internal structure of S-boxes and keeping the model based on satisfiability. Third, we introduce a CP model for the partial-sum technique for the first time. To show the usefulness and versatility of our approach, we apply it to various designs, from strongly aligned ones like ForkSKINNY and QARMAv2 to weakly aligned ones such as Ascon and PRESENT, yielding significantly improved results. To mention a few of our results, we improve the integral distinguisher of QARMAv2-128 (resp. QARMAv2-64) by 7 (resp. 5) rounds, and the integral distinguisher of ForkSKINNY by 1 round, only thanks to our cell-wise distinguisher modelings. By using our new bit-wise modeling, our tool can find a group of 2^{155} 5-round ID and ZC distinguishers for Ascon in only one run, taking a few minutes on a regular laptop. The new CP model for the partial-sum technique enhances integral attacks on all SKINNY variants, notably improving the best attack on SKINNY- n - n in the single-key setting by 1 round. We also enhance ID attacks on ForkSKINNY and provide the first analysis of this cipher in a limited reduced-round setting. Our methods are generic and applicable to other block ciphers.

Keywords: Integral attacks · Partial-sum technique · Impossible differential attacks · Zero-correlation attacks · Ascon · SKINNY · SKINNYe · ForkSKINNY · QARMAv2 · MANTIS · PRESENT

1 Introduction

Three important attacks on block ciphers are integral, impossible differential (ID), and zero-correlation (ZC) attacks. For example, the best attack on 6 rounds of AES so far is the integral attack [FKL⁺00], taking advantage of the partial-sum technique in key recovery. As another example, the first 7-round attack on AES-128 was the ID attack [LDKK08]. The integral attack was initially introduced as a theoretical generalization of differential analysis by Lai [Lai94], and as a practical attack by Daemen et al. [DKR97]. The core idea of the integral attack is to find a set of inputs whose outputs sum to a key independent value in some positions. The ID attack, independently introduced by Biham et al. [BBS99] and Knudsen [Knu98], exploits an impossible differential in a block cipher to retrieve the master key. The ZC attack, first introduced by Bogdanov and Rijmen [BR14], is the dual method of the ID attack when it comes to linear analysis. These attacks were discovered and studied independently at different times. At ASIACRYPT 2012 [BLNW12], Bogdanov et al. established a link between the (multidimensional) ZC approximation and integral distinguishers. At CRYPTO 2015 [SLR⁺15], Sun et al. developed further the links among the ID, ZC, and integral attacks. These links are handy because a discovery or tool regarding one of these attacks may be used to improve another. For example, the miss-in-the-middle technique that was first introduced to find ID distinguishers [BBS99, BLNW12] was later used to find ZC distinguishers [BLNW12]. At ToSC 2019 [ADG⁺19], Ankele et al. studied the impact of the tweakey schedule in ZC distinguishers and used a miss-in-the-middle like approach to find ZC and integral distinguishers for tweakable block ciphers.

The core idea of the miss-in-the-middle technique is to find a pair of differences (resp. linear masks) that propagate halfway through the cipher forward and backward with certainty but contradict each other somewhere in the middle. The inconsistency between these propagations yields an impossible differential (resp. unbiased linear hull). Yet, this technique involves tracking how the differences (resp. linear masks) propagate through the cipher, either in the word- or bit-level, which is a time-consuming and potentially error-prone process using a manual approach. Moreover, finding a complete integral, ID, or ZC attack includes two phases: finding the distinguisher is only the first phase, and the key recovery phase should also follow that. Although finding the distinguishers for ID, ZC, and integral attacks can benefit from the same technique (miss-in-the-middle), the key recovery techniques for these three attacks differ. For example, the early-abort technique is a common approach for the key recovery of ID attacks, and the partial-sum technique is the typical method for the key recovery of ZC and integral attacks.

Expanding the distinguisher on either one or both ends is necessary when constructing the key recovery. This involves tracking the spread of additional cryptographic characteristics while considering various important factors. For example, the position of the involved cells/bits in the key recovery and the order of guessing variables can affect the final complexity of the attack. Overall, finding an optimum complete integral, ID, or ZC attack is a combinatorial optimization problem, which is very hard to solve using a manual approach. This is why developing automatic tools to evaluate the security of block ciphers against these attacks is becoming increasingly important. Particularly for new lightweight designs, more accurate and efficient security evaluation helps designers minimize security margins, and improve performance.

One of the common approaches to solving the optimization problems stemming from cryptanalytic attacks is converting the cryptanalytic problem into a constraint satisfaction problem (CSP) or a constraint optimization problem (COP) and then solving it with off-the-shelf general-purpose CP/MILP/SMT/SAT solvers. As a pioneering work at EUROCRYPT 2017, Sasaki and Todo proposed an automatic tool based on MILP solvers to find ID distinguishers [ST17]. At the same time, Cui et al. proposed a similar approach to finding ID and ZC distinguishers [CCJ⁺16]. However, all of these works were focused only

on finding the distinguishers. As another primary limitation, all of these modelings were based on the unsatisfiability of the models where the input/output difference/mask is fixed. This has also been the case in all existing CP models to search for integral distinguishers based on division property [Tod15, XZBL16] or monomial prediction [HSWW20, HE22].

At EUROCRYPT 2023, Hadipour et al. [HSE23] proposed a very generic and efficient CP-based method to search for the complete integral, ID, and ZC attacks. Unlike all previous works, the core idea in [HSE23] was creating a CP model based on satisfiability to search for ID, ZC, and integral distinguishers. This key feature allows extending the distinguisher model to a unified optimization model for directly finding a nearly optimum attack, including the key recovery and complexity evaluation. Another advantage of the method in [HSE23] is considering some key recovery techniques, such as key-bridging, to find a better attack. However, the method in [HSE23] has some limitations as mentioned by the authors:

- In the CP model for distinguishers in [HSE23], one should determine the contradiction’s location in advance. It means the model should be run for several possible contradiction locations, which is sometimes inefficient. So, one question is how to relax this limitation.
- The method in [HSE23] is a cell-wise model that was proved to work very well for strongly aligned or cell-wise designs such as SKINNY, but not for the weakly aligned and bit-wise ciphers, such as Ascon [DEMS21] and PRESENT [BKL⁺07]. So, the second question is how to extend the method in [HSE23] to a bit-wise model.
- Most importantly, the partial-sum technique, which is an important optimization technique in the key recovery of integral attacks, is not integrated into the CP model in [HSE23]. The authors only propose a separate tool to apply after a distinguisher has already been fixed, which is suboptimal for two reasons: First, the best distinguisher does not necessarily lead to the best key-recovery attack. Second, their tool does not fully optimize the order of partial-sum steps. So, the third question is how to convert the partial-sum technique into a CP model, which the authors left open as future work.

Our contributions. In this paper, we answer all of these questions and improve the method in [HSE23] in several ways:

- We first remove the limitation of determining the contradiction location in advance. Our new modeling automatically locates contradictions to optimize the objective function while still keeping the model based on satisfiability. We need not split the distinguisher into two parts, as in [HSE23]. Given only one integer number r_D as the number of rounds for the distinguisher, a CP model based on satisfiability is created to find the best distinguisher for r_D rounds. We applied this new improved cell-wise modeling to ForkSKINNY [ALP⁺19], MANTIS [BJK⁺16], QARMAv2 [ABD⁺23] and got a series of significantly improved results.

Regarding QARMAv2, we discovered a 12-round (rep. 10-round) integral distinguisher for QARMAv2-128 (rep. QARMAv2-64), outperforming the best integral distinguisher by 7 (rep. 5) rounds (see Table 1). The existing best integral distinguisher for QARMAv2 is a 5-round one based on the division property claimed by the designers [ABD⁺23]. This highlights the effectiveness of our tool for strongly aligned ciphers. Note that our findings for QARMAv2 do not threaten its security.

For ForkSKINNY, we could efficiently scan many variants of this cipher (depending on the SKINNY version, the position of the fork (\mathbf{r}_1), and the length of output branches ($\mathbf{r}_0, \mathbf{r}_1$). Attacks on SKINNY can be adapted to ForkSKINNY, but not vice versa. So,

without previous ID, ZC, and integral distinguishers for specific ForkSKINNY variants, we have to compare our findings with top SKINNY ID/ZC and integral distinguishers for these variants. For instance, we introduce a 22-round related-tweakey ID (resp. 17-round integral) distinguisher for ForkSKINNY-64-192, improving the best-known ID (resp. integral) distinguisher for this cipher by 5 (resp. 1) rounds; see Table 1. Extending our distinguisher modeling to a unified model for key recovery, we find the best ID attacks on many variants of ForkSKINNY. We also analyze ForkSKINNY for the first time in the limited reduced-round setting (see Appendix D). In all cases, we propose the best ID attacks; see Table 2.

- Next, we show how to extend the distinguisher model to a bit-wise model, taking advantage of DDT and LAT of S-boxes yet keeping the model based on satisfiability. To show the efficiency and versatility of our approach, we applied it to very diverse designs, including Ascon and PRESENT, and could reproduce the best-known ID and ZC distinguishers with much less effort than the methods based on unsatisfiability [ST17, CCJ⁺16]. For example, only one run of our tool, taking a few minutes on a regular laptop, can find a cluster of 2^{155} ID/ZC distinguishers for 5 rounds of Ascon (see Appendix L, Figure 50, Figure 51, Figure 52, and Figure 53). Similar to the cell-wise model, our bit-wise model is a simple CP/MILP model based on satisfiability that can be extended to a unified optimization model for key recovery.
- Lastly, we focus on the key recovery part and introduce a CP model for the partial-sum technique for the first time. We applied our tool to SKINNY and ForkSKINNY. For SKINNY, we could improve the best-known integral attacks on all variants of this cipher. Notably, we introduce an 18-round integral attack on SKINNY- n - n , which improves the best single-key attack on this variant of SKINNY by 1 round. We provided 27-round (resp. 23-round) integral attacks on ForkSKINNY-64-192 (resp. ForkSKINNY-128-256) that are the best attacks in the single-key setting; see Table 2.

The source code of our tool is available at <https://github.com/hadipourh/zeroplus>.

Table 1: Integral and ID distinguishers. †: Distinguishers of SKINNY that can be directly applied to ForkSKINNY. ‡: #Rounds by skipping the first S-box layer.

Cipher	#Rounds	Dist.	Data complexity	Ref.
QARMAv2-64	5	Integral	-	[ABD ⁺ 23]
QARMAv2-64 ($\mathcal{T} = 1$)	7 / 8 / 9	Integral	$2^8 / 2^{16} / 2^{44}$	I
QARMAv2-64 ($\mathcal{T} = 2$)	8 / 9 / 10	Integral	$2^8 / 2^{16} / 2^{44}$	I
QARMAv2-128 ($\mathcal{T} = 2$)	10 / 11 / 12	Integral	$2^{16} / 2^{44} / 2^{96}$	I
MANTIS	9	Integral	2^{44}	[ADG ⁺ 19]
MANTIS	7 / 8 / 9	Integral	$2^8 / 2^{16} / 2^{44}$	J
ForkSKINNY-64-192 [†]	16	Integral	2^{72}	[NLSW21]
ForkSKINNY-64-192 [†]	16	Integral	2^{60}	[HSE23]
ForkSKINNY-64-192	17	Integral	2^{60}	F
ForkSKINNY-64-192 [†]	16 (17 [‡])	ID	-	[HSE23]
ForkSKINNY-64-192	21 (22[‡])	ID	-	F
ForkSKINNY-128-256 [†]	14	Integral	2^{56}	[HSE23]
ForkSKINNY-128-256	15	Integral	2^{56}	F
ForkSKINNY-128-256	17 (18 [‡])	ID	-	[BDL20]
ForkSKINNY-128-256	17 (18 [‡])	ID	-	F
ForkSKINNY-128-288	21 (22 [‡])	ID	-	G.9

Table 2: Summary of our cryptanalytic results for SKINNY. ID/Int = impossible differential, integral. STK/RTK = single/related-tweakey. SK = single-key with given keysize, CP/KP = chosen/known plaintext, CT = chosen tweak. †: attack has minor issues. * : limited setting for ForkSKINNY, i.e., reduced-round by $r_1 - x$ rounds before the fork and $r_0 - x = r_1 - x$ rounds in each branch after the fork for an integer x (see Appendix D).

Cipher	#R	Time	Data	Mem.	Attack	Setting / Model	Ref.
SKINNY-64-64	17	$2^{61.80}$	$2^{59.50}$	$2^{49.60}$	ID	STK / CP	[YQC17]
	17	2^{59}	$2^{58.79}$	2^{40}	ID	STK / CP	[HSE23]
	18	$2^{53.58}$	$2^{53.58}$	2^{48}	Int	60,SK / CP,CT	4.2
SKINNY-128-128	17	$2^{120.80}$	$2^{118.50}$	$2^{97.50}$	ID	STK / CP	[YQC17]
	17	$2^{116.51}$	$2^{116.37}$	2^{80}	ID	STK / CP	[HSE23]
	18	$2^{105.58}$	$2^{105.58}$	2^{96}	Int	120,SK / CP,CT	4.2
SKINNY-64-128	20 [†]	$2^{97.50}$	$2^{68.40}$	2^{82}	Int [†]	120,SK / CP,CT	[ADG ⁺ 19]
	22	2^{110}	$2^{57.58}$	2^{108}	Int	120,SK / CP,CT	[HSE23]
	22	2^{106}	2^{58}	2^{104}	Int	120,SK / CP,CT	C.1
SKINNY-128-256	22	2^{216}	$2^{113.58}$	2^{216}	Int	240,SK / CP,CT	[HSE23]
	22	2^{213}	2^{112}	2^{208}	Int	240,SK / CP,CT	C.1
SKINNY-64-192	23 [†]	$2^{155.60}$	$2^{73.20}$	2^{138}	Int [†]	180,SK / CP,CT	[ADG ⁺ 19]
	26	2^{172}	2^{61}	2^{172}	Int	180,SK / CP,CT	[HSE23]
	26	2^{166}	2^{62}	2^{164}	Int	180,SK / CP,CT	C.2
SKINNY-128-384	26	2^{344}	2^{121}	2^{340}	Int	360,SK / CP,CT	[HSE23]
	26	2^{331}	2^{124}	2^{328}	Int	360,SK / CP,CT	C.2
SKINNY-128-288	26	$2^{286.38}$	2^{122}	2^{194}	ID	288,RTK / CP	H.1
	23	$2^{126.73}$	$2^{120.80}$	$2^{88.80}$	ID	128,RTK / CP	H.2
SKINNYe-v2	30	2^{232}	2^{65}	2^{228}	Int	240,SK / CP,CT	[HSE23]
	31	$2^{251.14}$	2^{62}	2^{110}	ID	RTK / CP	H.3
ForkSKINNY-64-192	27	2^{166}	2^{58}	2^{164}	Int	180,SK / CP,CT	E.2
	32	$2^{186.27}$	2^{62}	2^{114}	ID	192,RTK / CP	G.3
	30	$2^{123.73}$	2^{62}	2^{86}	ID	128,RTK / CP	G.4
	28*	$2^{169.60}$	2^{60}	2^{104}	ID	192,RTK / CP	G.1
	28*	$2^{123.73}$	2^{62}	2^{86}	ID	128,RTK / CP	G.2
	23	2^{214}	2^{114}	2^{208}	Int	240,SK / CP,CT	E.1
ForkSKINNY-128-256	26	$2^{254.6}$	2^{125}	2^{160}	ID	256,RTK / CP	[BDL20]
	26	$2^{250.3}$	2^{127}	2^{160}	ID	256,RTK / CP	[BDL20]
	26	$2^{238.50}$	$2^{127.60}$	$2^{175.60}$	ID	256,RTK / CP	G.5
	26	$2^{244.50}$	2^{127}	2^{175}	ID	256,RTK / CP	G.5
	24	$2^{124.5}$	$2^{122.5}$	$2^{97.5}$	ID	128,RTK / CP	[BDL20]
	24	$2^{123.17}$	$2^{118.40}$	$2^{118.40}$	ID	128,RTK / CP	G.7
	24*	$2^{246.62}$	$2^{126.70}$	$2^{158.70}$	ID	256,RTK / CP	G.7
	20*	$2^{102.20}$	2^{101}	2^{93}	ID	128,RTK / CP	G.8
ForkSKINNY-128-288	31	$2^{280.52}$	$2^{126.50}$	$2^{190.50}$	ID	288,RTK / CP	G.9
	28	$2^{126.68}$	$2^{124.80}$	$2^{100.69}$	ID	128,RTK / CP	G.10
	28*	$2^{277.23}$	$2^{126.90}$	$2^{158.90}$	ID	288,RTK / CP	G.11
	26*	$2^{126.74}$	$2^{124.50}$	$2^{68.50}$	ID	128,RTK / CP	G.12

Outline. In Section 2, we provide background on ID, ZC, and integral attacks with the partial-sum technique, and recall CP models for deterministic trails. In Subsection 3.1, we introduce our new cell-wise CP model. In Subsection 3.2, we present our bit-wise CP model. In Section 4, we propose our unified optimization model for integral attacks using the partial-sum technique and apply it to SKINNY. Finally, we conclude in Section 5.

2 Background

Here, we briefly review ID attacks, then explain the connection between ZC and integral distinguishers to help the reader understand the terminology for integral distinguishers in this paper. Next, we discuss the partial-sum technique for key recovery in integral attacks. Finally, we review constraint programming models in automatic cryptanalysis.

2.1 Impossible Differential Attacks

Biham et al. [BBS99] and Knudsen [Knu98] independently introduced the impossible differential (ID) attack. The ID attack relies on an impossible differential to distinguish the block cipher from a random permutation. An impossible differential for E_D is a pair of differences (Δ_U, Δ_L) such that Δ_U can never lead to Δ_L . Consider E as a block cipher with n -bit block size and k -bit key. As depicted in Figure 1, let's assume the existence of an impossible differential $\Delta_U \not\rightarrow \Delta_L$ for r_D rounds of E denoted by E_D . Suppose that Δ_U (Δ_L) propagates backward (resp. forward) with probability 1 through E_B^{-1} (resp. E_F) to Δ_B (Δ_F), and $|\Delta_B|$ ($|\Delta_F|$) denotes the dimension of the vector space Δ_B (resp. Δ_F). Let c_B (c_F) be the number of bit-conditions that should be satisfied for $\Delta_B \rightarrow \Delta_U$ (resp. $\Delta_L \leftarrow \Delta_F$), i.e., $\Pr(\Delta_B \rightarrow \Delta_U) = 2^{-c_B}$ (resp. $\Pr(\Delta_L \leftarrow \Delta_F) = 2^{-c_F}$). Additionally, we denote k_B (k_F) as the key information, typically subkey bits, used in E_B (or E_F). Then, we can divide the ID attacks into three steps [HSE23]:

- *Step 1: Pair Generation.* We generate N pairs $(x, y) \in \{0, 1\}^{2n}$ such that $x \oplus y \in \Delta_B$ and $E(x) \oplus E(y) \in \Delta_F$ and store them. This is a limited birthday problem, and according to [BNPS14] the complexity of this step is:

$$T_0 = \max \left\{ \min_{\Delta \in \{\Delta_B, \Delta_F\}} \left\{ \sqrt{N 2^{n+1-|\Delta|}} \right\}, N 2^{n+1-|\Delta_B|-|\Delta_F|} \right\} \quad (1)$$

- *Step 2: Guess-and-Filter.* The objective of this stage is to eliminate all subkeys in $k_B \cup k_F$ that are invalidated by any of the generated pairs. Instead of attempting to guess all subkeys in $k_B \cup k_F$ simultaneously and testing them against all pairs, we can optimize this step by using the *early abort* technique [LKKD08]: We divide $k_B \cup k_F$ into smaller subsets, typically the round keys, and guess them step by step. At each step, we reduce the remaining pairs by checking if they satisfy the conditions of the truncated differential trail through E_B and E_F . The average number of partial encryptions/decryptions in this step is [BLNPS18]:

$$T_1 + T_2 = N + 2^{|k_B \cup k_F|} \frac{N}{2^{c_B + c_F}} \quad (2)$$

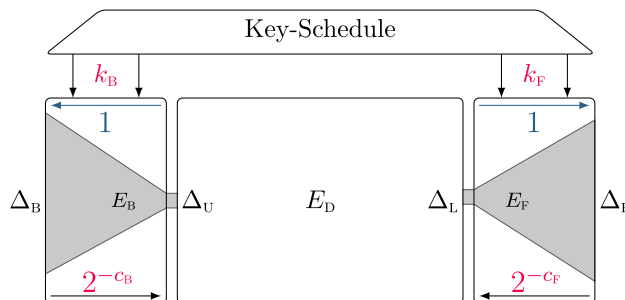


Figure 1: The effective parameters of the ID attack.

- *Step 3: Exhaustive Search.* The probability of a wrong key surviving the guess-and-filter step is $P = (1 - 2^{-(c_b+c_r)})^N$. Consequently, the average number of candidates after the guess-and-filter is $P \cdot 2^{|k_b \cup k_r|}$. Additionally, the guess-and-filter step doesn't involve $k - |k_b \cup k_r|$ bits of key information. Therefore, for a unique key determination, we need to search exhaustively in a space of size $T_3 = 2^{k - |k_b \cup k_r|} \cdot P \cdot 2^{|k_b \cup k_r|} = 2^k \cdot P$.

Assuming C_E is the cost of one full encryption and $C_{E'}$ is the ratio of the cost for one partial encryption to the full encryption, the total time complexity of the ID attack is:

$$T_{\text{tot}} = (T_0 + (T_1 + T_2)C_{E'} + T_3)C_E, \quad (3)$$

To maintain data complexity below the full codebook, we need $T_0 < 2^n$. Additionally, to extract at least one bit of key information in the guess-and-filter we need $P < \frac{1}{2}$. It's important to note that Equation 2 represents the average time complexity of the guess-and-filter step. Therefore, for each ID attack, a precise evaluation of its complexity is necessary to ensure we meet this bound.

In the related-(twea)key setting, an attacker can access two encryption (or decryption) oracles utilizing the related keys K , and $K \oplus \Delta K$, where the attacker knows ΔK . The aim is to retrieve K . In the related (twea)key setting, any plaintext structure is encrypted with two different (twea)keys. However, for any plaintext P in each structure, we can build two different pairs $((K, P), (K \oplus \Delta K, P \oplus \Delta P))$, and $((K \oplus \Delta K, P), (K, P \oplus \Delta P))$. As a result, the complexity formulations of the related-(twea)key setting are the same as the single-key setting, except that the data complexity and term T_0 should be modified as follows:

$$T_0^{RTK} = \max \left\{ \min_{\Delta \in \{\Delta_b, \Delta_r\}} \left\{ 2\sqrt{N2^{n-|\Delta|}} \right\}, N2^{n+1-|\Delta_b|-|\Delta_r|} \right\}, D^{RTK} = T_0^{RTK}/2 \quad (4)$$

Moreover, the condition $T_0 < 2^n$ is replaced by $T_0^{RTK} < 2^{n+1}$ in the related-key setting.

As discussed in some previous works, e.g., [Der16], the above formulations typically provide a *lower bound* of the complexity of key recovery phase, and the exact complexity may deviate from the above formulations. However, to create an optimization model for finding a nearly optimum attack, the above formulation is sufficient and, more importantly, easy to model. The only little effort we should make is to evaluate the exact complexity of the discovered attack. This approach is much more practical and faster than creating an optimization model that considers the exact complexity of the attack. Because, creating an optimization model that considers the exact complexity of the ID attack yields a highly complex/heavy model that is not solvable in a reasonable time and still likely returns the same solution as the simple model, even if it can be solved in a reasonable time.

2.2 Integral Attacks and ZC Attacks

The idea of integral distinguishers was first introduced as a theoretical extension of differential distinguishers by Lai in [Lai94]. Later on, Daemen et al. in [DKR97] demonstrated it as a practical attack. Knudsen and Wagner further formalized this concept in [KW02]. The central concept behind integral distinguishers is to find inputs where their corresponding outputs sum up to zero (or generally key-independent value) in certain bit/cell positions. ZC attacks [BR14] are essentially the dual of the ID attack when it comes to linear analysis. They rely on a linear approximation with zero correlation to distinguish the block ciphers from a random permutation. At CRYPTO 2015, Sun et al. [SLR⁺15] proposed **Theorem 1**, according to which a ZC linear hull for block ciphers defined over \mathbb{F}_2^n always yield an integral distinguisher.

Theorem 1 (Sun et al. [SLR⁺15]). *Let $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ be a vectorial Boolean function. Assume A is a subspace of \mathbb{F}_2^n and $\beta \in \mathbb{F}_2^n \setminus \{0\}$ such that (α, β) is a ZC approximation for any $\alpha \in A$. Then, for any $\lambda \in \mathbb{F}_2^n$, $\langle \beta, F(x + \lambda) \rangle$ is balanced over the set*

$$A^\perp = \{x \in \mathbb{F}_2^n \mid \forall \alpha \in A : \langle \alpha, x \rangle = 0\}.$$

Based on [Theorem 1](#), the data complexity of the integral distinguisher derived from a ZC linear hull is 2^{n-m} , where n stands for the block size and m is the dimension of the linear space created by the input linear masks in the corresponding ZC linear hull.

In ToSC 2019, Ankele et al. [[ADG⁺19](#)] explored the impact of the tweakkey on ZC distinguishers of tweakable block ciphers (TBCs). Their findings revealed that considering the tweakkey schedule can lead to a longer ZC distinguisher and, consequently, a longer integral distinguisher. They introduced [Theorem 2](#), providing an algorithmic approach to finding ZC linear hulls for TBCs based on the super-position tweakkey (STK) construction of the TWEAKEY framework [[JNP14](#)] (see [Figure 2](#))

Theorem 2 (Ankele et al. [[ADG⁺19](#)]). *Let $E_K(T, P) : \mathbb{F}_2^{t \times n} \rightarrow \mathbb{F}_2^n$ be a TBC following the STK construction. Assume that the tweakkey schedule of E_K has z parallel paths and applies a permutation h on the tweakkey cells in each path. Let (Γ_0, Γ_r) be a pair of linear masks for r rounds of E_K , and $\Gamma_1, \dots, \Gamma_{r-1}$ represents a possible sequence for the intermediate linear masks. If there is a cell position i such that any possible sequence $\Gamma_0[i], \Gamma_1[h^{-1}(i)], \Gamma_2[h^{-2}(i)], \dots, \Gamma_r[h^{-r}(i)]$ has at most z linearly active cells, then (Γ_0, Γ_r) yields a ZC linear hull for r rounds of E .*

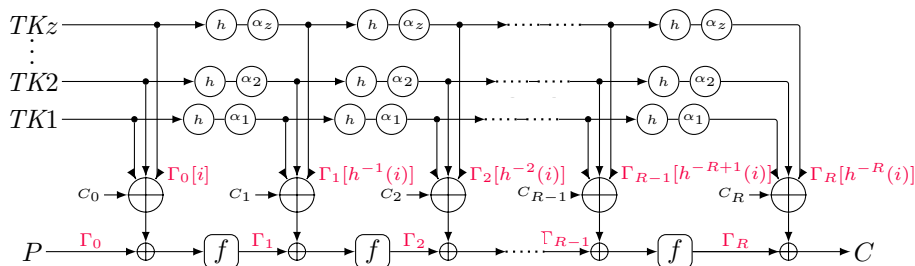


Figure 2: The STK construction of the TWEAKEY framework [[HSE23](#)].

Using [Theorem 2](#) Ankele et al. manually discovered ZC linear hulls for several tweakable block ciphers including SKINNY, and MANTIS [[BJK⁺16](#)]. At EUROCRYPT 2023, Hadipour et al. [[HSE23](#)], introduced a new CP/MILP modeling to find ZC linear hulls for TBCs based on [Theorem 2](#), and significantly improved the ZC and integral attacks on all variants of SKINNY and other tweakable block ciphers.

2.3 Partial-Sum Key Recovery

The partial-sum technique optimizes key recovery in integral attacks [[FKL⁺00](#)]. Assume that we are provided with an integral distinguisher for a certain number of rounds of a block cipher. The distinguishing property of the (zero-sum) integral distinguishers is that the sum (XOR) of the distinguisher’s output should be zero for certain cell/bit positions, which are typically referred to as *balanced* positions. In the key recovery of integral attacks, we add a few rounds after the distinguisher. Subsequently, starting from the ciphertexts, we guess the involved key bits and partially decrypt the ciphertexts to obtain the value of balanced positions at the end of the distinguisher. Then, we verify if these values collectively sum to zero. If so, we keep the corresponding guessed key as a candidate; otherwise, we discard it.

In the partial-sum technique, instead of simultaneously guessing all the key bits, we break down the partial decryption into steps. At each step, we guess only a portion of the involved key bits while storing intermediate values. We continue this process until we reach the output of the distinguisher. This way, we can use memory in favor of time and reduce the time complexity of the key recovery. Note that at each step, we only need to derive a portion of the internal state, whose values are necessary to compute the sum of

the balanced positions at the output of the distinguisher. We refer to these intermediate cell/bit positions as *involved cell/bit positions*.

Recall $\bigoplus_{i=0}^{n-1} x = x$ if n is odd; otherwise, $\bigoplus_{i=0}^{n-1} x = 0$. Thus, values that appear an even number of times do not contribute to the final sum. Therefore, to calculate the final sum at the balanced positions, we only need to know whether each value appears an odd or even number of times. To achieve this, we use an array of binary values whose indices correspond to all possible values of the involved cells/bits at each step, and whose values indicate the parity of occurrences for each possible value (1 for odd occurrences and 0 for even occurrences). The primary advantage is that the number of involved cell/bit positions diminishes as we approach the output of the distinguisher. After a certain point, the count of possible values for the involved cells/bits falls below the count of actual intermediate values for all ciphertexts. Here, we switch from tracking parity across all ciphertexts to tracking parity for each possible intermediate value, reducing memory usage.

Example. To show how the partial-sum technique works, we explain the integral attack on 6 rounds of AES based on a 4-round distinguisher [FKL⁺00,DKR97]. As illustrated in Figure 3a, we assume the internal state is arranged column-wise in a 4×4 array of bytes, with the last round not including MixColumns. Also, K_0 is the initial whitening key, and K_5, K_6 are the 5th and 6th round keys, respectively. To derive the 4-round integral distinguisher, we encrypt a set of 2^{32} plaintexts where all bytes are equal except for the main diagonal, which takes all possible values exactly once. Then, the sum of outputs after 4 rounds is zero. As depicted in Figure 3a, for the 5th round, we aim to retrieve the equivalent subkey \bar{K}_5 , the subkey that is derived by transforming the original subkey before the MixColumns. According to Figure 3a, the relation between the targeted balanced position $C_4[0]$ and the ciphertexts is as follows:

$$\begin{aligned} C_4[0] = & \mathcal{S}^{-1} (\bar{K}_5[0] \oplus 0E \cdot \mathcal{S}^{-1} (C_6[0] \oplus K_6[0]) \oplus 09 \cdot \mathcal{S}^{-1} (C_6[7] \oplus K_6[7]) \\ & \oplus 0D \cdot \mathcal{S}^{-1} (C_6[10] \oplus K_6[10]) \oplus 0B \cdot \mathcal{S}^{-1} (C_6[13] \oplus K_6[13])), \end{aligned} \quad (5)$$

where \mathcal{S} is the S-box, and multiplications are performed in the finite field \mathbb{F}_{2^8} . As shown in Equation 5, 40 key bits are involved. Checking the integral property in a balanced position provides an 8-bit filter. So, we check 6 sets of 2^{32} plaintext to retrieve the key bits. Hence, the time complexity of naive key recovery is $6 \cdot 2^{32} \cdot 2^{40} \approx 2^{74.58}$ partial decryptions.

Now we explain how to optimize the key recovery using the partial-sum technique [FKL⁺00]. We denote $0E \cdot \mathcal{S}^{-1}(\cdot)$ by $\mathcal{S}_0(\cdot)$, $09 \cdot \mathcal{S}^{-1}(\cdot)$ by $\mathcal{S}_1(\cdot)$, $0D \cdot \mathcal{S}^{-1}(\cdot)$ by $\mathcal{S}_2(\cdot)$, and $0B \cdot \mathcal{S}^{-1}(\cdot)$ by $\mathcal{S}_3(\cdot)$. Note that $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2$, and \mathcal{S}_3 can be implemented as lookup tables. Additionally, we need to store 2^{32} ciphertexts indexed as C^i , $0 \leq i < 2^{32}$.

As illustrated in Figure 3b, we first guess $K_6[0, 7]$ and compute $p_1 = \mathcal{S}_0(C_6[0] \oplus K_6[0]) \oplus \mathcal{S}_1(C_6[7] \oplus K_6[7])$ for all ciphertexts. We store how often each value of $(p_1, C_6[10, 13])$ occurs among all ciphertexts. This step is denoted by **count** in Figure 3b. To do so, we create a list \mathcal{L}_1 of 2^{24} bits whose indices are all possible values for $(p_1, C_6[10, 13])$ and whose entries represent whether the corresponding value appears an odd (marked by 1) or even (marked by 0) number of times. The complexity of computing \mathcal{L}_1 for all possible values of $K_6[0, 7]$ is $2^{16} \cdot 2^{32} = 2^{48}$ S-box lookups. The memory complexity of this step is 2^{24} bits.

In the second step, as Figure 3b shows, we guess $K_6[10]$, and compute $p_2 = p_1 \oplus \mathcal{S}_2(C_6[10] \oplus K_6[10])$ for all $(p_1, C_6[10, 13])$ such that $\mathcal{L}_1[(p_1, C_6[10, 13])] = 1$. We create a list \mathcal{L}_2 of 2^{16} bits such that $\mathcal{L}_2[(p_2, C_6[13])]$ represents the occurrence parity of $(p_2, C_6[13])$. We compute \mathcal{L}_2 for all possible values of $K_6[0, 7, 10]$ and also for all indices of \mathcal{L}_1 whose corresponding value is 1. Hence, the time complexity of this step is at most 2^{48} S-box lookups, and the memory complexity is 2^{16} bits.

In the third step, we guess $K_6[13]$ and compute the parity of occurrences of $p_3 = p_2 \oplus \mathcal{S}_3(C_6[13] \oplus K_6[13])$ for all $(p_2, C_6[13])$ such that $\mathcal{L}_2[(p_2, C_6[13])] = 1$. We store the

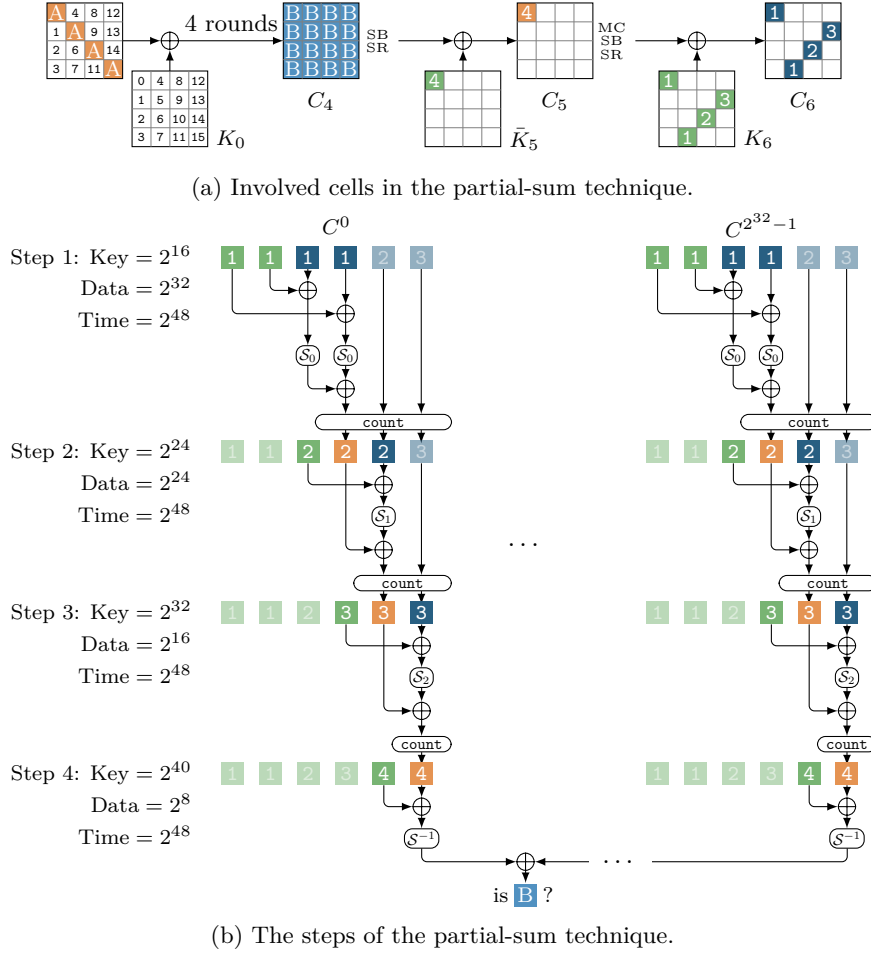


Figure 3: Overview of the partial-sum technique applied on AES. Inspired by [Jea16].

parity of occurrences of p_3 in a list \mathcal{L}_3 of 2^8 bits. Therefore the cost of computing \mathcal{L}_3 for all possible values of $K_6[0, 7, 10, 13]$ is at most 2^{48} S-box lookups. The memory complexity of this step is 2^8 bits.

Finally, we guess $\bar{K}_5[0]$, and compute $\bigoplus_{p_3 \in \mathbb{F}_2^8: \mathcal{L}_3[p_3]=1} \mathcal{S}^{-1}(\bar{K}_5[0] \oplus p_3)$. If the result is zero, we add $\bar{K}_5[0] || K_6[0, 7, 11, 13]$ to the set of key candidates; otherwise, we discard it. Algorithm 1 briefly describes the entire procedure. The total complexity for a single set of 2^{32} plaintexts is $2^{16} \cdot (2^{32} + 2^8 \cdot (2^{24} + 2^8 \cdot (2^{16} + 2^8 \cdot 2^8))) = 2^{50}$ S-box lookups. Since checking one balanced byte provides an 8-bit filter and 40 bits of the key are involved, we need to repeat the whole procedure for 6 different sets of 2^{32} plaintexts to retrieve the involved key bits uniquely. Therefore, the total time complexity is about $6 \cdot 2^{50} \approx 2^{52.58}$ S-box lookups. One AES encryption employs about 2^8 S-box lookups. As a result, the complexity of the partial-sum technique is equivalent to about $2^{44.58}$ AES encryptions. Storing 2^{32} ciphertexts dominates the memory complexity, which amounts to approximately 2^{32} 128-bit blocks. Data complexity is also $6 \cdot 2^{32} \approx 2^{34.58}$ chosen plaintexts. We can similarly retrieve other key bits by using the other balanced positions.

Algorithm 1: Partial-sum key recovery attack on 6 rounds of AES

Input: 2^{32} ciphertexts, and the set of discarded keys \mathcal{DK} (empty in the first run)
Output: A set of candidates \mathcal{K} for $\bar{K}_5[0]||K_6[0, 7, 10, 13]$

```

1  $\mathcal{K} \leftarrow \emptyset$ ;
2 forall  $K_6[0, 7]$  do
3   Initialize a list  $\mathcal{L}_1$  of size  $2^{24}$  with zeros;
4   forall  $2^{32}$  ciphertexts do
5      $p_1 \leftarrow \mathcal{S}_0(C_6[0] \oplus K_6[0]) \oplus \mathcal{S}_1(C_6[7] \oplus K_6[7])$ ;
6      $\mathcal{L}_1[(p_1, C_6[10, 13])] \leftarrow \mathcal{L}_1[(p_1, C_6[10, 13])] \oplus 1$ ;
7   forall  $K_6[10]$  do
8     Initialize a list  $\mathcal{L}_2$  of size  $2^{16}$  with zeros;
9     forall  $(p_1, C_6[10, 13])$  s.t.  $\mathcal{L}_1[(p_1, C_6[10, 13])] = 1$  do
10       $p_2 \leftarrow p_1 \oplus \mathcal{S}_2(C_6[10] \oplus K_6[10])$ ;
11       $\mathcal{L}_2[(p_2, C_6[13])] \leftarrow \mathcal{L}_2[(p_2, C_6[13])] \oplus 1$ ;
12    forall  $K_6[13]$  do
13      Initialize a list  $\mathcal{L}_3$  of size  $2^8$  with zeros;
14      forall  $(p_2, C_6[13])$  s.t.  $\mathcal{L}_2[(p_2, C_6[13])] = 1$  do
15         $p_3 \leftarrow p_2 \oplus \mathcal{S}_3(C_6[13] \oplus K_6[13])$ ;
16         $\mathcal{L}_3[p_3] \leftarrow \mathcal{L}_3[p_3] \oplus 1$ ;
17      forall  $\bar{K}_5[0]$  do
18        if  $\bar{K}_5[0]||K_6[0, 7, 10, 13] \notin \mathcal{DK}$  then
19           $\text{Result} \leftarrow \bigoplus_{p_3 \in \mathbb{F}_2^8: \mathcal{L}_3[p_3]=1} \mathcal{S}^{-1}(\bar{K}_5[0] \oplus p_3)$ ;
20          if  $\text{Result} = 0$  then  $\mathcal{K} \leftarrow \mathcal{K} \cup \{\bar{K}_5[0]||K_6[0, 7, 10, 13]\}$ ;
21          else  $\mathcal{DK} \leftarrow \mathcal{DK} \cup \{\bar{K}_5[0]||K_6[0, 7, 10, 13]\}$ 
22 return  $\mathcal{K}$ ;
```

2.4 Constraint Programming

A mathematical problem involving variables and constraints is known as a Constraint Satisfaction Problem (CSP). Formally, a CSP is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, with $\mathcal{X} = \{X_0, X_1, \dots, X_{n-1}\}$ representing the variable set; $\mathcal{D} = \{\mathcal{D}_0, \mathcal{D}_1, \dots, \mathcal{D}_{n-1}\}$ as the domain set, where $X_i \in \mathcal{D}_i$ for $0 \leq i \leq n-1$; and $\mathcal{C} = \{\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_{n-1}\}$ serving as the constraint set. Each constraint $\mathcal{C}_j \in \mathcal{C}$ is a tuple $(\mathcal{S}_j, \mathcal{R}_j)$, where $\mathcal{S}_j = \{X_{i_0}, \dots, X_{i_{k-1}}\} \subseteq \mathcal{X}$, and \mathcal{R}_j is a relation defined on the corresponding domain, that is, $\mathcal{R}_j \subseteq \mathcal{D}_{i_0} \times \dots \times \mathcal{D}_{i_{k-1}}$. A feasible solution is an assignment of values to the variables in a CSP problem that satisfies all constraints.

A constraint optimization problem (COP) is a CSP with an objective function. Searching for the solution of a CSP or COP problem is referred to as constraint programming (CP), and CP solvers are the tools that can solve CSPs and COPs. We use MiniZinc [NSB⁺07] to create a CSP or a COP over integer variables in this paper. MiniZinc can model CSP and COP problems in a high-level and solver-independent way. It converts the model into FlatZinc, a standard language supported by a wide range of CP solvers. We also use Or-Tools [PF] as the CP solver.

2.5 Cell-Wise Model for Deterministic Differential and Linear Trails

Before explaining our new modelings in the following sections, we first recall the cell-wise model in [SGWW20] for deterministic differential trails. The duality relation between differential and linear analysis allows one to derive the corresponding model for linear trails. Thus, we skip the details for linear trails.

To encode deterministic truncated differential trails, we introduce two types of variables. Let $\Delta X = (\Delta X[0], \dots, \Delta X[m-1])$ denote the difference value of the internal state X in an n -bit block cipher E , where $n = m \cdot c$, and $\Delta X[i] \in \mathbb{F}_2^c$ for all $0 \leq i \leq m-1$. We use the variable $\text{AX}[i] \in \{0, \dots, 3\}$ to represent the activity pattern of $\Delta X[i]$, and another variable $\text{DX}[i] \in \{-2, -1, 0, \dots, 2^c - 1\}$ indicates the actual c -bit value of $\Delta X[i]$. We use -2 and -1 for $\text{DX}[i]$ to differentiate between cases where $\Delta X[i]$ can take any nonzero value

and cases where $\Delta X[i]$ can take any value, respectively. These variables should satisfy the following constraints:

$$\text{Link}(\text{AX}[i], \text{DX}[i]) := \begin{cases} \text{if } \text{AX}[i] = 0 \text{ then } \text{DX}[i] = 0 \\ \text{elseif } \text{AX}[i] = 1 \text{ then } \text{DX}[i] > 0 \\ \text{elseif } \text{AX}[i] = 2 \text{ then } \text{DX}[i] = -1 \\ \text{else } \text{DX}[i] = -2 \text{ endif} \end{cases}$$

Following that, we will provide a concise explanation of the propagation rules for deterministic truncated linear trails.

Proposition 1 (XOR). For $F : \mathbb{F}_2^c \rightarrow \mathbb{F}_2^c$, $F(X, Y) = Z$ where $Z = X \oplus Y$, the valid transitions for deterministic truncated linear trails satisfy

$$\text{XOR}(\text{AX}, \text{DX}, \text{AY}, \text{DY}, \text{AZ}, \text{DZ}) := \begin{cases} \text{if } \text{AX} + \text{AY} > 2 \text{ then } \text{AZ} = 3 \wedge \text{DZ} = -2 \\ \text{elseif } \text{AX} + \text{AY} = 1 \text{ then } \text{AZ} = 1 \wedge \text{DZ} = \text{DX} + \text{DY} \\ \text{elseif } \text{AX} = \text{AY} = 0 \text{ then } \text{AZ} = 0 \wedge \text{DZ} = 0 \\ \text{elseif } \text{DX} + \text{DY} < 0 \text{ then } \text{AZ} = 2 \wedge \text{DZ} = -1 \\ \text{elseif } \text{DX} = \text{DY} \text{ then } \text{AZ} = 0 \wedge \text{DZ} = 0 \\ \text{else } \text{AZ} = 1 \wedge \text{DZ} = \text{DX} \oplus \text{DY} \text{ endif} \end{cases}$$

Proposition 2 (Branching). For $F : \mathbb{F}_2^c \rightarrow \mathbb{F}_2^c$, $F(X) = (Y, Z)$ where $Z = Y \oplus X$, the valid transitions for deterministic truncated linear trails satisfy

$$\text{XOR}(\text{AX}, \text{DX}, \text{AY}, \text{DY}, \text{AZ}, \text{DZ}) := (\text{AZ} = \text{AX} \wedge \text{DZ} = \text{DX} \wedge \text{DY} = \text{DX} \wedge \text{DY} = \text{DX})$$

Proposition 3 (S-box). Assume that $S : \mathbb{F}_2^c \rightarrow \mathbb{F}_2^c$ is a c -bit bijective S-box and $Y = S(X)$. The valid transitions for deterministic truncated linear trails satisfy

$$\text{S-box}(\text{AX}, \text{AY}) := (\text{AY} \neq 1 \wedge \text{AX} + \text{AY} \in \{0, 3, 4, 6\} \wedge \text{AY} \geq \text{AX} \wedge \text{AY} - \text{AX} \leq 1)$$

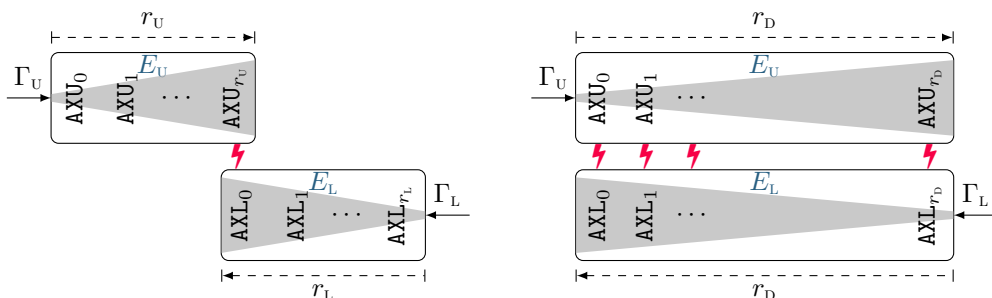
For modeling more complex operations, e.g., the non-MDS matrix employed in SKINNY, we can use the rules of XOR and branching to encode the propagation as in Appendix B.

3 Modeling the Distinguishers

In this section, we present our improvements to the CSP model presented in [HSE23] for identifying ID, ZC, and integral distinguishers.

3.1 Improved Cell-Wise Model to Find Distinguishers

We start by explaining how we relax the requirement for the contradiction to occur at a specific fixed meeting point. In the CSP model introduced in [HSE23], the distinguisher is divided into two parts. For this, it is necessary to specify the distinguisher using two integer numbers, i.e., r_U , and r_L as depicted in Figure 4a, which determine the length of each segment. Following this, deterministic truncated trails are propagated in opposing directions, moving towards the designated meeting point. By introducing certain constraints for this meeting point, one can ensure that there is an inconsistency between the two opposite propagations, creating a contradiction. In our CSP model, we refrain from splitting the distinguisher into two parts. Instead, as illustrated in Figure 4b, we encode the propagation of deterministic trails in opposite directions across the entire distinguisher. Subsequently, we introduce additional constraints to ensure inconsistency between the two propagations at least at one point throughout the entire distinguisher. This adjustment enables the CSP model to automatically determine the optimal location for the contradiction, thereby optimizing the objective function.



(a) Modeling the distinguishers in [HSE23].

(b) Our modeling of the distinguishers.

Figure 4: Modeling the ID, ZC, and integral distinguishers as a CSP problem.

We explain our modeling for ZC and integral distinguishers, but the same approach applies to ID distinguishers. Assume that we aim to find a distinguisher for r_D rounds of a block cipher E . We also assume that the block size of E is n bits, where $n = m \cdot c$ and c represents the cell size. As represented by Figure 4b, we define integer variables AXU_r (resp. AXL_r) to represent the activeness pattern of the internal state after r rounds in the forward direction (resp. background direction). Similarly, we also define the integer variables LXU_r (resp. LXL_r) to represent the actual values of linear masks. Next, as represent by Figure 4b, we model the propagation of the deterministic trails in forward and backward direction over r_D rounds independently. We add the constraints $\sum_{i=0}^{m-1} AXU_0[i] \neq 0$, and $\sum_{i=0}^{m-1} AXL_{r_D}[i] \neq 0$ to exclude all trivial solutions in the single-tweakey setting¹. Let $CSP_U(AXU_0, LXU_0, \dots, AXU_{r_D}, LXU_{r_D})$ and $CSP_L(AXL_0, LXL_0, \dots, AXL_{r_D}, LXL_{r_D})$ denote the CSP models for the forward and backward propagation, respectively. Now we add the following constraints to ensure the inconsistency between the two deterministic propagations at least one point throughout the distinguisher:

$$CSP_M := \bigvee_{i=0}^{r_D} \left(\bigvee_{j=0}^{m-1} \left((AXU_i[j] + AXL_i[j] < 3) \wedge \begin{matrix} AXU_i[j] \neq AXL_i[j] \\ LXU_i[j] \neq LXL_i[j] \end{matrix} \right) \right) \vee \bigvee_{j=0}^{m-1} \left(\begin{matrix} AXU_i[j] = 1 \wedge \\ AXL_i[j] = 1 \wedge \\ LXU_i[j] \neq LXL_i[j] \end{matrix} \right) \quad (6)$$

The conjunction of the CSP models above, i.e., $CSP_D = CSP_U \wedge CSP_M \wedge CSP_L$, creates a unified CP or MILP model based on satisfiability, whose all feasible solutions are the ZC or integral distinguishers. Hence, when provided with only an integer value r_D , this model yields a distinguisher for r_D rounds of the block cipher, eliminating the need to specify the location of the meeting point (contradiction). We can also include an objective function, e.g., $\max \left(\sum_{i=0}^{m-1} AXU_0[i] + \sum_{i=0}^{m-1} AXL_{r_D}[i] \right)^2$, to maximize the number of differentially (linearly) active cells at the input and output. This objective function is useful for finding integral distinguishers with minimum data complexity or ID/ZC distinguishers with maximum multiplicity.

We can also extend our idea to automatically find integral distinguishers using Theorem 2. Let E be a tweakable block cipher following the STK framework, as represented in Figure 2. Assume that E has z parallel independent paths in the tweakey schedule that apply the permutation h to shuffle the position of cells in each path. Additionally, let $STK_r[i]$ denote the i th cell of the subtweakey after r rounds. We define an integer variable $ASTK_r[i] \in \{0, 1, 2, 3\}$ to encode the activation pattern of $STK_r[i]$. By the way, let $AYU_r[i]$ (resp. $AYL_r[i]$) encode the activation pattern of the internal state at the subtweakey addition in round r in forward (resp. backward) propagation. Now, we add the new

¹In the related-tweakey setting, we ensure that the difference in $\Delta P || \Delta TK$ and ΔC are not zero.

²The second summation is not present when searching for the integral distinguishers.

constraint $\text{ASTK}_r[i] = \min\{\text{AYU}_r[i], \text{AYL}_r[i]\}$ for all $0 \leq r \leq r_D - 1$, and $0 \leq i \leq m - 1$, to link the activeness pattern of the subkey to the activeness pattern of the internal state. This simple constraint causes the subkey to follow the activeness pattern in one of the forward or backward propagations with less active cells. Finally, we add the following constraint to ensure the conditions of [Theorem 2](#) are met:

$$\text{CSP}_{TK}(\text{ASTK}_0, \dots, \text{ASTK}_{r_D-1}) := \bigvee_{i=0}^{m-1} \left(\left(\sum_{r=0}^{r_D-1} \text{bool2int}(\text{ASTK}_r[h^{-r}(i)] \neq 0) \leq z \right) \vee \left(\bigwedge_{r=0}^{r_D-1} \text{ASTK}_r[h^{-r}(i)] = 0 \right) \right) \quad (7)$$

Consequently, we do not need to define the border between the forward and backward propagations, as the model automatically finds the best configuration.

Before explaining our following enhancement, we would like to highlight some insightful discoveries we made using only the CP model described above. At FSE 2020, Bariant et al. conducted an extensive manual analysis on ForkSKINNY-128-256 [[BDL20](#)]. They demonstrated that one of the ID distinguishers of SKINNY-128-256 could be extended by up to 5 rounds for a particular reduced-round version of ForkSKINNY-128-256 (where $\mathbf{r}_0 = 27$). However, their analysis was based solely on the existing ID distinguishers of SKINNY. Furthermore, they did not explore various variants of ForkSKINNY (e.g., ForkSKINNY-64-192, and ForkSKINNY-128-288) due to the tedious and time-consuming nature of identifying ID distinguishers/attacks using the previous automated tools. They also deferred the analysis of more significant reduced-round versions of ForkSKINNY where $\mathbf{r}_0 = \mathbf{r}_1$ to future work.

Thanks to the efficiency of our new CP/MILP modeling for distinguishers, we identified longer ID distinguishers for many versions of ForkSKINNY, whether $\mathbf{r}_0 = \mathbf{r}_1$ or $\mathbf{r}_0 \neq \mathbf{r}_1$. [Figure 28](#) and [Figure 27](#) illustrate some of these distinguishers. Moreover, we applied our modeling to find integral distinguishers for ForkSKINNY. We discovered several integral distinguishers for reduced-round versions of ForkSKINNY that are one round longer than the integral distinguishers of SKINNY. [Figure 26](#) and [Figure 25](#) illustrate one of our 17-round (for ForkSKINNY-64-192) and 15-round (for ForkSKINNY-128-256) integral distinguishers. Our results show that ForkSKINNY can be generally weaker than SKINNY against the integral attack for some $(\mathbf{r}_i, \mathbf{r}_0, \mathbf{r}_1)$.

As another interesting application, we used our cell-wise model to find integral distinguishers for QARMAv2-64 and QARMAv2-128. The designers of QARMAv2 managed to find a 5-round integral distinguisher for QARMAv2-64 by using the division property. For QARMAv2-128, they could not apply the division property due to the challenges posed by its large block size and complex MILP/SAT models. With our CP/MILP model, we identified several 12-round (for QARMAv2-128) and 10-round (for QARMAv2-64) integral distinguishers. Additionally, we found multiple 11-round integral distinguishers for QARMAv2-128 with the data complexity of only 2^{44} , which is lower than the threshold of 2^{80} for data complexity of this version. Note that our integral distinguishers do not threaten the security of QARMAv2 as it has enough security margin against our distinguishers.

We point out that the cell-wise model proposed in [[HSE23](#)] can theoretically also find the integral distinguishers we discovered for QARMAv2 and ForkSKINNY, but it requires several runs for several predefined contradiction locations to find these distinguishers. However, our improved cell-wise model can find the distinguisher in just one run and automatically locates the positions of contradictions to optimize the objective function. One might be concerned that adding the new inconsistency checkers in [Equation 7](#) could increase the solving time of the CP model, compared to solving one CP model based on [[HSE23](#)]. However, in all our implementations, including these new inconsistency checkers in our model has only a minor effect on solving time. The CP model remains solvable in seconds on a standard laptop. For example, in the case of a cipher like ForkSKINNY, where the fork point is also a critical parameter, the model in [[HSE23](#)] would need to be run for

many different configurations before finding the optimal attack, which can be inefficient. In contrast, our model can find the optimal distinguisher in just one run, taking only a few seconds on a standard laptop. As a result, our new cell-wise model enhances the one from [HSE23] by eliminating the need to specify contradiction locations and reducing the effort required to discover the optimal ID, ZC, or integral distinguishers.

3.2 Bit-Wise CP Model to Find ID, ZC and Integral Distinguishers

We now explain our second enhancement to the distinguisher modeling in [HSE23], which is to create a bit-wise model for searching ID, ZC, and integral distinguishers while maintaining a satisfiability-based approach. This model takes advantage of the internal structure of S-boxes and is more appropriate for weakly aligned primitives, such as Ascon.

In our bit-wise model, for each bit of the internal state, we define an integer variable with the domain of $\{-1, 0, 1\}$ to indicate whether the difference (linear mask) is unknown, zero, or one, respectively. Then, we impose some constraints to model the deterministic propagation of differential (linear) trails at the level of bits. For this aim, we first define some simple rules for deterministic propagations through the building block operations, i.e., XOR, Branching, and S-boxes or generally a vectorial Boolean function. We explain our modeling for deterministic differential trails; the same approach applies to linear trails.

Proposition 4 (Branching-Bitwise). *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*

$$\text{Branch}_b(\text{AX}, \text{AY}[0], \dots, \text{AY}[n-1]) := \bigwedge_{i=0}^{n-1} (\text{AY}[i] = \text{AX}),$$

where AX , and $\text{AY}[i]$ are integer variables with the domain of $\{-1, 0, 1\}$ for all $0 \leq i \leq n-1$.

Proposition 5 (XOR-Bitwise). *For $f : \mathbb{F}_2 \rightarrow \mathbb{F}_2^n$, $f(x_0, x_1, \dots, x_{n-1}) = y$, where $y = x_0 \oplus x_1 \oplus \dots \oplus x_{n-1}$, the valid deterministic differential transitions satisfy the following:*





$$\text{XOR}_b(\text{AY}, \text{AX}[0], \dots, \text{AX}[n-1]) := \begin{cases} \text{if } \bigvee_{i=0}^{n-1} (\text{AX}[i] = -1) \text{ then } \text{AY} = -1 \\ \text{else } \text{AY} = \text{AX}[0] + \text{AX}[1] + \dots + \text{AX}[n-1] \pmod{2} \text{ endif} \end{cases}$$

where $\text{AX}[i]$, and AY are integer variables with the domain of $\{-1, 0, 1\}$ for all $0 \leq i \leq n-1$.

For the S-boxes, or generally a vectorial Boolean function, we refer to its DDT (resp. LAT) to identify differential (resp. linear) transitions in which the difference (resp. linear mask) is known with certainty for at least one bit. We refer to these transitions as *bit-wise deterministic differential (linear) transitions* and use some simple CP constraints to model them. We recall that the deterministic differential transitions are already used in [Tez14] under the name of *undisturbed bits*. To explain our bit-wise model for the S-boxes, we give a basic example. Take the 4-bit S-box of MANTIS as an example. We define the integer variables $\text{AX}[i]$ and $\text{AY}[i]$ for $0 \leq i \leq 3$ to represent the bit differences at the input and output of this S-box, respectively, where $\text{AX}[0]$ and $\text{AY}[0]$ correspond to left-most bit, i.e., MSB. The domain of these variables is $\{-1, 0, 1\}$, where -1 indicates that the difference is unknown. Referring to the DDT of the S-box, the following constraints can model all valid deterministic differential transitions (see Appendix M):

$$\left\{ \begin{array}{l} \text{if } (\text{AX}[0] = 0 \wedge \text{AX}[1] = 0 \wedge \text{AX}[2] = 0 \wedge \text{AX}[3] = 0) \text{ then } (\text{AY}[0] = 0 \wedge \text{AY}[1] = 0 \wedge \text{AY}[2] = 0 \wedge \text{AY}[3] = 0) \\ \text{elseif } (\text{AX}[0] = 0 \wedge \text{AX}[1] = 0 \wedge \text{AX}[2] = 1 \wedge \text{AX}[3] = 0) \text{ then } (\text{AY}[0] = -1 \wedge \text{AY}[1] = -1 \wedge \text{AY}[2] = 0 \wedge \text{AY}[3] = -1) \\ \text{elseif } (\text{AX}[0] = 0 \wedge \text{AX}[1] = 0 \wedge \text{AX}[2] = -1 \wedge \text{AX}[3] = 0) \text{ then } (\text{AY}[0] = -1 \wedge \text{AY}[1] = -1 \wedge \text{AY}[2] = 0 \wedge \text{AY}[3] = -1) \\ \text{elseif } (\text{AX}[0] = 1 \wedge \text{AX}[1] = 1 \wedge \text{AX}[2] = 0 \wedge \text{AX}[3] = 1) \text{ then } (\text{AY}[0] = -1 \wedge \text{AY}[1] = -1 \wedge \text{AY}[2] = 1 \wedge \text{AY}[3] = -1) \\ \text{elseif } (\text{AX}[0] = 1 \wedge \text{AX}[1] = 1 \wedge \text{AX}[2] = 1 \wedge \text{AX}[3] = 1) \text{ then } (\text{AY}[0] = -1 \wedge \text{AY}[1] = -1 \wedge \text{AY}[2] = 1 \wedge \text{AY}[3] = -1) \\ \text{elseif } (\text{AX}[0] = 1 \wedge \text{AX}[1] = 1 \wedge \text{AX}[2] = -1 \wedge \text{AX}[3] = 1) \text{ then } (\text{AY}[0] = -1 \wedge \text{AY}[1] = -1 \wedge \text{AY}[2] = 1 \wedge \text{AY}[3] = -1) \\ \text{else } (\text{AY}[0] = -1 \wedge \text{AY}[1] = -1 \wedge \text{AY}[2] = -1 \wedge \text{AY}[3] = -1) \text{ endif} \end{array} \right.$$

Using the bit-wise propagation rules above, we can independently model the deterministic differential (linear) trails forward and backward. Similar to our cell-wise modeling, we include some constraints to ensure the inconsistency between the two deterministic propagations at least one point throughout the distinguisher. More precisely, assume that $AXU_r[i]$, (resp. $AXL_r[i]$) represents the difference (linear mask) in the i th bit of internal state in round r in forward (resp. backward) propagation. Then, we include $CSP_M := \bigvee_{r=0}^{r_v-1} \left(\bigvee_{i=0}^{n-1} (AXU_r[i] + AXL_r[i] = 1) \right)$ to ensure that the two deterministic propagations are inconsistent in at least one bit throughout the distinguisher. As a result, the conjunction of CSP models for forward and backward propagations together with CSP_M creates a unified CP or MILP model based on satisfiability, whose all feasible solutions are the ID distinguishers. The same approach applies to ZC and integral distinguishers.

To show the usefulness of our bit-wise model, we applied it to two bit-wise (weakly aligned) designs Ascon, and PRESENT. When searching for ID and ZC distinguishers of Ascon, we include the objective function $\min. \left(\sum_{i=0}^{n-1} AXU_0[i] + \sum_{i=0}^{n-1} AXL_{r_v}[i] \right)$ to maximize the number of unknown bits at the input/output of distinguisher. This way, any model’s solution is indeed a cluster of ID, ZC, or integral distinguishers. The more unknown bits at the input/output of the distinguisher, the more distinguishers it contains. Figure 50, Figure 51, Figure 52, Figure 53, and Figure 54 illustrate some of the 5-round ZC and ID distinguishers discovered by our tool for Ascon, respectively. The unknown bit (in terms of difference value or mask value) in the forward and backward propagations are represented by , and , respectively. In addition, the bit difference (or linear mask) 1 is represented by  and  in forward and backward propagations, respectively. As seen in Figure 50, the contradiction happens in the bit level (at the output of the first round), which is not detectable by a cell-wise model. The result represented in Figure 50 can be derived with our tool running on a regular laptop in a few minutes. However, as seen in Figure 50 it is a cluster of 2^{155} ZC distinguishers. Regarding the ID and ZC distinguisher of PRESENT, we also achieved the best previous results (see Appendix K). This highlights the advantage of our search method over previous works [ST17, CCJ+16], where each ZC or ID distinguisher had to be derived separately when fixing the input/output to specific differences or linear masks in each run.

3.3 Comparing Our Distinguisher Modeling to Previous Methods

We emphasize that, similar to [HSE23], our primary aim with our CP models for distinguishers is to create satisfiability-based models that extend to a unified COP for deriving the complete key-recovery ID, ZC, and integral attacks. Unlike previous tools such as [ST17, CCJ+16], which rely on unsatisfiability and require fixing input/output differences or linear masks to find distinguishers, our models are satisfiability-based, eliminating the need for fixing input/output differences or linear masks. Consequently, both the approach in [HSE23] and our new models are the only methods for ID, ZC, and integral distinguishers that can be extended into a unified CP model for key recovery. While our primary goal in distinguisher modeling is not to create tools for proving the non-existence of ID and ZC distinguishers, one might wonder if our models, especially the bit-wise one, could serve this purpose. We do not assert that our models can be used for proving the non-existence of ID or ZC distinguishers. Nevertheless, both the findings from [HSE23] and our recent results have demonstrated their efficiency in discovering the longest ID and ZC distinguishers for various applications, including SKINNY, ForkSKINNY, SKINNYe, SKINNYe, CRAFT, Deoxys-BC, MANTIS, QARMAv2, Ascon, and PRESENT. For example, our CP model to find an ID distinguisher for 6 (resp. 7) rounds of Ascon (resp. PRESENT) returns UNSATISFIABLE, meaning that there is no ID distinguisher for 6 (resp. 7) rounds of Ascon (resp. PRESENT) under the assumption of our model, e.g., round independence

and round-key independence assumptions³. However, all of the existing tools have yet to discover any 6 (resp. 7) ID distinguisher for Ascon (resp. PRESENT). This is the case in all other applications studied in [HSE23] or this paper.

Note that previous tools such as [ST17, CCJ⁺16] also cannot be directly used to prove the non-existence of ID distinguishers due to their inherent limitation, which involves fixing the input/output differences. To demonstrate the non-existence of ID distinguishers using the methods from [ST17, CCJ⁺16] for a specific number of rounds in a block cipher, one must examine all possible combinations of input/output differences to ensure the presence of at least one differential trail for each input/output difference⁴. However, checking all possible combinations of input/output differences or linear masks is not feasible. Thus, a very few heuristic methods have been proposed to simplify this task, e.g., [HPW22]. These methods primarily involve dividing the space of all possible input/output differences into equivalent classes and then examining only one representative from each class. Nonetheless, this approach remains computationally demanding, particularly for larger block sizes, and often encounters computational complexity challenges, as highlighted in [HPW22], particularly in the *related-(twea)key* setting.

In summary, our new cell-wise and bit-wise models, inspired by [HSE23], bridge the gap in developing a satisfiability-based model that can extend into a unified COP for key recovery in ID, ZC, or integral attacks. Similar to [HSE23], we expand our modeling for near-optimal complete ID, ZC, or integral attacks (demonstrated for SKINNY and ForkSKINNY). We use the same model as in [HSE23] for key recovery of ID attack. However, our key recovery model for integral attacks is novel, as explained in the next section.

4 Modeling the Key Recovery for Integral Attacks

This section introduces a COP model for optimizing the partial-sum key recovery of integral attacks. When combined with the CSP model for integral distinguishers in Section 3, it is possible to construct a unified COP for finding an optimized full integral attack. This unified model receives three integer numbers (r_B, r_D, r_F), corresponding to the lengths of each part in Figure 1, and outputs an optimized integral attack with the corresponding time, memory, and data complexity. Moreover, due to our modular approach, our COP for the partial-sum technique can be used together with other models for distinguishers.

Previous Work If we apply the partial-sum technique to AES as seen in Section 2.3, the order in which the partial sums are calculated does not affect the final complexity. However, this may not be the case for ciphers where each input of the MixColumns operation only affects some cells in the same column of the output, particularly when more rounds are appended to the distinguisher. A straightforward way to optimize the key guessing order is to use the partial-sum reduction at the MixColumns step by guessing the keys column-wise. Previous works [ADG⁺19, HSE23] also optimize which column should be determined first to reduce the overall time complexity. However, they only do this one round at a time.

Improving the partial-sum key guessing order For ciphers like SKINNY, the optimizations in previous works are insufficient for finding the best partial-sum guessing order, as demonstrated by the example in Figure 5. Here, we compare optimizing the partial sums locally, one round at a time, with a global optimization. The time complexity of the local optimization is listed in Table 3. If we apply a global optimization and instead guess

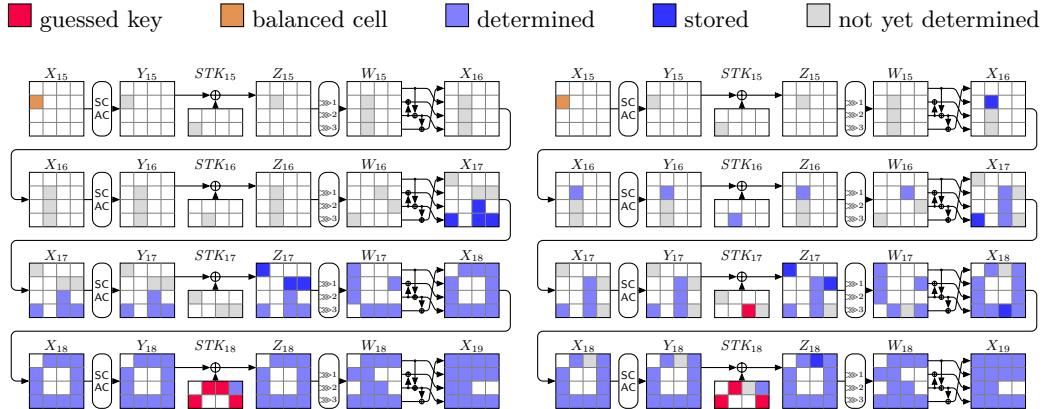
³Note that all existing ID/ZC distinguisher tools rely on these two assumptions, without considering cross-round or round-key dependencies.

⁴This approach fails for ZC distinguishers as multiple linear trails can exist between input and output masks, each with different signed correlations, leading to a total correlation of zero. Currently, there is no systematic method for proving the non-existence of ZC distinguishers.

$STK_{17}[6]$ in step 4, we have to store on partial-sum less, as shown in Figure 5. This reduces the data complexity of step 4 and, as a result, the time complexity of step 5. Because step 5 dominates the time complexity of the key recovery, we significantly decrease the time to recover the key.

Table 3: Complexity of local 4 round partial-sum optimization with balanced cell $X_{15}[4]$.

Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{32} = 2^{32}$	$2^{32-5.2}$	$Z_{18}[1, 2, 4, 7]; X_{18}[8, 11-15]; X_{17}[12]; STK_{16}[5]$
1	$STK_{18}[4]$	$2^4 \times 2^{32} = 2^{36}$	$2^{36-7.2}$	$Z_{18}[1, 2, 7]; X_{18}[11, 13-15]; Z_{17}[0, 7]; X_{17}[10, 12]; STK_{16}[5]$
2	$STK_{18}[1]$	$2^8 \times 2^{32} = 2^{40}$	$2^{40-7.2}$	$Z_{18}[2, 7]; X_{18}[11, 14, 15]; Z_{17}[0, 7]; X_{17}[10, 12, 14]; STK_{16}[5]$
3	$STK_{18}[7]$	$2^{12} \times 2^{32} = 2^{44}$	$2^{44-8.2}$	$Z_{18}[2]; X_{18}[14]; Z_{17}[0, 6, 7]; X_{17}[10, 12, 14]; STK_{16}[5]$
4	$STK_{18}[2]$	$2^{16} \times 2^{32} = 2^{48}$	$2^{48-7.2}$	$Z_{17}[0, 6, 7]; X_{17}[10, 12, 14, 15]; STK_{16}[5]$
5	$STK_{17}[6]$	$2^{20} \times 2^{20} = 2^{40}$	$2^{52-7.2}$	$Z_{17}[0, 7]; X_{17}[12, 15]; X_{16}[5]$
6	$STK_{17}[0]$	$2^{24} \times 2^{16} = 2^{40}$	$2^{44-7.2}$	$Z_{17}[7]; X_{17}[15]; X_{16}[5, 13]$
7	$STK_{17}[7]$	$2^{28} \times 2^4 = 2^{32}$	$2^{44-6.7}$	$X_{15}[4]$
Σ		2^{48}	$2^{44.86}$	



(a) When optimizing the partial-sum guessing order one round at a time, 7 cells need to be stored in step 4.

(b) When optimizing the partial-sum guessing order globally, 6 cells need to be stored in step 4.

Figure 5: Advantage of an advanced guessing order for the partial-sum technique. For ciphers like SKINNY, it is often better to guess some keys in earlier rounds first, instead of guessing one round at a time. In this example, this allows us to store one cell less in the partial-sum recovery step 4, reducing the time and memory complexity by 2^4 .

Making use of this global optimization, our novel approach uses a COP model that takes into account every key guess possible at each step. The main idea is to assign to the cells of each state the step number in which it is guessed. The various internal parts of ciphers affect how these step numbers propagate to the next state.

Overview of the COP Model. Our unified COP model comprises the following modules:

- **Model the distinguisher** as in Section 3.
- **Model the meet-in-the-middle part.** A distinguisher with two or more balanced cells may yield a better attack when using the meet-in-the-middle technique [SW12].

Algorithm 2: CSP model to find which cells are involved in the partial-sum key-recovery of SKINNY

Input: integer numbers (start round R_s , end round R , output of distinguisher AXL_{R_s})
Output: CSP_{IC}

- 1 Declare an empty CSP model \mathcal{M} ;
- 2 $\mathcal{M}.var \leftarrow \{IXF_r[i] \in \{0, 1\} : R_s \leq r \leq R + 1, 0 \leq i \leq 15\}$;
- 3 $\mathcal{M}.var \leftarrow \{IWF_r[i] \in \{0, 1\} : R_s \leq r \leq R, 0 \leq i \leq 15\}$;
- 4 **for** $i = 0, \dots, 15$ **do**
- 5 $\mathcal{M}.con \leftarrow IXF_{R_s}[i] = (\text{if } AXL_{R_s}[i] > 0 \text{ then } 1 \text{ else } 0)$;
- 6 **for** $r = R_s, \dots, R$; $i = 0, \dots, 15$ **do**
- 7 $\mathcal{M}.con \leftarrow IWF_r[i] = IXF_r[ShiftRows[i]]$;
- 8 $\mathcal{M}.con \leftarrow IXF_{r+1}[i] = MixColumnsForwards(IWF_r, i)$;
- 9 **return** \mathcal{M} ;

If that is the case, the following parts of our model are applied to each of the balanced cells, and the overall complexity is the sum of each of these branches.

- **Model which state cells are involved** in the key recovery part.
- **Model the key-schedule.** We take advantage of the linear key-schedule and make sure that we exclude the active sub-(twea)keys from the optimization.
- **Model the step assignment** to define in which order the (twea)key cells are guessed in the key-recovery attack.
- **Model the data usage** of each step. We want to know how many partial sums we have to store per step to model the time complexity.
- **Model the complexity.** The complexity of each step consists of all the key guesses so far, times the data usage of the previous step. The overall complexity we want to minimize is the sum of the complexities of each step. If we use the meet-in-the-middle technique, we also have to sum over all branches.

4.1 Detailed model for SKINNY

In this section, we explain our model in more detail using the cipher SKINNY, for which the specification can be found in [Appendix A](#). We combine this key recovery COP model with the distinguisher model of [Section 3](#) to create a unified model. However, for simplicity, we describe just the partial-sum optimization given the position of the balanced cell AXL_{R_s} and the active input cells AI from the distinguisher. Furthermore, the tweakkey setting z , the start ($R_s = r_B + r_D$) and end ($R = r_B + r_D + r_F$) round of the key recovery, and the active tweakkey AT need to be specified.

Model the Involved State Cells First, we want to model which state cells are involved in the partial-sum key recovery in [Algorithm 2](#). For this, we create two Boolean arrays IXF, IWF representing the cells of state X and W in each round, respectively (see [Figure 10](#)). If a cell at index i equals True, then this cell is involved in the key recovery. We set IXF_{R_s} to *True* (1) at the balanced positions; otherwise it takes *False* (0). Then, using the properties of MixColumns and the permutation *ShiftRows*, we deduce the involvement of each cell until we reach the last round. *MixColumnsForwards* models the inverse MixColumns matrix M^{-1} (see [Appendix A](#)):

$$MixColumnsForwards(IWF, i) := \begin{cases} \text{if } i < 4 \text{ then } IWF[i + 12] \\ \text{elseif } i < 8 \text{ then } IWF[i - 4] \vee IWF[i] \vee IWF[i + 4] \\ \text{elseif } i < 12 \text{ then } IWF[i - 4] \\ \text{else } IWF[i - 8] \vee IWF[i - 4] \vee IWF[i] \text{ endif.} \end{cases}$$

Algorithm 3: CSP model of properties of the tweakey schedule of SKINNY

Input: $CSP_{IC.var}$, integer numbers (start round R_s , end round R , tweakey setting z , active tweakey AT)

Output: CSP_{TW}

- 1 Declare an empty CSP model \mathcal{M} ;
- 2 $\mathcal{M}.var \leftarrow \{STKC[i] \in \{0, \dots, (R - R_s)/2 + 1\} : 0 \leq i \leq 15\}$;
- 3 $\mathcal{M}.var \leftarrow \{ISKF_r[i] \in \{-1, \dots, 15\} : R_s \leq r \leq R, 0 \leq i \leq 7\}$;
- 4 $\mathcal{M}.var \leftarrow \{STKIR_r[i] \in \{-1, \dots, 15\} : R_s \leq r \leq R, 0 \leq i \leq 15\}$;
- 5 $\mathcal{M}.con \leftarrow STKC[i] = \sum_{r=R_s}^R \sum_{j=0}^7 IXF_r[j] = 1 \wedge KF_r[j] = i$;
- 6 **for** $r = R_s, \dots, R$; $i = 0, \dots, 7$ **do**
- 7 **if** $IXF_r[i] = 1 \wedge stkp_r[i] \neq AT$ **then**
- 8 **if** $STKC[stkp_r[i]] \leq z$ **then**
- 9 $\mathcal{M}.con \leftarrow ISKF_r[i] = stkp_r[i]$;
- 10 **else**
- 11 $\mathcal{M}.con \leftarrow ISKF_r[i] = stkp_r[i] \vee ISKF_r[i] = -1$;
- 12 **else**
- 13 $\mathcal{M}.con \leftarrow ISKF_r[i] = -1$;
- 14 **for** $i = \{0, \dots, 15\} \setminus AT$ **do**
- 15 $\mathcal{M}.con \leftarrow z \geq \sum_{r=R_s}^R \sum_{j=0}^7 ISKF_r[j] = i$;
- 16 **for** $i = 0, \dots, 15$ **do**
- 17 **if** $STKC[i] > z \wedge i \neq AT$ **then**
- 18 **for** $r = R_s, \dots, R$ **do**
- 19 **if** $i \in_j ISKF_r$ **then**
- 20 $\mathcal{M}.con \leftarrow STKIR_r[i] = KF_r[j]$;
- 21 **else**
- 22 $\mathcal{M}.con \leftarrow STKIR_r[i] = -1$;
- 23 **else**
- 24 $\mathcal{M}.con \leftarrow STKIR_{R_s, \dots, R}[i] = -1$;
- 25 **for** $r = R_s, \dots, R$; $i = 0, \dots, 7$ **do**
- 26 **if** $IXF_r[i] = 1 \wedge stkp_r[i] \neq AT \wedge stkp_r[i] \neq ISKF_r[i]$ **then**
- 27 $\mathcal{M}.con \leftarrow STKIR_{R_s, \dots, R}[stkp_r[i]] < XF_r[i]$;
- 28 **return** \mathcal{M} ;

Model the Key-Schedule In SKINNY, each subtweakey is linearly derived from the z tweakey cells. If we already determined z subtweakeys derived from the same cell position of the tweakeys, we can recover this part of the tweakey. Furthermore, in our integral attacks on SKINNY- $n-z \cdot n$, z cells of the tweakey are known and under the attacker's control. Thus, only the other positions ($15 \cdot z$ cells) must be recovered.

In Algorithm 3, we ensure that the active tweakey cell is not considered in the partial-sum optimization and that at most z subtweakeys per tweakey position have to be guessed. The precomputed input $stkp$ maps each subtweakey position to a tweakey position. In STKC, we store how often a tweakey cell is involved in the forward propagation done in Algorithm 2. Based on the conditions mentioned in the previous paragraph, we store which subtweakey positions are involved in the key recovery in ISKF. If in STKC a certain tweakey position is guessed more than z times, we store the rounds in which it is guessed in STKIR.

Model the Step Assignment The main part of our model assigns a step number between 0 and the maximum number of involved tweakey cells S to each state cell (Algorithm 4). On the subtweakey state, KF, this number represents in which step we want to guess this subtweakey. The other states XF, ZF, WF are used to propagate the step numbers backward. Uninvolved cells are represented by -1 . First, we assign to each involved cell in the last round the step number 0. We then trace the step assignment backward until we reach the output of the distinguisher. An example step assignment is illustrated in Figure 6. All states not depending on each other can have the same step number. This makes it possible that multiple tweakeys are guessed at the same time.

In detail, we first model that the step number in WF is always the maximum step number of the cells it depends on in XF. By looking at the dependencies in M^{-1} , we model *MixColumnsBackwards* as

$$\text{MixColumnsBackwards}(\text{XF}, i) := \begin{cases} \text{if } i < 4 \text{ then } \text{XF}[i + 4] \\ \text{elseif } i < 8 \text{ then } \max\{\text{XF}[i], \text{XF}[i + 4], \text{XF}[i + 8]\} \\ \text{elseif } i < 12 \text{ then } \max\{\text{XF}[i - 4], \text{XF}[i + 4]\} \\ \text{else } \max\{\text{XF}[i - 12], \text{XF}[i]\} \text{ endif.} \end{cases}$$

Afterward, the step numbers in ZF are the values in WF shifted by *ShiftRows*. Each new subtweakey guess corresponds to a step. Therefore, we impose the condition that KF is larger than ZF.

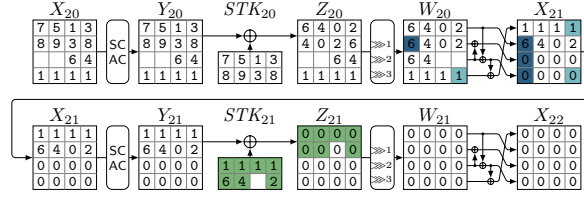


Figure 6: Example of a step assignment to the various states. We highlight how the subtweakey addition and the MixColumns operation affects the step number.

Algorithm 4: CSP model for the step assignment of each state of SKINNY

Input: $\text{CSP}_{IC}.\text{var}$, $\text{CSP}_{TW}.\text{var}$, integer numbers (start round R_s , end round R , maximum step number S)

Output: CSP_{SA}

- 1 Declare an empty CSP model \mathcal{M} ;
- 2 $\mathcal{M}.\text{var} \leftarrow \{\text{XF}_r[i] \in \{-1, \dots, S-1\} : R_s \leq r \leq R, 0 \leq i \leq 15\}$;
- 3 $\mathcal{M}.\text{var} \leftarrow \{\text{KF}_r[i] \in \{-1, \dots, S-1\} : R_s \leq r \leq R, 0 \leq i \leq 7\}$;
- 4 $\mathcal{M}.\text{var} \leftarrow \{\text{ZF}_r[i] \in \{-1, \dots, S-1\} : R_s \leq r \leq R, 0 \leq i \leq 15\}$;
- 5 $\mathcal{M}.\text{var} \leftarrow \{\text{WF}_r[i] \in \{-1, \dots, S-1\} : R_s \leq r \leq R, 0 \leq i \leq 15\}$;
- 6 **for** $i = 0, \dots, 15$ **do**
- 7 $\mathcal{M}.\text{con} \leftarrow \text{XF}_{R+1}[i] = \text{IXF}_{R+1}[i] - 1$ //Assign either 0 or -1;
- 8 **for** $r = R_s, \dots, R$; $i = 0, \dots, 15$ **do**
- 9 **if** $\text{IWF}_r[i] = 1$ **then**
- 10 $\mathcal{M}.\text{con} \leftarrow \text{WF}_r[i] = \text{MixColumnsBackwards}(\text{XF}_{r+1}, i)$;
- 11 **else**
- 12 $\mathcal{M}.\text{con} \leftarrow \text{WF}_r[i] = -1$;
- 13 $\mathcal{M}.\text{con} \leftarrow \text{ZF}_r[\text{ShiftRows}[i]] = \text{WF}_r[i]$;
- 14 **if** $i < 8$ **then**
- 15 **if** $\text{ISKF}_r[i] \geq 0$ **then**
- 16 $\mathcal{M}.\text{con} \leftarrow (\text{KF}_r[i] > \text{ZF}_r[i]) \wedge (\text{XF}_r[i] = \text{KF}_r[i])$;
- 17 **else**
- 18 $\mathcal{M}.\text{con} \leftarrow (\text{KF}_r[i] = -1) \wedge (\text{XF}_r[i] = \text{ZF}_r[i])$;
- 19 **else**
- 20 $\mathcal{M}.\text{con} \leftarrow \text{XF}_r[i] = \text{ZF}_r[i]$;
- 21 **return** \mathcal{M} ;

Model the Data Usage of Each Step To calculate the time and data complexity of the partial-sum technique, we first need to calculate how much data needs to be stored at each step. This is done in [Algorithm 5](#). The number of subtweakeys determined at each step is stored in STKPS. Furthermore, the variables ZC and XC store how many partial sums have to be stored per step in the state Z and X, respectively. In SBC, we store how many S-Boxes are passed in each step. Lastly, in ASTKC, we store whether an active tweakey cell

Algorithm 5: CSP model for data use of the partial-sum technique on SKINNY

Input: $CSP_{IC}.var$, $CSP_{TW}.var$, $CSP_{SA}.var$, integer numbers (start round R_s , end round R , maximum step number S)

Output: CSP_{MEM}

- 1 Declare an empty CSP model \mathcal{M} ;
- 2 $\mathcal{M}.var \leftarrow \{\text{STKPS}[i] \in \{0, \dots, 8\} : 1 \leq i \leq S-1\}$;
- 3 $\mathcal{M}.var \leftarrow \{\text{ZC}[i] \in \{0, \dots, 16\} : 0 \leq i \leq S-1\}$;
- 4 $\mathcal{M}.var \leftarrow \{\text{XC}[i] \in \{0, \dots, 16\} : 0 \leq i \leq S-1\}$;
- 5 $\mathcal{M}.var \leftarrow \{\text{SBC}[i] \in \{0, \dots, 16\} : 1 \leq i \leq S-1\}$;
- 6 $\mathcal{M}.var \leftarrow \{\text{ASTKC}[i] \in \{0, \dots, (R-R_s)/2\} : 0 \leq i \leq S-1\}$;
- 7 **for** $i = 1, \dots, S-1$ **do**
- 8 $\mathcal{M}.con \leftarrow \text{STKPS}[i] = \sum_{r=R_s}^R \sum_{j=0}^7 \text{KF}_r[j] = i$;
- 9 $\mathcal{M}.con \leftarrow \text{SBC}[i] = \sum_{r=R_s}^R \sum_{j=0}^{15} \text{XF}_r[j] = i$;
- 10 **for** $i = 0, \dots, S-1$ **do**
- 11 $\mathcal{M}.con \leftarrow \text{ZC}[i] = \sum_{r=R_s}^R \sum_{j=0}^{15} \text{ZF}_r[j] \leq i \wedge \text{KF}_r[j] > i$;
- 12 $\mathcal{M}.con \leftarrow \text{XC}[i] = \sum_{r=R_s}^R \sum_{j=0}^{15} \text{XF}_{r+1}[j] \leq i \wedge \text{MCBigger}(\text{WF}_r, i, j)$;
- 13 $\mathcal{M}.con \leftarrow \text{ASTKC}[i] = \sum_{r=R_s}^R \sum_{j=0}^7 \text{stkp}_r[j] = \text{AT} \wedge \text{XF}_r[j] > i$;
- 14 **return** \mathcal{M} ;

is involved in a step. We again use M^{-1} to obtain the relation used by *MCBigger*, defined as

$$\text{MCBigger}(\text{WF}, i, j) := \begin{cases} \text{if } j < 4 \text{ then } \text{WF}[j+12] > i \\ \text{elseif } j < 8 \text{ then } \text{WF}[j-4] > i \vee \text{WF}[j] > i \vee \text{WF}[j+4] > i \\ \text{elseif } j < 12 \text{ then } \text{WF}[j-4] > i \\ \text{else } \text{WF}[j-8] > i \vee \text{WF}[j-4] > i \vee \text{WF}[j] > i \text{ endif.} \end{cases}$$

Model the Complexity Combining all previous models, we can create the COP model, minimizing the time complexity displayed in Algorithm 6. We do this by calculating the cost of each step. As described in Subsection 2.3, we do this by multiplying all the keys guessed this far by the number of partial sums stored in the previous step. First, we ensure that at least one tweakkey is guessed in each step, up to the point where all are determined. In T_{exp} , we store the exponents of the time complexity of each step and use them to calculate the time complexities T of each step. Lastly, we sum each T multiplied by the number of S-box evaluations used in the corresponding step. This is done to obtain the time complexity in equivalent cipher encryptions. Since the total amount of S-Box usage per encryption remains constant for a fixed number of rounds, we do not have to divide T by this number for the objective function. For our model, this has the advantage that we stay in an integer domain and can use a wide range of CP/MILP solvers, including OrTools, to solve it.

4.2 Results

An overview of the improved integral attacks on SKINNY and ForkSKINNY obtained by using our new method can be found in Table 2. In this section, we show an integral distinguisher and the corresponding key-recovery attack for SKINNY found with our model. We list additional results for SKINNY in Appendix C and ForkSKINNY in Appendix E.

Integral Key-Recovery Attack on 18-Round SKINNY- n - n The distinguisher is obtained from ZC distinguishers by inverting the activity pattern at the ZC distinguisher's input. More precisely, plaintext cells with active linear masks take a fixed value, and plaintext cells with zero linear masks take all possible values. Besides, the active tweakkey cells

Algorithm 6: COP model for the partial-sum optimization on SKINNY

Input: integer numbers (tweakey setting z , start round R_s , end round R , output of distinguisher AXL_{R_s} , active tweakey AT , input active AI , maximum step number S)

Output: COP

```

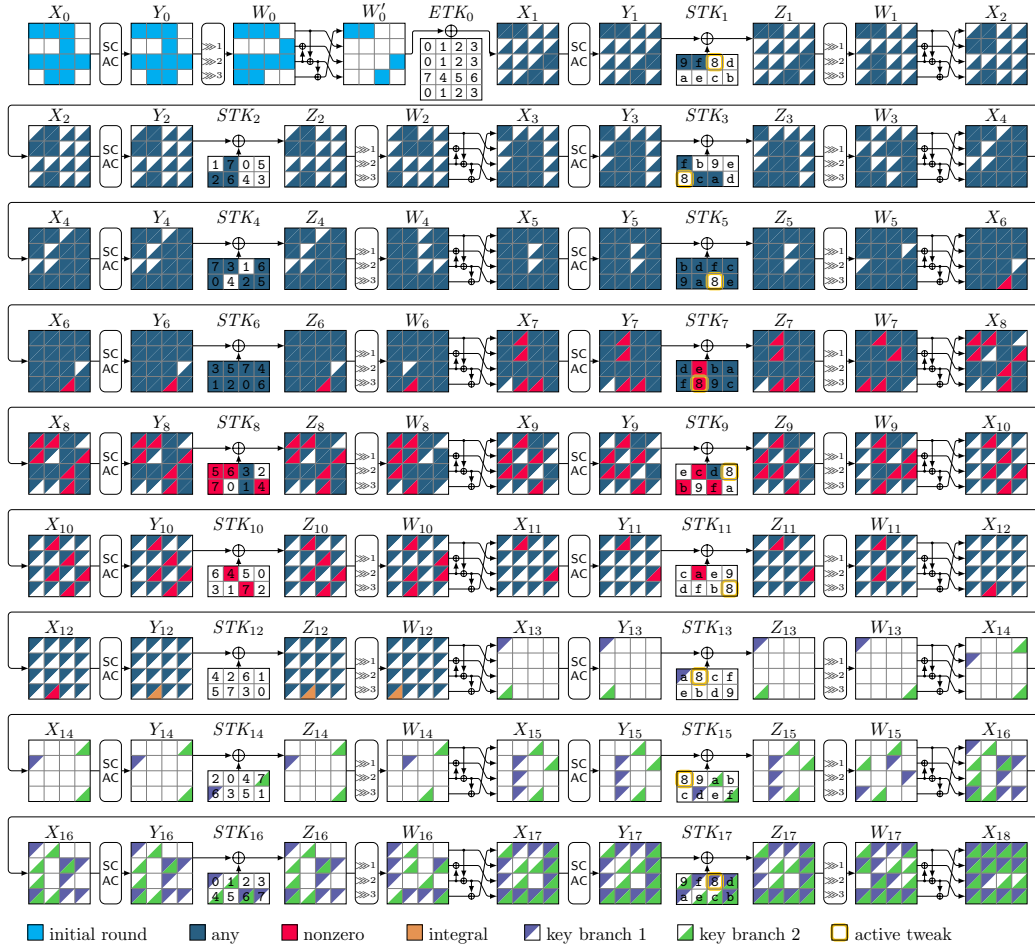
1  $DI \leftarrow 16 - AI + z$ ;
2 Declare an empty COP model  $\mathcal{M}$ ;
3  $\mathcal{M} \leftarrow CSP_{IC} \wedge CSP_{TW} \wedge CSP_{SA} \wedge CSP_{MEM}$ ;
4  $\mathcal{M}.var \leftarrow \{T_{exp}[i] \in \{0, \dots, z \cdot 16 \cdot c\} : 1 \leq i \leq S - 1\}$ ;
5  $\mathcal{M}.var \leftarrow \{T[i] \in \{0, \dots, 2^{z \cdot 16 \cdot c}\} : 1 \leq i \leq S - 1\}$ ;
6  $\mathcal{M}.var \leftarrow T_{total} \in [0, 2^{z \cdot 16 \cdot c}]$ ;
7 for  $i = 1, \dots, S - 2$  do
8   if  $STKPS[i + 1] \neq 0$  then
9      $\mathcal{M}.con \leftarrow STKPS[i] > 0$ ;
10 for  $i = 1, \dots, S - 1$  do
11   if  $STKPS[i] > 0$  then
12     if  $DI > (ASTKC[i - 1] + ZC[i - 1] + XC[i - 1])$  then
13        $\mathcal{M}.con \leftarrow T_{exp}[i] = c \cdot (ASTKC[i - 1] + ZC[i - 1] + XC[i - 1] + \sum_{j=1}^i STKPS[j])$ ;
14     else
15        $\mathcal{M}.con \leftarrow T_{exp}[i] = c \cdot (DI + \sum_{j=1}^i STKPS[j])$ ;
16   else
17      $\mathcal{M}.con \leftarrow T_{exp}[i] = 0$ ;
18      $\mathcal{M}.con \leftarrow T[i] = 2^{T_{exp}[i]}$ ;
19  $\mathcal{M}.con \leftarrow T_{total} = \sum_{i=1}^{S-1} T[i] \cdot SBC[i]$ ;
20  $\mathcal{M}.obj \leftarrow \text{Minimize } T_{total}$ ;
21 return  $\mathcal{M}$ ;
```

involved in the attack take all possible values, and the remaining tweakey cells take a fixed value. Consequently, the sum of the active output cells forms a balanced Boolean function over the input set.

The attack is based on a 12-round ZC distinguisher, extended with one free initial round plus a final key-recovery phase over 5 rounds. In this distinguisher, shown in Figure 7, the tweakey cell 8 is only active in exactly $z = 1$ round (‘nonzero’ in STK_7). At the input to the distinguisher, 4 cells are active. Thus, we can convert it to an integral distinguisher [ADG⁺19] with data complexity $2^{4 \cdot (16-4+1)} = 2^{52}$ for $n = 64$, where the values in the active input cells and the tweakey cell with index 8 iterate over all values. The inactive input cells are constant; the other tweakey cells form the $4 \cdot 15 = 60$ -bit key. Then, the distinguisher’s outputs in $W_{12}[12]$ sum to zero. We can trivially prepend 1 round because adding the equivalent tweakey does not change the input structure (key cell 8 is not involved), and all other operations in the first round are unkeyed.

Key recovery. For the key recovery, we separately recover the sums in $X_{13}[0]$ (branch 1) and $X_{13}[12]$ (branch 2) using the partial-sum technique [FKL⁺00] and merge the results following the meet-in-the-middle approach [SW12]. For the merging, we combine each key candidate for the first branch with each candidate for the second branch that produces the same sum, so that the overall sum is zero. We can repeat the attack a few times to reduce the size of this remaining set of key candidates.

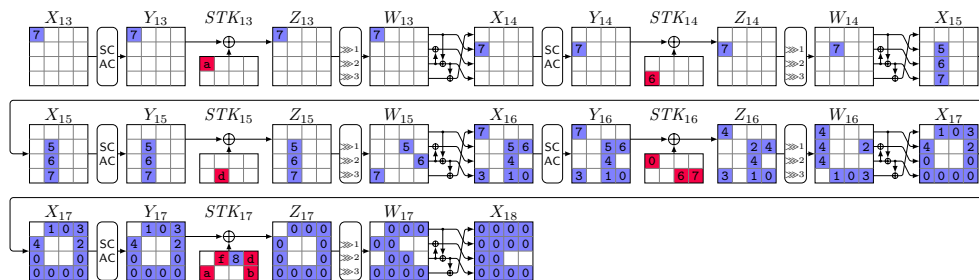
The procedures for the two sums are summarized in Figure 8 and Figure 9. For each sum, we start with Step 0 by storing the obtained ciphertexts (after unwrapping the last linear layer, i.e., Z_{17}) together with their corresponding chosen tweakey values. For the tweakey, we either store the required subtweakey values (i.e., $STK_{17}[2]$) or, if this is more efficient, the input tweakey values from which all subtweakeys can be reconstructed. At this point, we can already recover all intermediate cells that do not depend on any key values, and do not need to further store any cells without further dependencies (i.e., nodes in the dependency tree whose parents are already recovered). For example, consider the

Figure 7: ZC-based integral attack on 18 rounds of SKINNY- n - n .

recovery of $X_{13}[0]$ in Figure 8. From $STK_{17}[2]$, $Z_{17}[2]$, $Z_{17}[14]$ we can recover $X_{16}[15]$ using 3 S-box computations via $Y_{16}[15]$ from $Z_{16}[15] = W_{16}[14] = X_{17}[2] \oplus X_{17}[14]$ from $Y_{17}[2]$, $Y_{17}[14]$. None of these intermediate cells appear in any other computation paths for $X_{13}[0]$, so the only of all these cells we store is $X_{16}[15]$. Similarly, we can derive all other cells labelled with step 0 without any guesses and only need to store 10 cells for this step: $Z_{17}[1, 3, 4, 7]$, $X_{17}[8, 11, 12, 13, 15]$, $X_{16}[15]$. In each of the following steps, we guess one or more cells of involved subkey STK_r , add new cells we can derive from this guess, and remove any cells that are no longer needed. For example, in step 1, we guess $STK_{17}[1]$ (tweakey cell index f), allowing us to derive $X_{16}[14]$ with 2 S-box computations from $Z_{17}[1]$, $X_{17}[13]$; the latter are no longer needed, reducing the remaining data by one cell. Note that the figure shows step indices in each state cell, but tweakey cell indices (instead of steps) in the round tweakey cells. If a subkey index is involved more than $z = 1$ time (such as indices 6, a, d), we only guess the first z times and derive the remaining values afterwards. The order of guessing steps is chosen to optimize the overall time complexity.

Complexity. The complexity of each step is determined by the number of guessed key cells so far and the number of new stored cells (for memory) or previously stored cells (for time). We use the number of S-box lookups as unit for the time complexity, as

customary in previous attacks (although in reality, the memory accesses would likely be more expensive). Overall, we obtain a mapping from values of the sum in $X_{13}[0]$ to disjoint subsets of the $2^{4 \cdot 7} = 2^{28}$ key candidates with complexity $2^{41.32}$, and for the sum in $X_{13}[12]$ for 2^{44} candidates with complexity $2^{45.23}$. These can be merged to obtain $2^{60-4} = 2^{56}$ key candidates that produce zero-sums. This remaining keyspace can either be brute-forced (complexity 2^{56}), or the attack can be repeated 3 times (complexity $3 \cdot 2^{45.23} = 2^{46.81}$ plus merging plus $2^{60-3 \cdot 4} = 2^{48}$). As merging can be done efficiently, the total complexity is less than 2^{49} encryptions equivalents plus $3 \cdot 2^{52} = 2^{53.58}$ data for 18-round SKINNY-64-64 with 60-bit keys, so the data querying phase dominates overall. The same approach yields a complexity of about $3 \cdot 2^{104-7.2} + 2^{120-3 \cdot 8} \approx 2^{99}$ plus $3 \cdot 2^{104} = 2^{105.58}$ data for 18-round SKINNY-128-128 with 120-bit keys.



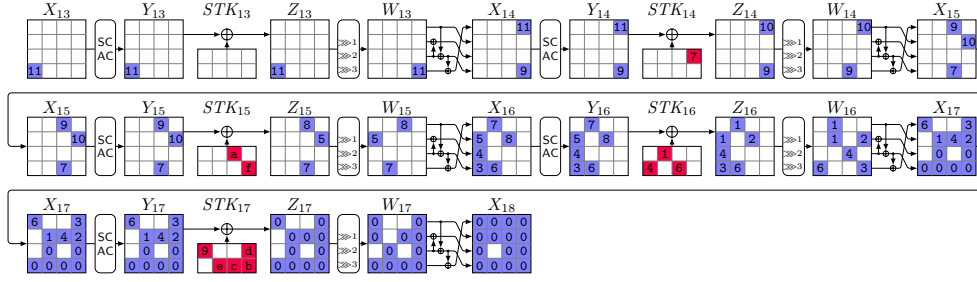
Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{40} = 2^{40}$	$2^{40-5.2}$	$Z_{17}[1, 3, 4, 7]; X_{17}[8, 11, 12, 13, 15]; X_{16}[15]$
1	$STK_{17}[1]$	$2^4 \times 2^{36} = 2^{40}$	$2^{44-7.2}$	$Z_{17}[3, 4, 7]; X_{17}[8, 11, 12, 15]; X_{16}[14, 15]$
2	$STK_{17}[7]$	$2^8 \times 2^{32} = 2^{40}$	$2^{44-8.2}$	$Z_{17}[3, 4]; X_{17}[8, 12, 15]; Z_{16}[6]; X_{16}[14, 15]$
3	$STK_{17}[3]$	$2^{12} \times 2^{28} = 2^{40}$	$2^{44-7.2}$	$Z_{17}[4]; X_{17}[8, 12]; Z_{16}[6]; X_{16}[12, 14, 15]$
4	$STK_{17}[4]$	$2^{16} \times 2^{28} = 2^{44}$	$2^{44-7.2}$	$Z_{16}[0, 6, 7]; X_{16}[10, 12, 14, 15]$
5	$STK_{16}[6]$	$2^{20} \times 2^{20} = 2^{40}$	$2^{48-7.2}$	$Z_{16}[0, 7]; X_{16}[12, 15]; X_{15}[5]$
6	$STK_{16}[7]$	$2^{24} \times 2^{16} = 2^{40}$	$2^{44-7.2}$	$Z_{16}[0]; X_{16}[12]; X_{15}[5, 9]$
7	$STK_{16}[0]$	$2^{28} \times 2^4 = 2^{32}$	$2^{44-6.2}$	$X_{13}[0]$
Σ		2^{44}	$2^{41.32}$	

Figure 8: Complexity of partial-sum key-recovery of $X_{13}[0]$ for 18 rounds of SKINNY- n - n .

Discussion Our model only uses integer variables, enabling us to use any CP solvers that supports integer CP models. We use the CP solver Or-Tools to find optimized integral attacks. Due to the heavy use of sums in the partial-sum COP model, the model takes a while to solve, depending on the settings. For $z = 1$, we used a unified COP model to find the best integral attack within a few hours. However, when using $z = 2$ and $z = 3$, using a separate model for the distinguisher and the partial-sum optimization was faster. In these settings, our model took too long to find optimal solutions. Still, compared to previous works [HSE23], we found better partial-sum optimizations.

Our results show that minimizing the number of tweakey cells involved in key recovery, as in previous work [HSE23], is not optimal. For example, for an 18-round integral attack on SKINNY with $z = 1$, the best key recovery involving 9 key cells has a time complexity of $2^{45.40}$, compared to $2^{45.33}$ when 11 key cells are involved. It is also essential to optimize the guessing order globally rather than one round at a time, as highlighted by Figure 13 where two cells in round 18 are guessed before a cell in round 20.

There are still ways to improve the model. For example, it may be more efficient to compute an intermediate partial-sum in MixColumns. Moreover, there might be cases



Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{44} = 2^{44}$	$2^{44-5.6}$	$Z_{17}[0, 3, 5, 6, 7]; X_{17}[9, 11-15]$
1	$STK_{17}[5]$	$2^4 \times 2^{40} = 2^{44}$	$2^{48-8.2}$	$Z_{17}[0, 3, 6, 7]; X_{17}[11, 12, 14, 15]; Z_{16}[1, 4]$
2	$STK_{17}[7]$	$2^8 \times 2^{36} = 2^{44}$	$2^{48-8.2}$	$Z_{17}[0, 3, 6]; X_{17}[12, 14, 15]; Z_{16}[1, 4, 6]$
3	$STK_{17}[3]$	$2^{12} \times 2^{32} = 2^{44}$	$2^{48-7.2}$	$Z_{17}[0, 6]; X_{17}[12, 14]; Z_{16}[1, 4, 6]; X_{16}[12]$
4	$STK_{17}[6]$	$2^{16} \times 2^{28} = 2^{44}$	$2^{48-7.2}$	$Z_{17}[0]; X_{17}[12]; Z_{16}[1, 4, 6]; X_{16}[8, 12]$
5	$STK_{16}[4]$	$2^{20} \times 2^{20} = 2^{40}$	$2^{48-8.2}$	$Z_{17}[0]; X_{17}[12]; Z_{16}[1, 6]; Z_{15}[7]$
6	$STK_{17}[0]$	$2^{24} \times 2^{16} = 2^{40}$	$2^{44-7.2}$	$Z_{16}[1, 6]; X_{16}[13]; Z_{15}[7]$
7	$STK_{16}[1]$	$2^{28} \times 2^{12} = 2^{40}$	$2^{44-7.2}$	$Z_{16}[6]; Z_{15}[7]; X_{15}[14]$
8	$STK_{16}[6]$	$2^{32} \times 2^{12} = 2^{44}$	$2^{44-8.2}$	$Z_{15}[2, 7]; X_{15}[14]$
9	$STK_{15}[2]$	$2^{36} \times 2^8 = 2^{44}$	$2^{48-7.2}$	$Z_{15}[7]; X_{14}[15]$
10	$STK_{15}[7]$	$2^{40} \times 2^8 = 2^{48}$	$2^{48-8.2}$	$Z_{14}[3]; X_{14}[15]$
11	$STK_{14}[3]$	$2^{44} \times 2^4 = 2^{48}$	$2^{52-7.2}$	$X_{13}[12]$
Σ		2^{48}	$2^{45.23}$	

Figure 9: Complexity of partial-sum key-recovery of $X_{13}[12]$ for 18 rounds of SKINNY- n - n .

where we should wait to apply the S-Box operation for a few steps to decrease the complexity of one step at the cost of increasing the cost later.

5 Conclusion and Future Works

This paper improved the automated search for ID, ZC, and integral attacks. In three aspects, we addressed the limitations of the approach introduced by Hadipour et al. at EUROCRYPT 2023. First, we removed the need to determine the contradiction location in advance. Second, we introduced a bit-wise CP model based on satisfiability to search for ID, ZC, and integral distinguishers. Third, we proposed a CP model for the partial-sum technique for the first time. We implemented our approach, applied it to several block ciphers following different design strategies, and got improved results. Our results show that the proposed approach is efficient and helpful in analyzing and designing block ciphers. One interesting future work is applying our bit-wise CP model to ARX designs. Another interesting future work is extending the idea of our distinguisher modeling, i.e., CP/MILP models based on satisfiability, to other cryptanalytic techniques, such as division property and monomial prediction techniques. Furthermore, one can consider extending our CP model for the partial-sum technique to the key recovery of ZC attacks.

Acknowledgments. This work has been supported in part by the Austrian Science Fund (FWF SFB project SPyCoDe). The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

- [ABD⁺23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The QARMAv2 family of tweakable block ciphers. IACR Cryptology ePrint Archive, Report 2023/929, 2023. URL: <https://eprint.iacr.org/2023/929>.
- [ADG⁺19] Ralph Ankele, Christoph Dobraunig, Jian Guo, Eran Lambooj, Gregor Leander, and Yosuke Todo. Zero-correlation attacks on tweakable block ciphers with linear tweakkey expansion. *IACR Transactions on Symmetric Cryptology*, 2019(1):192–235, Mar. 2019. doi:10.13154/tosc.v2019.i1.192-235.
- [ALP⁺19] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. Forkcipher: A new primitive for authenticated encryption of very short messages. In Steven D. Galbraith and Shihō Moriai, editors, *ASIACRYPT 2019*, volume 11922 of *LNCS*, pages 153–182. Springer, 2019. doi:10.1007/978-3-030-34621-8_6.
- [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.
- [BDL20] Augustin Bariant, Nicolas David, and Gaëtan Leurent. Cryptanalysis of Forkciphers. *IACR Trans. Symmetric Cryptol.*, 2020(1):233–265, 2020. doi:10.13154/tosc.v2020.i1.233-265.
- [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.
- [BKL⁺07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007. doi:10.1007/978-3-540-74735-2_31.
- [BLNPS18] Christina Boura, Virginie Lallemand, María Naya-Plasencia, and Valentin Suder. Making the impossible possible. *Journal of Cryptology*, 31(1):101–133, 2018. doi:10.1007/s00145-016-9251-7.
- [BLNW12] Andrey Bogdanov, Gregor Leander, Kaisa Nyberg, and Meiqin Wang. Integral and multidimensional linear distinguishers with correlation zero. In *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 244–261. Springer, 2012. doi:10.1007/978-3-642-34961-4_16.
- [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*, 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.

- [CCJ⁺16] Tingting Cui, Shiyao Chen, Keting Jia, Kai Fu, and Meiqin Wang. New automatic search tool for impossible differentials and zero-correlation linear approximations. IACR Cryptology ePrint Archive, Report 2016/689, 2016. URL: <https://eprint.iacr.org/2016/689>.
- [DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schl affer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, 2021. doi:10.1007/s00145-021-09398-9.
- [Der16] Patrick Derbez. Note on impossible differential attacks. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 416–427. Springer, 2016. doi:10.1007/978-3-662-52993-5_21.
- [DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In *FSE 1997*, volume 1267 of *LNCS*, pages 149–165. Springer, 1997. doi:10.1007/BFb0052343.
- [DQSW22] Xiaoyang Dong, Lingyue Qin, Siwei Sun, and Xiaoyun Wang. Key guessing strategies for linear key-schedule algorithms in rectangle attacks. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022*, volume 13277 of *LNCS*, pages 3–33. Springer, 2022. doi:10.1007/978-3-031-07082-2_1.
- [FKL⁺00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. Improved cryptanalysis of Rijndael. In *FSE 2000*, volume 1978 of *LNCS*, pages 213–230. Springer, 2000. doi:10.1007/3-540-44706-7_15.
- [HE22] Hosein Hadipour and Maria Eichlseder. Integral cryptanalysis of WARP based on monomial prediction. *IACR Trans. Symmetric Cryptol.*, 2022(2):92–112, 2022. doi:10.46586/tosc.v2022.i2.92-112.
- [HNE22] Hosein Hadipour, Marcel Nageler, and Maria Eichlseder. Throwing boomerangs into feistel structures application to clefia, warp, lblock, lblocks and TWINE. *IACR Trans. Symmetric Cryptol.*, 2022(3):271–302, 2022. doi:10.46586/tosc.v2022.i3.271-302.
- [HPW22] Kai Hu, Thomas Peyrin, and Meiqin Wang. Finding all impossible differentials when considering the DDT. *IACR Cryptol. ePrint Arch.*, page 1034, 2022. URL: <https://eprint.iacr.org/2022/1034>.
- [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.
- [HSWW20] Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums. In *ASIACRYPT 2020*, volume 12491 of *LNCS*, pages 446–476. Springer, 2020. doi:10.1007/978-3-030-64837-4_15.
- [Jea16] J r my Jean. TikZ for Cryptographers. <https://www.iacr.org/authors/tikz/>, 2016.
- [JNP14] J r my Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In *ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 274–288. Springer, 2014. doi:10.1007/978-3-662-45608-8_15.

- [Knu98] Lars Knudsen. DEAL – a 128-bit block cipher. *Complexity*, 258(2):216, 1998.
- [KW02] Lars R. Knudsen and David A. Wagner. Integral cryptanalysis. In *FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, 2002. doi:10.1007/3-540-45661-9_9.
- [Lai94] Xuejia Lai. Higher order derivatives and differential cryptanalysis. *Communications and Cryptography: Two Sides of One Tapestry*, pages 227–233, 1994. doi:10.1007/978-1-4615-2694-0_23.
- [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.
- [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.
- [NLSW21] Chao Niu, Muzhou Li, Siwei Sun, and Meiqin Wang. Zero-correlation linear cryptanalysis with equal treatment for plaintexts and tweakeys. In *CT-RSA 2021*, volume 12704 of *LNCS*, pages 126–147. Springer, 2021. doi:10.1007/978-3-030-75539-3_6.
- [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 *CP 2007*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.
- [PF] Laurent Perron and Vincent Furnon. OR-Tools. URL: <https://developers.google.com/optimization/>.
- [QDW⁺21] Lingyue Qin, Xiaoyang Dong, Xiaoyun Wang, Keting Jia, and Yunwen Liu. Automated search oriented to key recovery on ciphers with linear key schedule applications to boomerangs in SKINNY and ForkSkinny. *IACR Trans. Symmetric Cryptol.*, 2021(2):249–291, 2021.
- [Sag22] Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.5.0)*, 2022. URL: <https://www.sagemath.org>.
- [SGWW20] Ling Sun, David Gerault, Wei Wang, and Meiqin Wang. On the usage of deterministic (related-key) truncated differentials and multidimensional linear approximations for SPN ciphers. *IACR Transactions on Symmetric Cryptology*, 2020(3):262–287, Sep. 2020. doi:10.13154/tosc.v2020.i3.262-287.
- [SLR⁺15] Bing Sun, Zhiqiang Liu, Vincent Rijmen, Ruilin Li, Lei Cheng, Qingju Wang, Hoda AlKhzaimi, and Chao Li. Links among impossible differential, integral and zero correlation linear cryptanalysis. In *CRYPTO 2015*, volume 9215 of *LNCS*, pages 95–115. Springer, 2015. doi:10.1007/978-3-662-47989-6_5.
- [SMB18] Sadegh Sadeghi, Tahereh Mohammadi, and Nasour Bagheri. Cryptanalysis of reduced round SKINNY block cipher. *IACR Trans. Symmetric Cryptol.*, 2018(3):124–162, 2018. doi:10.13154/tosc.v2018.i3.124-162.
- [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 International Publishing. doi:10.1007/978-3-319-56617-7_7.

- [SW12] Yu Sasaki and Lei Wang. Meet-in-the-middle technique for integral attacks against Feistel ciphers. In *SAC 2012*, volume 7707 of *LNCS*, pages 234–251. Springer, 2012. doi:[10.1007/978-3-642-35999-6_16](https://doi.org/10.1007/978-3-642-35999-6_16).
- [Tez14] Cihangir Tezcan. Improbable differential attacks on Present using undisturbed bits. *J. Comput. Appl. Math.*, 259:503–511, 2014. doi:[10.1016/j.cam.2013.06.023](https://doi.org/10.1016/j.cam.2013.06.023).
- [Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 287–314. Springer, 2015. doi:[10.1007/978-3-662-46800-5_12](https://doi.org/10.1007/978-3-662-46800-5_12).
- [XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In *ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 648–678, 2016. doi:[10.1007/978-3-662-53887-6_24](https://doi.org/10.1007/978-3-662-53887-6_24).
- [YQC17] Dong Yang, Wen-Feng Qi, and Hua-Jin Chen. Impossible differential attacks on the SKINNY family of block ciphers. *IET Inf. Secur.*, 11(6):377–385, 2017. doi:[10.1049/iet-ifs.2016.0488](https://doi.org/10.1049/iet-ifs.2016.0488).

— Supplementary Material —

A Specification of the SKINNY family of tweakable block ciphers

SKINNY is a family of lightweight tweakable block ciphers proposed by Beierle et al. in CRYPTO 2016 [BJK⁺16]. SKINNY suggests two block sizes, $n \in \{64, 128\}$, and for each block size it offers three tweakable sizes, $t \in \{n, 2n, 3n\}$. SKINNY- n - t represents SKINNY with n -bit block size and t -bit tweakable size. The internal state of SKINNY is a 4×4 array of cells arranged in a row-major order. The tweakable state is composed of z 4×4 array of cells, where $z = \frac{t}{n} \in \{1, 2, 3\}$. We use $TK1$, $TK2$, and $TK3$ to denote the tweakable arrays. The cell size is 4 (or 8) bits when $n = 64$ (resp. $n = 128$).

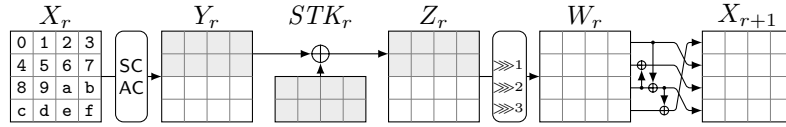


Figure 10: Round function of SKINNY

In each round of SKINNY, five basic operations are applied to the internal state, as illustrated in Figure 10: SubCells (SC), AddConstants (AC), AddRoundTweakey (ART), ShiftRows (SR), and MixColumns (MC). The SC operation applies a 4-bit (or an 8-bit) S-box on each cell. AC combines the round constant with the internal state using the bitwise exclusive-or (XOR). In ART layer, the cells in the first and the second rows of sub-tweakey are XORed to the corresponding cells in the internal state. SR applies a permutation P on the position of the state cells, where $P = [0, 1, 2, 3, 7, 4, 5, 6, 10, 11, 8, 9, 13, 14, 15, 12]$. MC multiplies each column of the internal state by a non-MDS matrix M . M and its inverse are as follows:

$$M = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad M^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

To denote the internal states of SKINNY after r rounds, we use the variables represented in Figure 10. We also represent the difference of state X_r as ΔX_r and the linear mask as ΓX_r . To refer to the i th cell of state X_r , we use $X_r[i]$, where i ranges from 0 to 15. We use STK_r to denote the sub-tweakey after r rounds, and ETK_r , referred to as the equivalent sub-tweakey in round r , is calculated as $ETK_r = MC \circ SR(STK_r)$. Figure 11 shows the relation between STK_r , ETK_r .

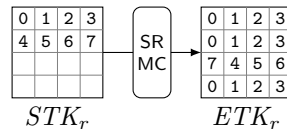


Figure 11: Relation between the sub-tweakey and the equivalent sub-tweakey

The tweakable schedule of SKINNY splits the master tweakable into z tweakable arrays ($TK1, \dots, TKz$), each consisting of n bits. Here, z can be one of three values: 1, 2, or 3. After this division, each tweakable array follows its own separate schedule. The sub-tweakey

for the i th round is generated as follows:

$$\begin{cases} STK_r = TK1_r & \text{if } t = n \\ STK_r = TK1_r \oplus TK2_r & \text{if } t = 2n \\ STK_r = TK1_r \oplus TK2_r \oplus TK3_r & \text{if } t = 3n, \end{cases} \quad (8)$$

where $TK1_r$, $TK2_r$, and $TK3_r$ represent the tweakable arrays in round r , and they're generated as follows:. First, we use a permutation h on each tweakable array. It means that we set $TKm_r[n] \leftarrow TKm_{r-1}[h(n)]$, for $0 \leq n \leq 15$. This applies to m from 1 to 3. Then, we apply an LFSR to every cell in the first and second rows of $TK2_r$ and $TK3_r$. For more in-depth information about SKINNY, you can refer to [BJK⁺16].

B Encoding the Matrix of SKINNY

Here, we recall the CP constraints for the propagation of deterministic differential trails though the matrix of SKINNY in [HSE23]. To see the constraints for deterministic linear trails, refer to [HSE23]. Let $Y = M(X)$, where $X, Y \in \mathbb{F}_2^{4s}$, and M represents the matrix of SKINNY. Additionally, we use a compact representation for the constraint encoding the branching operation $Copy(Y[k], X[i], X[j])$ as $Branch(AX[i], DX[i], AX[j], DX[j], AY[k], DY[k])$. The following CP constraints describe valid transitions for deterministic truncated linear trails through M :

$$Mdiff(AX, DX, AY, DY) := \begin{cases} AY[1] = AX[0] \wedge DY[1] = DX[0] \wedge \\ XOR(AX[1], DX[1], AX[2], DX[2], AY[2], DY[2]) \wedge \\ XOR(AX[0], DX[0], AX[2], DX[2], AY[3], DY[3]) \wedge \\ XOR(AY[3], DY[3], AX[3], DX[3], AY[0], DY[0]) \end{cases}$$

Given that $Y = M^{-1}(X)$, we apply the following constraints to represent the propagation of deterministic truncated linear trails through M^{-1} :

$$Minvdiff(AX, DX, AY, DY) := \begin{cases} AY[0] = AX[1] \wedge DY[0] = DX[1] \wedge \\ XOR(AX[1], DX[1], AX[3], DX[3], AY[2], DY[2]) \wedge \\ XOR(AX[0], DX[0], AX[3], DX[3], AY[3], DY[3]) \wedge \\ XOR(AY[2], DY[2], AX[2], DX[2], AY[1], DY[1]) \end{cases}$$

C Integral Attacks on SKINNY

C.1 Integral Key-Recovery Attack on 22-Round SKINNY- $n-2n$

Following a similar approach as in Section 4.2, we obtain the following complexity. For SKINNY-64-128 with 120-bit keys, we repeat the attack 4 times for a total data complexity of $4 \cdot 2^{4 \cdot (16-4+2)} = 2^{58}$ and a brute-force complexity of $2^{120-4 \cdot 4} = 2^{104}$. Based on the partial-sum approach in Figure 13 and Figure 14, the memory complexity is 2^{104} and the summation time complexity about $4 \cdot (2^{102.23} + 2^{99.07}) \approx 2^{104.38}$ encryption equivalents plus merging. The total time complexity is less than 2^{106} .

For SKINNY-128-256 with 240-bit keys, we repeat the attack 4 times for a total data complexity of $4 \cdot 2^{8 \cdot (16-4+2)} = 2^{116}$ and a brute-force complexity of $2^{240-8 \cdot 4} = 2^{208}$. Based on the partial-sum approach, the memory complexity is 2^{208} and the summation time complexity about $4 \cdot (2^{210.08} + 2^{202.96}) \approx 2^{212.09}$ encryption equivalents plus merging. The total time complexity is less than 2^{213} .

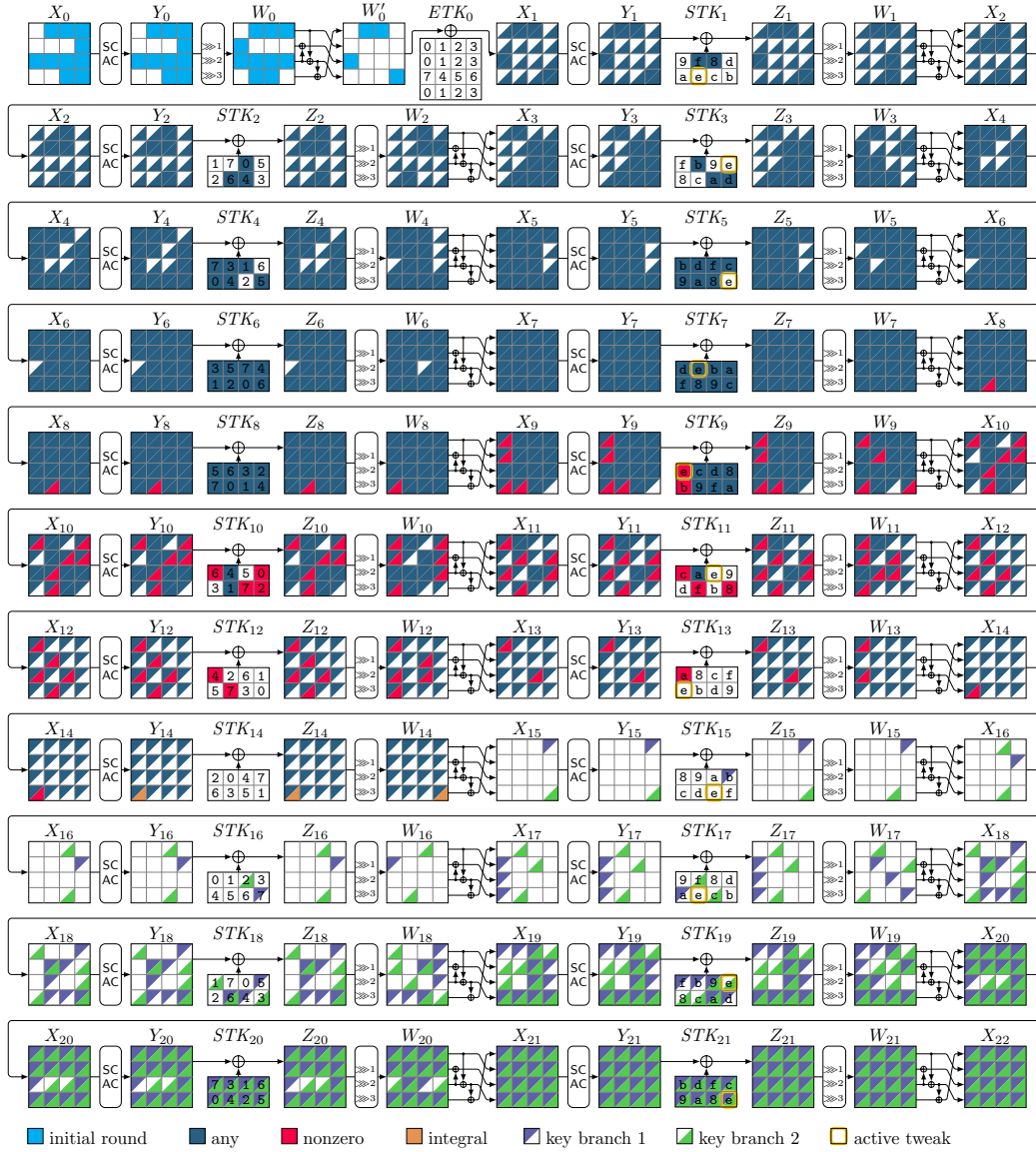
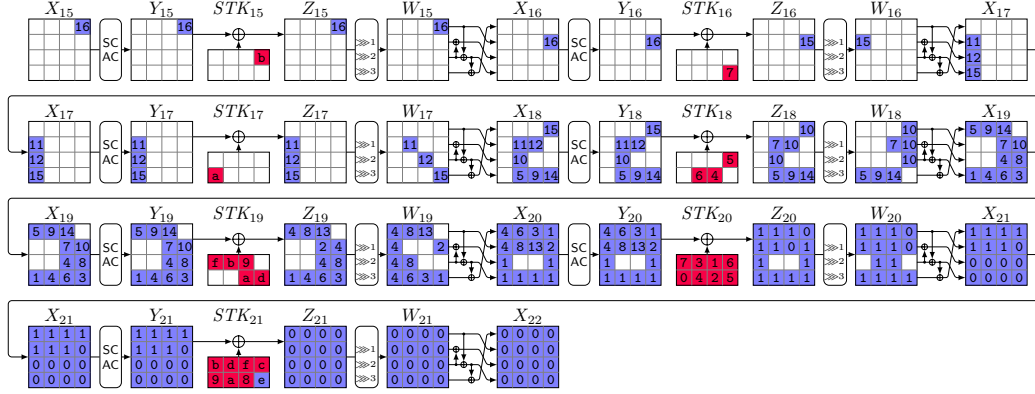
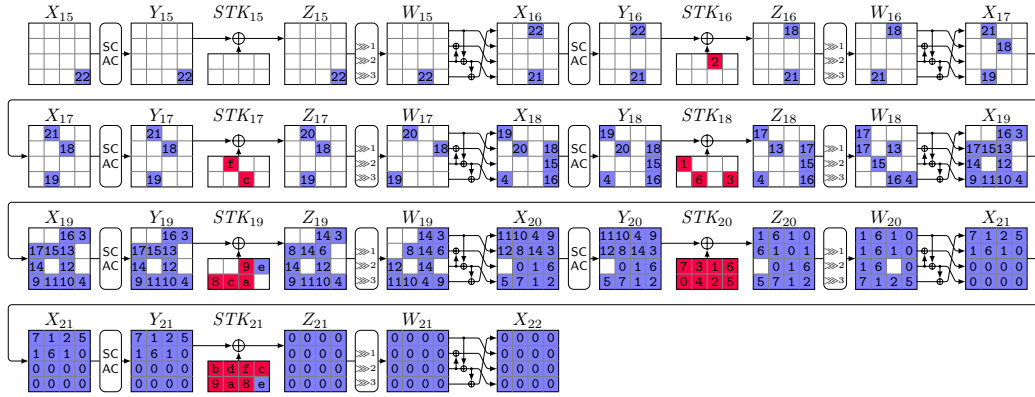


Figure 12: ZC-based integral attack on 22 rounds of SKINNY- $n-2n$.



Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{56} = 2^{56}$	$2^{56-5.3}$	$Z_{21}[0-6]; X_{21}[8-15]; Z_{20}[3, 6]$
1	$STK_{21}[0-6];$ $STK_{20}[3]$	$2^{32} \times 2^{56} = 2^{88}$	$2^{88-4.6}$	$Z_{20}[0, 1, 2, 4-7]; X_{20}[8, 11-15]; X_{19}[12]$
2	$STK_{20}[7]$	$2^{36} \times 2^{48} = 2^{84}$	$2^{92-8.5}$	$Z_{20}[0, 1, 2, 4, 5, 6]; X_{20}[8, 12, 13, 14]; Z_{19}[6]; X_{19}[12]$
3	$STK_{20}[2]$	$2^{40} \times 2^{44} = 2^{84}$	$2^{88-7.5}$	$Z_{20}[0, 1, 4, 5, 6]; X_{20}[8, 12, 13]; Z_{19}[6]; X_{19}[12, 15]$
4	$STK_{20}[0, 4]$	$2^{48} \times 2^{44} = 2^{92}$	$2^{92-6.5}$	$Z_{20}[1, 5, 6]; X_{20}[13]; Z_{19}[0, 6, 7]; X_{19}[10, 12, 13, 15]$
5	$STK_{19}[0]$	$2^{52} \times 2^{40} = 2^{92}$	$2^{96-7.5}$	$Z_{20}[1, 5, 6]; X_{20}[13]; Z_{19}[6, 7]; X_{19}[10, 13, 15]; X_{18}[13]$
6	$STK_{20}[1]$	$2^{56} \times 2^{40} = 2^{96}$	$2^{96-7.5}$	$Z_{20}[5, 6]; X_{20}[13]; Z_{19}[6, 7]; X_{19}[10, 13, 14, 15]; X_{18}[13]$
7	$STK_{19}[6]$	$2^{60} \times 2^{36} = 2^{96}$	$2^{100-8.5}$	$Z_{20}[5, 6]; X_{20}[13]; Z_{19}[7]; X_{19}[13, 14, 15]; Z_{18}[5]; X_{18}[13]$
8	$STK_{20}[5]$	$2^{64} \times 2^{36} = 2^{100}$	$2^{100-7.5}$	$Z_{20}[6]; Z_{19}[1, 7]; X_{19}[11, 13, 14, 15]; Z_{18}[5]; X_{18}[13]$
9	$STK_{19}[1]$	$2^{68} \times 2^{32} = 2^{100}$	$2^{104-7.5}$	$Z_{20}[6]; Z_{19}[7]; X_{19}[11, 14, 15]; Z_{18}[5]; X_{18}[13, 14]$
10	$STK_{19}[7]$	$2^{72} \times 2^{32} = 2^{104}$	$2^{104-7.5}$	$Z_{20}[6]; X_{19}[14]; Z_{18}[3, 5, 6]; X_{18}[9, 13, 14]$
11	$STK_{18}[5]$	$2^{76} \times 2^{24} = 2^{100}$	$2^{108-7.5}$	$Z_{20}[6]; X_{19}[14]; Z_{18}[3, 6]; X_{18}[14]; X_{17}[4]$
12	$STK_{18}[6]$	$2^{80} \times 2^{20} = 2^{100}$	$2^{104-7.5}$	$Z_{20}[6]; X_{19}[14]; Z_{18}[3]; X_{17}[4, 8]$
13	$STK_{20}[6]$	$2^{84} \times 2^{20} = 2^{104}$	$2^{104-8.5}$	$Z_{19}[2]; X_{19}[14]; Z_{18}[3]; X_{17}[4, 8]$
14	$STK_{19}[2]$	$2^{88} \times 2^{16} = 2^{104}$	$2^{108-7.5}$	$Z_{18}[3]; X_{18}[15]; X_{17}[4, 8]$
15	$STK_{18}[3]$	$2^{92} \times 2^4 = 2^{96}$	$2^{108-7.5}$	$Z_{16}[7]$
16	$STK_{16}[7]$	$2^{96} \times 2^4 = 2^{100}$	$2^{100-7.5}$	$X_{15}[3]$
Σ		2^{104}	$2^{102.23}$	

Figure 13: Complexity of partial-sum key-recovery of $X_{15}[3]$ for 22 rounds of SKINNY- $n-2n$.



Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{56} = 2^{56}$	$2^{56-5.1}$	$Z_{21}[0-6]; X_{21}[8, 9, 10-15]; Z_{20}[3, 6]; X_{20}[9]; STK_{19}[3]$
1	$STK_{21}[1, 4, 6]$	$2^{12} \times 2^{56} = 2^{68}$	$2^{68-6.1}$	$Z_{21}[0, 2, 3, 5]; X_{21}[9, 12-15]; Z_{20}[0, 2-7]; X_{20}[9, 10, 14]; STK_{19}[3]$
2	$STK_{21}[2]$	$2^{16} \times 2^{56} = 2^{72}$	$2^{72-7.5}$	$Z_{21}[0, 3, 5]; X_{21}[9, 12, 13, 15]; Z_{20}[0, 2, 3, 5, 6, 7]; X_{20}[9, 10, 14, 15]; STK_{19}[3]$
3	$STK_{20}[7]$	$2^{20} \times 2^{56} = 2^{76}$	$2^{76-7.5}$	$Z_{21}[0, 3, 5]; X_{21}[9, 12, 13, 15]; Z_{20}[0, 2, 3, 5, 6]; X_{20}[7, 9, 10, 14, 15]; X_{19}[3]$
4	$STK_{20}[2]$	$2^{24} \times 2^{56} = 2^{80}$	$2^{80-6.9}$	$Z_{21}[0, 3, 5]; X_{21}[9, 12, 13, 15]; Z_{20}[0, 3, 5, 6]; X_{20}[7, 9, 10, 14, 15]; X_{18}[12]$
5	$STK_{21}[3]$	$2^{28} \times 2^{56} = 2^{84}$	$2^{84-7.5}$	$Z_{21}[0, 5]; X_{21}[9, 12, 13]; Z_{20}[0, 3, 5, 6]; X_{20}[7, 9, 10, 12, 14, 15]; X_{18}[12]$
6	$STK_{21}[5]$	$2^{32} \times 2^{56} = 2^{88}$	$2^{88-7.5}$	$Z_{21}[0]; X_{21}[12]; Z_{20}[0, 1, 3-6]; X_{20}[9, 10, 12, 14, 15]; Z_{19}[6]; X_{18}[12]$
7	$STK_{21}[0]$	$2^{36} \times 2^{56} = 2^{92}$	$2^{92-7.5}$	$Z_{20}[0, 1, 3-6]; X_{20}[9, 10, 12-15]; Z_{19}[6]; X_{18}[12]$
8	$STK_{20}[5]$	$2^{40} \times 2^{52} = 2^{92}$	$2^{96-8.5}$	$Z_{20}[0, 1, 3, 4, 6]; X_{20}[10, 12-15]; Z_{19}[4, 6]; X_{18}[12]$
9	$STK_{20}[3]$	$2^{44} \times 2^{48} = 2^{92}$	$2^{96-7.5}$	$Z_{20}[0, 1, 4, 6]; X_{20}[10, 12, 13, 14]; Z_{19}[4, 6]; X_{19}[12]; X_{18}[12]$
10	$STK_{20}[1]$	$2^{48} \times 2^{44} = 2^{92}$	$2^{96-7.5}$	$Z_{20}[0, 4, 6]; X_{20}[10, 12, 14]; Z_{19}[4, 6]; X_{19}[12, 14]; X_{18}[12]$
11	$STK_{20}[0]$	$2^{52} \times 2^{44} = 2^{96}$	$2^{96-7.5}$	$Z_{20}[4, 6]; X_{20}[10, 12, 14]; Z_{19}[4, 6]; X_{19}[12, 13, 14]; X_{18}[12]$
12	$STK_{20}[4]$	$2^{56} \times 2^{40} = 2^{96}$	$2^{100-7.5}$	$Z_{20}[6]; X_{20}[10, 14]; Z_{19}[4, 6]; X_{19}[10, 12, 13, 14]; X_{18}[12]$
13	$STK_{19}[6]$	$2^{60} \times 2^{36} = 2^{96}$	$2^{100-8.5}$	$Z_{20}[6]; X_{20}[10, 14]; Z_{19}[4]; X_{19}[12, 13, 14]; Z_{18}[5]; X_{18}[12]$
14	$STK_{20}[6]$	$2^{64} \times 2^{36} = 2^{100}$	$2^{100-7.5}$	$Z_{19}[2, 4, 5]; X_{19}[8, 12, 13, 14]; Z_{18}[5]; X_{18}[12]$
15	$STK_{19}[5]$	$2^{68} \times 2^{32} = 2^{100}$	$2^{104-7.5}$	$Z_{19}[2, 4]; X_{19}[8, 12, 14]; Z_{18}[5]; X_{18}[11, 12]$
16	$STK_{19}[2]$	$2^{72} \times 2^{28} = 2^{100}$	$2^{104-7.5}$	$Z_{19}[4]; X_{19}[8, 12]; Z_{18}[5]; X_{18}[11, 12, 15]$
17	$STK_{19}[4]$	$2^{76} \times 2^{24} = 2^{100}$	$2^{104-8.5}$	$Z_{18}[0, 5, 7]; X_{18}[11, 12, 15]$
18	$STK_{18}[7]$	$2^{80} \times 2^{16} = 2^{96}$	$2^{104-7.5}$	$Z_{18}[0, 5]; X_{18}[12]; Z_{16}[2]$
19	$STK_{18}[0]$	$2^{84} \times 2^{12} = 2^{96}$	$2^{100-7.5}$	$Z_{18}[5]; X_{17}[13]; Z_{16}[2]$
20	$STK_{18}[5]$	$2^{88} \times 2^{12} = 2^{100}$	$2^{100-8.5}$	$Z_{17}[1]; X_{17}[13]; Z_{16}[2]$
21	$STK_{17}[1]$	$2^{92} \times 2^8 = 2^{100}$	$2^{104-7.5}$	$Z_{16}[2]; X_{16}[14]$
22	$STK_{16}[2]$	$2^{96} \times 2^4 = 2^{100}$	$2^{104-7.5}$	$X_{15}[15]$
Σ		2^{100}	$2^{99.07}$	

Figure 14: Complexity of partial-sum key-recovery of $X_{15}[15]$ for 22-round SKINNY- $n-2n$.

C.2 Integral Key-Recovery Attack on 26-Round SKINNY- $n-3n$

For SKINNY-64-192 with 180-bit keys, we repeat the attack 4 times for a total data complexity of $4 \cdot 2^{4 \cdot (16-4+3)} = 2^{62}$ and a brute-force complexity of $2^{180-4 \cdot 4} = 2^{164}$. Based on the partial-sum approach in Figure 16 and Figure 17, the memory complexity is 2^{164} and the summation time complexity about $4 \cdot (2^{162.38} + 2^{160.76}) \approx 2^{164.49}$ encryption equivalents plus merging. The total time complexity is less than 2^{166} .

For SKINNY-128-384 with 360-bit keys, we repeat the attack 4 times for a total data complexity of $4 \cdot 2^{8 \cdot (16-4+3)} = 2^{124}$ and a brute-force complexity of $2^{360-8 \cdot 4} = 2^{328}$. Based on the partial-sum approach, the memory complexity is 2^{328} and the summation time complexity about $4 \cdot (2^{330.30} + 2^{328.30}) \approx 2^{330.62}$ encryption equivalents plus merging. The total time complexity is less than 2^{331} .

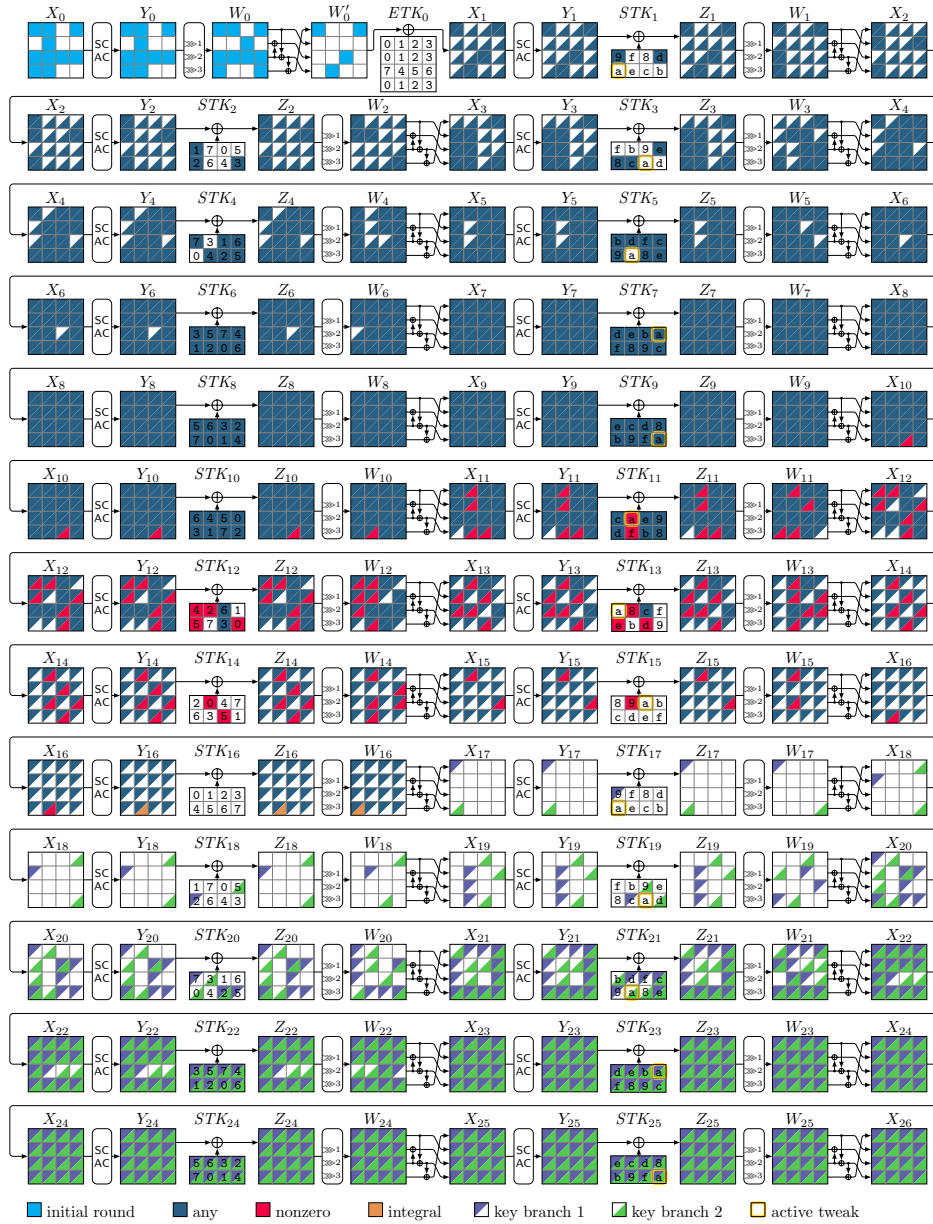
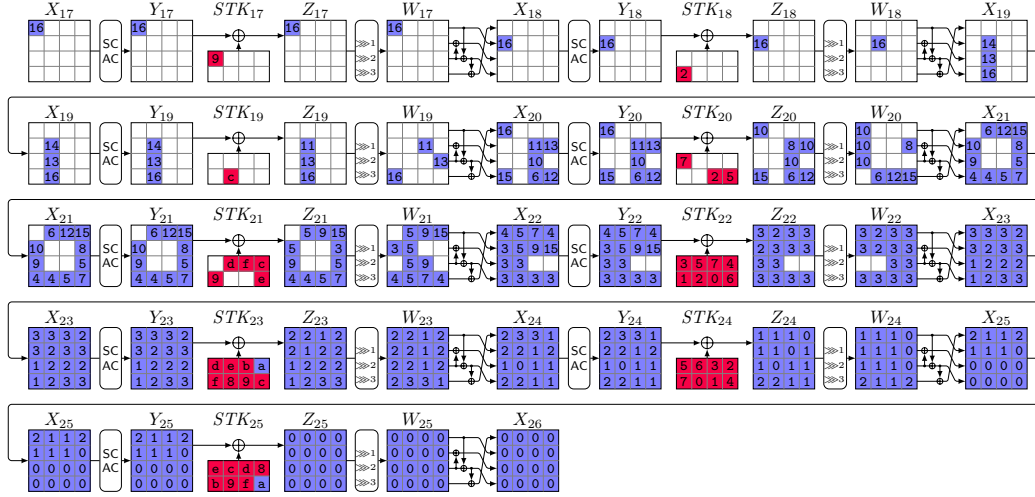
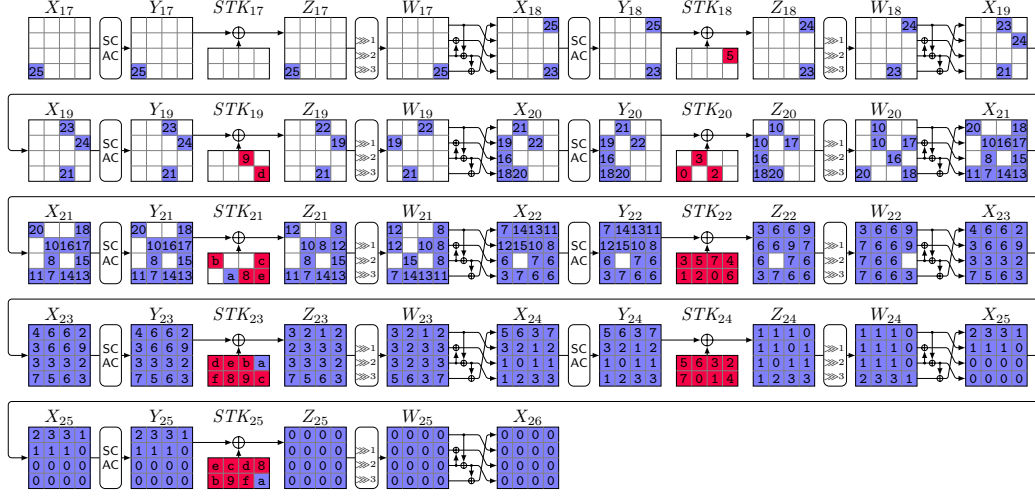


Figure 15: ZC-based integral attack on 26 rounds of SKINNY- $n-3n$.



Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{60} = 2^{60}$	$2^{60-5.4}$	$Z_{25}[0-6]; X_{25}[8, 9, 10, 12-15]; Z_{24}[3, 6]; X_{24}[9]; STK_{23}[3]$
1	$STK_{25}[1, 2, 4, 5, 6];$ $STK_{24}[3, 6]$	$2^{28} \times 2^{60} = 2^{88}$	$2^{88-4.9}$	$Z_{25}[0, 3]; X_{25}[12, 15]; Z_{24}[0, 1, 2, 4, 5, 7]; X_{24}[8, 9, 11, 14, 15]; Z_{23}[2, 5]; X_{23}[8, 12]; STK_{23}[3]$
2	$STK_{25}[0, 3];$ $STK_{24}[0, 4, 5, 7];$ $STK_{23}[5]$	$2^{56} \times 2^{60} = 2^{116}$	$2^{116-4.9}$	$Z_{24}[1, 2]; X_{24}[13, 14]; Z_{23}[0, 1, 2, 4, 6, 7]; X_{23}[3, 8, 10-13]; Z_{22}[1, 4]$
3	$STK_{24}[1, 2];$ $STK_{23}[0, 1, 2, 4, 6, 7];$ $STK_{22}[4]$	$2^{92} \times 2^{52} = 2^{144}$	$2^{152-4.6}$	$Z_{22}[0-3, 5, 6, 7]; X_{22}[9, 12-15]; Z_{21}[7]$
4	$STK_{22}[0, 3]$	$2^{100} \times 2^{44} = 2^{144}$	$2^{152-6.7}$	$Z_{22}[1, 2, 5, 6, 7]; X_{22}[9, 13, 14]; Z_{21}[7]; X_{21}[12, 13]$
5	$STK_{22}[1, 5]$	$2^{108} \times 2^{44} = 2^{152}$	$2^{152-6.7}$	$Z_{22}[2, 6, 7]; X_{22}[14]; Z_{21}[1, 4, 7]; X_{21}[11-14]$
6	$STK_{21}[1]$	$2^{112} \times 2^{40} = 2^{152}$	$2^{156-7.7}$	$Z_{22}[2, 6, 7]; X_{22}[14]; Z_{21}[4, 7]; X_{21}[11, 12, 14]; X_{20}[14]$
7	$STK_{22}[2]$	$2^{116} \times 2^{40} = 2^{156}$	$2^{156-7.7}$	$Z_{22}[6, 7]; X_{22}[14]; Z_{21}[4, 7]; X_{21}[11, 12, 14, 15]; X_{20}[14]$
8	$STK_{21}[7]$	$2^{120} \times 2^{36} = 2^{156}$	$2^{160-8.7}$	$Z_{22}[6, 7]; X_{22}[14]; Z_{21}[4]; X_{21}[12, 14, 15]; Z_{20}[6]; X_{20}[14]$
9	$STK_{22}[6]$	$2^{124} \times 2^{36} = 2^{160}$	$2^{160-7.7}$	$Z_{22}[7]; Z_{21}[2, 4]; X_{21}[8, 12, 14, 15]; Z_{20}[6]; X_{20}[14]$
10	$STK_{21}[4]$	$2^{128} \times 2^{36} = 2^{164}$	$2^{164-7.7}$	$Z_{22}[7]; Z_{21}[2]; X_{21}[14, 15]; Z_{20}[0, 6, 7]; X_{20}[10, 14]$
11	$STK_{20}[6]$	$2^{132} \times 2^{28} = 2^{160}$	$2^{168-8.7}$	$Z_{22}[7]; Z_{21}[2]; X_{21}[14, 15]; Z_{20}[0, 7]; Z_{19}[5]$
12	$STK_{21}[2]$	$2^{136} \times 2^{24} = 2^{160}$	$2^{164-7.7}$	$Z_{22}[7]; X_{21}[15]; Z_{20}[0, 7]; X_{20}[15]; Z_{19}[5]$
13	$STK_{20}[7]$	$2^{140} \times 2^{20} = 2^{160}$	$2^{164-7.7}$	$Z_{22}[7]; X_{21}[15]; Z_{20}[0]; Z_{19}[5]; X_{19}[9]$
14	$STK_{19}[5]$	$2^{144} \times 2^{20} = 2^{164}$	$2^{164-8.7}$	$Z_{22}[7]; X_{21}[15]; Z_{20}[0]; X_{19}[5, 9]$
15	$STK_{22}[7]$	$2^{148} \times 2^{16} = 2^{164}$	$2^{168-7.1}$	$Z_{20}[0]; X_{20}[12]; X_{19}[5, 9]$
16	$STK_{20}[0]$	$2^{152} \times 2^4 = 2^{156}$	$2^{168-6.7}$	$X_{17}[0]$
Σ		2^{164}	$2^{162.38}$	

Figure 16: Complexity of partial-sum key-recovery of $X_{17}[0]$ for 26 rounds of SKINNY- n - $3n$.



Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{60} = 2^{60}$	$2^{60-5.4}$	$Z_{25}[0-6]$; $X_{25}[8, 9, 10, 12-15]$; $Z_{24}[3, 6]$; $X_{24}[9]$; $STK_{23}[3]$; $STK_{21}[5]$
1	$STK_{25}[3, 4, 5, 6]$; $STK_{24}[6]$	$2^{20} \times 2^{60} = 2^{80}$	$2^{80-5.5}$	$Z_{25}[0, 1, 2]$; $X_{25}[12, 13, 14]$; $Z_{24}[0-5, 7]$; $X_{24}[6, 8-12]$; $Z_{23}[2]$; $STK_{23}[3]$; $STK_{21}[5]$
2	$STK_{25}[0]$; $STK_{24}[5, 7]$	$2^{32} \times 2^{60} = 2^{92}$	$2^{92-6.1}$	$Z_{25}[1, 2]$; $X_{25}[13, 14]$; $Z_{24}[0-4]$; $X_{24}[6, 7, 8, 10-13]$; $Z_{23}[1, 2, 4]$; $X_{23}[3, 11]$; $STK_{21}[5]$
3	$STK_{25}[1, 2]$; $STK_{24}[2, 4]$; $STK_{23}[4]$	$2^{52} \times 2^{60} = 2^{112}$	$2^{112-5.1}$	$Z_{24}[0, 1, 3]$; $X_{24}[12, 13, 15]$; $Z_{23}[0, 1, 2, 5, 6, 7]$; $X_{23}[4, 8, 9, 10, 11, 15]$; $Z_{22}[0]$; $X_{22}[12]$; $STK_{21}[5]$
4	$STK_{23}[0]$	$2^{56} \times 2^{60} = 2^{116}$	$2^{116-8.7}$	$Z_{24}[0, 1, 3]$; $X_{24}[12, 13, 15]$; $Z_{23}[1, 2, 5, 6, 7]$; $X_{23}[0, 4, 8, 9, 10, 11, 15]$; $Z_{22}[0]$; $X_{22}[12]$; $STK_{21}[5]$
5	$STK_{24}[0]$	$2^{60} \times 2^{60} = 2^{120}$	$2^{120-7.7}$	$Z_{24}[1, 3]$; $X_{24}[13, 15]$; $Z_{23}[1, 2, 5, 6, 7]$; $X_{23}[0, 4, 8, 9, 10, 11, 13, 15]$; $Z_{22}[0]$; $X_{22}[12]$; $STK_{21}[5]$
6	$STK_{24}[1]$; $STK_{23}[1, 2, 5, 6]$	$2^{80} \times 2^{60} = 2^{140}$	$2^{140-5.4}$	$Z_{24}[3]$; $X_{24}[15]$; $Z_{23}[7]$; $X_{23}[0, 4, 8, 11, 15]$; $Z_{22}[0, 1, 2, 4, 5]$; $X_{22}[8, 11, 12, 14, 15]$; $STK_{21}[5]$
7	$STK_{24}[3]$; $STK_{22}[0]$	$2^{88} \times 2^{60} = 2^{148}$	$2^{148-6.1}$	$Z_{23}[7]$; $X_{23}[11, 15]$; $Z_{22}[1, 2, 4, 5, 7]$; $X_{22}[8, 10-15]$; $X_{21}[13]$; $STK_{21}[5]$
8	$STK_{22}[7]$	$2^{92} \times 2^{60} = 2^{152}$	$2^{152-7.7}$	$Z_{23}[7]$; $X_{23}[11, 15]$; $Z_{22}[1, 2, 4, 5]$; $X_{22}[8, 10, 12-15]$; $Z_{21}[3, 6]$; $X_{21}[9, 13]$; $STK_{21}[5]$
9	$STK_{23}[7]$	$2^{96} \times 2^{60} = 2^{156}$	$2^{156-8.7}$	$Z_{22}[1-6]$; $X_{22}[8, 10, 12-15]$; $Z_{21}[3, 6]$; $X_{21}[9, 13]$; $STK_{21}[5]$
10	$STK_{22}[6]$	$2^{100} \times 2^{56} = 2^{156}$	$2^{160-7.7}$	$Z_{22}[1-5]$; $X_{22}[8, 12-15]$; $Z_{21}[3, 6]$; $Z_{20}[1, 4]$
11	$STK_{22}[3]$	$2^{104} \times 2^{52} = 2^{156}$	$2^{160-7.7}$	$Z_{22}[1, 2, 4, 5]$; $X_{22}[8, 12, 13, 14]$; $Z_{21}[3, 6]$; $X_{21}[12]$; $Z_{20}[1, 4]$
12	$STK_{22}[4]$	$2^{108} \times 2^{48} = 2^{156}$	$2^{160-8.7}$	$Z_{22}[1, 2, 5]$; $X_{22}[13, 14]$; $Z_{21}[0, 3, 6, 7]$; $X_{21}[12]$; $Z_{20}[1, 4]$
13	$STK_{22}[2]$	$2^{112} \times 2^{44} = 2^{156}$	$2^{160-7.7}$	$Z_{22}[1, 5]$; $X_{22}[13]$; $Z_{21}[0, 3, 6, 7]$; $X_{21}[12, 15]$; $Z_{20}[1, 4]$
14	$STK_{22}[1]$	$2^{116} \times 2^{44} = 2^{160}$	$2^{160-7.7}$	$Z_{22}[5]$; $X_{22}[13]$; $Z_{21}[0, 3, 6, 7]$; $X_{21}[12, 14, 15]$; $Z_{20}[1, 4]$
15	$STK_{22}[5]$	$2^{120} \times 2^{40} = 2^{160}$	$2^{164-7.7}$	$Z_{21}[0, 3, 6, 7]$; $X_{21}[11, 12, 14, 15]$; $Z_{20}[1, 4]$
16	$STK_{21}[6]$	$2^{124} \times 2^{36} = 2^{160}$	$2^{164-7.7}$	$Z_{21}[0, 3, 7]$; $X_{21}[11, 12, 15]$; $Z_{20}[1, 4]$; $X_{20}[8]$
17	$STK_{21}[7]$	$2^{128} \times 2^{32} = 2^{160}$	$2^{164-8.7}$	$Z_{21}[0, 3]$; $X_{21}[12, 15]$; $Z_{20}[1, 4, 6]$; $X_{20}[8]$
18	$STK_{21}[3]$	$2^{132} \times 2^{28} = 2^{160}$	$2^{164-7.7}$	$Z_{21}[0]$; $X_{21}[12]$; $Z_{20}[1, 4, 6]$; $X_{20}[8, 12]$
19	$STK_{20}[4]$	$2^{136} \times 2^{20} = 2^{156}$	$2^{164-8.7}$	$Z_{21}[0]$; $X_{21}[12]$; $Z_{20}[1, 6]$; $Z_{19}[7]$
20	$STK_{21}[0]$	$2^{140} \times 2^{16} = 2^{156}$	$2^{160-7.7}$	$Z_{20}[1, 6]$; $X_{20}[13]$; $Z_{19}[7]$
21	$STK_{20}[1]$	$2^{144} \times 2^{12} = 2^{156}$	$2^{160-7.7}$	$Z_{20}[6]$; $Z_{19}[7]$; $X_{19}[14]$
22	$STK_{20}[6]$	$2^{148} \times 2^{12} = 2^{160}$	$2^{160-8.7}$	$Z_{19}[2, 7]$; $X_{19}[14]$
23	$STK_{19}[2]$	$2^{152} \times 2^8 = 2^{160}$	$2^{164-7.7}$	$Z_{19}[7]$; $X_{18}[15]$
24	$STK_{19}[7]$	$2^{156} \times 2^8 = 2^{164}$	$2^{164-8.7}$	$Z_{18}[3]$; $X_{18}[15]$
25	$STK_{18}[3]$	$2^{160} \times 2^4 = 2^{164}$	$2^{168-7.7}$	$X_{17}[12]$
Σ		2^{164}	$2^{160.76}$	

Figure 17: Complexity of partial-sum key-recovery of $X_{17}[12]$ for 26-round SKINNY- $n-3n$.

D Specification of ForkSKINNY

In this section we briefly describe the specification of ForkSKINNY. ForkSKINNY is a tweakable block cipher following the Forkcipher design strategy [ALP⁺19], that advanced to the second-round of the NIST LWC standardization process. As can be seen in Figure 18, Forkcipher receives the key, tweak and n -bit plaintext and can generate $2n$ -bit ciphertext. ForkSKINNY is one instantiation of the Forkcipher framework that uses SKINNY as its underlying primitive. Following the same notation as in [BDL20], we represent the number of rounds before the fork point by \mathbf{r}_i . To represent the number of rounds in the C_0 and C_1 branches (see Figure 18), we use \mathbf{r}_0 and \mathbf{r}_1 , respectively. Depending on the underlying SKINNY variant and the parameters $(\mathbf{r}_i, \mathbf{r}_0, \mathbf{r}_1)$, ForkSKINNY has four variants that are listed in Table 4. For more details, we refer the reader to [ALP⁺19].

Given that ForkSKINNY has two branches after the fork, there are several ways of interpreting the concept of reduced-round instances of ForkSKINNY. In addition, one can attack the path between the C_0 and C_1 branches, called the reconstruction type of attack. However, the designers of ForkSKINNY define the meaning of reduced-round instance of ForkSKINNY as follows. A reduced-round instance has the same parameters as the full-round (see Table 4), except that it has $\mathbf{r}_i - x$ rounds before the fork, and $\mathbf{r}_0 - x = \mathbf{r}_1 - x$ rounds in each branch after the fork for an integer x . It essentially means reducing the number of rounds from three ends by the same amount. Nevertheless, none of the previous analysis on ForkSKINNY falls into this category, including [BDL20, QDW⁺21, DQSW22]. In this paper, for the first time, we provide a limited setting where we limit ourselves to the definition of the designers of ForkSKINNY. For the sake of completeness and to have a fair comparison with previous works, we also provide attacks in arbitrary settings, where we do not limit ourselves to the definition of the designers similar to [BDL20, QDW⁺21, DQSW22].

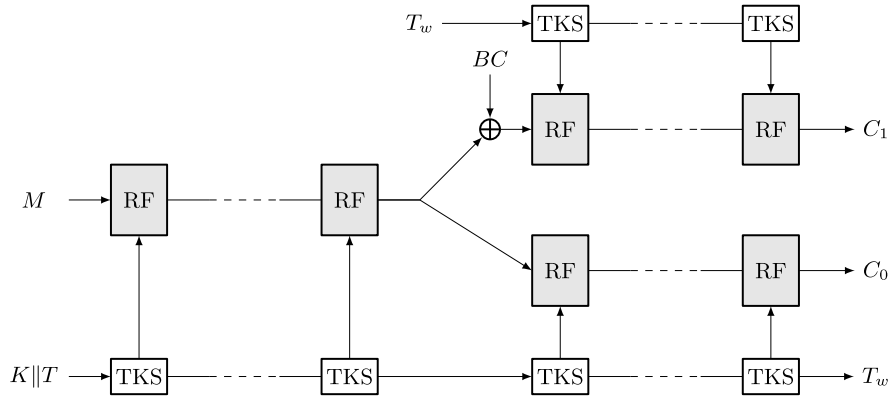


Figure 18: The Forkcipher framework [ALP⁺19].

Table 4: Different variants of ForkSKINNY.

Primitive	block	tweak	tweakey	\mathbf{r}_i	\mathbf{r}_0	\mathbf{r}_1
ForkSKINNY-64-192	64	64	192	17	23	23
ForkSKINNY-128-192	128	64	192	21	27	27
ForkSKINNY-128-256	128	128	256	21	27	27
ForkSKINNY-128-288	128	128	288	25	31	31

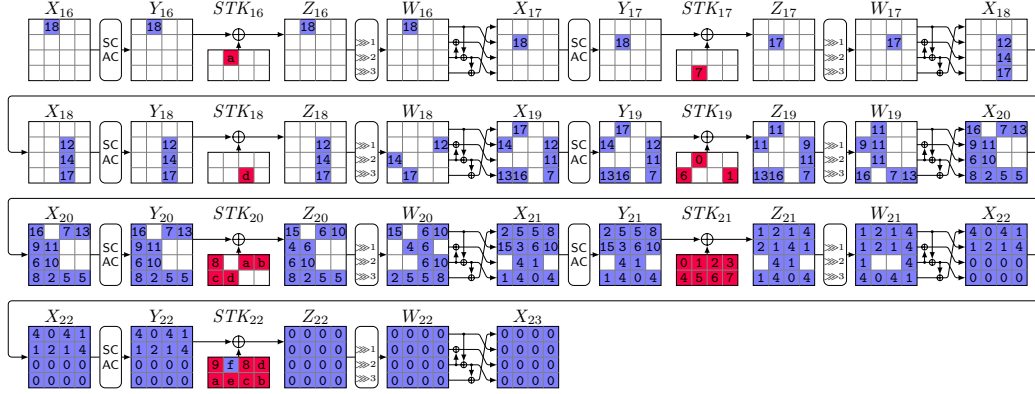
E Integral Attacks on ForkSKINNY

E.1 Integral Key-Recovery Attack on 23-Round ForkSKINNY-128-256 with $r_i = 9$ and $r_o = 27$

We follow the same approach as for SKINNY above. For our attacks, we assume that the tweak is similarly flexible as in SKINNY, i.e., we choose each tweak cell to be either key or tweak material, as has been done in previous published analysis of ForkSKINNY. For ForkSKINNY-128-256 with 240-bit keys, we repeat the attack 4 times for a total data complexity of $4 \cdot 2^{8 \cdot (16-4+2)} = 2^{114}$ and a brute-force complexity of $2^{240-8 \cdot 4} = 2^{208}$. Based on the partial-sum approach in Figure 20 and Figure 21, the memory complexity is 2^{208} and the summation time complexity about $4 \cdot (2^{210.1} + 2^{210.08}) \approx 2^{213.49}$ encryption equivalents plus merging. The total time complexity is less than 2^{214} .

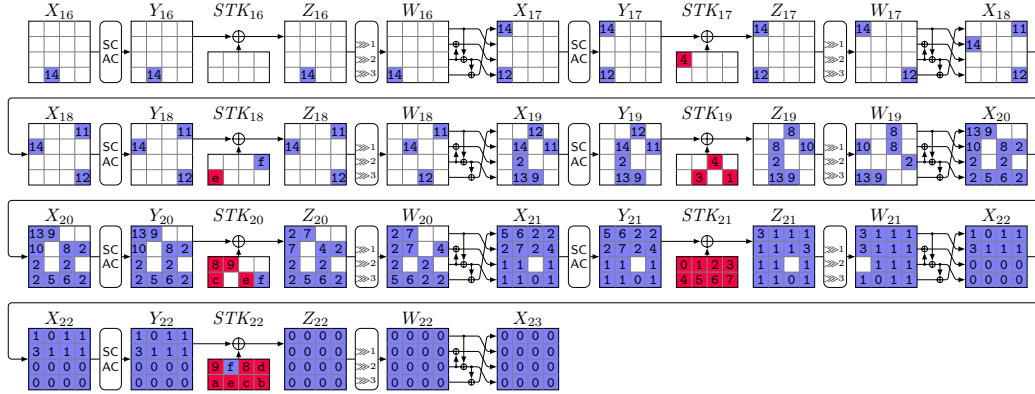


Figure 19: ZC-based integral attack on 23 rounds of ForkSKINNY-128-256.



Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{112} = 2^{112}$	$2^{112-5.2}$	$Z_{22}[0, 2-7]; X_{22}[8-15]; X_{21}[14]$
1	$STK_{22}[3, 4, 6]$	$2^{24} \times 2^{112} = 2^{136}$	$2^{136-6.2}$	$Z_{22}[0, 2, 5, 7]; X_{22}[9, 11-15]; Z_{21}[0, 2, 5, 7]; X_{21}[10, 12, 14]$
2	$STK_{22}[5];$ $STK_{21}[0]$	$2^{40} \times 2^{112} = 2^{152}$	$2^{152-6.9}$	$Z_{22}[0, 2, 7]; X_{22}[11, 12, 14, 15]; Z_{21}[1, 2, 4, 5, 7]; X_{21}[10, 14];$ $X_{20}[13]$
3	$STK_{21}[5]$	$2^{48} \times 2^{112} = 2^{160}$	$2^{160-8.5}$	$Z_{22}[0, 2, 7]; X_{22}[11, 12, 14, 15]; Z_{21}[1, 2, 4, 7]; X_{21}[5, 10, 14];$ $X_{20}[13]$
4	$STK_{22}[0, 2, 7]$	$2^{72} \times 2^{96} = 2^{168}$	$2^{184-5.9}$	$Z_{21}[1-4, 6, 7]; X_{21}[10, 13, 14, 15]; Z_{20}[4]; X_{20}[13]$
5	$STK_{21}[1, 2]$	$2^{88} \times 2^{88} = 2^{176}$	$2^{184-6.5}$	$Z_{21}[3, 4, 6, 7]; X_{21}[10, 14, 15]; Z_{20}[4]; X_{20}[13, 14, 15]$
6	$STK_{21}[6]$	$2^{96} \times 2^{88} = 2^{184}$	$2^{184-7.5}$	$Z_{21}[3, 4, 7]; X_{21}[15]; Z_{20}[2, 4, 5]; X_{20}[8, 13, 14, 15]$
7	$STK_{20}[2]$	$2^{104} \times 2^{80} = 2^{184}$	$2^{192-7.5}$	$Z_{21}[3, 4, 7]; X_{21}[15]; Z_{20}[4, 5]; X_{20}[8, 13, 15]; X_{19}[15]$
8	$STK_{21}[3]$	$2^{112} \times 2^{80} = 2^{192}$	$2^{192-7.5}$	$Z_{21}[4, 7]; X_{21}[15]; Z_{20}[4, 5]; X_{20}[8, 12, 13, 15]; X_{19}[15]$
9	$STK_{20}[4]$	$2^{120} \times 2^{72} = 2^{192}$	$2^{200-8.5}$	$Z_{21}[4, 7]; X_{21}[15]; Z_{20}[5]; X_{20}[12, 13, 15]; Z_{19}[7]; X_{19}[15]$
10	$STK_{21}[7]$	$2^{128} \times 2^{72} = 2^{200}$	$2^{200-7.5}$	$Z_{21}[4]; Z_{20}[3, 5]; X_{20}[9, 12, 13, 15]; Z_{19}[7]; X_{19}[15]$
11	$STK_{20}[5]$	$2^{136} \times 2^{72} = 2^{208}$	$2^{208-7.5}$	$Z_{21}[4]; Z_{20}[3]; X_{20}[12, 15]; Z_{19}[1, 4, 7]; X_{19}[11, 15]$
12	$STK_{19}[7]$	$2^{144} \times 2^{56} = 2^{200}$	$2^{216-7.5}$	$Z_{21}[4]; Z_{20}[3]; X_{20}[12, 15]; Z_{19}[1, 4]; X_{18}[6]$
13	$STK_{20}[3]$	$2^{152} \times 2^{48} = 2^{200}$	$2^{208-7.5}$	$Z_{21}[4]; X_{20}[12]; Z_{19}[1, 4]; X_{19}[12]; X_{18}[6]$
14	$STK_{19}[4]$	$2^{160} \times 2^{40} = 2^{200}$	$2^{208-7.5}$	$Z_{21}[4]; X_{20}[12]; Z_{19}[1]; X_{18}[6, 10]$
15	$STK_{21}[4]$	$2^{168} \times 2^{40} = 2^{208}$	$2^{208-8.5}$	$Z_{20}[0]; X_{20}[12]; Z_{19}[1]; X_{18}[6, 10]$
16	$STK_{20}[0]$	$2^{176} \times 2^{32} = 2^{208}$	$2^{216-7.5}$	$Z_{19}[1]; X_{19}[13]; X_{18}[6, 10]$
17	$STK_{19}[1]$	$2^{184} \times 2^8 = 2^{192}$	$2^{216-7.5}$	$Z_{17}[5]$
18	$STK_{17}[5]$	$2^{192} \times 2^8 = 2^{200}$	$2^{200-7.5}$	$X_{16}[1]$
Σ		2^{208}	$2^{210.1}$	

Figure 20: Complexity of partial-sum key-recovery of $X_{16}[1]$ for 23 rounds of ForkSKINNY-128-256.



Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{112} = 2^{112}$	$2^{112-5.2}$	$Z_{22}[0, 2-7]; X_{22}[8-15]; X_{21}[14]; STK_{20}[7]; STK_{18}[3]$
1	$STK_{22}[0, 2, 3, 5, 6, 7]$	$2^{48} \times 2^{112} = 2^{160}$	$2^{160-4.9}$	$Z_{22}[4]; X_{22}[8, 12]; Z_{21}[1-6]; X_{21}[8, 9, 11-15]; STK_{20}[7]; STK_{18}[3]$
2	$STK_{21}[2, 3, 4, 6]$	$2^{80} \times 2^{112} = 2^{192}$	$2^{192-5.2}$	$Z_{22}[4]; X_{22}[8, 12]; Z_{21}[1, 5]; X_{21}[9, 11, 12, 13, 15]; Z_{20}[0]; X_{20}[8, 10, 12]; X_{19}[9]; STK_{18}[3]$
3	$STK_{22}[4]$	$2^{88} \times 2^{112} = 2^{200}$	$2^{200-8.5}$	$Z_{21}[0, 1, 5, 7]; X_{21}[9, 11, 12, 13, 15]; Z_{20}[0]; X_{20}[8, 10, 12]; X_{19}[9]; STK_{18}[3]$
4	$STK_{21}[7]$	$2^{96} \times 2^{104} = 2^{200}$	$2^{208-8.5}$	$Z_{21}[0, 1, 5]; X_{21}[9, 12, 13]; Z_{20}[0, 6]; X_{20}[8, 10, 12]; X_{19}[9]; STK_{18}[3]$
5	$STK_{21}[0]$	$2^{104} \times 2^{96} = 2^{200}$	$2^{208-7.5}$	$Z_{21}[1, 5]; X_{21}[9, 13]; Z_{20}[0, 6]; X_{20}[8, 10, 12, 13]; X_{19}[9]; STK_{18}[3]$
6	$STK_{21}[1]$	$2^{112} \times 2^{96} = 2^{208}$	$2^{208-7.5}$	$Z_{21}[5]; X_{21}[9, 13]; Z_{20}[0, 6]; X_{20}[8, 10, 12, 13, 14]; X_{19}[9]; STK_{18}[3]$
7	$STK_{21}[5]$	$2^{120} \times 2^{88} = 2^{208}$	$2^{216-8.5}$	$Z_{20}[0, 1, 4, 6]; X_{20}[8, 10, 12, 13, 14]; X_{19}[9]; STK_{18}[3]$
8	$STK_{20}[6]$	$2^{128} \times 2^{80} = 2^{208}$	$2^{216-8.5}$	$Z_{20}[0, 1, 4]; X_{20}[8, 12, 13]; Z_{19}[2, 5]; X_{19}[9]; STK_{18}[3]$
9	$STK_{20}[1]$	$2^{136} \times 2^{72} = 2^{208}$	$2^{216-7.5}$	$Z_{20}[0, 4]; X_{20}[8, 12]; Z_{19}[2, 5]; X_{19}[9, 14]; STK_{18}[3]$
10	$STK_{20}[4]$	$2^{144} \times 2^{64} = 2^{208}$	$2^{216-8.5}$	$Z_{20}[0]; X_{20}[12]; Z_{19}[2, 5, 7]; X_{19}[9, 14]; STK_{18}[3]$
11	$STK_{19}[7]$	$2^{152} \times 2^{56} = 2^{208}$	$2^{216-7.5}$	$Z_{20}[0]; X_{20}[12]; Z_{19}[2, 5]; X_{19}[9, 14]; X_{18}[3]$
12	$STK_{19}[2]$	$2^{160} \times 2^{40} = 2^{200}$	$2^{216-6.9}$	$Z_{20}[0]; X_{20}[12]; Z_{19}[5]; X_{19}[9]; X_{17}[12]$
13	$STK_{20}[0]$	$2^{168} \times 2^{32} = 2^{200}$	$2^{208-7.5}$	$Z_{19}[5]; X_{19}[9, 13]; X_{17}[12]$
14	$STK_{19}[5]$	$2^{176} \times 2^8 = 2^{184}$	$2^{208-6.5}$	$X_{16}[13]$
Σ		2^{208}	$2^{210.8}$	

Figure 21: Complexity of partial-sum key-recovery of $X_{16}[13]$ for 23 rounds of ForkSKINNY-128-256.

E.2 Integral Key-Recovery Attack on 27-Round ForkSKINNY-64-192 with $r_i = 9$ and $r_o = 23$

For ForkSKINNY-64-192 with 180-bit keys, we repeat the attack 4 times for a total data complexity of $4 \cdot 2^{4 \cdot (16-4+2)} = 2^{58}$ and a brute-force complexity of $2^{180-4 \cdot 4} = 2^{164}$. Based on the partial-sum approach in Figure 23 and Figure 24, the memory complexity is 2^{164} and the summation time complexity about $4 \cdot (2^{162.11} + 2^{158.77}) \approx 2^{164.25}$ encryption equivalents plus merging. The total time complexity is less than 2^{166} .

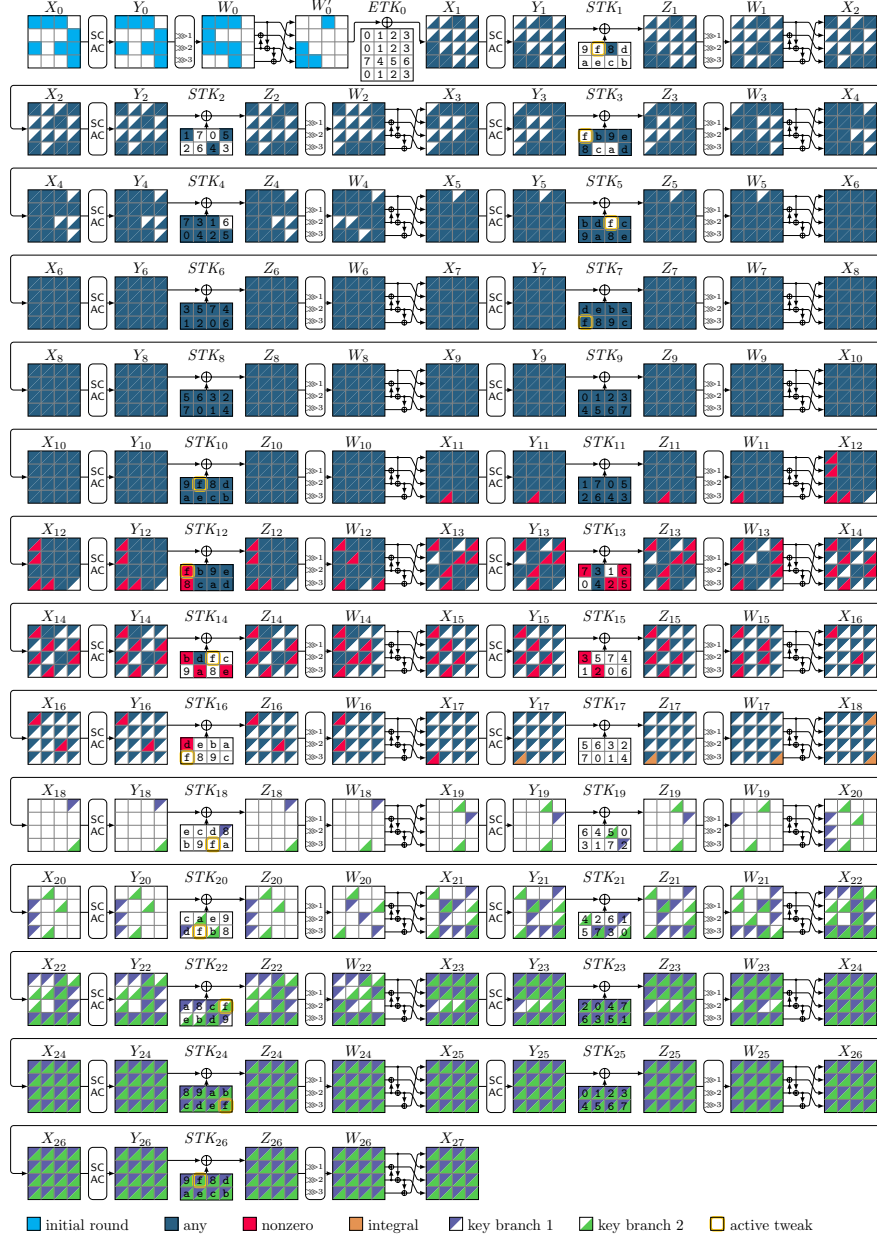
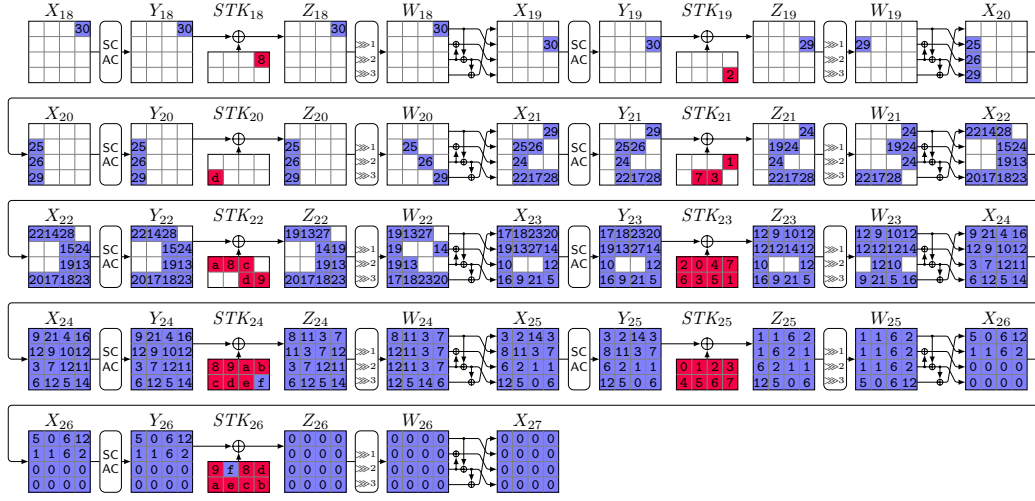
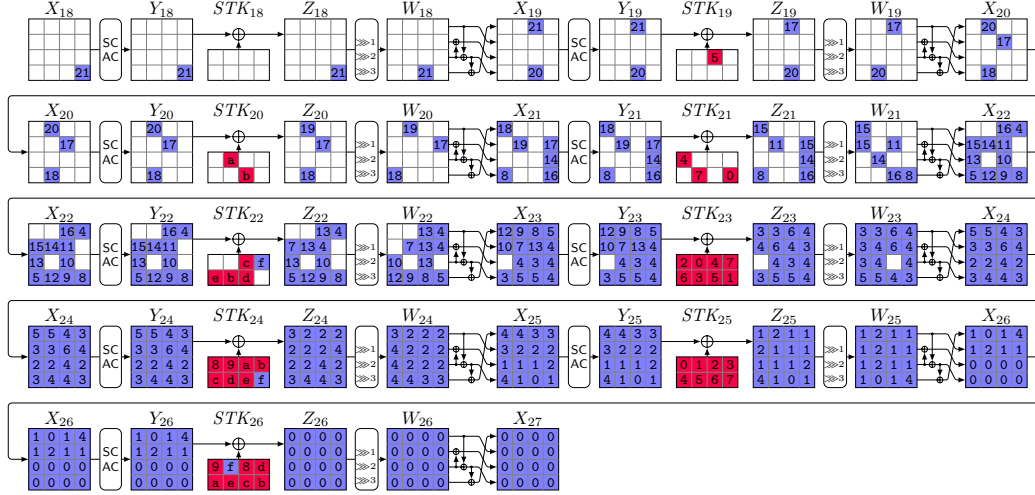


Figure 22: ZC-based integral attack on 27 rounds of ForkSKINNY-64-192.



Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{60} = 2^{60}$	$2^{60-5.4}$	$Z_{26}[0, 2-7]; X_{26}[8-15]; X_{25}[14]; STK_{24}[7]$
1	$STK_{26}[4, 5]$	$2^8 \times 2^{60} = 2^{68}$	$2^{68-6.8}$	$Z_{26}[0, 2, 3, 6, 7]; X_{26}[10, 11, 12, 14, 15]; Z_{25}[0, 1, 4, 7]; X_{25}[10, 11, 14]; STK_{24}[7]$
2	$STK_{26}[7]; STK_{25}[1]$	$2^{16} \times 2^{60} = 2^{76}$	$2^{76-7.2}$	$Z_{26}[0, 2, 3, 6]; X_{26}[10, 12, 14, 15]; Z_{25}[0, 3, 4, 6, 7]; X_{25}[1, 9, 10, 11, 14]; STK_{24}[7]$
3	$STK_{25}[0, 3]$	$2^{28} \times 2^{60} = 2^{88}$	$2^{88-6.8}$	$Z_{26}[0, 2, 3, 6]; X_{26}[10, 12, 14, 15]; Z_{25}[4, 7]; X_{25}[0, 1, 3, 9, 11, 14]; Z_{24}[2, 5]; X_{24}[8]; STK_{24}[7]$
4	$STK_{24}[2]$	$2^{32} \times 2^{60} = 2^{92}$	$2^{92-8.8}$	$Z_{26}[0, 2, 3, 6]; X_{26}[10, 12, 14, 15]; Z_{25}[4, 7]; X_{25}[0, 1, 3, 9, 11, 14]; Z_{24}[5]; X_{24}[2, 8]; STK_{24}[7]$
5	$STK_{26}[0]$	$2^{36} \times 2^{60} = 2^{96}$	$2^{96-6.8}$	$Z_{26}[2, 3, 6]; X_{26}[10, 14, 15]; Z_{25}[4, 7]; X_{25}[0, 3, 9, 11, 13, 14]; Z_{24}[5]; X_{24}[8, 14]; X_{23}[15]; STK_{24}[7]$
6	$STK_{26}[2, 6]$	$2^{44} \times 2^{60} = 2^{104}$	$2^{104-6.4}$	$Z_{26}[3]; X_{26}[15]; Z_{25}[2, 4, 5, 7]; X_{25}[0, 8, 9, 11, 13, 14, 15]; Z_{24}[5]; X_{24}[8, 12, 14]; X_{23}[15]; STK_{24}[7]$
7	$STK_{25}[7]$	$2^{48} \times 2^{60} = 2^{108}$	$2^{108-7.8}$	$Z_{26}[3]; X_{26}[15]; Z_{25}[2, 4, 5]; X_{25}[0, 8, 9, 13, 14]; Z_{24}[3, 5, 6]; X_{24}[8, 9, 12, 14]; X_{23}[15]; STK_{24}[7]$
8	$STK_{25}[4]$	$2^{52} \times 2^{60} = 2^{112}$	$2^{112-8.8}$	$Z_{26}[3]; X_{26}[15]; Z_{25}[2, 5]; X_{25}[0, 4, 8, 9, 13, 14]; Z_{24}[0, 3, 5, 6]; X_{24}[8, 9, 12, 14]; X_{23}[15]; STK_{24}[7]$
9	$STK_{24}[0, 5]$	$2^{60} \times 2^{60} = 2^{120}$	$2^{120-7.2}$	$Z_{26}[3]; X_{26}[15]; Z_{25}[2, 5]; X_{25}[0, 4, 8, 9, 13, 14]; Z_{24}[3, 6]; X_{24}[5, 8, 9, 12, 14]; Z_{23}[1]; X_{23}[13, 15]; STK_{24}[7]$
10	$STK_{24}[6]$	$2^{64} \times 2^{60} = 2^{124}$	$2^{124-7.8}$	$Z_{26}[3]; X_{26}[15]; Z_{25}[2, 5]; X_{25}[0, 4, 8, 9, 13, 14]; Z_{24}[3]; X_{24}[5, 6, 8, 9, 12, 14]; Z_{23}[1, 2]; X_{23}[8, 13, 15]; STK_{24}[7]$
11	$STK_{25}[5]$	$2^{68} \times 2^{60} = 2^{128}$	$2^{128-7.8}$	$Z_{26}[3]; X_{26}[15]; Z_{25}[2]; X_{25}[0, 4, 8, 14]; Z_{24}[1, 3, 4]; X_{24}[5, 6, 8, 9, 11, 12, 14]; Z_{23}[1, 2]; X_{23}[8, 13, 15]; STK_{24}[7]$
12	$STK_{26}[3]; STK_{24}[4]$	$2^{76} \times 2^{60} = 2^{136}$	$2^{136-5.9}$	$Z_{25}[2]; X_{25}[14]; Z_{24}[1, 3]; X_{24}[7, 11, 13]; Z_{23}[0, 1, 2, 3, 4, 5, 7]; X_{23}[8, 11, 13, 15]$
13	$STK_{23}[5]$	$2^{80} \times 2^{60} = 2^{140}$	$2^{140-7.8}$	$Z_{25}[2]; X_{25}[14]; Z_{24}[1, 3]; X_{24}[7, 11, 13]; Z_{23}[0, 1, 2, 3, 4, 7]; X_{23}[8, 11, 13, 15]; Z_{22}[1]; X_{22}[11]$
14	$STK_{25}[2]; STK_{23}[7]; STK_{22}[1]$	$2^{92} \times 2^{60} = 2^{152}$	$2^{152-6.8}$	$Z_{24}[1, 3]; X_{24}[13, 15]; Z_{23}[0, 1, 2, 3, 4, 6]; X_{23}[8, 13, 15]; Z_{22}[6]; X_{22}[1, 11]$
15	$STK_{22}[6]$	$2^{96} \times 2^{60} = 2^{156}$	$2^{156-8.8}$	$Z_{24}[1, 3]; X_{24}[13, 15]; Z_{23}[0, 1, 2, 3, 4, 6]; X_{23}[8, 13, 15]; X_{22}[1, 6, 11]$
16	$STK_{24}[3]$	$2^{100} \times 2^{60} = 2^{160}$	$2^{160-7.8}$	$Z_{24}[1]; X_{24}[13]; Z_{23}[0, 1, 2, 3, 4, 6]; X_{23}[8, 12, 13, 15]; X_{22}[1, 6, 11]$
17	$STK_{23}[0]$	$2^{104} \times 2^{56} = 2^{160}$	$2^{164-7.2}$	$Z_{24}[1]; X_{24}[13]; Z_{23}[1, 2, 3, 4, 6]; X_{23}[8, 12, 13, 15]; X_{22}[6, 11]; X_{21}[14]$
18	$STK_{23}[1]$	$2^{108} \times 2^{52} = 2^{160}$	$2^{164-7.8}$	$Z_{24}[1]; X_{24}[13]; Z_{23}[2, 3, 4, 6]; X_{23}[8, 12, 15]; X_{22}[6, 11, 14]; X_{21}[14]$
19	$STK_{23}[4]$	$2^{112} \times 2^{48} = 2^{160}$	$2^{164-7.8}$	$Z_{24}[1]; X_{24}[13]; Z_{23}[2, 3, 6]; X_{23}[15]; Z_{22}[0, 7]; X_{22}[11, 14]; Z_{21}[5]; X_{21}[14]$
20	$STK_{23}[3]$	$2^{116} \times 2^{44} = 2^{160}$	$2^{164-7.8}$	$Z_{24}[1]; X_{24}[13]; Z_{23}[2, 6]; Z_{22}[0, 7]; X_{22}[11, 12, 14]; Z_{21}[5]; X_{21}[14]$
21	$STK_{24}[1]$	$2^{120} \times 2^{40} = 2^{160}$	$2^{164-7.8}$	$Z_{23}[2, 6]; X_{23}[14]; Z_{22}[0, 7]; X_{22}[11, 12, 14]; Z_{21}[5]; X_{21}[14]$
22	$STK_{22}[0]$	$2^{124} \times 2^{36} = 2^{160}$	$2^{164-7.8}$	$Z_{23}[2, 6]; X_{23}[14]; Z_{22}[7]; X_{22}[11, 14]; Z_{21}[5]; X_{21}[13, 14]$
23	$STK_{23}[2]$	$2^{128} \times 2^{32} = 2^{160}$	$2^{164-7.8}$	$Z_{23}[6]; Z_{22}[7]; X_{22}[11, 14, 15]; Z_{21}[5]; X_{21}[13, 14]$
24	$STK_{22}[7]$	$2^{132} \times 2^{32} = 2^{164}$	$2^{164-7.8}$	$Z_{23}[6]; X_{22}[14]; Z_{21}[3, 5, 6]; X_{21}[9, 13, 14]$
25	$STK_{21}[5]$	$2^{136} \times 2^{24} = 2^{160}$	$2^{168-7.8}$	$Z_{23}[6]; X_{22}[14]; Z_{21}[3, 6]; X_{21}[14]; X_{20}[4]$
26	$STK_{21}[6]$	$2^{140} \times 2^{20} = 2^{160}$	$2^{164-7.8}$	$Z_{23}[6]; X_{22}[14]; Z_{21}[3]; X_{20}[4, 8]$
27	$STK_{23}[6]$	$2^{144} \times 2^{20} = 2^{164}$	$2^{164-8.8}$	$Z_{22}[2]; X_{22}[14]; Z_{21}[3]; X_{20}[4, 8]$
28	$STK_{22}[2]$	$2^{148} \times 2^{16} = 2^{164}$	$2^{168-7.8}$	$Z_{21}[3]; X_{21}[15]; X_{20}[4, 8]$
29	$STK_{21}[3]$	$2^{152} \times 2^4 = 2^{156}$	$2^{168-7.8}$	$Z_{19}[7]$
30	$STK_{19}[7]$	$2^{156} \times 2^4 = 2^{160}$	$2^{160-7.8}$	$X_{18}[3]$
Σ		2^{164}	$2^{162.11}$	

Figure 23: Complexity of partial-sum key-recovery of $X_{18}[3]$ for 27 rounds of ForkSKINNY-64-192.



Step	Guessed	$K \times D = \text{Mem}$	Time	Stored Texts
0	–	$2^0 \times 2^{60} = 2^{60}$	$2^{60-5.4}$	$Z_{26}[0, 2-7]; X_{26}[8-15]; X_{25}[14]; STK_{24}[7]; STK_{22}[3]$
1	$STK_{26}[0, 2, 4, 6, 7]$	$2^{20} \times 2^{60} = 2^{80}$	$2^{80-5.4}$	$Z_{26}[3, 5]; X_{26}[9, 13, 15]; Z_{25}[0, 2, 3, 5, 6, 7]; X_{25}[8, 9, 10, 13, 14, 15]; STK_{24}[7]; STK_{22}[3]$
2	$STK_{26}[5]; STK_{25}[5, 6, 7]$	$2^{36} \times 2^{60} = 2^{96}$	$2^{96-5.8}$	$Z_{26}[3]; X_{26}[15]; Z_{25}[0-4]; X_{25}[8, 13, 14, 15]; Z_{24}[1-6]; X_{24}[8, 9, 11]; STK_{24}[7]; STK_{22}[3]$
3	$STK_{25}[2, 3, 4]; STK_{24}[3, 4, 5]$	$2^{60} \times 2^{60} = 2^{120}$	$2^{120-5.4}$	$Z_{26}[3]; X_{26}[15]; Z_{25}[0, 1]; X_{25}[4, 8, 13]; Z_{24}[0, 1, 2, 6]; X_{24}[5, 9, 11, 12, 15]; Z_{23}[0, 1, 7]; X_{23}[10, 12]; STK_{24}[7]; STK_{22}[3]$
4	$STK_{26}[3]; STK_{25}[0, 1]; STK_{24}[2]; STK_{23}[7]$	$2^{80} \times 2^{60} = 2^{140}$	$2^{140-4.9}$	$Z_{24}[0, 1, 6]; X_{24}[10, 12, 13, 14]; Z_{23}[0, 1, 3, 4, 6]; X_{23}[9, 10, 12, 15]; Z_{22}[6]; X_{22}[3]$
5	$STK_{24}[0, 1]; STK_{23}[3]$	$2^{92} \times 2^{60} = 2^{152}$	$2^{152-6.2}$	$Z_{24}[6]; X_{24}[10, 14]; Z_{23}[0, 1, 4, 6]; X_{23}[9, 10, 12, 13, 14]; Z_{22}[6]; X_{22}[3, 12]$
6	$STK_{24}[6]$	$2^{96} \times 2^{56} = 2^{152}$	$2^{156-8.8}$	$Z_{23}[0, 1, 2, 4, 5, 6]; X_{23}[9, 10, 12, 13, 14]; Z_{22}[6]; X_{22}[3, 12]$
7	$STK_{23}[5]$	$2^{100} \times 2^{52} = 2^{152}$	$2^{156-8.8}$	$Z_{23}[0, 1, 2, 4, 6]; X_{23}[10, 12, 13, 14]; Z_{22}[4, 6]; X_{22}[3, 12]$
8	$STK_{23}[2]$	$2^{104} \times 2^{48} = 2^{152}$	$2^{156-7.2}$	$Z_{23}[0, 1, 4, 6]; X_{23}[10, 12, 13, 14]; Z_{22}[4, 6]; X_{22}[12]; X_{21}[12]$
9	$STK_{23}[1]$	$2^{108} \times 2^{44} = 2^{152}$	$2^{156-7.8}$	$Z_{23}[0, 4, 6]; X_{23}[10, 12, 14]; Z_{22}[4, 6]; X_{22}[12, 14]; X_{21}[12]$
10	$STK_{23}[4]$	$2^{112} \times 2^{44} = 2^{156}$	$2^{156-7.8}$	$Z_{23}[0, 6]; X_{23}[10, 12, 14]; Z_{22}[4, 6]; X_{22}[10, 12, 14]; X_{21}[12]$
11	$STK_{22}[6]$	$2^{116} \times 2^{40} = 2^{156}$	$2^{160-8.8}$	$Z_{23}[0, 6]; X_{23}[10, 12, 14]; Z_{22}[4]; X_{22}[12, 14]; Z_{21}[5]; X_{21}[12]$
12	$STK_{23}[0]$	$2^{120} \times 2^{36} = 2^{156}$	$2^{160-7.8}$	$Z_{23}[6]; X_{23}[10, 14]; Z_{22}[4]; X_{22}[12, 13, 14]; Z_{21}[5]; X_{21}[12]$
13	$STK_{23}[6]$	$2^{124} \times 2^{36} = 2^{160}$	$2^{160-7.8}$	$Z_{22}[2, 4, 5]; X_{22}[8, 12, 13, 14]; Z_{21}[5]; X_{21}[12]$
14	$STK_{22}[5]$	$2^{128} \times 2^{32} = 2^{160}$	$2^{164-7.8}$	$Z_{22}[2, 4]; X_{22}[8, 12, 14]; Z_{21}[5]; X_{21}[11, 12]$
15	$STK_{22}[4]$	$2^{132} \times 2^{28} = 2^{160}$	$2^{164-8.8}$	$Z_{22}[2]; X_{22}[14]; Z_{21}[0, 5, 7]; X_{21}[11, 12]$
16	$STK_{22}[2]$	$2^{136} \times 2^{24} = 2^{160}$	$2^{164-7.8}$	$Z_{21}[0, 5, 7]; X_{21}[11, 12, 15]$
17	$STK_{21}[7]$	$2^{140} \times 2^{16} = 2^{156}$	$2^{164-7.8}$	$Z_{21}[0, 5]; X_{21}[12]; Z_{19}[2]$
18	$STK_{21}[0]$	$2^{144} \times 2^{12} = 2^{156}$	$2^{160-7.8}$	$Z_{21}[5]; X_{20}[13]; Z_{19}[2]$
19	$STK_{21}[5]$	$2^{148} \times 2^{12} = 2^{160}$	$2^{160-8.8}$	$Z_{20}[1]; X_{20}[13]; Z_{19}[2]$
20	$STK_{20}[1]$	$2^{152} \times 2^8 = 2^{160}$	$2^{164-7.8}$	$Z_{19}[2]; X_{19}[14]$
21	$STK_{19}[2]$	$2^{156} \times 2^4 = 2^{160}$	$2^{164-7.8}$	$X_{18}[15]$
Σ		2^{160}	$2^{158.77}$	

 Figure 24: Complexity of partial-sum key-recovery of $X_{18}[15]$ for 27 rounds of ForkSKINNY-64-192.

F ID, ZC and Integral Distinguishers for ForkSKINNY

This section presents the ID, ZC, and integral distinguishers for ForkSKINNY. To illustrate how our CP model works, we represent both forward and backward propagations in the shape of our distinguishers. To represent a cell with a fixed, nonzero, or unknown difference value (linear mask), we use \square , \blacksquare , \square , and \square , \blacksquare , \square in forward and backward propagations, respectively. In addition, we represent the active tweak cell in our integral distinguishers by \square .

Figure 25, and Figure 26 illustrate some of our discovered integral distinguishers for ForkSKINNY in the single-key and chosen-tweak setting. Figure 27, and Figure 28 illustrate some of our discovered ID distinguishers for ForkSKINNY in the related-tweakey setting. For example, as can be seen in Figure 28, the type of activeness pattern for $Y_0[5]$ and $STK_0[5]$ is nonzero fixed. It means that the difference value in these cells should be equal but can take any of the 15 possible nonzero values.

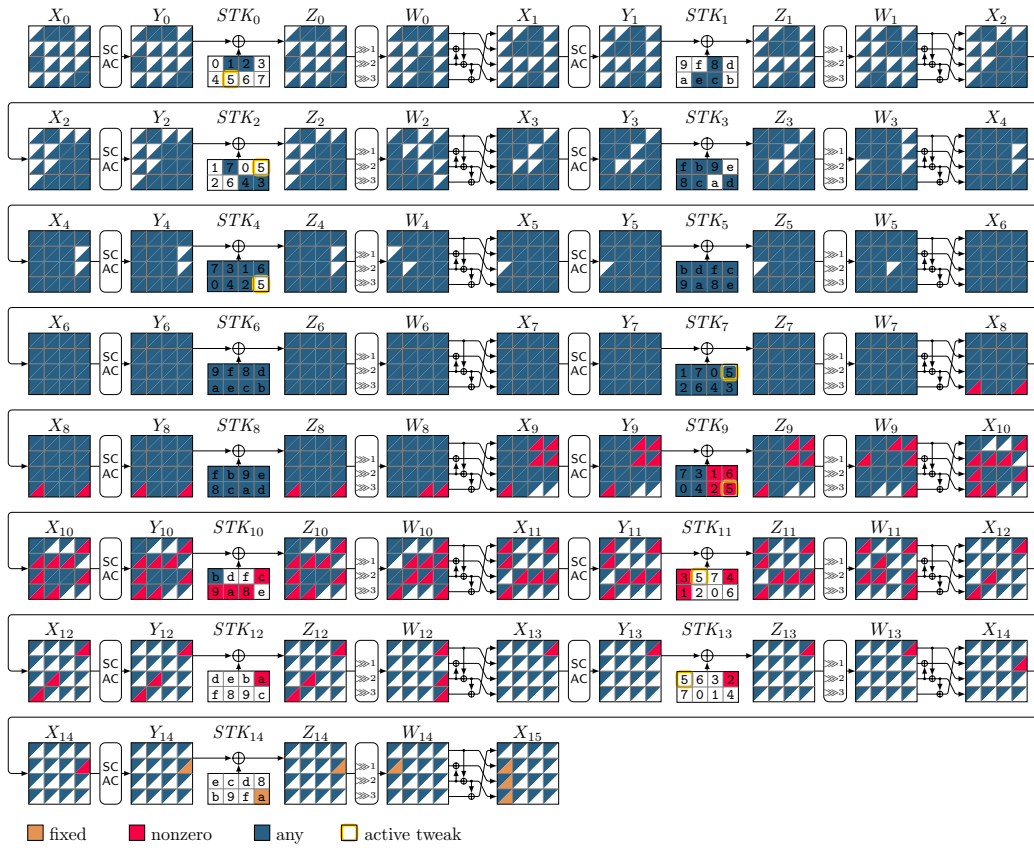


Figure 25: Integral and ZC distinguishers for 15 rounds of ForkSKINNY-128-256, where $r_i = 6$, $r_0 = 27$, and $r_1 = 9$.



Figure 26: Integral and ZC distinguishers for 17 rounds of ForkSKINNY-64-192, where $r_i = 6$, $r_0 = 23$, and $r_1 = 11$.

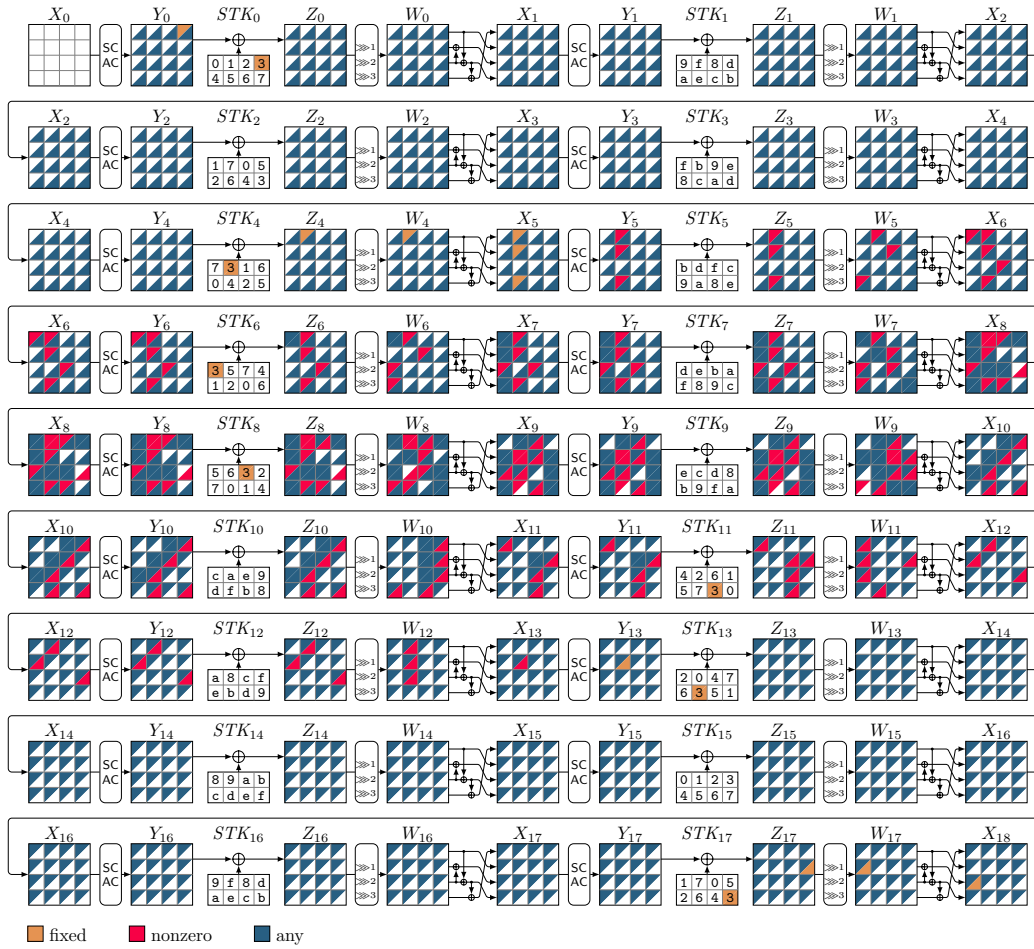


Figure 27: A cluster of 15 related-tweakey ID distinguishers for 18 rounds of ForkSKINNY-128-256, where $r_1 = 10$, $r_0 = 17$, and $r_1 = 8$.



Figure 28: A cluster of 15 related-tweakey ID distinguishers for 22 rounds of ForkSKINNY-64-192, where $r_1 = 8$, $r_0 = 15$, and $r_1 = 14$.

G Related-Tweakey ID Attack on ForkSKINNY

Similar to the tool in [HSE23], our tool for finding complete ID attack provides only an estimation of the time complexity for the Guess-and-Filter step in the ID attack. Therefore, after identifying the nearly optimal attack through our automated tool, we offer a more comprehensive complexity analysis by accurately calculating the complexity of each step within the Guess-and-Filter phase. Additionally, we incorporate Lemma 1 into the complexity analysis of our ID attacks.

Lemma 1. *For a given S -box S , and any input/output difference $\delta_i, \delta_o \neq 0$, the equation $S(x \oplus \delta_i) \oplus S(x) = \delta_o$ has one solution on average that can be derived efficiently. In addition, the inverse of S has a similar property.*

G.1 28-round Related-Tweakey ID Attack on ForkSKINNY-64-192-192 ($\mathbf{r}_i = 11, \mathbf{r}_o = 17$)

In this section, we introduce a 28-round ID attack on ForkSKINNY-64-192-192 within the related-tweakey setting. Figure 29 illustrates the attack identified by our tool. This attack occurs in a limited setting, where there are $\mathbf{r}_i = 17 - X$ rounds before the fork, and $\mathbf{r}_o = \mathbf{r}_i = 23 - X$ rounds in each branch after the fork, with $X = 6$.

In our case, the distinguisher is placed at rounds 5 to 24, and we use 225 related-key impossible differences for 20 rounds, with the input differences $\Delta Y_4 = 0\delta_0 000\delta'_0 0 \dots 0$, and $\Delta STK_4 = 0\delta_0 000\delta'_0 0 \dots 0$ and the output difference $\Delta Z_{23} = 00\delta_{18} 0000\delta'_{18} 0 \dots 0$, where δ_0 , and δ'_0 can take any nonzero differences ($1 \leq \delta_0, \delta'_0 \leq 15$) and δ_i (resp. δ'_i) is determined after a linear transformation implemented for i times with the LFSRs (in TK_2 , and TK_3) on δ_0 (resp. δ'_0)⁵. We extend the 20-round related-tweakey impossible differential 4 rounds on the top and 4 rounds at the bottom, which is illustrated in Figure 29.

Pair Generation We should prepare 2^x structures $S_t, 0 \leq t \leq 2^x - 1$ at W'_0 and evaluate all possible values in 13 cells $W'_0[1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15]$ for each structure, while the other cells assume a fixed value. For each plaintext P_u in S_t , we ask the encryption oracle to get $(C_u = E_K(P_u), \overline{C}_u = E_{K \oplus \Delta}(P_u))$, where

$$\Delta = \Delta ETK_0 = 000\Delta_1 000\Delta_1 0\Delta_2 00000\Delta_1, \quad \Delta_1, \Delta_2 \neq 0.$$

This step needs a total of 2^{x+52+1} encryption calls. By using 2^{x+52} plaintexts, we can have 2^{x+104} pairs of ciphertexts. The expected number of the remaining pairs of ciphertexts is approximately $N = 2^{x+104-(n-|\Delta_f|)} = 2^{x+96}$ pairs.

Guess-and-Filter

Step 1. For each guess of $ETK_0[0-3, 8-11] = \mathcal{K}_0$, encrypt all N pairs. For each guess, create a set \mathcal{L}_1 containing pairs that satisfy both ΔY_1 and Y_1 , as illustrated in Figure 29. Due to the MC^{-1} operation on active cells in X_2 , we have $\Delta Y_1[1] = \Delta Y_1[4] = \Delta Y_1[11]$, $\Delta Y_1[2] \oplus \Delta Y_1[8] = \Delta Y_1[15]$, $\Delta Y_1[3] \oplus \Delta Y_1[6] = \Delta Y_1[12]$, and $\Delta Y_1[6] = \Delta Y_1[9]$. Consequently, a 20-bit filter is established. On average, the size of a single \mathcal{L}_1 set is approximately $N2^{-20}$. The time complexity of this step is approximately $N2^{32}$.

Step 2. For each guess of \mathcal{K}_0 , and $STK_1[0-7] = \mathcal{K}_1$, encrypt all pairs within the corresponding \mathcal{L}_1 set associated with \mathcal{K}_0 . Additionally, construct a subset $\mathcal{L}_2 \subseteq \mathcal{L}_1$ that contains all the pairs satisfying ΔY_2 and Y_2 as shown in Figure 29. Due to

⁵Suppose the one-cell tweakey differences of TK_1 , TK_2 , and TK_3 are d_1 , d_2 , and d_3 , respectively. If we set $d_1 \oplus d_2 \oplus d_3 = \delta_0$, then $\delta_i = d_1 \oplus LFSR_2^i(d_2) \oplus LFSR_3^i(d_3)$.

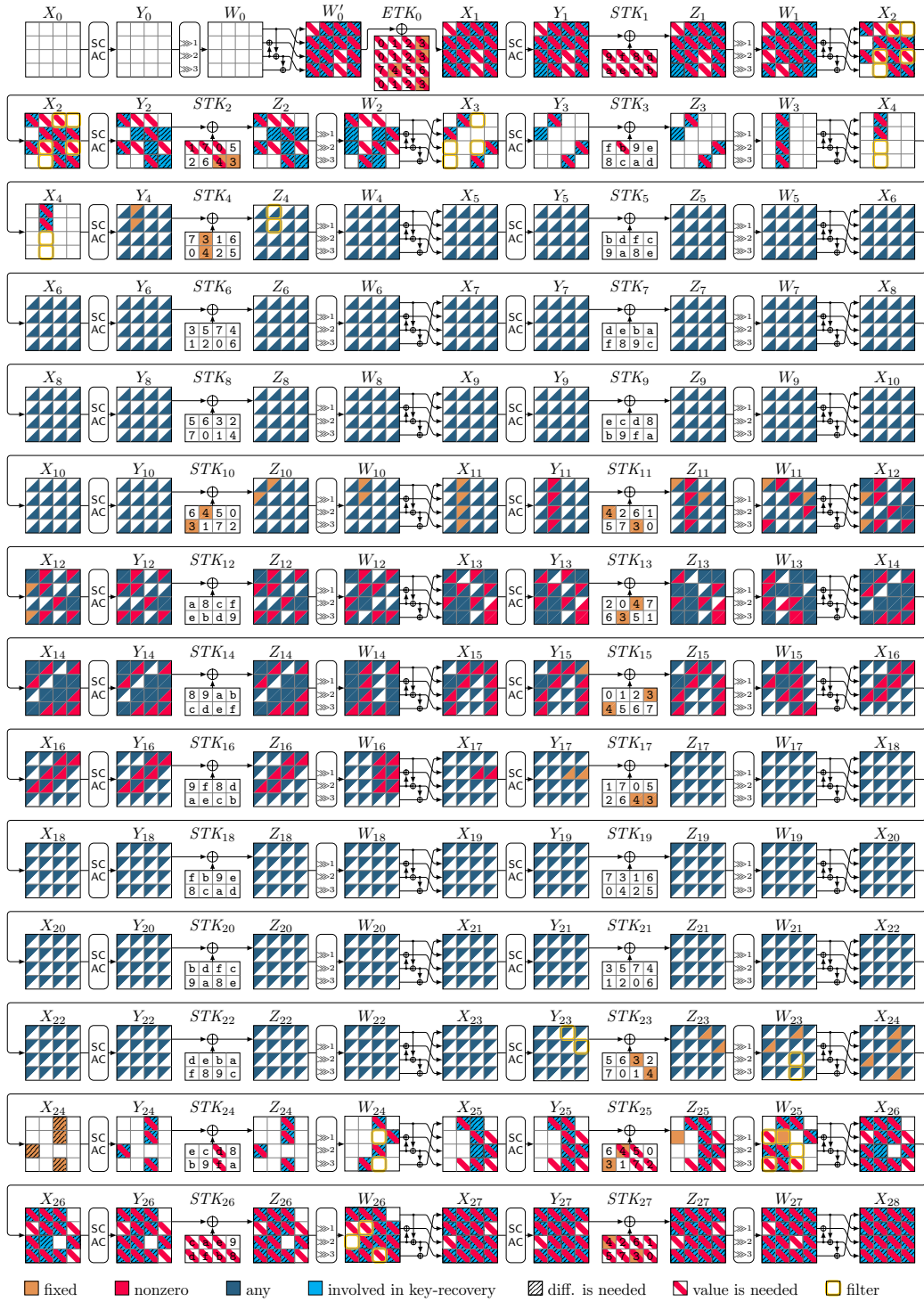


Figure 29: Related-Tweakey ID attack on 28 rounds of ForkSKINNY-64-192-192 in the limited setting ($r_i = 11, r_0 = 17$). $|k_B \cup k_F| = 160, c_B = 44, c_F = 56, \Delta_B = 52, \Delta_F = 56$. In this case we have 15×15 related-key impossible differences.

MC^{-1} operation on the active cells in the X_3 , we will have a 16-bit filter on ΔY_2 . On average, the size of a single \mathcal{L}_2 set is approximately $N2^{-20-16} = N2^{-36}$. The time complexity of this step is approximately $N2^{-20+32+32} = N2^{44}$.

Step 3. For each guess of \mathcal{K}_0 , \mathcal{K}_1 , and $\text{STK}_2[0-2, 6] \parallel \text{STK}_3[1] = \mathcal{K}_{2,3}$, encrypt all pairs within the corresponding \mathcal{L}_2 set associated with $\mathcal{K}_0 \parallel \mathcal{K}_1$. Additionally, construct a subset $\mathcal{L}_3 \subseteq \mathcal{L}_2$ that contains all the pairs satisfying ΔW_3 and W_3 as shown in Figure 29. Due to MC^{-1} operation on the active cells in the X_4 , we will have an 8-bit filter on ΔW_3 . Therefore, we can determine $\Delta Y_4[1, 5]$. On average, the size of a single \mathcal{L}_3 set is approximately $N2^{-36-8} = N2^{-44}$. The time complexity of this step is approximately $N2^{-36+32+32+20} = N2^{48}$. Here, the established relationships $\Delta Y_4[1] = \text{STK}_4[1]$, and $\Delta Y_4[5] = \text{STK}_4[5]$ have the potential to result in an 8-bit filter. Nevertheless, to improve the attack by exploiting multiple impossible differentials. In this step, we are aware that there are 15×15 distinct nonzero values for the pair $(\text{STK}_4[1], \text{STK}_4[5]) = (\delta_0, \delta'_0)$. To handle this, drawing inspiration from [SMB18], we utilize a hash table with 225 rows, labeled as $h(\delta_0, \delta'_0)$, where δ_0 ranges from 1 to 15 and δ'_0 ranges from 1 to 15. This hash table facilitates the partitioning of the remaining pairs within the \mathcal{L}_3 set. The partitioning is based on the values of $(\Delta Y_4[1], \Delta Y_4[5]) = (\delta_0, \delta'_0)$, which correspond to row (δ_0, δ'_0) in the hash table.

During the attack procedure, data associated with fixed nonzero differences $(\Delta Y_4[1], \Delta Y_4[5])$ is effectively stored within the $h(\delta_0, \delta'_0)$. We continue the attack in each set based on each row of the hash table.

Step 4. For each guess of \mathcal{K}_0 , \mathcal{K}_1 , $\mathcal{K}_{2,3}$, and $\text{STK}_{27}[0, 1, 3, 6, 7] = \mathcal{K}_{27}$ decrypt all pairs within the corresponding \mathcal{L}_3 set associated with $\mathcal{K}_0 \parallel \mathcal{K}_1 \parallel \mathcal{K}_{2,3}$. Additionally, construct a subset $\mathcal{L}_4 \subseteq \mathcal{L}_3$ that contains all the pairs satisfying ΔZ_{27} and Z_{27} as shown in Figure 29. There is no requirement for any tweaked information to compute $\Delta X_{27}[12]$. We get $\Delta X_{27}[12] = \Delta X_{27}[4]$ because of the MC operation on the active cells in the first column of W_{26} . From the knowledge of $\Delta X_{27}[4]$, we can determine $Y_{27}[4]$ by applying Lemma 1. Thus, we can determine $\text{STK}_{27}[4]$ (due to $\text{STK}_{27}[4] = Z_{27}[4] \oplus Y_{27}[7]$). Similarly, based on the MC operation on the active cells in the second, and the third column of W_{26} , Lemma 1 helps us to derive the tweakey cells $\text{STK}_{27}[2, 5]$. We compute Z_{26} and ΔZ_{26} as shown in Figure 29. We can determine $\Delta X_{26}[8, 12]$ from the knowledge of $\Delta Z_{26}[8, 12]$. Due to MC operation on the active cells in the first column of W_{25} , we have $\Delta X_{26}[0] = \Delta X_{26}[8] = \Delta X_{26}[12]$. Checking if $\Delta X_{26}[8] = \Delta X_{26}[12]$ will lead to a 4-bit filter. Also, the knowledge of $\Delta X_{26}[0]$ can help us to determine $Y_{26}[0]$ by applying Lemma 1, and so we can drive the tweakey cell $\text{STK}_{26}[0]$. Similarly, based on the MC operation on the active cells in the third column of W_{25} , and using Lemma 1, we can derive the tweakey cells $\text{STK}_{26}[2, 6]$. On average, the size of a single \mathcal{L}_4 set is approximately $N2^{-44-4} = N2^{-48}$. The time complexity of this step is approximately $N2^{-44+32+32+20+20} = N2^{60}$.

Step 5. For each guess of \mathcal{K}_0 , \mathcal{K}_1 , $\mathcal{K}_{2,3}$, \mathcal{K}_{27} , and $\text{STK}_{26}[5] = \mathcal{K}_{26}$ decrypt all pairs within the corresponding \mathcal{L}_4 set associated with $\mathcal{K}_0 \parallel \mathcal{K}_1 \parallel \mathcal{K}_{2,3} \parallel \mathcal{K}_{27}$. Additionally, construct a subset $\mathcal{L}_5 \subseteq \mathcal{L}_4$ that contains all the pairs satisfying $\Delta W_{25}[5]$ as shown in Figure 29. For each pair in the row (δ_0, δ'_0) of hash table h of set \mathcal{L}_5 , checking if $\Delta W_{25}[5] = \delta_{19}$ will generally lead to a 4-bit filter on all rows. On average, the size of a single \mathcal{L}_5 set is approximately $N2^{-48-4} = N2^{-52}$. The time complexity of this step is approximately $N2^{-48+32+32+20+20+4} = N2^{60}$.

Step 6. For each guess of \mathcal{K}_0 , \mathcal{K}_1 , $\mathcal{K}_{2,3}$, \mathcal{K}_{27} , \mathcal{K}_{26} , and $\text{STK}_{26}[1, 4, 7] = \mathcal{K}'_{26}$ decrypt all pairs within the corresponding \mathcal{L}_5 set associated with

$$\mathcal{K}_0 \parallel \mathcal{K}_1 \parallel \mathcal{K}_{2,3} \parallel \mathcal{K}_{27} \parallel \mathcal{K}_{26}.$$

Additionally, construct a subset $\mathcal{L}_6 \subseteq \mathcal{L}_5$ that contains all the pairs satisfying Z_{25} and ΔZ_{25} as shown in Figure 29. We know the values of $ETK_0[8, 9]$, $STK_2[1, 6]$, and $STK_{27}[0, 5]$ from the previous steps. Therefore, we can deduce $STK_{25}[1, 6]$. The values $STK_{25}[6]$ help us to compute $\Delta X_{25}[6]$. Due to MC operation on the active cells in the third column of W_{24} , we have $\Delta X_{25}[6] \oplus \Delta X_{25}[10] = \Delta X_{25}[14]$. This equality will lead to a 4-bit filter. Also, Due to MC operation on the active cells in the third column of W_{24} , we have $\Delta X_{25}[14] = \Delta X_{25}[2]$. From the knowledge of $\Delta X_{25}[2]$, Lemma 1 helps us to derive $STK_{25}[2]$. On average, the size of a single \mathcal{L}_6 set is approximately $N2^{-52-4} = N2^{-56}$. The time complexity of this step is approximately $N2^{-52+32+32+20+20+4+12} = N2^{68}$.

Step 7. For each guess of $\mathcal{K}_0, \mathcal{K}_1, \mathcal{K}_{2,3}, \mathcal{K}_{27}, \mathcal{K}_{26}, \mathcal{K}'_{26}$, and $STK_{25}[7] = \mathcal{K}_{25}$ decrypt all pairs within the corresponding \mathcal{L}_6 set associated with

$$\mathcal{K}_0 || \mathcal{K}_1 || \mathcal{K}_{2,3} || \mathcal{K}_{27} || \mathcal{K}_{26} || \mathcal{K}'_{26}.$$

Additionally, construct a subset $\mathcal{L}_7 \subseteq \mathcal{L}_6$ that contains all the pairs satisfying Z_{24} and ΔZ_{24} as shown in Figure 29. Due to MC operation on the active cells in the third column of W_{23} , we have $\Delta X_{24}[2] = \Delta X_{24}[6] = \Delta X_{24}[14]$. There is no requirement for any tweaked information to compute $\Delta X_{24}[14]$. Therefore, we can determine $\Delta X_{24}[2]$, and $\Delta X_{24}[6]$. Now, Lemma 1 helps us to determine $STK_{24}[2, 6]$. We compute Z_{23} and ΔZ_{23} as shown in Figure 29. For each pair in the row (δ_0, δ'_0) of hash table h of \mathcal{L}_7 set, checking if $\Delta Z_{23}[2] = \delta_{18}$, and also $\Delta Z_{23}[7] = \delta'_{18}$ will generally lead to an 8-bit filter on all pairs in h . On average, the size of a single \mathcal{L}_6 set is approximately $N2^{-56-8} = N2^{-64}$. The time complexity of this step is approximately $N2^{-56+32+32+20+20+4+12+4} = N2^{68}$.

We can confirm that a guess is an incorrect key guess if and only if one of the N pairs under the guess satisfies the following property:

$$\Delta Y_4 = 0\delta_0 000\delta'_0 0 \dots 0, \Delta Z_{23} = 00\delta_{18} 0000\delta'_{18} 0 \dots 0.$$

After the above steps, each of the rows in each of \mathcal{L}_7 sets which is nonempty suggests the wrong keys. The probability of a given pair surviving the filtering step satisfying ΔY_4 , and ΔZ_{23} under a random key guess is about $(1 - 2^{-100})^{2^{x+96}} = e^{-2^{x-4}}$. Therefore, the total number of remaining tweakeys is $2^{|\mathcal{K}_B \cup \mathcal{K}_F|} \times e^{-2^{x-4}} = 2^{160} \times e^{-2^{x-4}}$. We guess the remaining 32-bit tweakey cells and exhaustively search the $2^{32} \times 2^{160} \times e^{-2^{x-4}}$ tweakeys to find the correct tweakey.

Complexity analysis The time complexity of the pair generation step, analyzing N pairs, and exhausted search is about:

$$2^{x+53} + \frac{1}{28} \times (N2^{32} + N2^{44} + N2^{48} \times 2 + N2^{60} \times 2 + N2^{68} \times 2) + 2^{192} \times e^{-2^{x-4}}$$

28-round encryptions. The attack needs a data complexity of $D = 2^{x+52}$. Memory complexity is storing plaintext/ciphertext pairs. Hence, to optimize the time complexity of the attack, we select $x = 8$. Thus, the data, time, and memory complexities of the attack on ForkSKINNY are 2^{60} , $2^{169.60}$, and 2^{104} , respectively.

We can also find the specifics of this attack in Table 5, which is divided into three parts: *Pair Generation*, *Guess-and-Filter*, and *Complexity analysis*. In this table, for the sake of simplicity, we utilize the notation $\Delta X[a = b = c]$ or $\Delta X[a \oplus b = c]$ in place of the lengthier expressions $\Delta X[a] = \Delta X[b] = \Delta X[c]$, or $\Delta X[a] \oplus \Delta X[b] = \Delta X[c]$, respectively. Moreover, the keys enclosed within parentheses represent the keys that are derivable from one another through the linear relationships established by the key schedule algorithm.

For other (Related-Tweakey) ID attacks, we follow the approach outlined in this section. To streamline the presentation, avoid redundancy, and provide comprehensive details about each attack, we will present the attack specifics in their respective figures and tables.

Table 5: Key-recovery procedure for ID attack on 28 rounds of ForkSKINNY-64-192-192, where $\mathbf{r}_i = 17 - X$, $\mathbf{r}_0 = \mathbf{r}_1 = 23 - X$ for $X = 6$.

		Pair Generation			
	# Plaintexts	# Pairs	N	# Enc.	
	2^{x+52}	2^{x+104}	2^{x+96}	2^{x+53}	
		Guess-and-Filter			
Step	Gussed keys {#}	Condition(s) {#filters}	# Remaining pairs ($ \mathcal{L}_i $)	Time	Deduced
$i=1$	$ETK_0[0-3, 8-11] = \mathcal{K}_0$ {32}	$\Delta Y_1[1 = 4 = 11]$ $\Delta Y_1[2 \oplus 8 = 15]$ $\Delta Y_1[3 \oplus 6 = 12]$ $\Delta Y_1[6 = 9]$ {20}	$N2^{-20}$	$N2^{32}$	$\Delta Y_1, Y_1$
$i=2$	\mathcal{K}_0 $STK_1[0-7] = \mathcal{K}_1$ {32}	$\Delta Y_2[0 = 7 = 10]$ $\Delta Y_2[5 = 8 = 15]$ {16}	$N2^{-36}$	$N2^{44}$	$\Delta Y_2, Y_2$
$i=3$	$\mathcal{K}_0, \mathcal{K}_1$ $STK_2[0-2, 6]$ $STK_3[1] = \mathcal{K}_{2,3}$ {20}	$\Delta W_3[1 = 5 = 9]$ {8}	$N2^{-44}$	$N2^{48}$	$\Delta W_3, W_3$
		$(\Delta Y_4[1], \Delta Y_4[5]) = (\delta_0, \delta'_0)$ (Building hash table h)			$\Delta Y_4[1, 5]$
$i=4$	$\mathcal{K}_0, \mathcal{K}_1, \mathcal{K}_{2,3}$ $STK_{27}[0, 1, 3, 6, 7]$ $= \mathcal{K}_{27}$ {20}	$\Delta X_{27}[12 = 4]$ $\Delta X_{27}[5 \oplus 9 = 13]$ $\Delta X_{27}[2 = 14]$ $\Delta X_{26}[8 = 12]$ {4} $\Delta X_{26}[0 = 8]$ $\Delta X_{26}[2 = 6 = 14]$	$N2^{-48}$	$N2^{60}$	$STK_{27}[4]$ $STK_{27}[5]$ $STK_{27}[2]$ $Z_{26}, \Delta Z_{26}$ $STK_{26}[0]$ $STK_{26}[2, 6]$
$i=5$	$\mathcal{K}_0, \mathcal{K}_1, \mathcal{K}_{2,3}, \mathcal{K}_{27}$ $STK_{26}[5] = \mathcal{K}_{26}$ {4}	$\Delta W_{25}[5] = \delta_{19}$ {4}	$N2^{-52}$	$N2^{60}$	
$i=6$	$\mathcal{K}_0, \mathcal{K}_1, \mathcal{K}_{2,3}, \mathcal{K}_{27}, \mathcal{K}_{26}$ $STK_{26}[1, 4, 7] = \mathcal{K}'_{26}$ ($ETK_0[8, 9]$) ($STK_2[1, 6]$) ($STK_{27}[0, 5]$) {12}	$\Delta X_{25}[6 \oplus 10 = 14]$ {4} $\Delta X_{25}[14 = 2]$	$N2^{-56}$	$N2^{68}$	$Z_{25}, \Delta Z_{25}$ ($STK_{25}[1, 6]$) $STK_{25}[2]$
$i=7$	$\mathcal{K}_0, \mathcal{K}_1, \mathcal{K}_{2,3}, \mathcal{K}_{27}, \mathcal{K}_{26},$ \mathcal{K}'_{26} $STK_{25}[7] = \mathcal{K}_{25}$ {4}	$\Delta X_{24}[2 = 6 = 14]$ $\Delta Z_{23}[2] = \delta'_{18}$ $\Delta Z_{23}[7] = \delta'_{18}$ {8}	$N2^{-64}$	$N2^{68}$	$Z_{24}, \Delta Z_{24}$ $STK_{24}[2, 6]$ $Z_{23}, \Delta Z_{23}$
Total	-	-	$N2^{64.19}$ 28-round Enc.		
		Complexity analysis			
x	Data	Memory	Total time		
8	2^{60}	2^{104}	$2^{61} + N2^{64.19} + 2^{192}e^{-2^x-4} = 2^{169.60}$		

G.2 28-round Related-Tweakey ID Attack on ForkSKINNY-64-192-128 ($\mathbf{r}_i = 11, \mathbf{r}_0 = 17$)

The 28-round related-tweakey ID Attack on ForkSKINNY-64-192-128 occurs in a limited setting, where there are $\mathbf{r}_i = 17 - X$ rounds before the fork, and $\mathbf{r}_0 = \mathbf{r}_1 = 23 - X$ rounds in each branch after the fork, with $X = 6$. We give the pattern of the attack in Figure 30.

Since the attack given in this section has the same structure as observed in the 30-round related-tweakey ID Attack on ForkSKINNY-64-192-128 (refer to Subsection G.4, Figure 32), it

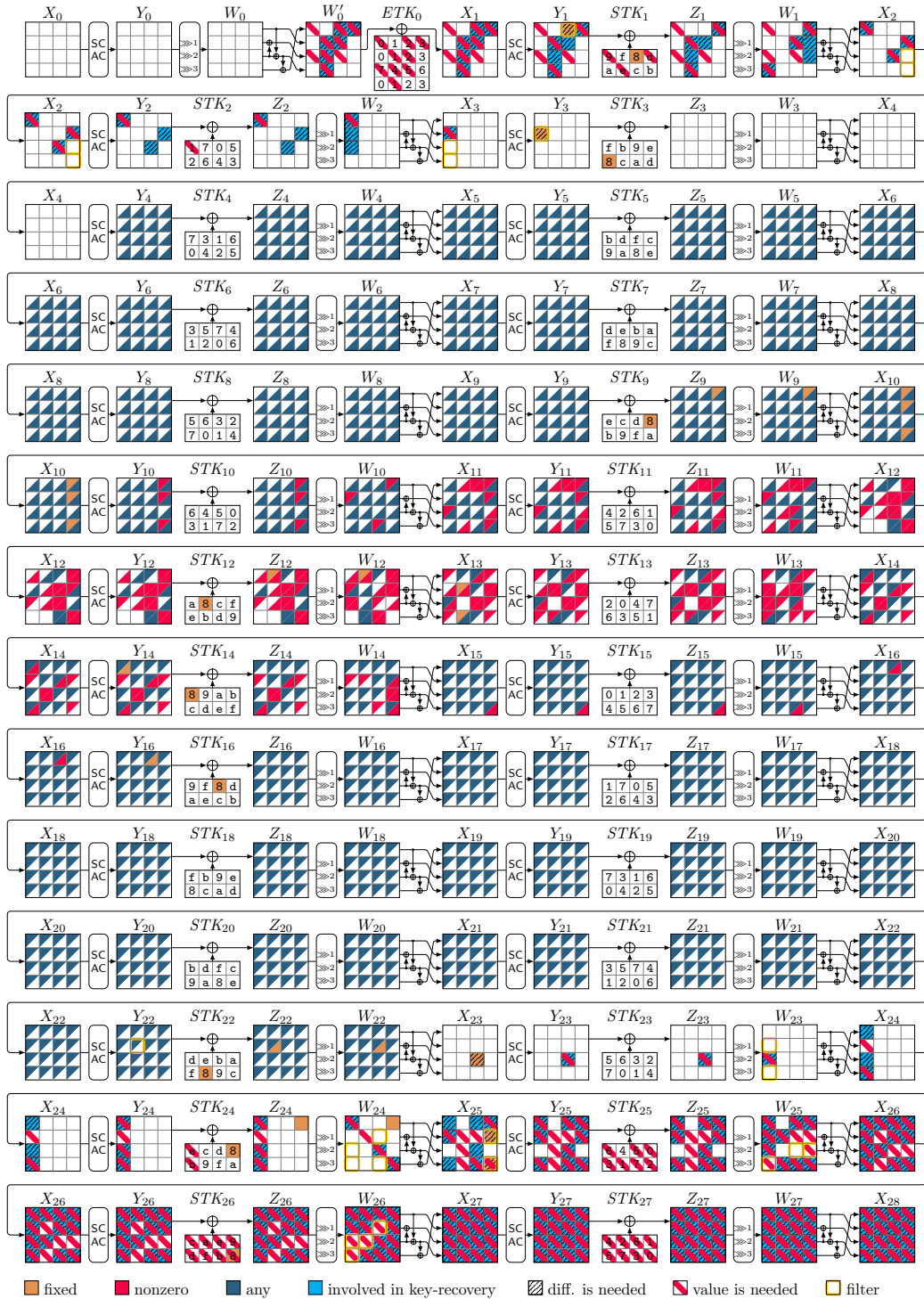


Figure 30: Related-Tweakey ID attack on 28 rounds of ForkSKINNY-64-192-128 in the limited setting ($r_i = 11, r_0 = 117$). $|k_B \cup k_F| = 112$, $c_B = 20$, $c_F = 64$, $\Delta_B = 24$, $\Delta_F = 64$.

becomes feasible to reuse the key recovery part of the aforementioned attack. Consequently, we proceed to target a modified version of ForkSKINNY-64-192-128 with 28 rounds in a constrained scenario, employing identical computational complexity as detailed in the attack of Subsection G.4. Therefore, the attack's data, memory, and time complexities are 2^{62} , 2^{86} , and $2^{123.73}$, respectively.

G.3 32-round Related-Tweakey ID Attack on ForkSKINNY-64-192-192 ($r_i = 11, r_0 = 15$)

Since the goal is to recover the full 192-bit tweakey, the generic bound is 2^{192} . We give the pattern of the attack in Figure 31. The key recovery details are given in Table 6.



Figure 31: Related-Tweakey ID attack on 32 rounds of ForkSKINNY-64-192-192 in the arbitrary setting ($\mathbf{r}_i = 11, \mathbf{r}_0 = 15$). $|k_B \cup k_F| = 176, c_B = 48, c_F = 64, \Delta_B = 52, \Delta_F = 64$. In this case we have 15 related-key impossible differences.

Table 6: Key-recovery procedure for ID attack on 32 rounds of ForkSKINNY-64-192-192, where $\mathbf{r}_i = 11$, $\mathbf{r}_0 = 15$, and $\mathbf{r}_1 = 21$.

		Pair Generation				
		# Plaintexts	# Pairs	N	# Enc.	
		2^{x+52}	2^{x+104}	2^{x+104}	2^{x+53}	
Guess-and-Filter						
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs($ \mathcal{L}_i $)	Time	Deduced	
$i=1$	$STK_{31}[0-7] = \mathcal{K}_{31}$ { 32 }	$\Delta X_{31}[4 \oplus 8 = 12]$ $\Delta X_{31}[5 \oplus 9 = 13]$ $\Delta X_{31}[6 = 14]$ $\Delta X_{31}[3 = 15]$ { 16 }	$N2^{-16}$	$N2^{32}$	$Z_{30}[0-15]$ $\Delta Z_{30}[0-15]$	
$i=2$	\mathcal{K}_{31} $STK_{30}[0-7] = \mathcal{K}_{30}$ { 32 }	$\Delta X_{30}[5 \oplus 9 = 13]$ $\Delta X_{30}[2 \oplus 6 = 14]$ $\Delta X_{30}[3 = 15]$ { 16 }	$N2^{-32}$	$N2^{48}$	$Z_{29}[0-15]$ $\Delta Z_{29}[0-15]$	
$i=3$	$\mathcal{K}_{31}, \mathcal{K}_{30}$ $STK_{29}[0, 1, 2, 4-7] = \mathcal{K}_{29}$ { 28 }	$\Delta X_{29}[2 = 6 = 14]$ { 8 }	$N2^{-40}$	$N2^{60}$	$Z_{28}[0-15]$ $\Delta Z_{28}[0-15]$ $\Delta X_{28}[14]$	
$i=4$	$\mathcal{K}_{31}, \mathcal{K}_{30}, \mathcal{K}_{29}$ $STK_{28}[1, 7] = \mathcal{K}_{28}$ { 8 }	$\Delta X_{28}[2 = 6 = 14]$ $\Delta X_{27}[2 = 6 = 14]$ $\Delta Z_{26}[2] = \delta_{19}(= \Delta STK_{26}[2])$ { 4 }	$N2^{-44}$	$N2^{60}$	$STK_{28}[2, 6]$ $Z_{27}[0-15]$ $\Delta Z_{27}[0-15]$ $STK_{27}[2, 6]$ $\Delta Z_{26}[2]$	
$i=5$	$\mathcal{K}_{31}, \mathcal{K}_{30}, \mathcal{K}_{29}, \mathcal{K}_{28}$ $ETK_0[11] = \mathcal{K}_0$ { 4 }		$N2^{-44}$	$N2^{60}$		
	$(ETK_0[8, 10])$ $(ETK_0[3])$ $(ETK_0[13])$	$\Delta Y_1[2 = 5 = 8]$ $\Delta Y_1[7 = 10]$ $\Delta Y_1[3 = 9]$ $\Delta Y_1[0 \oplus 13 = 7]$			$ETK_0[2, 5]$ $ETK_0[7]$ $ETK_0[9]$ $ETK_0[0]$ $Y_1, \Delta Y_1, \Delta X_2$	
$i=6$	$\mathcal{K}_{31}, \mathcal{K}_{30}, \mathcal{K}_{29}, \mathcal{K}_{28}, \mathcal{K}_0$ $STK_1[0, 2] = \mathcal{K}_1$ { 8 }	$\Delta Y_2[4 = 1 = 11]$ $X_2[1] = W_1[1 \oplus 9 \oplus 13]$ $X_2[11] = W_1[7 \oplus 9 \oplus 11]$ $\Delta Y_2[6 = 9] = \Delta Y_2[9]$ $X_2[9] = W_1[5 \oplus 9]$ $\Delta Y_2[6 \oplus 12 = 3]$ $X_2[3] = W_1[3 \oplus 11 \oplus 15]$	$N2^{-48}$	$N2^{68}$	$\Delta Y_2[4, 12]$ $X_2[1, 11]$ $STK_1[1]$ $STK_1[6]$ $X_2[9]$ $STK_1[4]$ $X_2[3]$ $STK_1[3]$ $\Delta Y_3[7]$	
	$(STK_2[3])$	$\Delta Y_3[7] = \delta_{-1}(= \Delta STK_3[7])$ { 4 }				
$i=7$	$\mathcal{K}_{31}, \mathcal{K}_{30}, \mathcal{K}_{29}, \mathcal{K}_{28}, \mathcal{K}_0, \mathcal{K}_1$ $STK_1[5, 7] = \mathcal{K}'_1$ { 8 }		$N2^{-56}$	$N2^{72}$	$Y_2, \Delta Y_2$	
	$(STK_2[1-4, 6, 7])$	$\Delta Y_3[5 = 8 = 12]$ { 8 }			$Y_3, \Delta Y_3$	
$i=8$	$\mathcal{K}_{31}, \mathcal{K}_{30}, \mathcal{K}_{29}, \mathcal{K}_{28}, \mathcal{K}_0, \mathcal{K}_1, \mathcal{K}'_1$ $STK_3[2] = \mathcal{K}_3$ $(STK_3[1, 6])$ $(STK_4[1])$		$N2^{-64}$	$N2^{68}$	$\Delta Y_5[1]$	
		$\Delta Y_5[1] = \delta(= \Delta STK_5[1])$ { 4 }				
Total	-	-		$N2^{67.17}$	32-round Enc.	
Complexity analysis						
x	Data	Memory	Total time			
10	2^{62}	2^{114}	$2^{63} + N2^{67.17} + 2^{192}e^{-2^{x-8}} = 2^{186.27}$			

G.4 30-round Related-Tweakey ID Attack on ForkSKINNY-64-192-128 ($r_i = 9, r_0 = 15$)

In this section, we give a 30-round related-tweakey ID Attack on ForkSKINNY-64-192-128. Since the goal is to recover the full 128-bit tweakey, the generic bound is 2^{128} . We give the pattern of the attack in [Figure 32](#). The key recovery details are given in [Table 7](#).

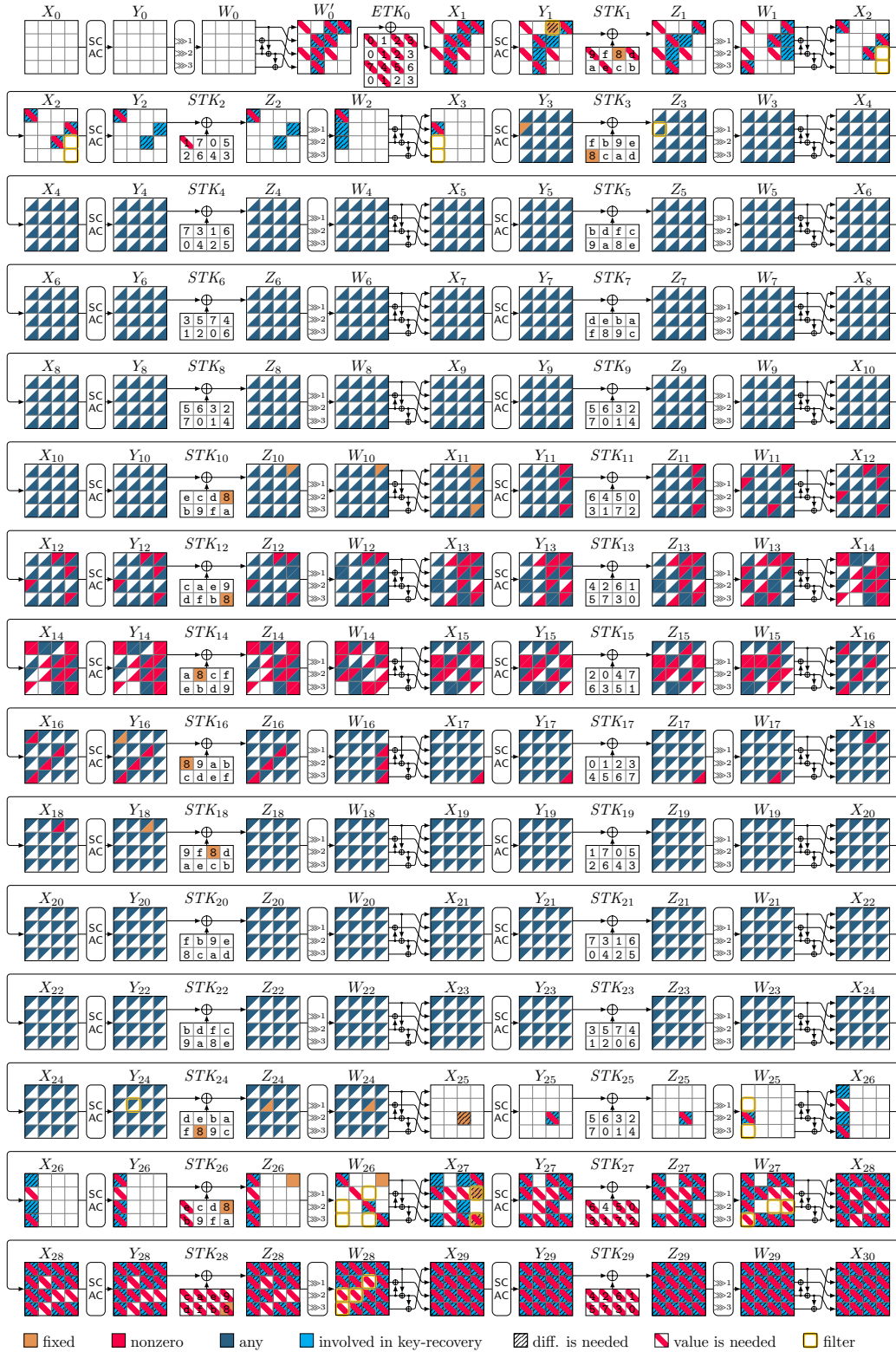


Figure 32: Related-Tweakey ID attack on 30 rounds of ForkSKINNY-64-192-128 in the arbitrary setting ($r_1 = 9, r_0 = 15$). $|k_B \cup k_F| = 112$, $c_B = 20$, $c_F = 64$, $\Delta_B = 24$, $\Delta_F = 64$.

Table 7: Key-recovery procedure for ID attack on 30 rounds of ForkSKINNY-64-192-128, where $\mathbf{r}_i = 9$, $\mathbf{r}_0 = 15$, and $\mathbf{r}_1 = 21$.

		Pair Generation			
		# Plaintexts	# Pairs	N	# Enc.
		2^{x+24}	2^{x+48}	2^{x+48}	2^{x+25}
		Guess-and-Filter			
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs(\mathcal{L}_i)	Time	Deduced
$i=1$	$ETK_0[2] $ $STK_{29}[0-7] = \mathcal{K}_{0,29}$ {36}	$\Delta Y_1[2] = \delta_{-1}(\Delta STK_1[2])$ $\Delta W_{28}[6 = 8 = 9 = 12] = 0$ {16} $\Delta X_{28}[0 = 12]$ $\Delta X_{28}[6 = 14]$ $\Delta X_{28}[7 = 15]$	$N2^{-16}$	$N2^{36}$	$Z_{28}, \Delta Z_{28}$ $\Delta X_{28}[12, 14, 15]$ $STK_{28}[0]$ $STK_{28}[6]$ $STK_{28}[7]$
	$\mathcal{K}_{0,29}$ $STK_{28}[1-5] = \mathcal{K}_{28}$ {20}	$\Delta X_{27}[0 = 4 = 12]$ $\Delta X_{27}[10 = 2]$ $\Delta X_{27}[10 = 14]$ {4} $\Delta X_{27}[7 = 15]$ $\Delta X_{27}[7] = \delta_{20}(\Delta STK_{26}[3])$ {4}	$N2^{-24}$	$N2^{40}$	$Z_{27}, \Delta Z_{27}$ $STK_{27}[0, 4]$ $STK_{27}[2]$ $STK_{27}[7]$ $(STK_{27}[3])$
$i=2$	$\mathcal{K}_{0,29}, \mathcal{K}_{28}$ $STK_{27}[5, 6] = \mathcal{K}_{27}$ {8}	$\Delta X_{26}[8 = 12]$ {4} $\Delta X_{26}[0 = 8]$	$N2^{-28}$	$N2^{40}$	$Z_{26}, \Delta Z_{26}$ $STK_{26}[0]$
$i=3$	$\mathcal{K}_{0,29}, \mathcal{K}_{28}, \mathcal{K}_{27}$ $STK_{27}[4] = \mathcal{K}_{26}$ {4}	$\Delta Z_{24}[5] = \delta_{19}(\Delta STK_{24}[5])$ {4}	$N2^{-32}$	$N2^{40}$	$\Delta Z_{24}[5]$ $(ETK_0[2, 3, 5, 8, 10])$
	$(STK_{29}[1, 3-6])$ $(STK_{27}[2, 4-7])$				
$i=4$	$\mathcal{K}_{0,29}, \mathcal{K}_{28}, \mathcal{K}_{27}, \mathcal{K}_{26}$ $ETK_0[9] = \mathcal{K}_0$ {4}	$\Delta Y_1[3 = 6 = 9]$ {8}	$N2^{-44}$	$N2^{40}$	$\Delta Y_1, Y_1$
	$(STK_{28}[2], STK_{26}[0])$	$\Delta Y_2[10 = 7 = 0]$ $X_2[0] = W_1[0 \oplus 8 \oplus 12]$ $X_2[7] = W_1[3]$			$(STK_1[5])$ $\Delta Y_2[10]$ $X_2[7, 0]$ $STK_1[0]$ $STK_1[3]$ $\Delta Y_2, Y_2$ $(STK_2[0])$ $\Delta Y_3[4]$
$i=5$	$(STK_{29}[3], STK_{27}[5])$	$\Delta Y_3[4] = \delta_0(\Delta STK_3[4])$ {4}			
Total	-	-		$N2^{37.11}$	30-round Enc.
		Complexity analysis			
x	Data	Memory	Total time		
38	2^{62}	2^{86}	$2^{63} + N2^{37.11} + 2^{128} e^{-2^{x-36}} = 2^{123.73}$		

G.5 26-round Related-Tweakey ID Attack on ForkSKINNY-128-256-256 ($r_i = 9, r_o = 17$)

In this section, we give a 26-round related-tweakey ID Attack on ForkSKINNY-128-256-256. Since the goal is to recover the full 256-bit tweakey, the generic bound is 2^{256} . We give the pattern of the attack in Figure 33. The key recovery details are given in Table 8.

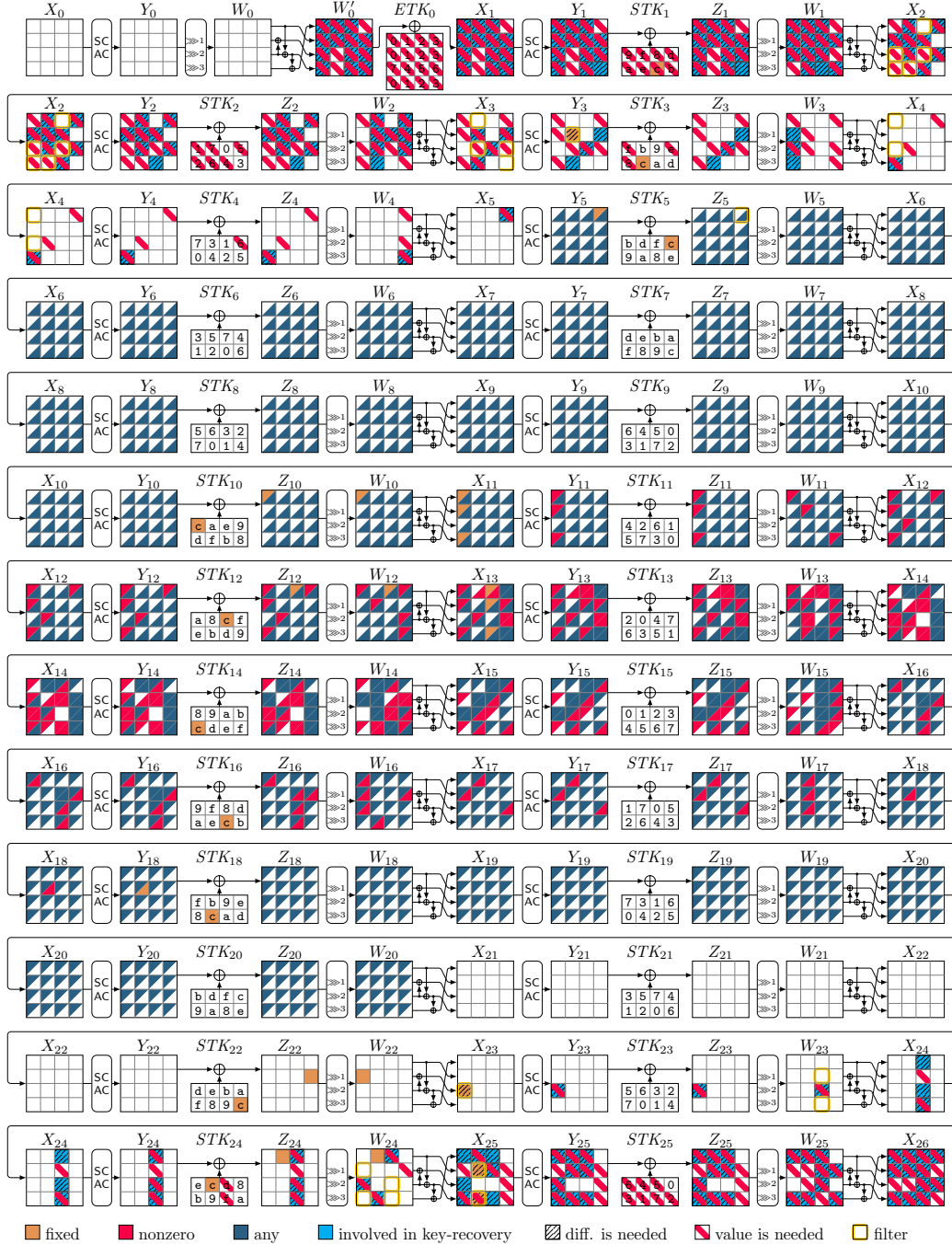


Figure 33: Related-Tweakey ID attack on 26 rounds of ForkSKINNY128-256-256 in the arbitrary setting ($r_i = 9, r_o = 17$). $|k_B \cup k_F| = 216$, $c_B = 96$, $c_F = 72$, $\Delta_B = 104$, $\Delta_F = 72$. In this case we have 15 related-key impossible differences.

Table 8: Key-recovery procedure for ID attack on 26 rounds of ForkSKINNY-128-256-256, where $\mathbf{r}_i = 9$, $\mathbf{r}_0 = 17$, and $\mathbf{r}_1 = 17$.

		Pair Generation			
	# Plaintexts	# Pairs	N	# Enc.	
	2^{x+104}	2^{x+208}	2^{x+152}	2^{x+105}	
		Guess-and-Filter			
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs(\mathcal{L}_i)	Time	
				Deduced	
$i=1$	$STK_{25}[1, 4, 7] = \mathcal{K}_{25}$ {24}	$\Delta X_{25}[5] = \delta_{19}(\Delta STK_{24}[1])$ (15×2^{-8} filters) \rightarrow {4} $\Delta X_{25}[8 = 12]$ {8} $\Delta X_{25}[0 = 12]$ $\Delta X_{25}[5 = 13]$ $\Delta X_{25}[2 = 6 = 14]$ $\Delta X_{24}[10 = 14]$ {8} $\Delta X_{24}[2 = 10]$	$N2^{-20}$	$N2^{24}$	$\Delta X_{25}[8, 12, 13, 14]$ $STK_{25}[0]$ $STK_{25}[5]$ $STK_{25}[2, 6]$ $\Delta Z_{24}, Z_{24}$ $STK_{24}[2]$
$i=2$	\mathcal{K}_{25} $STK_{24}[6] = \mathcal{K}_{24}$ {8}	$\Delta Z_{22}[7] = \delta_{18}(\Delta STK_{22}[7])$ {8}	$N2^{-28}$	$N2^{12}$	
$i=3$	$\mathcal{K}_{25}, \mathcal{K}_{24}$ $ETK_0[1, 3, 9] $ $STK_1[1] = \mathcal{K}_{0,1}$ {32}	$\Delta Y_1[0 = 7 = 10]$ $\Delta Y_1[1 = 11]$ $\Delta Y_1[5 = 8 = 2 \oplus 15]$ $\Delta Y_3[5] = \delta_{-1}(\Delta STK_3[5])$ {8}	$N2^{-36}$	$N2^{36}$	$\Delta Y_1[1, 5, 7, 15]$ $ETK_0[0, 10]$ $ETK_0[11]$ $ETK_0[2, 8]$ $\Delta Y_1, Y_1$ ($STK_2[0, 1, 3-6]$)
$i=4$	$\mathcal{K}_{25}, \mathcal{K}_{24}, \mathcal{K}_{0,1}$ $STK_1[0] = \mathcal{K}_1$ {8}	$\Delta Y_2[1 \oplus 14 = 4 = 11]$ $X_2[11] = W_1[7 \oplus 11]$ $X_2[14] = W_1[2 \oplus 10]$ $\Delta Y_2[3 = 6 = 9]$ $X_2[3] = W_1[3 \oplus 11 \oplus 15]$ $X_2[9] = W_1[5 \oplus 9]$ $\Delta Y_3[7 = 13]$ {8} $\Delta Y_3[7 = 10]$ $X_3[10] = W_2[6 \oplus 10]$ $X_2[8] = W_1[7 \oplus 11]$	$N2^{-44}$	$N2^{36}$	$\Delta Y_2[4]$ $X_2[11, 14]$ $STK_1[6]$ $STK_1[2], \Delta Y_2[6]$ $X_2[3, 9]$ $STK_1[3]$ $STK_1[4]$ $\Delta Y_3[7, 13]$ $X_3[10]$ $W_2[10], X_2[8]$ $STK_1[7]$
$i=5$	$\mathcal{K}_{25}, \mathcal{K}_{24}, \mathcal{K}_{0,1}, \mathcal{K}_1$ $STK_1[5] STK_3[3, 4] =$ $\mathcal{K}_{1,3}$ {24} ($STK_{24}[6], STK_1[1]$)	$\Delta Y_5[3] = \delta_0$ {8}	$N2^{-68}$	$N2^{52}$	$\Delta Y_3, Y_3$ ($STK_3[0]$)
Total	-	-	$N2^{48.29}$ 26-round Enc.		
Complexity analysis					
x	Data	Memory	Total time		
23	2^{127}	2^{175}	$2^{128} + N2^{48.29} + 2^{256}e^{-2^{x-20}} = 2^{244.45}$		
23.6	$2^{127.6}$	$2^{175.6}$	$2^{128.6} + N2^{48.29} + 2^{256}e^{-2^{x-20}} = 2^{238.5}$		

G.6 24-round Related-Tweakey ID Attack on ForkSKINNY-128-256-128 ($r_i = 17, r_0 = 27$)

In this section, we give a 24-round related-tweakey ID Attack on ForkSKINNY-128-256-128. Since the goal is to recover the full 128-bit tweakey, the generic bound is 2^{128} . We give the pattern of the attack in Figure 34. The key recovery details are given in Table 9.

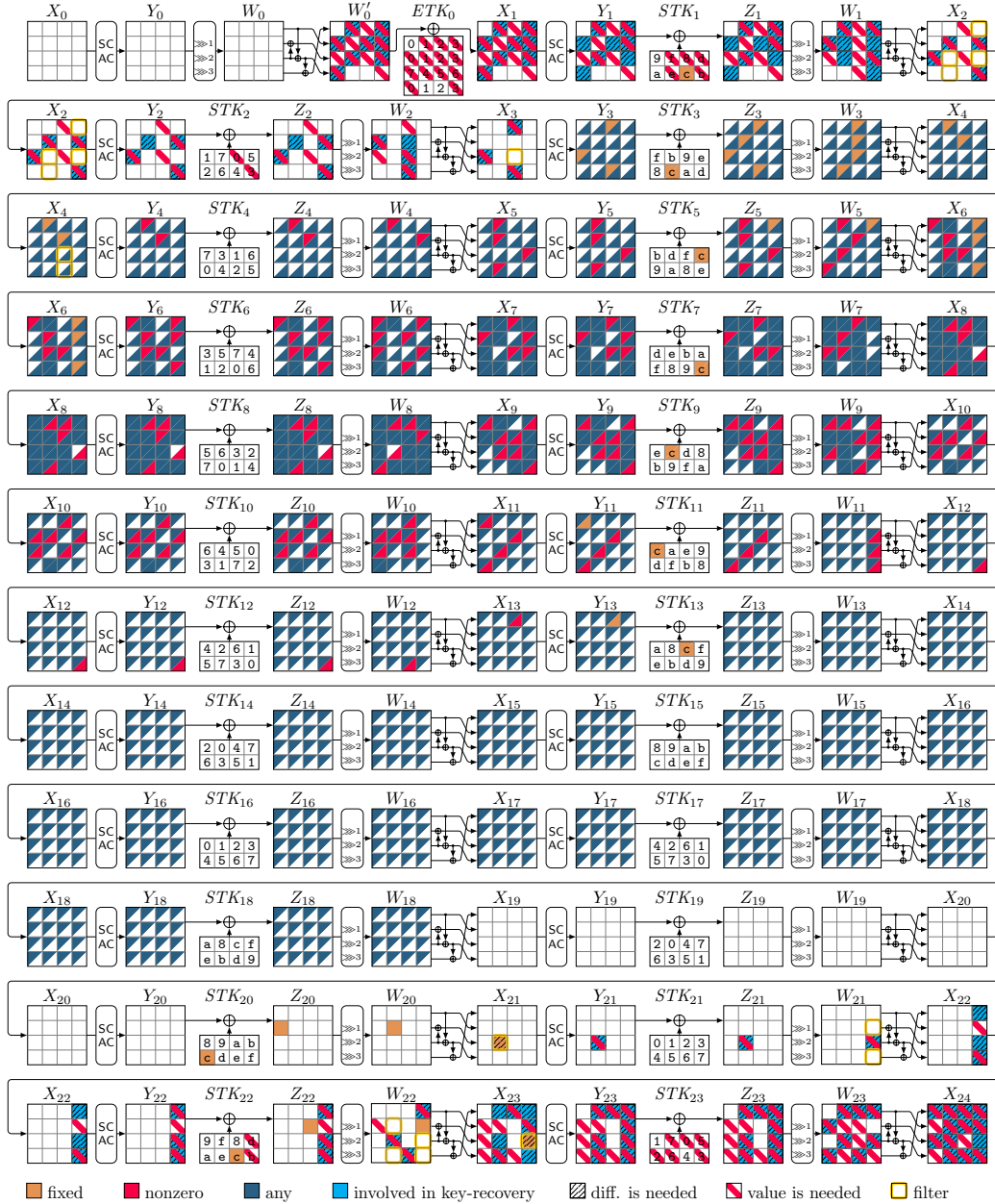


Figure 34: Related-Tweakey ID attack on 24 rounds of ForkSKINNY128-256-128 in the arbitrary setting ($r_i = 17, r_0 = 27$). $|k_B \cup k_F| = 104$, $c_B = 48$, $c_F = 64$, $\Delta_B = 64$, $\Delta_F = 64$. In this case, we have 15 related-key impossible differences

Table 9: Key-recovery procedure for ID attack on 24 rounds of ForkSKINNY-128-256-128, where $\mathbf{r}_i = 17$, $\mathbf{r}_0 = 27$, and $\mathbf{r}_1 = 7$.

		Pair Generation				
	# Plaintexts	# Pairs	N		# Enc.	
		2^{x+64}	2^{x+128}	2^{x+64}		2^{x+65}
Guess-and-Filter						
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs ($ \mathcal{L}_i $)	Time	Deduced	
$i=1$	$STK_{23}[5] = \mathcal{K}_{23}$ {8}	$\Delta X_{23}[9 = 13]$ {8} $\Delta X_{23}[1 = 9]$ $\Delta X_{23}[3 = 7 = 15]$ $\Delta X_{23}[11] = \delta_{23}(\Delta STK_{22}[6])$ $(15 \times 2^{-8}) \rightarrow$ {4} $\Delta Y_1[1 = 4 = 11]$	$N2^{-20}$	$N2^8$	$ETK_0[11]$ $STK_{23}[1]$ $STK_{23}[3, 7]$ $ETK_0[1, 4]$ $ETK_0[12]$ $(ETK_0[3])$ $ETK_0[6, 9]$ $Y_1, \Delta Y_1$ $Z_{22}, \Delta Z_{22}$	
		$(STK_{23}[7])$ $\Delta Y_1[12 \oplus 3 = 6 = 9]$ $\Delta X_{22}[11 = 15]$ {8} $\Delta X_{22}[3 = 11]$			$ETK_0[11]$ $STK_{22}[3]$	
$i=2$	\mathcal{K}_{23} $STK_{22}[7] = \mathcal{K}_{22}$ {8}	$\Delta Z_{20}[4] = \delta_{22}(\Delta STK_{20}[4])$ {8} $(STK_{22}[7])$ $\Delta Y_2[8 = 5]$ $X_2[5] = W_1[1]$ $(STK_{22}[3, 7])$ $(STK_{23}[2, 7])$ $\Delta Z_3[2 = 8] = \delta_0(\Delta STK_3[5])$ $X_3[2] = W_2[2 \oplus 10 \oplus 14]$ $X_2[2] = W_1[2 \oplus 10 \oplus 14]$ $X_3[8] = W_2[4 \oplus 8]$ $X_2[10] = W_1[6 \oplus 10]$	$N2^{-28}$	$N2^{-4}$	$\Delta Z_{20}[4]$ $(STK_1[7])$ $\Delta Y_2[8]$ $X_2[5]$ $STK_1[1]$ $(STK_1[3, 7])$ $(STK_2[2, 7])$ $\Delta Y_3[2, 8, 14]$ $\Delta Z_3[2, 8]$ $X_3[2, 8]$ $W_2[2]$ $X_2[2]$ $W_1[2]$ $STK_1[2]$ $W_2[8]$ $X_2[10]$ $W_1[6]$ $STK_1[5]$	
Total	-	-		$N2^{4.41}$	24-round Enc.	
Complexity analysis						
x	Data	Memory	Total time			
54.4	$2^{118.4}$	$2^{118.4}$	$2^{119.4} + N2^{4.41} + 2^{128}e^{-2^{x-52}} = 2^{123.17}$			

G.7 24-round Related-Tweakey ID Attack on ForkSKINNY-128-256-256 ($r_i = 9, r_0 = 15$)

In this section, we give a 24-round related-tweakey ID Attack on ForkSKINNY-128-256-256. Since the goal is to recover the full 256-bit tweakey, the generic bound is 2^{256} . This attack occurs in a limited setting, where there are $r_i = 21 - X$ rounds before the fork, and $r_0 = r_1 = 27 - X$ rounds in each branch after the fork, with $X = 12$. We give the pattern of the attack in Figure 35. The key recovery details are given in Table 10.



Figure 35: Related-Tweakey ID attack on 24 rounds of ForkSKINNY-128-256-256 in the limited setting ($r_i = 9, r_0 = 15$). $|k_B \cup k_F| = 224$, $c_B = 56$, $c_F = 96$, $\Delta_B = 64$, $\Delta_F = 96$. In this case we have 15 related-key impossible differences.

Table 10: Key-recovery procedure for ID attack on 24 rounds of ForkSKINNY-128-256-256, where $\mathbf{r}_i = 9$, and $\mathbf{r}_0 = \mathbf{r}_1 = 15$.

		Pair Generation			
		# Plaintexts	# Pairs	N	# Enc.
		2^{x+64}	2^{x+128}	2^{x+96}	2^{x+65}
		Guess-and-Filter			
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs($ \mathcal{L}_i $)	Time	Deduced
$i=1$	$STK_{23}[1, 3, 7] = \mathcal{K}_{23}$ {24}	$\Delta X_{23}[0 = 4 = 12]$ $\Delta X_{23}[5 = 13]$ $\Delta X_{23}[2 = 6 \oplus 10 = 14]$ $\Delta X_{22}[8 = 12]$ {8}	$N2^{-8}$	$N2^{24}$	$STK_{23}[0, 4]$ $STK_{23}[5]$ $STK_{23}[2, 6]$ $Z_{22}, \Delta Z_{22}$
$i=2$	\mathcal{K}_{23} $STK_{22}[1, 4, 7]$ $STK_{21}[6] = \mathcal{K}_{22,21}$ {32}	$\Delta X_{22}[0 = 8]$ $\Delta X_{22}[2 = 6 = 14]$ $\Delta X_{21}[10 = 14]$ {8} $\Delta X_{21}[2 = 14]$ $\Delta Z_{19}[7] = \delta_{15}(\Delta STK_{19}[7])$ (15×2^{-8} filters) \rightarrow {4}	$N2^{-16}$	$N2^{48}$	$STK_{22}[0]$ $STK_{22}[2, 6]$ $Z_{21}, \Delta Z_{21}$ $STK_{21}[2]$ ΔZ_{19} ΔZ_{19}
$i=3$	$\mathcal{K}_{23}, \mathcal{K}_{22,21}$ $ETK_0[8, 10]$ $STK_1[3] = \mathcal{K}_{0,1}$ {24} ($STK_{23}[4, 5]$) ($STK_{21}[2, 6]$)	$\Delta Y_1[1 = 4 = 11]$ $\Delta Y_1[3 \oplus 12 = 6 = 9]$ $\Delta Y_2[6] = \delta_{-1}(\Delta STK_2[7])$ {8} $\Delta Y_2[15 = 5 = 8]$ $X_2[5] = W_1[1]$ $X_2[8] = W_1[4 \oplus 8]$	$N2^{-28}$	$N2^{64}$	($ETK_0[1, 2, 6, 14]$) $ETK_0[4, 11, 12]$ $ETK_0[3, 9]$ $Y_1, \Delta Y_1$ $\Delta Y_2[6, 15]$ $X_2[5, 8]$ $STK_1[1]$ $STK_1[7]$
$i=4$	$\mathcal{K}_{23}, \mathcal{K}_{22,21}, \mathcal{K}_{0,1}$ $STK_1[2, 4, 6] = \mathcal{K}_1$ {24} ($STK_{23}[2, 3, 6]$) ($ETK_0[4, 8, 9]$) ($STK_{22}[0], STK_1[7]$)	$\Delta Y_4 = \delta_0(\Delta STK_4[1])$ {8}	$N2^{-36}$	$N2^{76}$	$Y_2, \Delta Y_2$ ($STK_2[1, 2, 6]$) ($STK_3[1]$) ΔY_4
Total	-	-		$N2^{72.41}$	24-round Enc.
Complexity analysis					
x	Data	Memory	Total time		
62.7	$2^{126.7}$	$2^{158.7}$	$2^{127.7} + N2^{72.41} + 2^{256}e^{-2^{x-60}} = 2^{246.62}$		

G.8 20-round Related-Tweakey ID Attack on ForkSKINNY-128-256-128 ($r_i = 7, r_0 = 13$)

In this section, we give a 20-round related-tweakey ID Attack on ForkSKINNY-128-256-128. Since the goal is to recover the full 128-bit tweakey, the generic bound is 2^{128} . This attack occurs in a limited setting, where there are $r_i = 21 - X$ rounds before the fork, and $r_0 = r_1 = 27 - X$ rounds in each branch after the fork, with $X = 14$. We give the pattern of the attack in Figure 36. The key recovery details are given in Table 11.

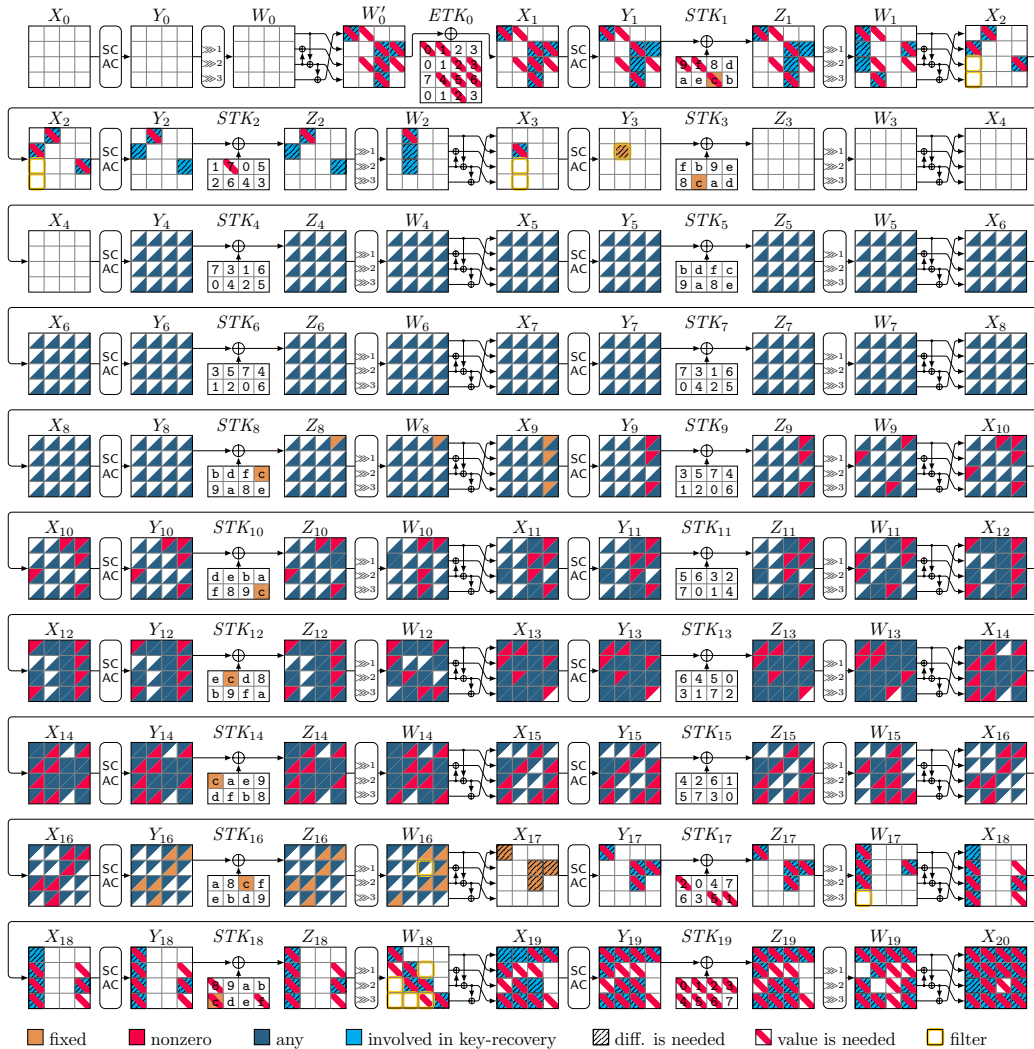


Figure 36: Related-Tweakey ID attack on 20 rounds of ForkSKINNY128-256-128 in the limited setting ($r_i = 7, r_0 = 13$). $|k_B \cup k_F| = 96, c_B = 32, c_F = 56, \Delta_B = 40, \Delta_F = 80$. In this case we have 255 related-key impossible differences

Table 11: Key-recovery procedure for ID attack on 20 rounds of ForkSKINNY-128-256-128, where $\mathbf{r}_i = 7$, and $\mathbf{r}_0 = \mathbf{r}_1 = 13$.

		Pair Generation			
		# Plaintexts	# Pairs	N	# Enc.
		2^{x+40}	2^{x+80}	2^{x+32}	2^{x+41}
		Guess-and-Filter			
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs($ \mathcal{L}_i $)	Time	Deduced
$i=1$	$STK_{19}[0] = \mathcal{K}_{19}$ {8}	$\Delta X_{19}[0 = 4 = 12]$ $\Delta X_{19}[1 = 13]$ $\Delta X_{19}[2 = 10]$ $\Delta X_{19}[10 = 14]$ {8}	$N2^{-8}$	$N2^8$	$STK_{19}[0, 4]$ $STK_{19}[1]$ $STK_{19}[2]$
	$(STK_{19}[0])$	$\Delta Y_1[0 = 7 = 10]$			$(ETK_0[0])$ $ETK_0[7, 10]$ $(STK_{19}[3, 5])$ $Z_{18}, \Delta Z_{18}$ $Y_1, \Delta Y_1$
$i=2$	\mathcal{K}_{19} $STK_1[0] = \mathcal{K}_1$ {8}	$\Delta Y_2[4 = 1 = 11]$ $\Delta X_{18}[0 = 12]$ $X_2[1] = W_1[1 \oplus 9 \oplus 13]$ $X_2[11] = W_1[7 \oplus 11]$	$N2^{-16}$	$N2^8$	$\Delta Y_2[4]$ $X_2[1, 11]$ $STK_{18}[0]$ $STK_1[1]$ $STK_1[6]$ $Y_2, \Delta Y_2$ $Z_{17}, \Delta Z_{17}$ $(STK_{17}[0, 6, 7])$ $X_{17}, \Delta X_{17}$
	$(STK_{19}[1, 2, 5])$	$\Delta X_{17}[6 = 10]$ {8}			
$i=2$	$\mathcal{K}_{19}, \mathcal{K}_1$ $STK_2[1] = \mathcal{K}_2$ {8}	$\Delta Y_3[5] = \delta_0(\Delta STK_3[5])$	$N2^{-16}$	$N2^8$	$\Delta Y_3[5]$
Total	-	-		$N2^{6.26}$	20-round Enc.
		Complexity analysis			
x	Data	Memory	Total time		
61	2^{101}	2^{93}	$2^{102} + N2^{6.26} + 2^{128}e^{-2^{x-56}} = 2^{102.20}$		

G.9 31-round Related-Tweakey ID Attack on ForkSKINNY-128-288-288 ($\mathbf{r}_i = 10, \mathbf{r}_0 = 25$)

This section gives a 31-round related-tweakey ID Attack on ForkSKINNY-128-288-288. Since the goal is to recover the full 288-bit tweakey, the generic bound is 2^{288} . This attack has $\mathbf{r}_i = 10$ rounds before the fork, and $\mathbf{r}_0 = 25/\mathbf{r}_1 = 21$ rounds in each branch after the fork. Figure 37 represent the complete ID attack discovered by our tool, Table 12 describes the key recovery procedure.

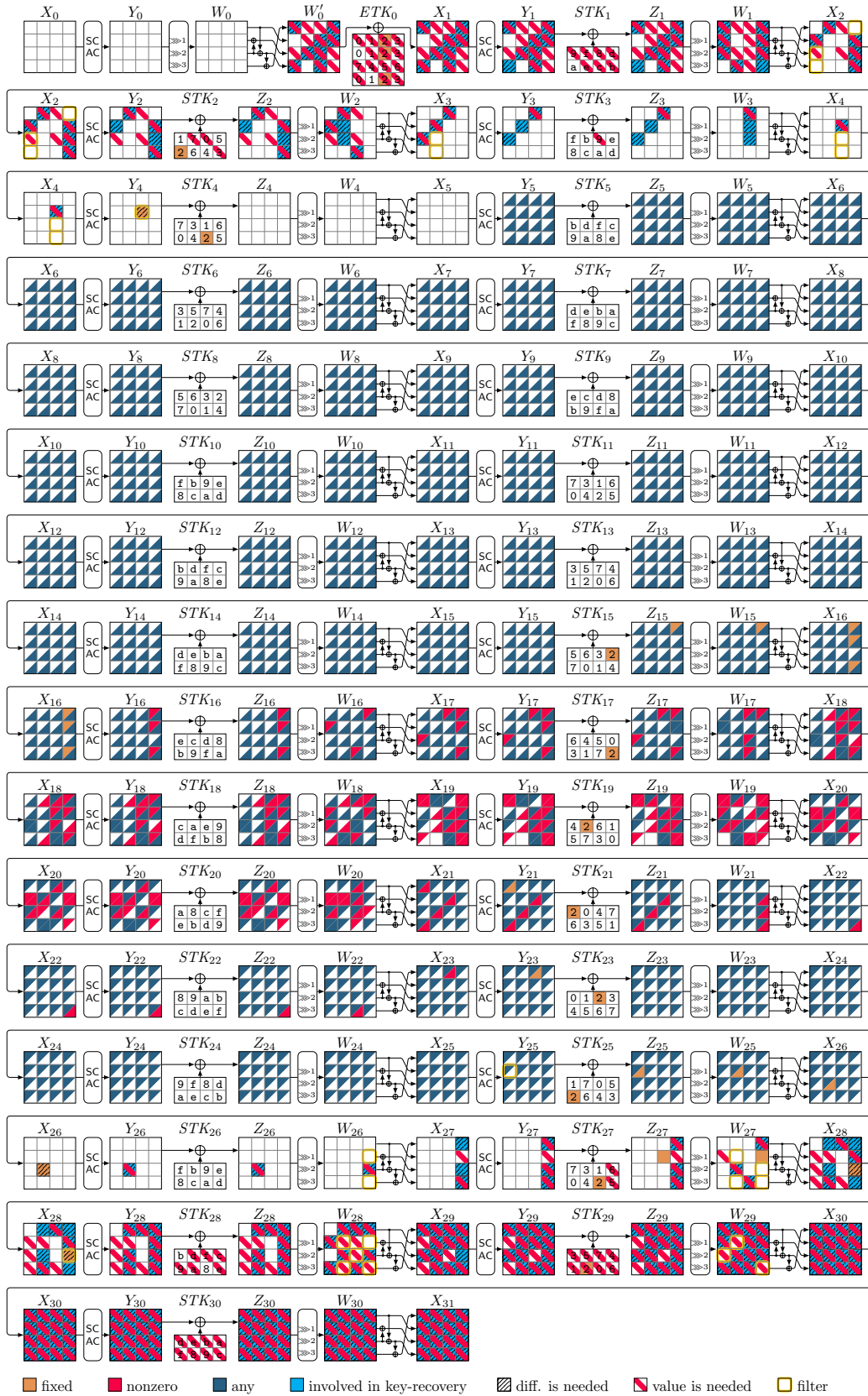


Figure 37: Related-Tweakey ID attack on 31 rounds of ForkSKINNY128-288-288 in the arbitrary setting ($\mathbf{r}_I = 10, \mathbf{r}_O = 25$). $|k_B \cup k_F| = 272, c_B = 56, c_F = 128, \Delta_B = 64, \Delta_F = 128$. In this case we have 15 related-key impossible differences.

Table 12: Key-recovery procedure for ID attack on 31 rounds of ForkSKINNY-128-288-288, where $\mathbf{r}_1 = 10$, $\mathbf{r}_0 = 25$, and $\mathbf{r}_1 = 21$.

		Pair Generation			
	# Plaintexts	# Pairs	N	# Enc.	
	2^{x+64}	2^{x+128}	2^{x+128}	2^{x+65}	
Guess-and-Filter					
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs($ \mathcal{L}_i $)	Time	Deduced
$i=1$	$STK_{30}[0-7] = \mathcal{K}_{30}$ {24}	$\Delta W_{29}[5 = 8 = 15 = 0]$ {24}	$N2^{-24}$	$N2^{64}$	$Z_{29}, \Delta Z_{29}$
$i=2$	\mathcal{K}_{30} $STK_{29}[0, 2, 4] = \mathcal{K}_{29}$ {24}	$\Delta X_{29}[1 = 13 = 5 \oplus 9]$ $\Delta X_{29}[6 = 14]$ $\Delta X_{29}[3 = 15 = 7 \oplus 11]$ $\Delta X_{28}[9 = 13]$ {8} $\Delta X_{28}[11] = \delta_{24}(\Delta STK_{27}[6])$ $(15 \times 2^{-8}) \rightarrow$ {4}	$N2^{-36}$	$N2^{64}$	$STK_{29}[1, 5]$ $STK_{29}[6]$ $STK_{29}[3, 7]$ $\Delta Z_{28}, Z_{28}$
$i=3$	$\mathcal{K}_{30}, \mathcal{K}_{29}$ $STK_{28}[2, 4, 5] = \mathcal{K}_{28}$ {24}	$\Delta X_{28}[1 = 9]$ $\Delta X_{28}[3 = 7 = 15]$ $\Delta X_{27}[11 = 15]$ {8}	$N2^{-44}$	$N2^{76}$	$STK_{28}[1]$ $STK_{28}[3, 7]$ $\Delta Z_{27}, Z_{27}$
$i=4$	$\mathcal{K}_{30}, \mathcal{K}_{29}, \mathcal{K}_{28}$ $STK_{27}[7] = \mathcal{K}_{27}$ {8}	$\Delta X_{27}[3 = 11]$ $\Delta Z_{25}[4] = \delta_{23}(\Delta STK_{25}[4])$ {8}	$N2^{-52}$	$N2^{76}$	$STK_{27}[3]$ $\Delta Z_{25}[4]$
$i=5$	$\mathcal{K}_{30}, \mathcal{K}_{29}, \mathcal{K}_{28}, \mathcal{K}_{27}$ $ETK_0[1, 2, 8] = \mathcal{K}_0$ {24} $(STK_{29}[1, 7])$ $(STK_{27}[3, 7])$	$\Delta Y_1[0 = 7 = 10]$ $\Delta Y_1[3 \oplus 12 = 9]$ $\Delta Y_2[1 = 4]$ {8} $\Delta Y_2[1 = 11]$ $X_2[11] = W_1[7 \oplus 11]$	$N2^{-60}$	$N2^{92}$	$(ETK_0[10, 11])$ $ETK_0[0, 7]$ $ETK_0[9], \Delta Y_1, Y_1$ $(STK_1[0, 1, 3, 5, 6])$ $Y_2[0, 7, 8, 10, 15]$ $\Delta Y_2[1, 4, 7, 15]$ $X_2[11]$ $STK_1[2]$
$i=6$	$\mathcal{K}_{30}, \mathcal{K}_{29}, \mathcal{K}_{28}, \mathcal{K}_{27},$ \mathcal{K}_0 $STK_1[7] = \mathcal{K}_1$ {8} $(STK_{29}[2])$ $(ETK_0[8])$	$\Delta Y_3[5 = 2 = 8]$ $X_3[2] = W_2[2 \oplus 10 \oplus 14]$ $X_3[8] = W_2[4 \oplus 8]$ $\Delta Y_4[6] = \delta_0(\Delta STK_4[6])$ {8}	$N2^{-68}$	$N2^{92}$	$Y_2, \Delta Y_2$ $(STK_2[1]), \Delta Y_3[5]$ $X_3[2, 8]$ $STK_2[2]$ $STK_2[7]$ $Y_3, \Delta Y_3$ $(STK_3[2]), \Delta Y_4[6]$
Total	-	-	$N2^{88.63}$ 31-round Enc.		
Complexity analysis					
x	Data	Memory	Total time		
62.5	$2^{126.5}$	$2^{190.5}$	$2^{127.5} + N2^{88.63} + 2^{288} e^{-2^{x-60}} = 2^{280.52}$		

G.10 28-round Related-Tweakey ID Attack on ForkSKINNY-128-288-128 ($r_i = 14, r_0 = 15$)

In this section, we give a 28-round related-tweakey ID Attack on ForkSKINNY-128-288-128. Since the goal is to recover the full 128-bit tweakey, the generic bound is 2^{128} . We give the pattern of the attack in Figure 38. The key recovery details are given in Table 13.

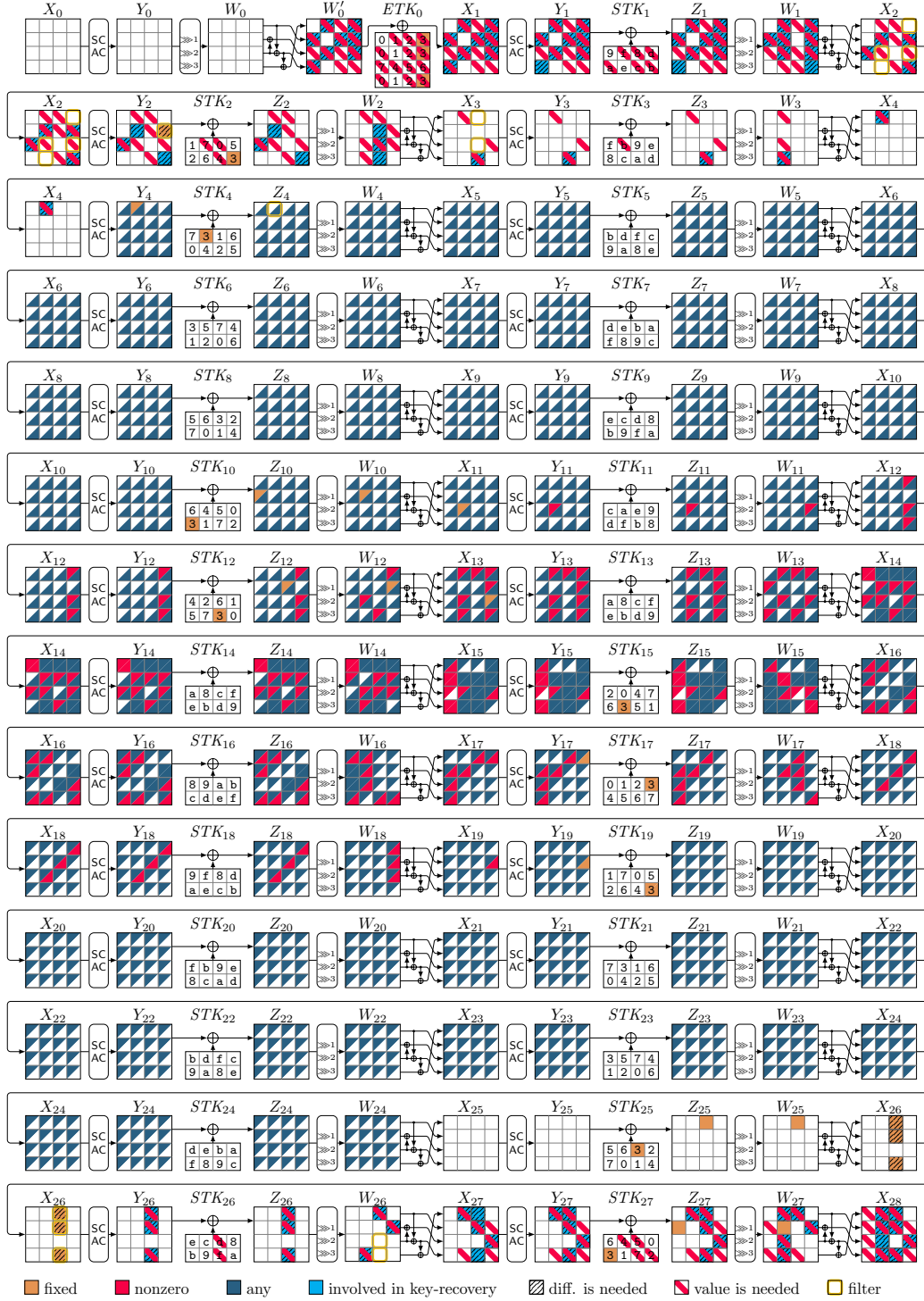


Figure 38: Related-Tweakey ID attack on 28 rounds of ForkSKINNY128-288-128 in the arbitrary setting ($r_i = 14, r_0 = 15$). $|k_B \cup k_F| = 112$, $c_B = 56$, $c_F = 40$, $\Delta_B = 64$, $\Delta_F = 40$. In this case, we have 15 related-key impossible differences.

Table 13: Key-recovery procedure for ID attack on 28 rounds of ForkSKINNY-128-288-128, where $\mathbf{r}_i = 14$, $\mathbf{r}_0 = 15$, and $\mathbf{r}_1 = 14$.

		Pair Generation			
		# Plaintexts	# Pairs	N	# Enc.
		2^{x+64}	2^{x+128}	2^{x+40}	2^{x+65}
Guess-and-Filter					
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs($ \mathcal{L}_i $)	Time	Deduced
$i=1$	$STK_{27}[1, 7] = \mathcal{K}_{27}$ {16}	$\Delta X_{27}[2 = 6 = 14]$	$N2^{-12}$	$N2^{16}$	$STK_{27}[2, 6]$ $Z_{26}, \Delta Z_{26}$ $STK_{26}[2, 6]$ $\Delta Z_{25}[2]$
		$\Delta X_{26}[2 = 6 = 14]$			
		$\Delta Z_{25}[2] = \delta_{18}(\Delta STK_{25}[2])$ (15×2^{-8}) \rightarrow {4}			
	$(STK_{27}[1, 2, 6, 7])$	$\Delta Y_1[6 = 9]$ {8}			$(ETK_0[6, 8, 9, 10])$
$i=2$	$\mathcal{K}_{27}, \mathcal{K}_0$ $ETK_0[4] = \mathcal{K}_0$ {8}	$\Delta Y_1[4 = 1 = 11]$ $\Delta Y_1[3 = 12 \oplus 9]$	$N2^{-28}$	$N2^{12}$	$ETK_0[12]$ $ETK_0[1, 11]$ $ETK_0[3]$ $Y_1, \Delta Y_1$ $(STK_1[1, 3])$ $\Delta Y_2[5, 7, 15]$
	$(STK_{26}[2, 6])$	$\Delta Y_2[7] = \delta_{-1}(\Delta STK_2[7])$ {8}			
		$\Delta Y_2[5 = 15]$ {8}			
		$\Delta Y_2[5 = 8]$ $X_2[8] = W_1[4 \oplus 8]$			$X_2[8]$ $STK_1[7]$
$i=2$	$\mathcal{K}_{27}, \mathcal{K}_0$ $STK_1[2, 4, 6] = \mathcal{K}_1$ {24}		$N2^{-36}$	$N2^{20}$	$Y_2, \Delta Y_2$
	$(ETK_0[4, 8, 9])$ $(STK_1[7])$	$\Delta Y_4[1] = \delta_0(\Delta STK_4[1])$ {8}			$(STK_2[1, 2, 6])$ $(STK_3[1])$ $\Delta Y_2[1]$
Total	-	-		$N2^{16.80}$	28-round Enc.
Complexity analysis					
x	Data	Memory	Total time		
60.8	$2^{124.80}$	$2^{100.69}$	$2^{125.8} + N2^{16.8} + 2^{128}e^{-2^{x-60}} = 2^{126.68}$		

G.11 28-round Related-Tweakey ID Attack on ForkSKINNY-128-288-288 ($\mathbf{r}_i = 11$, $\mathbf{r}_0 = 17$)

In this section, we give a 28-round related-tweakey ID Attack on ForkSKINNY-128-288-288. Since the goal is to recover the full 288-bit tweakey, the generic bound is 2^{288} . This attack occurs in a limited setting, where there are $\mathbf{r}_i = 25 - X$ rounds before the fork, and $\mathbf{r}_0 = \mathbf{r}_1 = 31 - X$ rounds in each branch after the fork, with $X = 14$. We give the pattern of the attack in Figure 39. The key recovery details are given in Table 14.



Figure 39: Related-Tweakey ID attack on 28 rounds of ForkSKINNY128-288-288 in the limited setting ($r_i = 11, r_0 = 17$). $|k_B \cup k_F| = 248, c_B = 48, c_F = 104, \Delta_B = 56, \Delta_F = 104$. In this case we have 15 related-key impossible differences.

Table 14: Key-recovery procedure for ID attack on 28 rounds of ForkSKINNY-128-288-288, where $\mathbf{r}_i = 11$, and $\mathbf{r}_0 = \mathbf{r}_1 = 17$.

		Pair Generation			
		# Plaintexts	# Pairs	N	# Enc.
		2^{x+56}	2^{x+112}	2^{x+88}	2^{x+57}
		Guess-and-Filter			
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs ($ \mathcal{L}_i $)	Time	Deduced
$i=1$	$STK_{27}[0, 2, 6] = \mathcal{K}_{27}$ {24}	$\Delta X_{27}[4 \oplus 8 = 12]$ $\Delta X_{27}[1 = 13 = 5 \oplus 9]$ $\Delta X_{27}[3 = 7 = 15]$ $\Delta X_{26}[11 = 15]$ {8} $\Delta X_{26}[3 = 15]$ $\Delta X_{26}[1 = 5 = 13]$ $\Delta X_{26}[10] = \delta_{20}(= STK_{25}[5])$ $(15 \times 2^{-8}) \rightarrow \{4\}$	$N2^{-12}$	$N2^{24}$	$STK_{27}[4]$ $STK_{27}[1, 5]$ $STK_{27}[3, 7]$ $\Delta Z_{26}, Z_{26}$ $STK_{26}[3]$ $STK_{26}[1, 5]$
$i=2$	$\mathcal{K}_{27}, \mathcal{K}_{26,25}$ $STK_{26}[0, 6, 7] $ $STK_{25}[5] = \mathcal{K}_{26,25}$ {32}	$\Delta X_{25}[9 = 13]$ {8} $\Delta X_{25}[1 = 9]$ $\Delta Z_{23}[6] = \delta_{18}(\Delta STK_{23}[6])$ {8}	$N2^{-28}$	$N2^{44}$	$\Delta Z_{25}, Z_{25}$ $STK_{25}[1]$ $\Delta Z_{23}[6]$
$i=3$	$\mathcal{K}_{27}, \mathcal{K}_{26,25}$ $ETK_0[0, 2, 7, 11] $ $STK_1[0] = \mathcal{K}_{0,1}$ {40} $(STK_{27}[0])$ $(STK_{25}[1])$	$\Delta Y_1[2 = 5 = 8]$ $\Delta Y_1[7 = 10]$ $\Delta Y_2[0] = \delta_{-1}(\Delta STK_2[0])$ {8}	$N2^{-36}$	$N2^{68}$	$\Delta Y_1[2, 7]$ $(ETK_0[9])$ $ETK_0[5, 8]$ $ETK_0[10]$ $\Delta Y_1, Y_1, \Delta Y_2[0]$
$i=4$	$\mathcal{K}_{27}, \mathcal{K}_{26,25}, \mathcal{K}_{0,1}$ $STK_1[2-5, 7] = \mathcal{K}_1$ {40}	$\Delta Y_2[6 = 9 = 12]$ {16}	$N2^{-52}$	$N2^{100}$	$\Delta Y_2, Y_2$
$i=5$	$\mathcal{K}_{27}, \mathcal{K}_{26,25}, \mathcal{K}_{0,1}$ \mathcal{K}_1 $STK_2[2, 7] = \mathcal{K}_2$ {16} $(STK_{27}[4])$ $(ETK_0[10])$ $(STK_{26}[3])$ $(STK_1[0])$	$\Delta Y_4[2] = \delta_0(\Delta STK_4[2])$ {8}	$N2^{-60}$	$N2^{100}$	$(STK_2[3])$ $(STK_3[2])$ $\Delta Y_4[2]$
Total	-	-		$N2^{96.19}$	28-round Enc.
		Complexity analysis			
x	Data	Memory	Total time		
70.9	$2^{126.9}$	$2^{158.9}$	$2^{127.9} + N2^{96.19} + 2^{288} e^{-2^{x-68}} = 2^{277.23}$		

G.12 26-round Related-Tweakey ID Attack on ForkSKINNY-128-288-128 ($r_i = 10, r_0 = 16$)

In this section, we give a 26-round related-tweakey ID Attack on ForkSKINNY-128-288-128. Since the goal is to recover the full 128-bit tweakey, the generic bound is 2^{128} . This attack occurs in a limited setting, where there are $r_i = 25 - X$ rounds before the fork, and $r_0 = r_1 = 31 - X$ rounds in each branch after the fork, with $X = 15$. We give the pattern of the attack in Figure 40. The key recovery details are given in Table 15.



Figure 40: Related-Tweakey ID attack on 26 rounds of ForkSKINNY128-288-128 in the limited setting ($r_i = 10, r_0 = 16$). $|k_B \cup k_F| = 88$, $c_B = 0$, $c_F = 64$, $\Delta_B = 8$, $\Delta_F = 64$. In this case we have 15 related-key impossible differences.

Table 15: Key-recovery procedure for ID attack on 26 rounds of ForkSKINNY-128-288-128, where $\mathbf{r}_i = 10$, and $\mathbf{r}_0 = \mathbf{r}_1 = 16$.

		Pair Generation			
	# Plaintexts	# Pairs	N	# Enc.	
	2^{x+8}	2^{x+16}	2^{x-48}	2^{x+9}	
		Guess-and-Filter			
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs ($ \mathcal{L}_i $)	Time	Deduced
$i=1$	$STK_{25}[2, 4, 5] = \mathcal{K}_{25}$ {24}	$\Delta X_{25}[1 = 9]$ $\Delta X_{25}[9 = 13]$ {8} $\Delta X_{25}[3 = 7 = 15]$	$N2^{-16}$	$N2^{24}$	$STK_{25}[1]$ $STK_{25}[3, 7]$ $Z_{24}, \Delta Z_{24}$
		$\Delta Z_{24}[6] = \delta_{19}(\Delta STK_{24}[6])$ $(15 \times 2^{-8}) \rightarrow$ {4} $\Delta X_{24}[11 = 15]$ {8} $\Delta X_{24}[3 = 15]$			$STK_{24}[3]$
$i=2$	\mathcal{K}_{25} $STK_{24}[7] = \mathcal{K}_{24}$ {8}	$\Delta Z_{22}[4] = \delta_{18}(\Delta STK_{22}[4])$ {8}	$N2^{-28}$	$N2^{16}$	$\Delta Z_{22}[4]$
$i=3$	$\mathcal{K}_{25}, \mathcal{K}_{24}$ $ETK_0[0, 10, 13] = \mathcal{K}_0$ {24} $(STK_{25}[5])$	$\Delta Y_2[0] = \delta_0(= STK_2[0])$ {8}	$N2^{-36}$	$N2^{28}$	$\Delta Y_1, Y_1$ $(STK_1[0])$ $\Delta Y_2[0]$
Total	-	-	$N2^{23.46}$ 26-round Enc.		
		Complexity analysis			
x	Data	Memory	Total time		
116.6	$2^{124.5}$	$2^{68.5}$	$2^{125.5} + N2^{23.46} + 2^{128} e^{-2^{x-116}} = 2^{126.74}$		

H Related-Tweakey ID Attack on SKINNY

H.1 26-round Related-Tweakey ID Attack on SKINNY-128-288-288

In this section, we give a 26-round related-tweakey ID Attack on SKINNY-128-288-288. Since the goal is to recover the full 288-bit tweakey, the generic bound is 2^{288} . We give the pattern of the attack in Figure 41. The key recovery details are given in Table 16.

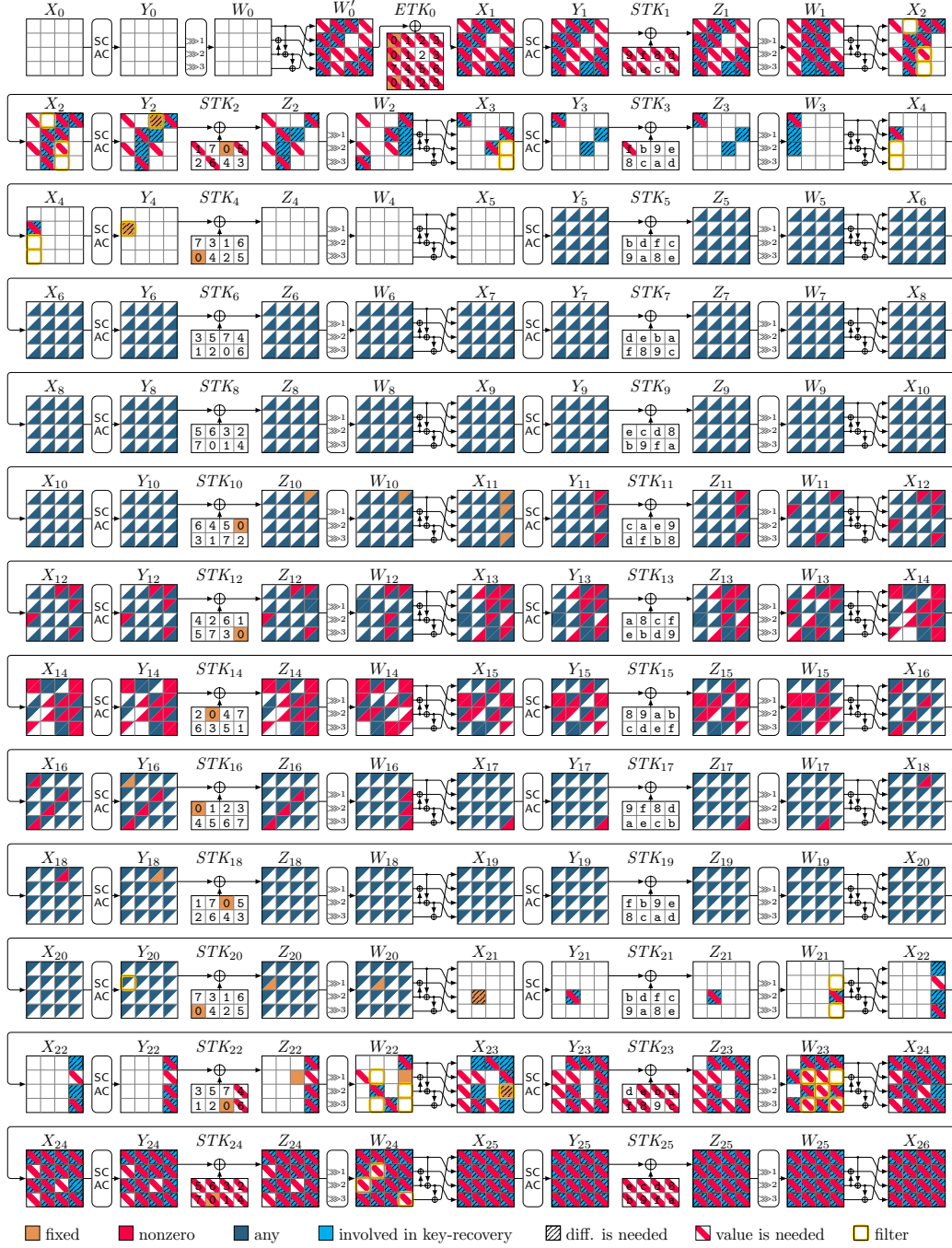


Figure 41: Related-Tweakey ID attack on 26 rounds of SKINNY128-288-288. $|k_B \cup k_F| = 264$, $c_B = 64$, $c_F = 128$, $\Delta_B = 72$, $\Delta_F = 128$. In this case, we have 255 related-key impossible differences.

Table 16: Key-recovery procedure for ID attack on 26 rounds of SKINNY-128-288-288

		Pair Generation			
	# Plaintexts	# Pairs	N	# Enc.	
	2^{x+72}	2^{x+144}	2^{x+144}	2^{x+73}	
Guess-and-Filter					
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs ($ \mathcal{L}_i $)	Time	Deduced
$i=1$	$STK_{25}[0-7] = \mathcal{K}_{25}$ {64}	$\Delta W_{24}[5 = 8 = 15] = 0$ {24}	$N2^{-24}$	$N2^{64}$	$Z_{24}, \Delta Z_{24}$
$i=2$	\mathcal{K}_{25} $STK_{24}[0, 2, 4] = \mathcal{K}_{24}$ {24}	$\Delta X_{24}[1 = 13 = 5 \oplus 9]$ $\Delta X_{24}[6 = 14]$ $\Delta X_{24}[3 = 15 = 7 \oplus 11]$ $\Delta X_{23}[9 = 13]$ {8} $\Delta X_{23}[11] = \delta_9(\Delta STK_{22}[6])$	$N2^{-32}$	$N2^{64}$	$STK_{24}[1, 5]$ $STK_{24}[6]$ $STK_{24}[3, 7]$ $\Delta Z_{23}, Z_{23}$
$i=3$	$\mathcal{K}_{25}, \mathcal{K}_{24}$ $STK_{23}[2, 4, 5] = \mathcal{K}_{23}$ {24}	$\Delta X_{23}[1 = 9]$ $\Delta X_{23}[3 = 7 = 15]$ $\Delta X_{22}[11 = 15]$ {8}	$N2^{-40}$	$N2^{80}$	$STK_{23}[1]$ $STK_{23}[3, 7]$ $\Delta Z_{22}, Z_{22}$
$i=4$	$\mathcal{K}_{25}, \mathcal{K}_{24}, \mathcal{K}_{23}$ $STK_{22}[7] = \mathcal{K}_{22}$ {8}	$\Delta X_{22}[3 = 11]$ $\Delta Z_{20}[4] = \delta_8(\Delta STK_{20}[4])$ {8}	$N2^{-48}$	$N2^{80}$	$STK_{22}[3]$ $\Delta Z_{25}[4]$
$i=5$	$\mathcal{K}_{25}, \mathcal{K}_{24}, \mathcal{K}_{23}, \mathcal{K}_{22}$ $ETK_0[1] = \mathcal{K}_0$ {8} $(STK_{24}[1, 7])$ $(STK_{22}[3, 7])$	$\Delta Y_1[1 \oplus 11 = 14]$ $\Delta Y_1[2 = 8]$ $\Delta Y_1[2 = 5]$ {8}	$N2^{-56}$	$N2^{80}$	$ETK_0[5]$ $(ETK_0[9, 11])$ $ETK_0[14]$ $ETK_0[8]$
$i=6$	$\mathcal{K}_{25}, \mathcal{K}_{24}, \mathcal{K}_{23}, \mathcal{K}_{22}$ \mathcal{K}_0 $ETK_0[0, 3, 10] = \mathcal{K}'_0$ {24} $(STK_{25}[0, 3, 4, 6, 7])$ $(STK_{23}[1-5])$	$\Delta Y_2[2] = \delta_{-1}(\Delta STK_2[2])$ $\Delta Y_2[6 = 9]$ {16} $\Delta Y_2[3 = 6]$ $X_2[3] = W_1[3 \oplus 11 \oplus 15]$	$N2^{-64}$	$N2^{96}$	$\Delta Y_1, Y_1$ $(STK_1[1, 2, 4, 5, 7])$ $\Delta Y_2[2, 6, 9]$ $X_2[3]$ $STK_1[3]$
$i=7$	$\mathcal{K}_{25}, \mathcal{K}_{24}, \mathcal{K}_{23}, \mathcal{K}_{22}$ $\mathcal{K}_0, \mathcal{K}'_0$ $STK_1[0] = \mathcal{K}_1$ {8}	$\Delta Y_2[3 = 6]$ {8}	$N2^{-80}$	$N2^{96}$	$\Delta Y_2, Y_2$
$i=8$	$\mathcal{K}_{25}, \mathcal{K}_{24}, \mathcal{K}_{23}, \mathcal{K}_{22}$ $\mathcal{K}_0, \mathcal{K}'_0, \mathcal{K}_1$ $STK_2[0] = \mathcal{K}_2$ {8} $(STK_{24}[0, 1])$ $(ETK_0[10, 11])$ $(STK_{25}[6])$ $(STK_{23}[4])$	$\Delta Y_3[0 = 7 = 10]$ {16} $\Delta Y_4[4] = \delta_0(\Delta STK_4[4])$ {8}	$N2^{-104}$	$N2^{88}$	$(STK_2[3, 5])$ $\Delta Y_3, Y_3$ $(STK_3[0]), \Delta Y_4$
Total	-	-		$N2^{92.30}$	26-round Enc.
Complexity analysis					
x	Data	Memory	Total time		
50	2^{122}	2^{194}	$2^{123} + N2^{92.30} + 2^{288} e^{-2^x - 48} = 2^{286.38}$		

H.2 23-round Related-Tweakey ID Attack on SKINNY-128-288-128

In this section, we give a 23-round related-tweakey ID Attack on SKINNY-128-288-128. Since the goal is to recover the full 128-bit tweakey, the generic bound is 2^{128} . We give the pattern of the attack in Figure 42. The key recovery details are given in Table 17.



Figure 42: Related-Tweakey ID attack on 23 rounds of SKINNY128-288-128. $|k_B \cup k_F| = 120$, $c_B = 48$, $c_F = 40$, $\Delta_B = 56$, $\Delta_F = 40$. In this case, we have 255 related-key impossible differences.

Table 17: Key-recovery procedure of ID attack on 23 rounds of SKINNY-128-288-128.

		Pair Generation			
	# Plaintexts	# Pairs	N		# Enc.
	2^{x+56}	2^{x+112}	2^{x+24}		2^{x+57}
Guess-and-Filter					
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs($ \mathcal{L}_i $)	Time	Deduced
$i=1$	$STK_{22}[1, 7] $ $ETK_0[9] = \mathcal{K}_{22,0}$ {24}	$\Delta X_{22}[2 = 6 = 14]$ $\Delta X_{21}[2 = 6 = 14]$ $\Delta Z_{20}[2] = \delta_8(\Delta STK_{20}[2])$	$N2^{-8}$	$N2^{24}$	$STK_{22}[2, 6]$ $Z_{21}, \Delta Z_{21}$ $STK_{21}[2, 6]$ $\Delta Z_{20}[2]$
	$(STK_{22}[1, 2, 6, 7])$	$\Delta Y_1[2 = 5 = 8]$ $\Delta Y_1[7 = 10]$			$(ETK_0[0, 8, 10, 11])$ $ETK_0[2, 5]$ $ETK_0[7]$ $\Delta Y_1, Y_1$ $(STK_1[2])$ $\Delta Y_2[6]$ $X_2[9, 12]$ $STK_1[4]$ $STK_1[0]$ $\Delta Y_2[0]$
	$(STK_{21}[6])$	$\Delta Y_2[6 = 9 = 12]$ $X_2[9] = W_1[5 \oplus 9]$ $X_2[12] = W_1[0 \oplus 8]$ $\Delta Y_2[0] = \delta_{-1}(\Delta STK_2[0])$ {8}			
				$N2^{-16}$	$N2^{40}$
$i=3$	$\mathcal{K}_{22,0}$ $STK_1[3, 5, 7] = \mathcal{K}_1$ {24} $(ETK_0[0, 3, 10])$ $(STK_1[0])$				
		$\Delta Y_4[2] = \delta_0(\Delta STK_4[2])$ {8}			
Total	-	-		$N2^{37.06}$	23-round Enc.
Complexity analysis					
x	Data	Memory	Total time		
64.8	$2^{120.80}$	$2^{88.80}$	$2^{121.80} + N2^{37.06} + 2^{128}e^{-2^{x-64}} = 2^{126.73}$		

H.3 31-round Related-Tweakey ID Attack on SKINNYe-v2

SKINNYe-v2 is essentially SKINNY- $n-4n$ with cell size $c = 4$. The pattern of the attack is illustrated in Figure 43, with key-recovery details given in Table 18.

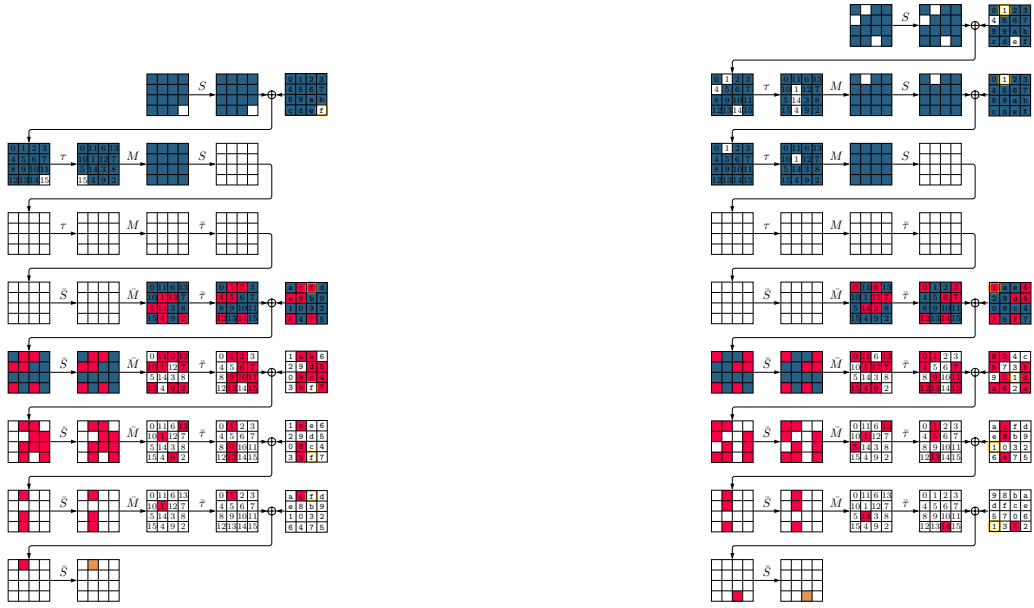


Figure 43: Related-Tweakey ID attack on 31 rounds of SKINNYe-v2. $|k_B \cup k_F| = 240$, $c_B = 44$, $c_F = 64$, $\Delta_B = 48$, $\Delta_F = 64$. In this case, we have 255 related-key impossible differences.

Table 18: Key-recovery procedure for ID attack on 31 rounds of SKINNYe-v2.

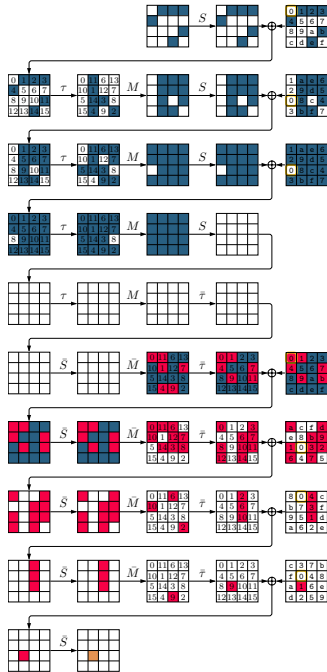
		Pair Generation				
		# Plaintexts	# Pairs	N	# Enc.	
		2^{x+48}	2^{x+96}	2^{x+96}	2^{x+49}	
		Guess-and-Filter				
Step	Guessed keys {#}	Condition(s) {#filters}	# Remaining pairs(\mathcal{L}_i)	Time	Deduced	
$i=1$	$STK_{31}[0-7] $ $STK_{30}[0-7] $ $STK_{29}[0-7] = \mathcal{K}_{31,30,29}$ {96}	$\Delta W_{27}[5 = 8 = 15] = 0$ {12}	$N2^{-12}$	$N2^{96}$	$Z_{27}, \Delta Z_{27}$	
$i=2$	$\mathcal{K}_{31,30,29}$ $STK_{27}[0-7] = \mathcal{K}_{27}$ {32}	$\Delta W_{26}[5 = 7 = 10 = 13 = 15] = 0$ {20} $\Delta X_{26}[11] = \delta_{10}(\Delta STK_{25}[6])$	$N2^{-32}$	$N2^{116}$	$\Delta Z_{26}, Z_{26}$	
$i=3$	$\mathcal{K}_{31,30,29}, \mathcal{K}_{27}$ $STK_{26}[1-5, 7] = \mathcal{K}_{26}$ {24}	$\Delta W_{25}[5 = 11 = 13 = 15] = 0$ {16}	$N2^{-48}$	$N2^{120}$	$\Delta Z_{25}, Z_{25}$	
$i=4$	$\mathcal{K}_{31,30,29}, \mathcal{K}_{27}, \mathcal{K}_{26}$ $STK_{26}[3, 7] = \mathcal{K}_{25}$ {8}	$\Delta W_{24}[7 = 15] = 0$ {8} $\Delta Z_{23}[4] = \delta_9(\Delta STK_{23}[4])$ {4}	$N2^{-60}$	$N2^{112}$	$\Delta Z_{23}[4]$	
$i=5$	$\mathcal{K}_{31,30,29}, \mathcal{K}_{27}, \mathcal{K}_{26}$ \mathcal{K}_{25} $ETK_0[0, 3, 8] $ $STK_1[0-7] = \mathcal{K}_{0,1}$ {44}	$\Delta Y_1[4 = 11]$ $\Delta Y_1[0 = 10]$ $\Delta Y_1[3 = 6 = 9]$ $\Delta Y_1[1 \oplus 14 = 4]$ $\Delta Y_2[2 = 5 = 8]$ $\Delta Y_2[7 = 10]$ {12} $\Delta Y_3[0] = \delta_{-1}(\Delta STK_3[0])$ {4}	$N2^{-76}$	$N2^{144}$	$ETK_0[4]$ $ETK_0[11]$ $ETK_0[10]$ $ETK_0[6, 9]$ $ETK_0[1]$ $\Delta Y_2, Y_2$	
$i=6$	$\mathcal{K}_{31,30,29}, \mathcal{K}_{27}, \mathcal{K}_{26}$ $\mathcal{K}_{25}, \mathcal{K}_{0,1}$ $STK_2[5] = \mathcal{K}_2$ {4}	$\Delta Y_3[6 = 9 = 12]$ {8}	$N2^{-84}$	$N2^{132}$	$\Delta Y_3, Y_3$	
$i=7$	$\mathcal{K}_{31,30,29}, \mathcal{K}_{27}, \mathcal{K}_{26}$ $\mathcal{K}_{25}, \mathcal{K}_{0,1}, \mathcal{K}_2$ $STK_3[2, 3, 7] = \mathcal{K}_3$ {12}	$\Delta Y_5[2] = \delta_0(= STK_5[2])$ {4}	$N2^{-88}$	$N2^{136}$	$\Delta Y_5[2]$	
Total	-	-	$N2^{140.04}$ 31-round Enc.			
		Complexity analysis				
x	Data	Memory	Total time			
14	2^{62}	2^{110}	$2^{63} + N2^{140.4} + 2^{256} e^{-2^x - 12} = 2^{251.14}$			

I Integral Distinguishers of QARMAv2



(a) Integral distinguisher for 7 rounds of QARMAv2-64 ($\mathcal{T} = 1$). Data complexity: 2^8 .

(b) Integral distinguisher for 8 rounds of QARMAv2-64 ($\mathcal{T} = 1$). Data complexity: 2^{16} .



(c) Integral distinguisher for 9 rounds of QARMAv2-64 ($\mathcal{T} = 1$). Data complexity: 2^{44} .

Figure 44: ZC-based integral distinguishers for QARMAv2-64 ($\mathcal{T} = 1$).

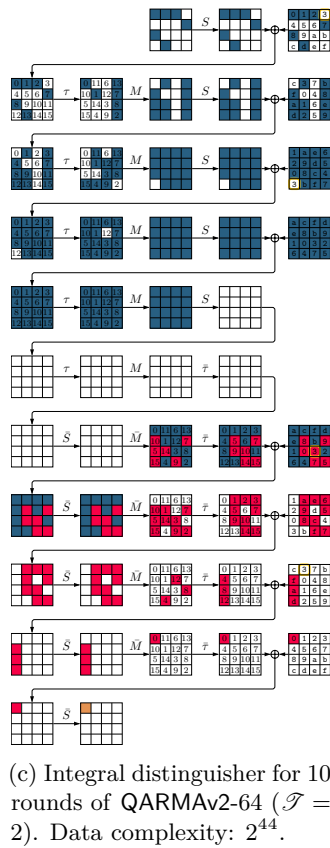
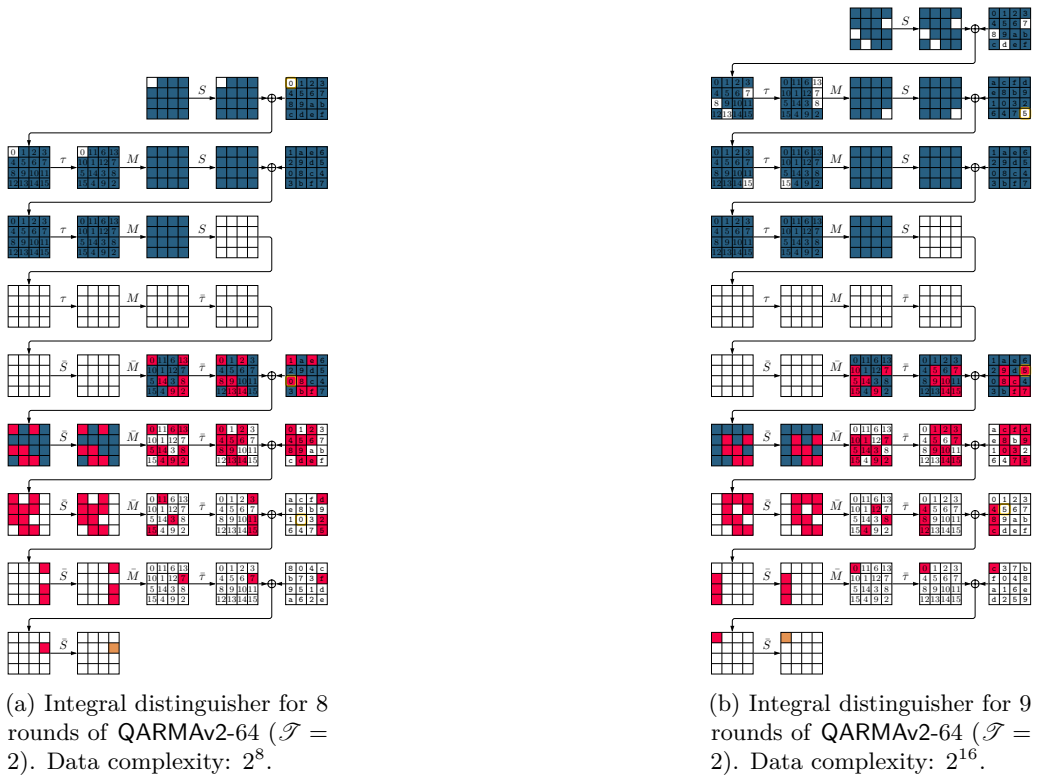


Figure 45: ZC-based integral distinguishers for QARMAv2-64 ($\mathcal{T} = 2$). Interpretation: For the integral distinguisher, if all blue (white) cells at the input are fixed (active), and the highlighted tweak cell is active, then the colored output cell will be balanced.



(a) Integral distinguisher for 10 rounds of QARMAv2-128. Data complexity: 2^{16} .

(b) Integral distinguisher for 11 rounds of QARMAv2-128. Data complexity: 2^{44} .

Figure 46: ZC-based integral distinguisher for 12 rounds of QARMAv2-128 ($\mathcal{T} = 2$). Data complexity: 2^{96} . Interpretation: In the ZC distinguisher, blue (red) cells take any (resp. an arbitrary nonzero) linear mask value. For the integral distinguisher, if all blue (white) cells at the input are fixed (active), and the highlighted tweak cell is active, then the colored output cell will be balanced.

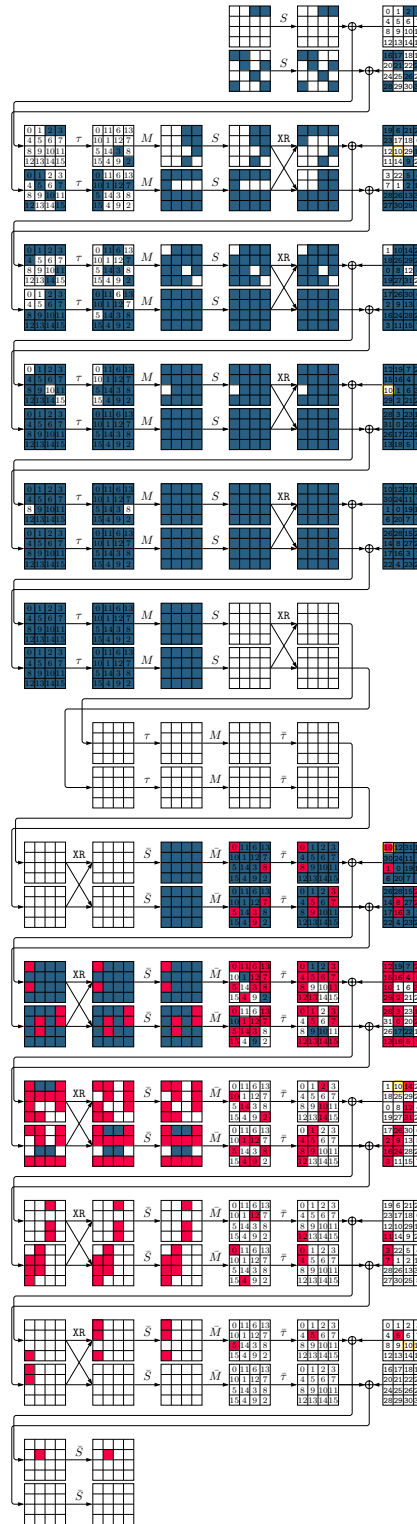


Figure 47: ZC-based integral distinguisher for 12 rounds of QARMAv2-128 ($\mathcal{T} = 2$). Data complexity: 2^{96} . Interpretation: In the ZC distinguisher, blue (red) cells take any (resp. an arbitrary nonzero) linear mask value. For the integral distinguisher, if all blue (white) cells at the input are fixed (active), and the highlighted tweak cell is active, then the colored output cell will be balanced.

J Integral Distinguishers of MANTIS

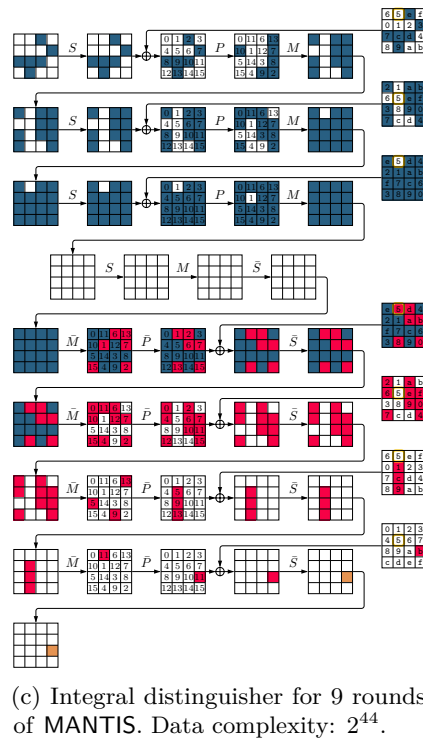
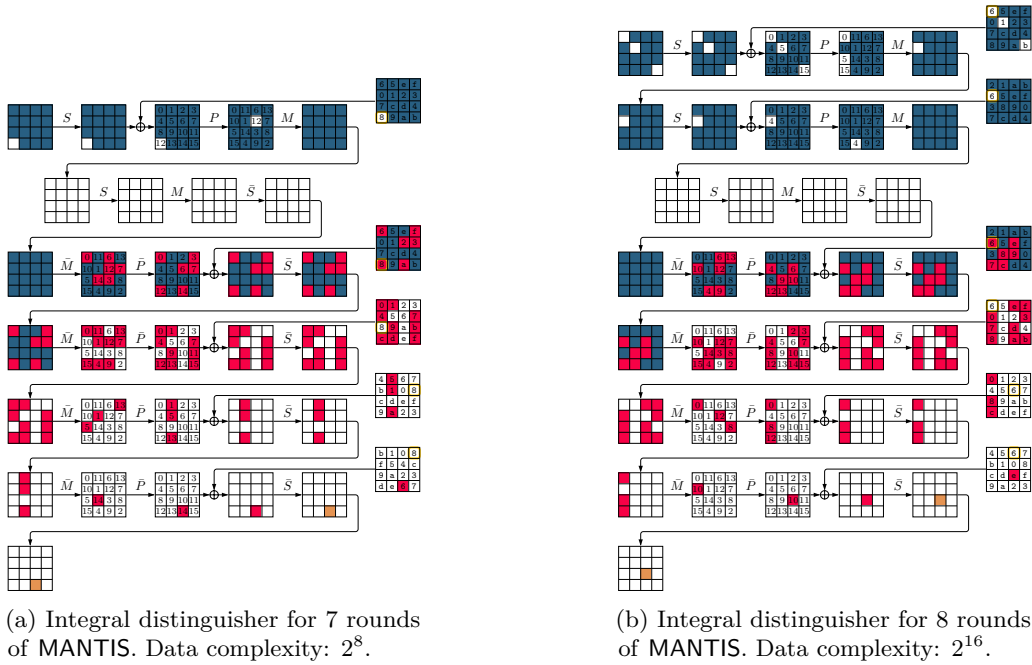
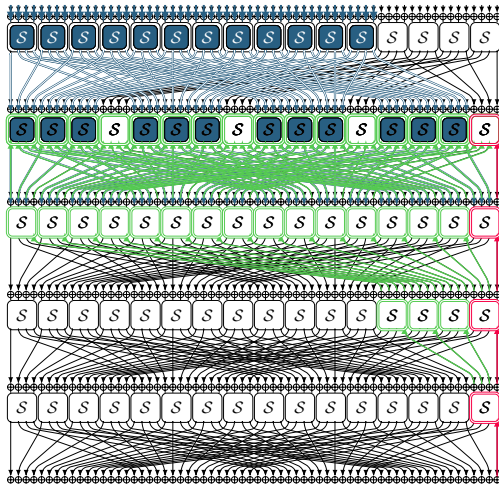
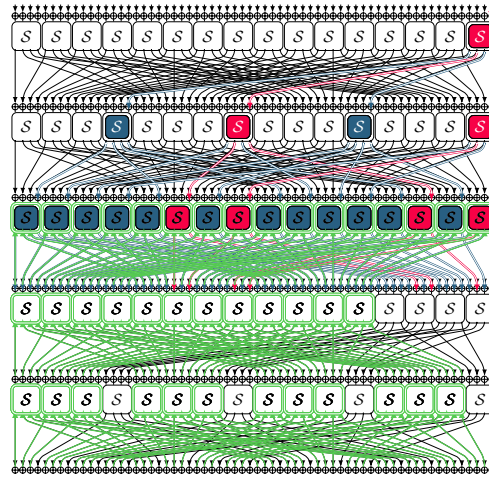


Figure 48: ZC-based integral distinguishers for MANTIS. Interpretation: For the integral distinguisher, if all blue (white) cells at the input are fixed (active), and the highlighted tweak cell is active, then the colored output cell will be balanced.

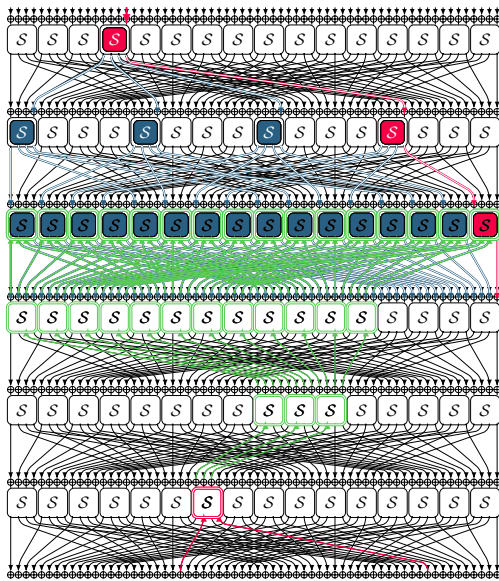
K ID and ZC Distinguishers of PRESENT



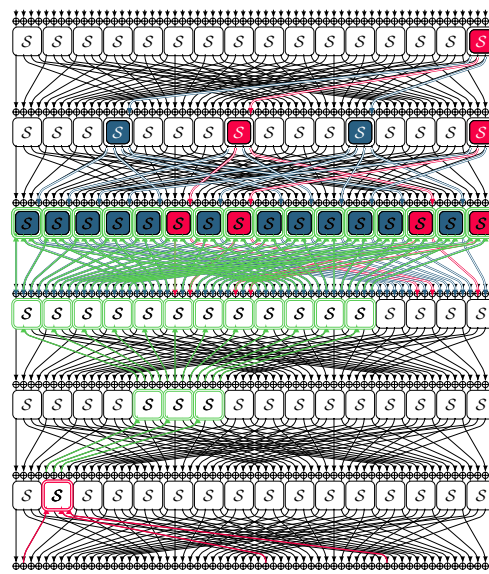
(a) A cluster of ID distinguishers for 5 rounds of PRESENT.



(b) A cluster of ZC distinguishers for 5 rounds of PRESENT.



(c) A cluster of ID distinguishers for 6 rounds of PRESENT.



(d) A cluster of ZC distinguishers for 6 rounds of PRESENT.

Figure 49: ID and ZC distinguishers for PRESENT.

L ID and ZC Distinguishers of Ascon

Here, we present the ZC and ID distinguishers that our tool derives for Ascon. When searching for the ZC or ID distinguisher, we set the objective function to maximize the number of unknown bits at the input/output of distinguishers. This way, any single output of the tool is indeed a cluster of distinguishers whose size depends on the number of unknown bits at the input/output of the distinguisher. The more unknown bits, the more distinguisher we have in the cluster. Figure 50, Figure 51, Figure 52, Figure 53, and Figure 54 illustrate some of the outputs of our tool when searching for a 5-round ZC/ID distinguisher for Ascon. We represent both forward and backward propagations in these figures. The unknown bit (in terms of difference value or mask value) in the forward and backward propagations are represented by \blacktriangleleft , and \blacktriangleright , respectively. In addition, the bit difference (or linear mask) 1 is represented by \blacktriangleleft and \blacktriangleright in forward and backward propagations, respectively. The result represented in Figure 50 can be derived with our tool running on a regular laptop in a few minutes. However, as seen in Figure 50 it is a cluster of 2^{155} ZC distinguishers. This should clarify the advantage of our searching method compared to the previous works [ST17, CCJ⁺16], where every single ZC or ID distinguisher should be derived separately when the input/output of the distinguishers are fixed to some specific difference or linear masks.

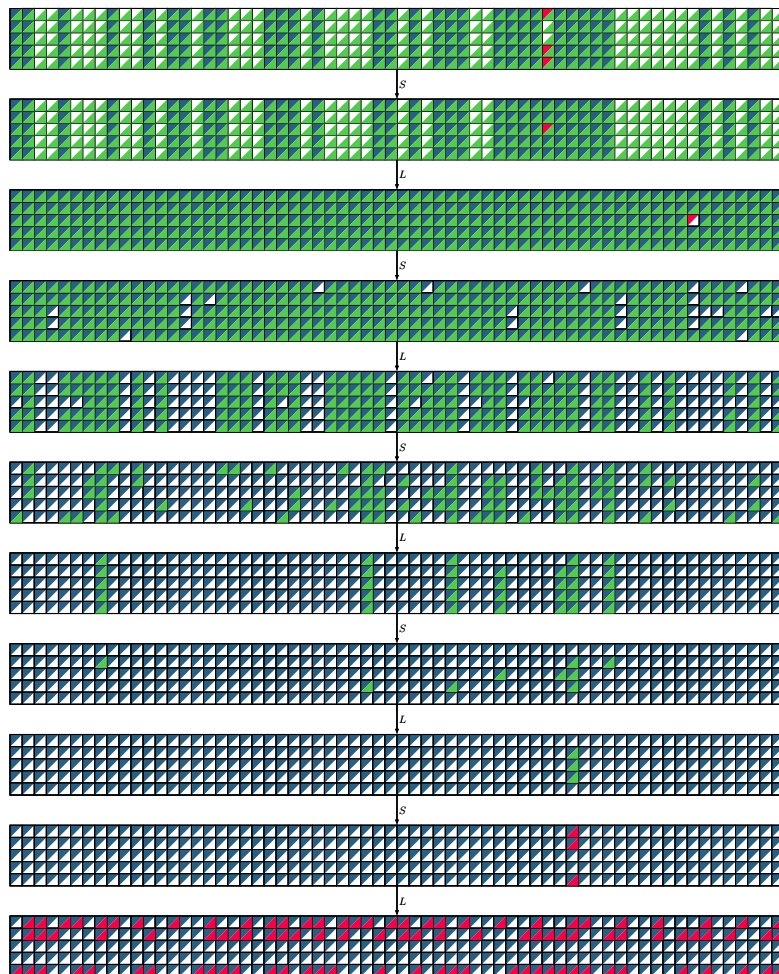


Figure 50: A cluster of 2^{155} ZC distinguishers for 5 rounds of Ascon.

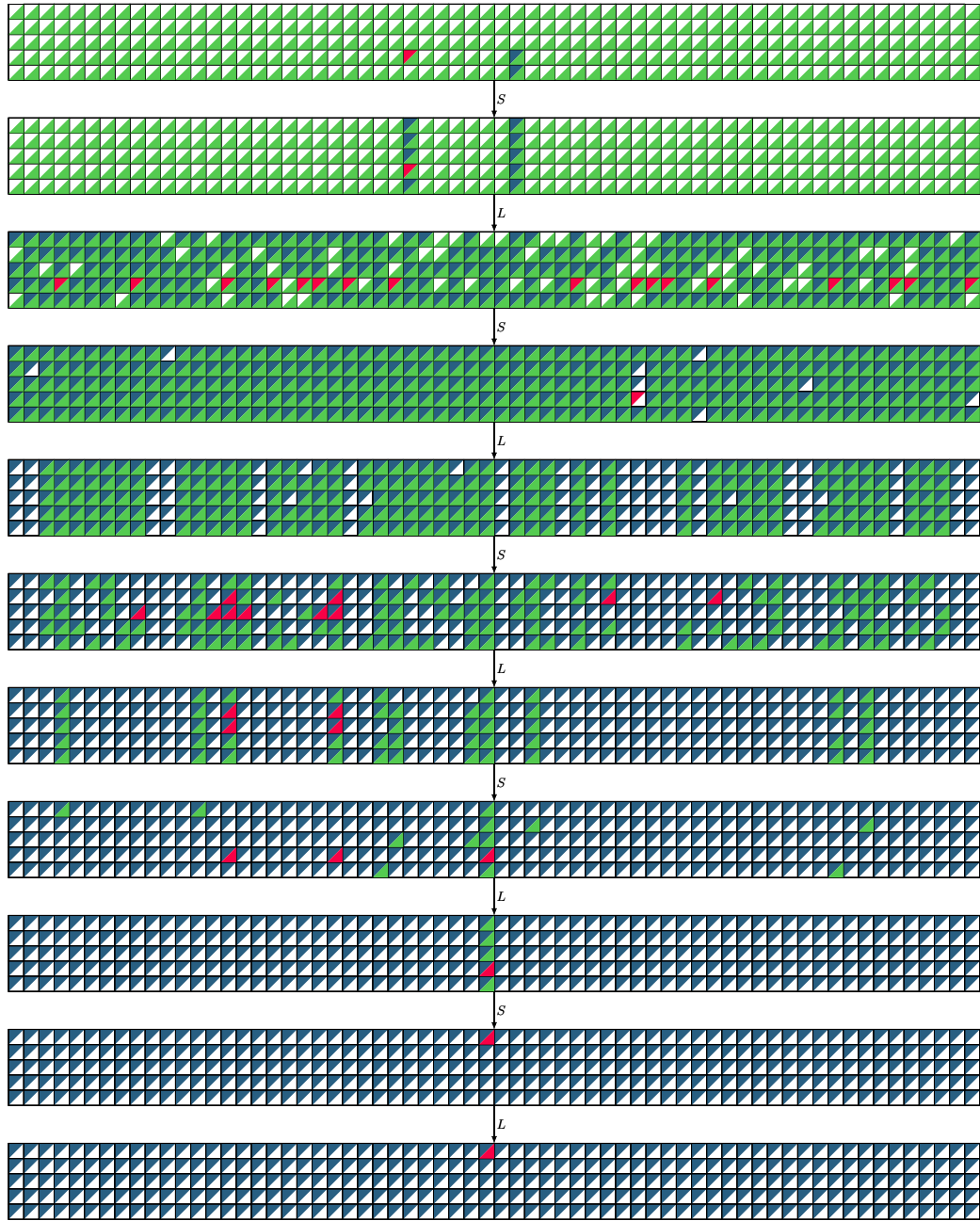


Figure 51: A cluster of 2^2 ZC distinguishers for 5 rounds of Ascon.

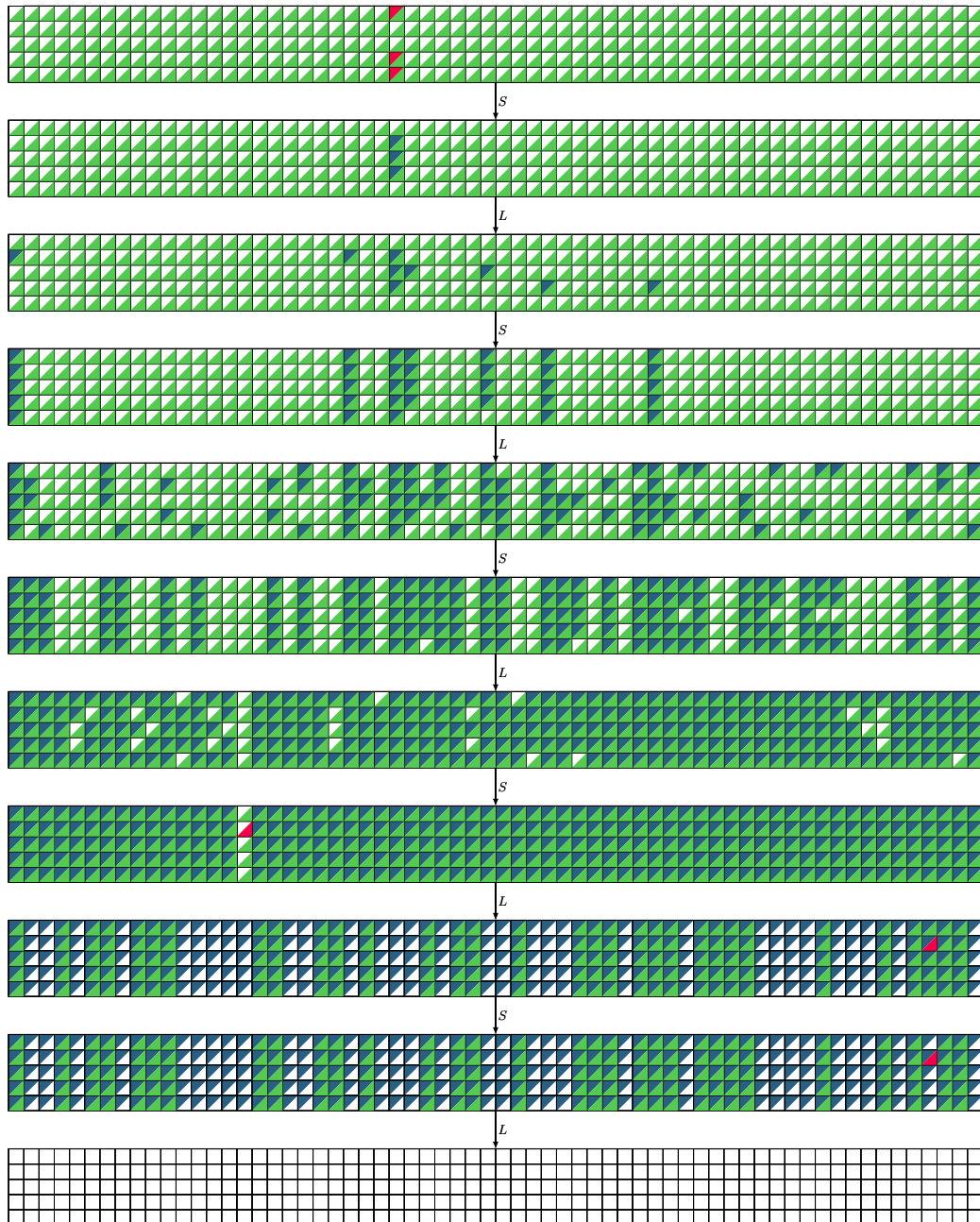


Figure 52: A cluster of 2^{155} ID distinguishers for 5 rounds of Ascon.

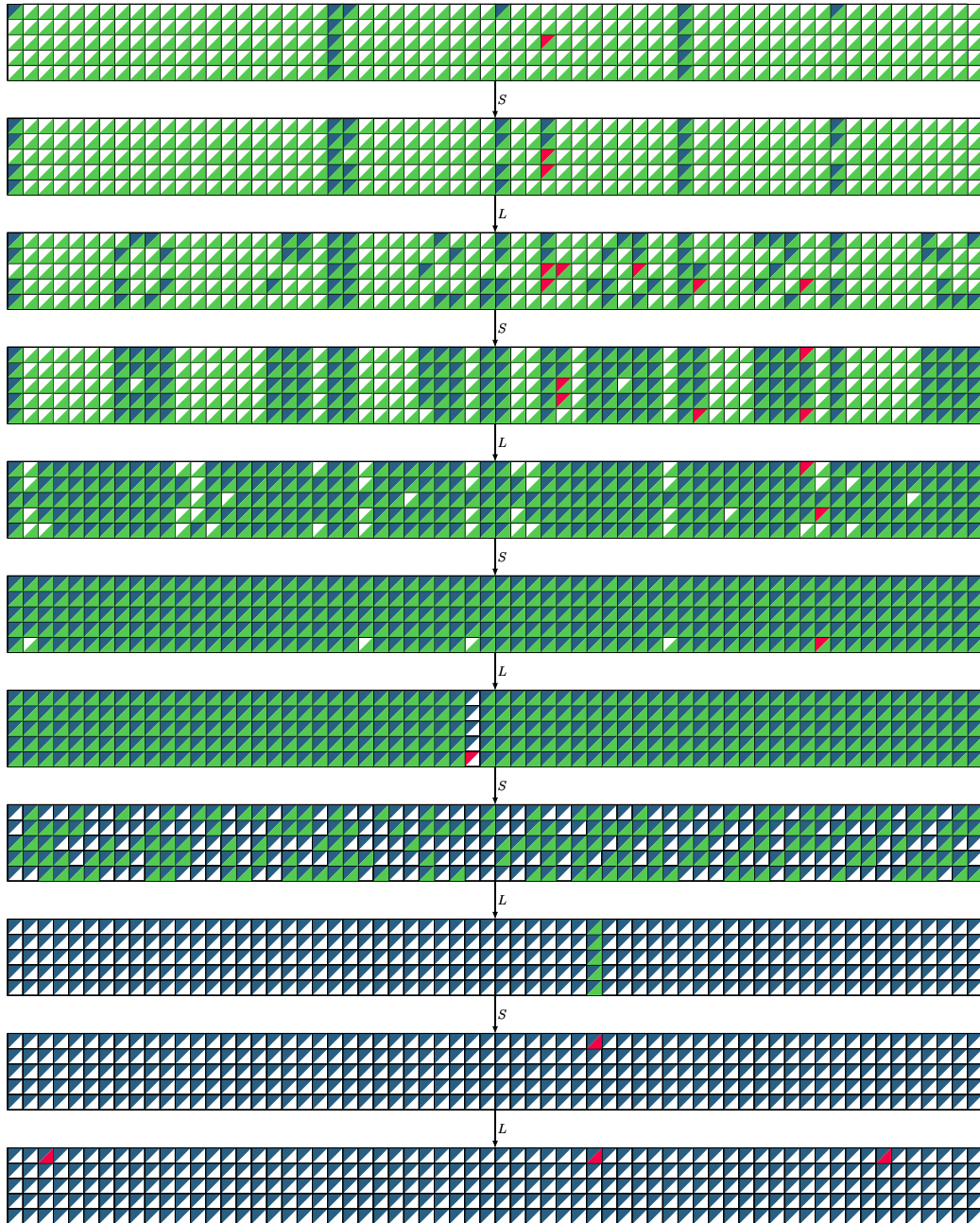


Figure 53: A cluster of 2^{14} ID distinguishers for 5 rounds of Ascon.

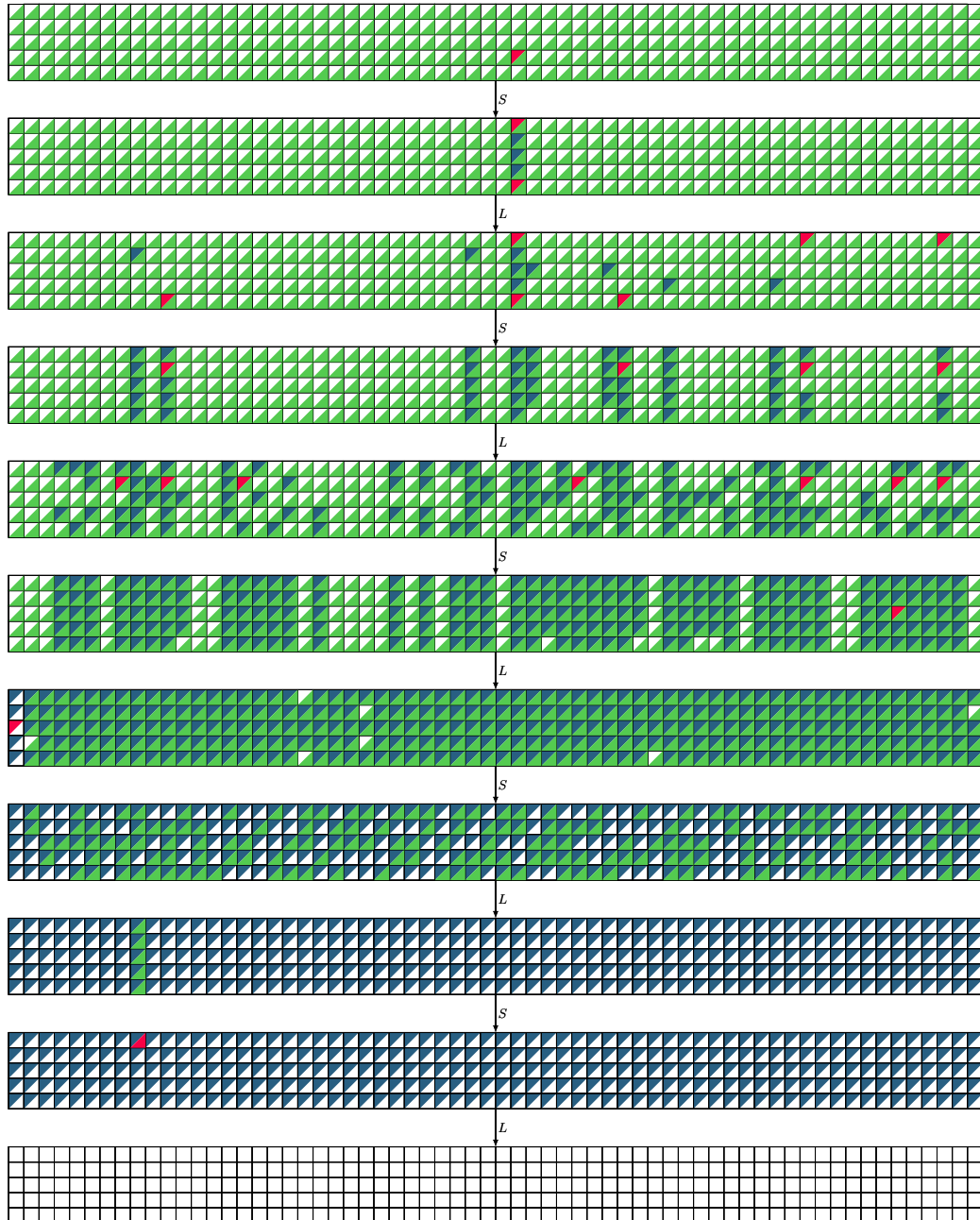


Figure 54: An ID distinguisher for 5 rounds of Ascon.

M Encoding Deterministic Behaviour of S-boxes

The S-box Analyzer is a SageMath [Sag22] module designed to encode S-boxes' differential, linear, and integral properties efficiently in MILP, SMT/SAT models. S-box Analyzer was applied to the differential, integral, and boomerang analysis of various block ciphers in [HNE22, HE22], and is publicly available at:

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

We expanded the S-box Analyzer to generate CP constraints in the MiniZinc language, making it easier for others to use our bit-wise method for finding ID, ZC, and integral distinguishers. Listing 1 briefly shows how to use the S-box Analyzer to derive CP constraints for deterministic differential/linear transitions through an S-box.

```

1 # Encoding deterministic differential behaviour of S-box
2 sage: from sboxanalyzer import *
3 sage: from sage.crypto.sboxes import MANTIS as sb
4 sage: sa = SboxAnalyzer(sb)
5 sage: diff = sa.encode_deterministic_differential_behavior()
6 sage: cpdiff = sa.generate_cp_constraints(diff)
7 sage: print(cpdiff)
8
9 # x3: msb, x0: lsb
10 if (x3 == 0 /\ x2 == 0 /\ x1 == 0 /\ x0 == 0) then (y3 = 0 /\ y2 = 0 /\ y1 = 0 /\ y0 = 0)
11 elseif (x3 == 0 /\ x2 == 0 /\ x1 == 1 /\ x0 == 0) then (y3 = -1 /\ y2 = -1 /\ y1 = 0 /\ y0 = -1)
12 elseif (x3 == 0 /\ x2 == 0 /\ x1 == -1 /\ x0 == 0) then (y3 = -1 /\ y2 = -1 /\ y1 = 0 /\ y0 = -1)
13 elseif (x3 == 1 /\ x2 == 1 /\ x1 == 0 /\ x0 == 1) then (y3 = -1 /\ y2 = -1 /\ y1 = 1 /\ y0 = -1)
14 elseif (x3 == 1 /\ x2 == 1 /\ x1 == 1 /\ x0 == 1) then (y3 = -1 /\ y2 = -1 /\ y1 = 1 /\ y0 = -1)
15 elseif (x3 == 1 /\ x2 == 1 /\ x1 == -1 /\ x0 == 1) then (y3 = -1 /\ y2 = -1 /\ y1 = 1 /\ y0 = -1)
16 else (y3 = -1 /\ y2 = -1 /\ y1 = -1 /\ y0 = -1)
17 endif
18
19 # Encoding deterministic linear behaviour of S-box
20 sage: lin = sa.encode_deterministic_linear_behavior()
21 sage: cplin = sa.generate_cp_constraints(lin)
22 sage: print(cplin)
23
24 # x3: msb, x0: lsb
25 if (x3 == 0 /\ x2 == 0 /\ x1 == 0 /\ x0 == 0) then (y3 = 0 /\ y2 = 0 /\ y1 = 0 /\ y0 = 0)
26 elseif (x3 == 0 /\ x2 == 0 /\ x1 == 1 /\ x0 == 0) then (y3 = -1 /\ y2 = -1 /\ y1 = 0 /\ y0 = -1)
27 elseif (x3 == 0 /\ x2 == 0 /\ x1 == -1 /\ x0 == 0) then (y3 = -1 /\ y2 = -1 /\ y1 = 0 /\ y0 = -1)
28 elseif (x3 == 0 /\ x2 == 1 /\ x1 == 0 /\ x0 == 1) then (y3 = 1 /\ y2 = -1 /\ y1 = 1 /\ y0 = -1)
29 elseif (x3 == 0 /\ x2 == 1 /\ x1 == 1 /\ x0 == 1) then (y3 = -1 /\ y2 = -1 /\ y1 = 1 /\ y0 = -1)
30 elseif (x3 == 0 /\ x2 == 1 /\ x1 == -1 /\ x0 == 1) then (y3 = -1 /\ y2 = -1 /\ y1 = 1 /\ y0 = -1)
31 else (y3 = -1 /\ y2 = -1 /\ y1 = -1 /\ y0 = -1)
32 endif

```

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