Improved Search for Integral, Impossible Differential and Zero-Correlation Attacks
Application to Ascon, ForkSKINNY, SKINNY, MANTIS, PRESENT and QARMAv2

Hosein Hadipour$^{(1)}$, Simon Gerhalter$^1$, Sadegh Sadeghi$^2$ and Maria Eichlseder$^{(1)}$

$^1$ Graz University of Technology, Graz, Austria
hossein.hadipour@iaik.tugraz.at, maria.eichlseder@iaik.tugraz.at, s.gerhalter@student.tugraz.at

$^2$ Department of Mathematics, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan 45137-66731, Iran
s.sadeghi@iasbs.ac.ir

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-$n$ 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 \cite{FKL00}, 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 \cite{LDKK08}. The integral attack was initially introduced as a theoretical generalization of differential analysis by Lai \cite{Lai94}, and as a practical attack by Daemen et al. \cite{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. \cite{BBS99} and Knudsen \cite{Knu98}, exploits an impossible differential in a block cipher to retrieve the master key. The ZC attack, first introduced by Bogdanov and Rijmen \cite{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 \cite{BLNW12}, Bogdanov et al. established a link between the (multidimensional) ZC approximation and integral distinguishers. At CRYPTO 2015 \cite{SLR15}, 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 \cite{BBS99} was later used to find ZC distinguishers \cite{BLNW12}. At ToSC 2019 \cite{ADG19}, 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, a 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 \cite{ST17}. At the same time, Cui et al. proposed a similar approach to finding ID and ZC distinguishers \cite{CCJ16}. 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 \cite{Tod15,XZBL16} or monomial prediction \cite{HSWW20,HE22}.

At EUROCRYPT 2023, Hadipour et al. \cite{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 \cite{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 \cite{HSE23} is considering some key recovery techniques, such as key-bridging, to find a better attack. However, the method in \cite{HSE23} has some limitations as mentioned by the authors:

- In the CP model for distinguishers in \cite{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 \cite{HSE23} is a cell-wise model that was proved to work very well for strongly aligned or cell-wise designs such as **SKINNY** \cite{BJK16}, but not for weakly aligned and bit-wise ciphers, such as **Ascon** \cite{DEMS21} and **PRESENT** \cite{BKL07}. So, the second question is how to extend the method in \cite{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 \cite{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 \cite{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 \cite{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** \cite{ALP19}, **MANTIS** \cite{BJK16}, **QARMAv2** \cite{ABD23} and got a series of significantly improved results.

Regarding **QARMAv2**, we discovered a 12-round (resp. 10-round) integral distinguisher for **QARMAv2-128** (resp. **QARMAv2-64**), outperforming the best integral distinguisher by 7 (resp. 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 \cite{ABD23}. 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 \( (r_f) \), and the length of output branches \( (r_0, r_1) \). 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 E). 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 the 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 M, 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 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 issue (see Appendix A). *: 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 E).

<table>
<thead>
<tr>
<th>Cipher</th>
<th>#R</th>
<th>Time</th>
<th>Data</th>
<th>Mem.</th>
<th>Attack</th>
<th>Setting / Model</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKINNY-64-64</td>
<td>17</td>
<td>2^{61.80}</td>
<td>2^{59.50}</td>
<td>2^{49.60}</td>
<td>ID</td>
<td>STK / CP</td>
<td>[YQC17]</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>2^{59}</td>
<td>2^{58.79}</td>
<td>2^{40}</td>
<td>ID</td>
<td>STK / CP</td>
<td>[HSE23]</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>2^{53.58}</td>
<td>2^{53.58}</td>
<td>2^{48}</td>
<td>Int</td>
<td>60,SK / CP,CT</td>
<td>4.2</td>
</tr>
<tr>
<td>SKINNY-128-128</td>
<td>17</td>
<td>2^{120.80}</td>
<td>2^{118.50}</td>
<td>2^{97.50}</td>
<td>ID</td>
<td>STK / CP</td>
<td>[YQC17]</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>2^{116.51}</td>
<td>2^{116.37}</td>
<td>2^{90}</td>
<td>ID</td>
<td>STK / CP</td>
<td>[HSE23]</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>2^{105.58}</td>
<td>2^{105.58}</td>
<td>2^{96}</td>
<td>Int</td>
<td>120,SK / CP,CT</td>
<td>4.2</td>
</tr>
<tr>
<td>SKINNY-64-128</td>
<td>20</td>
<td>2^{37.50}</td>
<td>2^{68.40}</td>
<td>2^{32}</td>
<td>Int</td>
<td>120,SK / CP,CT</td>
<td>[ADG+19]</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>2^{110}</td>
<td>2^{57.58}</td>
<td>2^{108}</td>
<td>Int</td>
<td>120,SK / CP,CT</td>
<td>[HSE23]</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>2^{106}</td>
<td>2^{58}</td>
<td>2^{104}</td>
<td>Int</td>
<td>120,SK / CP,CT</td>
<td>D.1</td>
</tr>
<tr>
<td>SKINNY-128-256</td>
<td>22</td>
<td>2^{216}</td>
<td>2^{113.58}</td>
<td>2^{216}</td>
<td>Int</td>
<td>240,SK / CP,CT</td>
<td>[HSE23]</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>2^{213}</td>
<td>2^{114}</td>
<td>2^{208}</td>
<td>Int</td>
<td>240,SK / CP,CT</td>
<td>D.1</td>
</tr>
<tr>
<td>SKINNY-64-192</td>
<td>23</td>
<td>2^{255.60}</td>
<td>2^{73.20}</td>
<td>2^{138}</td>
<td>Int</td>
<td>180,SK / CP,CT</td>
<td>[ADG+19]</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>2^{172}</td>
<td>2^{61}</td>
<td>2^{172}</td>
<td>Int</td>
<td>180,SK / CP,CT</td>
<td>[HSE23]</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>2^{166}</td>
<td>2^{62}</td>
<td>2^{164}</td>
<td>Int</td>
<td>180,SK / CP,CT</td>
<td>D.2</td>
</tr>
<tr>
<td>SKINNY-128-384</td>
<td>26</td>
<td>2^{144}</td>
<td>2^{121}</td>
<td>2^{240}</td>
<td>Int</td>
<td>360,SK / CP,CT</td>
<td>[HSE23]</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>2^{131}</td>
<td>2^{122}</td>
<td>2^{228}</td>
<td>Int</td>
<td>360,SK / CP,CT</td>
<td>D.2</td>
</tr>
<tr>
<td>SKINNY-128-288</td>
<td>26</td>
<td>2^{286.38}</td>
<td>2^{222}</td>
<td>2^{194}</td>
<td>ID</td>
<td>288,RTK / CP</td>
<td>L.1</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>2^{260.73}</td>
<td>2^{121.80}</td>
<td>2^{188}</td>
<td>ID</td>
<td>128,RTK / CP</td>
<td>L.2</td>
</tr>
<tr>
<td>SKINNY-e-v2</td>
<td>30</td>
<td>2^{232}</td>
<td>2^{65}</td>
<td>2^{228}</td>
<td>Int</td>
<td>240,SK / CP,CT</td>
<td>[HSE23]</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>2^{251.14}</td>
<td>2^{63}</td>
<td>2^{110}</td>
<td>ID</td>
<td>RTK / CP</td>
<td>L.3</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>2^{166}</td>
<td>2^{58}</td>
<td>2^{164}</td>
<td>Int</td>
<td>180,SK / CP,CT</td>
<td>F.2</td>
</tr>
<tr>
<td>ForkSKINNY-64-192</td>
<td>32</td>
<td>2^{186.27}</td>
<td>2^{63}</td>
<td>2^{114}</td>
<td>ID</td>
<td>192,RTK / CP</td>
<td>H.3</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>2^{123.73}</td>
<td>2^{63}</td>
<td>2^{86}</td>
<td>ID</td>
<td>128,RTK / CP</td>
<td>H.4</td>
</tr>
<tr>
<td></td>
<td>28+</td>
<td>2^{169.60}</td>
<td>2^{61}</td>
<td>2^{104}</td>
<td>ID</td>
<td>192,RTK / CP</td>
<td>H.1</td>
</tr>
<tr>
<td></td>
<td>28+</td>
<td>2^{123.73}</td>
<td>2^{63}</td>
<td>2^{86}</td>
<td>ID</td>
<td>128,RTK / CP</td>
<td>H.2</td>
</tr>
<tr>
<td>ForkSKINNY-128-256</td>
<td>23</td>
<td>2^{114}</td>
<td>2^{114}</td>
<td>2^{208}</td>
<td>Int</td>
<td>240,SK / CP,CT</td>
<td>F.1</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>2^{254.6}</td>
<td>2^{125}</td>
<td>2^{160}</td>
<td>ID</td>
<td>256,RTK / CP</td>
<td>[BDL20]</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>2^{250.3}</td>
<td>2^{127}</td>
<td>2^{160}</td>
<td>ID</td>
<td>256,RTK / CP</td>
<td>[BDL20]</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>2^{238.50}</td>
<td>2^{128.60}</td>
<td>2^{170.60}</td>
<td>ID</td>
<td>256,RTK / CP</td>
<td>H.5</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>2^{244.50}</td>
<td>2^{128}</td>
<td>2^{175}</td>
<td>ID</td>
<td>256,RTK / CP</td>
<td>H.5</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>2^{124.5}</td>
<td>2^{222.50}</td>
<td>2^{297}</td>
<td>ID</td>
<td>128,RTK / CP</td>
<td>[BDL20]</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>2^{123.17}</td>
<td>2^{119.40}</td>
<td>2^{118.40}</td>
<td>ID</td>
<td>128,RTK / CP</td>
<td>H.7</td>
</tr>
<tr>
<td></td>
<td>24+</td>
<td>2^{240.62}</td>
<td>2^{127.70}</td>
<td>2^{153.70}</td>
<td>ID</td>
<td>256,RTK / CP</td>
<td>H.7</td>
</tr>
<tr>
<td></td>
<td>20+</td>
<td>2^{102.20}</td>
<td>2^{102}</td>
<td>2^{53}</td>
<td>ID</td>
<td>128,RTK / CP</td>
<td>H.8</td>
</tr>
<tr>
<td>ForkSKINNY-128-288</td>
<td>31</td>
<td>2^{280.52}</td>
<td>2^{127.50}</td>
<td>2^{190.50}</td>
<td>ID</td>
<td>288,RTK / CP</td>
<td>H.9</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>2^{126.68}</td>
<td>2^{125.80}</td>
<td>2^{100.69}</td>
<td>ID</td>
<td>128,RTK / CP</td>
<td>H.10</td>
</tr>
<tr>
<td></td>
<td>28+</td>
<td>2^{277.23}</td>
<td>2^{127.90}</td>
<td>2^{158.90}</td>
<td>ID</td>
<td>288,RTK / CP</td>
<td>H.11</td>
</tr>
<tr>
<td></td>
<td>26+</td>
<td>2^{126.74}</td>
<td>2^{125.50}</td>
<td>2^{68.50}</td>
<td>ID</td>
<td>128,RTK / CP</td>
<td>H.12</td>
</tr>
</tbody>
</table>

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

Bihm 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 $E$ is a pair of differences $(\Delta_u, \Delta_l)$ such that $\Delta_u$ can never lead to $\Delta_l$. 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 $\Delta_u$ propagates backward (resp. forward) with probability 1 through $E_{b-1}$ (resp. $E_{f}$) to $\Delta_b$ (resp. $\Delta_f$), and $|\Delta_u|$ (|$\Delta_l$|) denotes the dimension of the vector space $\Delta_b$ (resp. $\Delta_f$). 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 $x \oplus y \in \Delta_u$ 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:

$$T_0 = \max \left\{ \min_{\Delta \in \{\Delta_u, \Delta_f\}} \left\{ \sqrt{N2^{n+1-|\Delta|}} \cdot N2^{n+1-|\Delta_u|-|\Delta_f|} \right\} \right\}$$

- **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]:

$$T_1 + T_2 = N + 2^{k_b \cup k_f} \cdot N \frac{2n+c}{2n+c+1}$$
As a result, the complexity formulations of the related-(twea)-key setting are the same as 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:

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

(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 $\Delta K$. 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:

$$T_{0}^{RTK} = \max \left\{ \min_{\Delta \in \{\Delta_l, \Delta_r\}} \left\{ 2\sqrt{N2^{n-|\Delta_l|}}, N2^{n+1-|\Delta_l|-|\Delta_r|} \right\}, D^{RTK} = T_{0}^{RTK} \right\}$$

(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 more practical and faster than creating an optimization model that considers the exact complexity of the attack. Making a model that considers the exact complexity of the ID attack becomes very complicated and hard to handle. This makes it impractical to solve in a reasonable time and likely gives the same result as a simpler model, even if the simpler one can be solved faster.

### 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. [SLR15] 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 $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ 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 $(\alpha, \beta)$ is a ZC approximation for any $\alpha \in A$. Then, for any $\lambda \in \mathbb{F}_2^n$, $(\beta, F(x + \lambda))$ is balanced over the set

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

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 tweakey on ZC distinguishers of tweakable block ciphers (TBCs). Their findings revealed that considering the tweakey 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 superposition tweakey (STK) construction of the TWEAKEY framework [JNP14] (see Figure 2).

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

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 $4 \times 4$ 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 $K_5$, 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:

$$C_4[0] = S^{-1} (K_5[0] \oplus 0E \cdot S^{-1} (C_6[0] \oplus K_6[0]) \oplus 09 \cdot S^{-1} (C_6[7] \oplus K_6[7])$$

$$\oplus 0D \cdot S^{-1} (C_6[10] \oplus K_6[10]) \oplus 0E \cdot S^{-1} (C_6[13] \oplus K_6[13]),$$

(5)

where $S$ 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 $2^{32}$ plaintexts 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 $0E \cdot S^{-1}()$ by $S_0()$, $09 \cdot S^{-1}()$ by $S_1()$, $0D \cdot S^{-1}()$ by $S_2()$, and $0E \cdot S^{-1}()$ by $S_3()$. Note that $S_0$, $S_1$, $S_2$, and $S_3$ 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 $p_1 = S_0(C_6[0] \oplus K_6[0]) \oplus S_1(C_6[7] \oplus K_6[7])$ 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 $L_1$ 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 $L_1$ 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 $p_2 = p_1 \oplus S_2(C_6[10] \oplus K_6[10])$ for all $(p_1, C_6[10,13])$ such that $L_1[(p_1, C_6[10,13])] = 1$. We create a
We compute

Algorithm 1 briefly describes the entire procedure. The total complexity for a single set storing $2$ complex of the partial-sum technique is equivalent to about $S$-box lookups. One involved key bits uniquely. Therefore, the total time complexity is about need to repeat the whole procedure for checking one balanced byte provides an $8$-bit filter and $2$ of $K$ is zero, we add of this step is all possible values of parity of occurrences of $p$ lookups, and the memory complexity is $2$ 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 $L_1$ 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 $p_3 = p_2 \oplus S_3(C_6[13] \oplus K_6[13])$ for all $(p_2, C_6[13])$ such that $\mathcal{L}_2[(p_2, C_6[13])] = 1$. We store the 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 $K_5[0]$, and compute $\bigoplus_{p_3 \in \mathbb{F}_2^8: \mathcal{L}_3[p_3]=1} S^{-1}(\bar{K}_5[0] \oplus p_3)$. If the result is zero, we add $\bar{K}_5[0] || K_5[0, 7, 11, 13]$ 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 $2^{16} \cdot (2^{32} + 2^8 \cdot (2^{24} + 2^8 \cdot (2^{16} + 2^8 \cdot 2^{48}))) = 2^{50}$ 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.

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 \((X, D, C)\), with \(X = \{X_0, X_1, \ldots, X_{n-1}\}\) representing the variable set; \(D = \{D_0, D_1, \ldots, D_{n-1}\}\) as the domain set, where \(X_i \in D_i\) for \(0 \leq i \leq n-1\); and \(C = \{C_0, C_1, \ldots, C_{n-1}\}\) serving as the constraint set. Each constraint \(C_j \in C\) is a tuple \((S_j, R_j)\), where \(S_j = \{X_{i_0}, \ldots, X_{i_k}\} \subseteq X\), and \(R_j\) is a relation defined on the corresponding domain, that is, \(R_j \subseteq D_{i_0} \times \cdots \times D_{i_k}\). 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 \(\Delta X = (\Delta X[0], \ldots, \Delta X[m-1])\) denote the difference value of the internal state \(X\) in an \(n\)-bit block cipher \(E\), where \(n = m \cdot c\), and \(\Delta X[i] \in F_2^c\) for all \(0 \leq i \leq m - 1\). We
use the variable $AX[i] \in \{0, \cdots, 3\}$ to represent the activity pattern of $\Delta X[i]$, and another variable $DX[i] \in \{-2, -1, 0, \cdots, 2^c - 1\}$ indicates the actual $c$-bit value of $\Delta X[i]$. We use $-2$ and $-1$ for $DX[i]$ 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:

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

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

**Proposition 1** (XOR). For $F : \mathbb{F}_2^{2c} \rightarrow \mathbb{F}_2$, $F(X, Y) = Z$ where $Z = X \oplus Y$, the valid transitions for deterministic truncated differential trails satisfy

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

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

\[
\text{Branch}(AX, DX, AY, DY, AZ, DZ) := (AZ = AX \land DZ = DX \land DY = DX \land DY = DX)
\]

**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 differential trails satisfy

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

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 C.

### 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., $r_0$, and $r_1$, 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.
This adjustment enables the CSP model to automatically determine the optimal location for the contradiction, thereby optimizing the objective function.

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 $r_0$ 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 $AXU_i$ (resp. $AXL_i$) to represent the activeness pattern of the internal state after $r$ rounds in the forward direction (resp. backward direction). Similarly, we also define the integer variables $LXU_i$ (resp. $LXL_i$) 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 independently. We add the constraints $\sum_{i=0}^{m-1} AXU_0[i] \neq 0$, and $\sum_{i=0}^{m-1} AXL_0[i] \neq 0$ to exclude all trivial solutions in the single-tweakey setting. Let $CSP_{z,0}(AXU_0, LXU_0, \ldots, AXU_{r_0}, LXU_{r_0})$ and $CSP_{z,1}(AXL_0, LXL_0, \ldots, AXL_{r_0}, LXL_{r_0})$ 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:

$$\text{CSP}_{M} := \bigcap_{i=0}^{r_0} \left( m^{-1} \bigwedge_{j=0} \left( AXU_i[j] + AXL_i[j] < 3 \right) \wedge \bigwedge_{j=0} \left( AXU_i[j] \neq AXL_i[j] \right) \right)$$

(6)

The conjunction of the CSP models above, i.e., $CSP_0 = CSP_{z,0} \land CSP_{z,1} \land CSP_{z,2}$, 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 $r_0$, this model yields a distinguisher for $r_0$ 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., $\max \left( \sum_{i=0}^{m-1} AXU_0[i] + \sum_{i=0}^{m-1} AXL_0[i] \right)^2$, 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 $STK_i[i]$ denote the $i$th cell of the subtweakey after $r$ rounds. We define an integer

---

1In the related-tweakey setting, we ensure $\Delta P \neq 0 \lor \Delta TK \neq 0 \lor \Delta C \neq 0$.

2The second summation is not present when searching for the integral distinguishers.
variable $\text{ASTK}^r[i] \in \{0, 1, 2, 3\}$ to encode the activation pattern of $STK^r[i]$. By the way, let $\text{AYL}_r[i]$ (resp. $\text{AYU}_r[i]$) encode the activation pattern of the internal state at the subwteakey addition in round $r$ in forward (resp. backward) propagation. Now, we add the new constraint $\text{ASTK}^r[i] = \min\{\text{AYL}_r[i], \text{AYU}_r[i]\}$ for all $0 \leq r \leq r_0 - 1$, and $0 \leq i \leq m - 1$, to link the activeness pattern of the subwteakey to the activeness pattern of the internal state. This simple constraint causes the subtweakey 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:

$$
\text{CSP}_{TK}(\text{ASTK}_0, ..., \text{ASTK}_{r_0 - 1}) := \\
\bigvee_{i=0}^{r_0-1} \left( \sum_{r=0}^{r_0-1} \text{bool2int}(\text{ASTK}^r[h^{-r}(i)] \neq 0) \leq z \right) \land \left( \bigwedge_{i=0}^{r_0-1} \text{ASTK}^r[h^{-r}(i)] = 1 \right) \lor \left( \bigwedge_{i=0}^{r_0-1} \text{ASTK}^r[h^{-r}(i)] = 0 \right)
$$

(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 $r_0 = 27$, see the specification of ForkSKINNY in Appendix E). 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-128-256) 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 $r_0 = r_1$ 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 $r_0 = r_1$ or $r_0 \neq r_1$. 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 $(r_1, r_0, r_1)$.

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 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 \( f : \mathbb{F}_2 \rightarrow \mathbb{F}_2^n \), \( f(x) = (y_0, y_1, \ldots, y_{n-1}) \) where \( y_0 = y_1 = \cdots = x \), the valid transitions for deterministic differential trails satisfy

\[
\text{Branch}_i(Ax, Ay[0], \ldots, Ay[n-1]) := \bigwedge_{i=0}^{n-1} (Ay[i] = Ax),
\]

where \( Ax \) and \( Ay[i] \) are integer variables with the domain of \{-1, 0, 1\} for all \( 0 \leq i \leq n-1 \).

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

\[
\text{XOR}_i(Ax, Ax[0], \ldots, Ax[n-1]) := \begin{cases} 
\text{if } \sqrt{\sum_{i=0}^{n-1} (Ax[i] = -1)} \text{ then } Ay = -1 \\
\text{else } Ay = Ax[0] + Ax[1] + \cdots + Ax[n-1] \mod 2 \end{cases}
\]

where \( Ax[i] \) 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 MANITIS as an example. We define the integer variables \( Ax[i] \) and \( Ay[i] \) for \( 0 \leq i \leq 3 \) to represent the bit differences at the input and output of this S-box, respectively, where \( Ax[0] \) and \( Ay[0] \) 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 N):

\[
\begin{align*}
&\text{if } (AX)[0] = 0 \land AX[1] = 0 \land AX[2] = 0 \land AX[3] = 0 \text{ then } (AY)[0] = 0 \land AY[1] = 0 \land AY[2] = 0 \land AY[3] = 0) \\
&\text{else } (AY)[0] = -1 \land AY[1] = -1 \land AY[2] = -1 \land AY[3] = -1) \text{ endif}
\end{align*}
\]

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 \(r\)th bit of internal state in round \(r\) in forward (resp. backward) propagation. Then, we include \(CSP_M := \bigvee_{r=0}^{r-1} \left( \bigvee_{i=0}^{i-1} (AXU_r[i] + AXL_r[i] = 1) \right)\) to ensure that the two deterministic propagations are inconsistent 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, e.g., Ascon, and PRESENT. When searching for ID and ZC distinguishers of Ascon, we include the objective function \(\min \left( \sum_{i=0}^{n-1} AXU_r[i] + \sum_{i=0}^{n-1} AXL_r[i] \right)\) 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 \(\Box\) and \(\blacksquare\) respectively. In addition, the bit difference (or linear mask) 1 is represented by \(\blacksquare\) and \(\Box\) in forward and backward propagations, respectively. As seen in Figure 50, the contradiction happens at 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 L). 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, SKINNyee, CRAFT, Deoxys-BC, MANTIS, QARMAv2, Ascon, and PRESENT. For instance, our CP model aimed at finding an ID distinguisher for 6 (or 7) rounds of Ascon (or PRESENT) returns UNSATISFIABLE. This indicates that, under the assumptions of our model, e.g., round independence and round-key independence, there is no ID distinguisher for 6 (or 7) rounds of Ascon (or PRESENT). Note that the majority of existing ID/ZC distinguisher tools also neglect cross-round or round-key dependencies.

Moreover, we assume that deterministic bit-wise (or word-wise) forward and backward propagations contradict each other in at least one bit (or word) throughout the distinguisher. 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 difference. 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

---

3It is worth mentioning that [HLJ+20] takes into account the impact of the key schedule, and [XLJ+22] considers specific cross-round 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.
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 \( STK_{17} \) 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 \( X_{15}[4] \).

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed</th>
<th>K × D = Mem</th>
<th>Time</th>
<th>Stored Texts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>( 2^0 \times 2^{32} = 2^{32} )</td>
<td>( 2^{32} - 5.2 )</td>
<td>( Z_{18}[1, 2, 4, 7]; X_{18}[8, 11 - 15]; X_{17}[12]; STK_{16}[5] )</td>
</tr>
<tr>
<td>1</td>
<td>( STK_{18} [4] )</td>
<td>( 2^4 \times 2^{32} = 2^{36} )</td>
<td>( 2^{36} - 7.2 )</td>
<td>( Z_{18}[1, 2, 7]; X_{18}[11, 13 - 15]; Z_{17}[0, 7]; X_{17}[10, 12]; STK_{16}[5] )</td>
</tr>
<tr>
<td>2</td>
<td>( STK_{18} [1] )</td>
<td>( 2^8 \times 2^{32} = 2^{40} )</td>
<td>( 2^{40} - 7.2 )</td>
<td>( Z_{18}[2, 7]; X_{18}[11, 14, 15]; Z_{17}[0, 7]; X_{17}[10, 12, 14]; STK_{16}[5] )</td>
</tr>
<tr>
<td>3</td>
<td>( STK_{18} [7] )</td>
<td>( 2^{12} \times 2^{32} = 2^{44} )</td>
<td>( 2^{44} - 8.2 )</td>
<td>( Z_{18}[2]; X_{18}[14]; Z_{17}[0, 6, 7]; X_{17}[10, 12, 14]; STK_{16}[5] )</td>
</tr>
<tr>
<td>4</td>
<td>( STK_{18} [2] )</td>
<td>( 2^{16} \times 2^{32} = 2^{48} )</td>
<td>( 2^{48} - 7.2 )</td>
<td>( Z_{17}[0, 6, 7]; X_{17}[10, 12, 14, 15]; STK_{16}[5] )</td>
</tr>
<tr>
<td>5</td>
<td>( STK_{17} [6] )</td>
<td>( 2^{20} \times 2^{20} = 2^{40} )</td>
<td>( 2^{40} - 7.2 )</td>
<td>( Z_{17}[0, 7]; X_{17}[12, 15]; X_{16}[5] )</td>
</tr>
<tr>
<td>6</td>
<td>( STK_{17} [0] )</td>
<td>( 2^{24} \times 2^{16} = 2^{40} )</td>
<td>( 2^{44} - 7.2 )</td>
<td>( Z_{17}[7]; X_{17}[15]; X_{16}[5, 13] )</td>
</tr>
<tr>
<td>7</td>
<td>( STK_{17} [7] )</td>
<td>( 2^{28} \times 2^4 = 2^{32} )</td>
<td>( 2^{44} - 6.7 )</td>
<td>( X_{15}[4] )</td>
</tr>
</tbody>
</table>

\[ \Sigma \]

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]. 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
(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 $2^4$.

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 B. 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 $\text{X}F$, and the active input cells $\text{AI}$ from the distinguisher. Furthermore, the tweakey setting $z$, the start ($R_s = r_n + r_0$) and end ($R = r_n + r_0 + r_s$) round of the key recovery, and the active tweakey $\text{AT}$ 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 $\text{IXF}, \text{IWF}$ 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 $\text{IXF}_R_i$ to True (1) at the balanced positions; otherwise it takes False (0). Then, using the properties of $\text{MixColumns}$ and the permutation $\text{ShiftRows}$, we deduce the involvement of each cell until we reach the last round. $\text{MixColumnsForwards}$ models the inverse $\text{MixColumns}$ matrix $M^{-1}$ (see Appendix B):

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

**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
Afterward, the step numbers in Algorithm 2. Based on the conditions mentioned in the previous paragraph, we store which subtweakey positions are involved in the key recovery in.

The precomputed input $20$ Improved Search for Integral, Impossible Differential and Zero-Correlation Attacks

S-Boxes are passed in each step. Lastly, in

Model the Data Usage of Each Step

In detail, we first model that the step number in $9$ should be stored per step in the state $Z$.

is stored in $8$ Improved Search for Integral, Impossible Differential and Zero-Correlation Attacks.

Furthermore, the variables $STKC$ in $7$ Improved Search for Integral, Impossible Differential and Zero-Correlation Attacks.

On the subtweakey state, $6$ Improved Search for Integral, Impossible Differential and Zero-Correlation Attacks.

The main part of our model assigns a step number between $5$ Improved Search for Integral, Impossible Differential and Zero-Correlation Attacks.

of the tweakeys, we can recover this part of the tweakey. Furthermore, in our integral attacks on $SKINNY$-$n$-$z$·$n$, $z$ cells of the tweakey are known and under the attacker’s control. Thus, only the other positions ($15$·$z$ 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

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 $ZF$.

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 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: $CSP_C$

1 Declare an empty CSP model $M$;
2 $M$.var $\leftarrow \{IXF_{R_s}[i]\} : R_s \leq r \leq R+1, 0 \leq i \leq 15$;
3 $M$.var $\leftarrow \{IXF_{R_s}[i]\} : R_s \leq r \leq R, 0 \leq i \leq 15$;
4 for $i = 0, \ldots, 15$ do
5 $M$.con $\leftarrow IAXL_{R_s}[i] = (\text{if } AXL_{R_s}[i] > 0 \text{ then } 1 \text{ else } 0)$;
6 for $r = R_s, \ldots, R$; $i = 0, \ldots, 15$ do
7 $M$.con $\leftarrow IWF_r[i] = IWF_r[\text{ShiftRows}[i]]$;
8 $M$.con $\leftarrow IWF_{r+4}[i] = MixColumnsForwards(IWF_r, i)$;
9 return $M$;
Algorithm 3: CSP model of properties of the tweakey schedule of SKINNY

Input: CSP1C.var, integer numbers (start round \( R_s \), end round \( R \), tweakey setting \( z \), active tweakey \( AT \))

Output: CSP2W

1. Declare an empty CSP model \( M \);
2. \( M.\text{var} \leftarrow \{\text{STKC}[i] \in \{0,\ldots,(R - R_s)/2 + 1\} : 0 \leq i \leq 15\};\)
3. \( M.\text{var} \leftarrow \{\text{ISKF}[i] \in \{-1,\ldots,15\} : R_s \leq r \leq R, 0 \leq i \leq 7\};\)
4. \( M.\text{var} \leftarrow \{\text{STKIR}[i] \in \{-1,\ldots,15\} : R_s \leq r \leq R, 0 \leq i \leq 15\};\)
5. \( M.\text{con} \leftarrow \text{STKC}[i] = \sum_{r=R_s}^R \sum_{j=0}^7 \text{IXF}_r[j] = 1 \land \text{KF}_r[j] = i;\)
6. for \( r = R_s,\ldots,R; i = 0,\ldots,7 \) do
   7. if \( \text{IXF}_r[i] = 1 \land \text{stkp}_r[i] \neq AT \) then
      8. \( M.\text{con} \leftarrow \text{ISKF}_r[i] = \text{stkp}_r[i];\)
      9. else
      10. \( M.\text{con} \leftarrow \text{ISKF}_r[i] = \text{stkp}_r[i] \lor \text{ISKF}_r[i] = -1;\)
      11. else
      12. \( M.\text{con} \leftarrow \text{STKC}[i] = \text{stkp}_r[i];\)
   13. for \( i = 0,\ldots,15 \) \( \land AT \) do
      14. \( M.\text{con} \leftarrow z \geq \sum_{r=R_s}^R \sum_{j=0}^7 \text{ISKF}_r[j] = i;\)
   15. for \( i = 0,\ldots,15 \) \( \lor AT \) do
      16. if \( \text{STKC}[i] > z \land i \neq AT \) then
      17. for \( r = R_s,\ldots,R \) do
         18. if \( i \in \text{JSKF} \) then
            19. \( M.\text{con} \leftarrow \text{STKIR}_r[i] = \text{KF}_r[j];\)
         20. else
         21. \( M.\text{con} \leftarrow \text{STKIR}_r[i] = -1;\)
         22. else
         23. \( M.\text{con} \leftarrow \text{STKIR}_r,\ldots,R[i] = -1;\)
      24. for \( r = R_s,\ldots,R, i = 0,\ldots,7 \) do
         25. if \( \text{IXF}_r[i] = 1 \land \text{stkp}_r[i] \neq AT \land \text{stkp}_r[i] \neq \text{ISKF}_r[i] \) then
         26. \( M.\text{con} \leftarrow \text{STKIR}_r,\ldots,R[\text{stkp}_r[i]] < \text{XF}_r[i];\)
   27. return \( M;\)

Figure 6: Example of a step assignment to the various states. We highlight how the sub-tweakey addition and the MixColumns operation affect the step number.

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

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

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

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

Output: $\text{CSP}_{\text{M.A}}$

1. Declare an empty CSP model $\mathcal{M}$;
2. $\mathcal{M}.\text{var} \leftarrow \{XF_r[i] \in \{-1, \ldots, S-1\} : R_s \leq r \leq R, 0 \leq i \leq 15\};$
3. $\mathcal{M}.\text{var} \leftarrow \{KF_r[i] \in \{-1, \ldots, S-1\} : R_s \leq r \leq R, 0 \leq i \leq 7\};$
4. $\mathcal{M}.\text{var} \leftarrow \{ZF_r[i] \in \{-1, \ldots, S-1\} : R_s \leq r \leq R, 0 \leq i \leq 15\};$
5. $\mathcal{M}.\text{var} \leftarrow \{WF_r[i] \in \{-1, \ldots, S-1\} : R_s \leq r \leq R, 0 \leq i \leq 15\};$
6. for $i = 0, \ldots, 15$ do
7.   $M.\text{con} \leftarrow XF_{R+1}[i] = 1$ if $n \geq 0$ or $-1;$
8. for $r = R_s, \ldots, R, i = 0, \ldots, 15$ do
9.   if $KF_r[i] = 1$ then
10.      $M.\text{con} \leftarrow WF_r[i] = \text{MixColumnsBackwards}(XF_{r+1}, i);$  
11.      $M.\text{con} \leftarrow WF_r[i] = -1;$
12.      $M.\text{con} \leftarrow ZF_r[S \times 	ext{ShiftRows}[i]] = WF_r[i];$
13.   if $i < 8$ then
14.      $M.\text{con} \leftarrow (ZF_r[i] > ZF_r[i]) \land (XF_r[i] = X[i]);$
15.      $M.\text{con} \leftarrow (ZF_r[i] = -1) \land (XF_r[i] = ZF_r[i]);$
16.   else
17.      $M.\text{con} \leftarrow (ZF_r[i] = -1) \land (XF_r[i] = ZF_r[i]);$
18.      $M.\text{con} \leftarrow XF_r[i] = ZF_r[i];$
19. return $\mathcal{M}$;

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

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

Output: $\text{CSP}_{\text{MEM}}$

1. Declare an empty CSP model $\mathcal{M}$;
2. $\mathcal{M}.\text{var} \leftarrow \{\text{STKPS}[i] \in \{0, \ldots, 8\} : 1 \leq i \leq S-1\};$
3. $\mathcal{M}.\text{var} \leftarrow \{ZC[i] \in \{0, \ldots, 16\} : 0 \leq i \leq S-1\};$
4. $\mathcal{M}.\text{var} \leftarrow \{XC[i] \in \{0, \ldots, 16\} : 0 \leq i \leq S-1\};$
5. $\mathcal{M}.\text{var} \leftarrow \{\text{SBC}[i] \in \{0, \ldots, 16\} : 1 \leq i \leq S-1\};$
6. $\mathcal{M}.\text{var} \leftarrow \{\text{ASTKC}[i] \in \{0, \ldots, (R - R_s)/2\} : 0 \leq i \leq S-1\};$
7. for $i = 1, \ldots, S - 1$ do
8.   $M.\text{con} \leftarrow \text{STKPS}[i] = \sum_{r=R_s}^{R} \sum_{j=0}^{7} KF_r[j] = i;$
9.   $M.\text{con} \leftarrow \text{SBC}[i] = \sum_{r=R_s}^{R} \sum_{j=0}^{15} XF_r[j] = i;$
10. for $i = 0, \ldots, S - 1$ do
11.   $M.\text{con} \leftarrow ZC[i] = \sum_{r=R_s}^{R} \sum_{j=0}^{15} ZF_r[j] = i \land KF_r[j];$
12.   $M.\text{con} \leftarrow XC[i] = \sum_{r=R_s}^{R} \sum_{j=0}^{15} XF_{r+1}[j] = i \land \text{MCBigger}(WF_r, i, j);$
13.   $M.\text{con} \leftarrow \text{ASTKC}[i] = \sum_{r=R_s}^{R} \sum_{j=0}^{7} \text{STKPS}[j] = \text{AT} \land XF_r[j] = i;$
14. return $\mathcal{M}$;

ensure that at least one tweakey 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.
We list additional results for An overview of the improved integral attacks on SKINNY. The inactive input cells are constant; the other tweakey cells form the values in the active input cells and the tweakey cell with index value. Consequently, the sum of the active output cells forms a balanced Boolean function candidate for the first branch with each candidate for the second branch that produces 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.

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 D and ForkSKINNY in Appendix F.

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 tweakey cells 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 \( 2^{4(16-4+1)} = 2^{52} \) 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.

---

**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 \( \CM; \)
3. \( \CM \leftarrow \CSPIC \land \CSPFW \land \CSPSA \land \CSPMEM; \)
4. \( \CM.var \leftarrow \{ \exp[i] \mid 0, \ldots, z \cdot 16 \cdot c \}; 1 \leq i \leq S - 1; \)
5. \( \CM.var \leftarrow \{ \fi[i] \mid 0, \ldots, 2^{n-16} \cdot c \}; 1 \leq i \leq S - 1; \)
6. \( \CM.var \leftarrow \T \text{total} \in [0, 2^{n-16}]; \)
7. for \( i = 1, \ldots, S - 2 \) do
8. if \( \STKPS[i+1] \neq 0 \) then
9. \( \M.con \leftarrow \STKPS[i] > 0; \)
10. for \( i = 1, \ldots, S - 1 \) do
11. if \( \STKPS[i] > 0 \) then
12. if \( \DI > (\ASTKC[i-1] + \ZC[i-1] + X[i-1]) \) then
13. \( \M.con \leftarrow \T \exp[i] = c \cdot (\ASTKC[i-1] + \ZC[i-1] + X[i-1] + \sum_{j=1}^{i} \STKPS[j]); \)
14. else
15. \( \M.con \leftarrow \T \exp[i] = 0; \)
16. \( \M.con \leftarrow \T[i] = 2^\si[i]; \)
17. \( \M.con \leftarrow \T \text{total} = \sum_{i=1}^{S-1} \T[i] \cdot \SBC[i]; \)
18. \( \M.obj \leftarrow \text{Minimize} \T \text{total}; \)
19. return \( \M; \)
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 recovery of $X_{16}[0]$ 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 $X_{16}[0]$, 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 subtweakey $STK_r$, 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 7), allowing us to derive $X_{16}[14]$ with 2 S-box computations.

![Figure 7: ZC-based integral attack on 18 rounds of SKINNY-$n$-m.](image-url)
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 subtweakey 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 (or 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 $2^{47} = 2^{28}$ 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 $2^{60-4} = 2^{60}$ 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 $2^{60-4} = 2^{46}$). As merging can be done efficiently, the total complexity is less than $2^{49}$ encryptions equivalents plus $3 \cdot 2^{52} = 2^{53.38}$ 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 $3 \cdot 2^{104-7.2} + 2^{120-3.8} \approx 2^{99}$ plus $3 \cdot 2^{104} = 2^{105.58}$ data for 18-round SKINNY-128-128 with 120-bit keys.

![Complexity of partial-sum key-recovery of $X_{13}[0]$ for 18 rounds of SKINNY-$n$-n.](image_url)

**Discussion** Our model only uses integer variables, enabling us to use any CP solvers that support 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
This paper improved the automated search for ID, ZC, and integral attacks. In three works [HSE23], we found better partial-sum optimizations.

Interesting future work is extending the idea of our distinguisher modeling, i.e., CP/MILP ciphers following different design strategies, and got improved results. Our results show a technique for the first time. We implemented our approach, applied it to several block 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 tweak key 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 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). We sincerely thank Xavier Bonnetain and Virginie
Lallemand for detecting a flaw in the key-recovery technique for related-tweakey ID attacks
in [SMB18] that also affects our related-tweakey ID attacks and more importantly, for
providing a new method to rectify it. Their solution fixes all of them, mostly with the
same time complexity. We would also 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

[ADG+19] Ralph Ankele, Christoph Dobrmuig, Jian Guo, Eran Lambooij, Gregor Leander,
and Yosuke Todo. Zero-correlation attacks on tweakable block ciphers
with linear tweakey expansion. IACR Transactions on Symmetric Cryptology,

[ALP+19] Elena Andreeva, Virginie Lallemand, Antooin Purnal, Reza Reyhanitabar,
Arnab Roy, and Damian Vizár. Forkcipher: A new primitive for authenticated
encryption of very short messages. In Steven D. Galbraith and Shiho Moriai,
editors, ASIACRYPT 2019, volume 11922 of LNCS, pages 153–182. Springer,

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced
to 31 rounds using impossible differentials. In EUROCRYPT 1999, volume

[BDL20] Augustin Bariant, Nicolas David, and Gaëtan Leurent. Cryptanalysis of

[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

[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.

impossible differential attacks. Personal communication, February 2024.

[BLNPS18] Christina Boura, Virginie Lallemand, María Naya-Plasencia, and Valentin


A Bonnetain and Lallemand’s Note on Related-Tweakey Impossible Differential Attacks

In our related-tweakey ID attacks, we reuse the same technique as [SMB18] to take advantage of multiple ID distinguishers with the same input/output activeness pattern (truncated ID distinguishers). As before, assume that $c_b$ denotes the number of bit conditions in $E_b$. Additionally, let $|\Delta_t|$ represent the number of (differentially) active bits in the tweakey. According to [SMB18], and also as we discuss in Subsection H.1, we assume that $|\Delta_t|$ bits of the exterior filters in $E_b$ can be passed for free. Therefore, to consider the impact of multiple distinguishers in our CP model on the time complexity formula, i.e., $T_0$ in Equation 4 and $T_1$ in Equation 2) we use $c_b - |\Delta_t|$ in our objective function terms. That is why the caption of our ID attacks in the related-tweakey setting includes two values for the number of bit conditions in $E_b$: $c'_b$, which denotes the number of bit conditions in $E_b$, and $c_b$, which represents the value of $c'_b - |\Delta_t|$ that we use in our CP model for related-tweakey settings to calculate time complexity terms.

However, on February 19, 2024, Xavier Bonnetain and Virginie Lallemand [BL24] informed us that the key recovery procedure for the related-tweakey ID attacks in [SMB18] has a logical flaw. Following the same technique as [SMB18], we assumed the attacker could access two related-tweakey oracles in our related-tweakey setting. While this assumption aligns with using a single distinguisher, according to Bonnetain and Lallemand [BL24], it presents a conflict when attempting to bypass exterior filters without cost when using multiple distinguishers. Bonnetain and Lallemand suggested using more than two related-tweakey oracles to fix this issue. Using this new setting, they could still take advantage of multiple distinguishers. Also, they showed that this change mostly does not change the time complexity of our related-tweakey ID attacks, whereas it can change the memory complexity. Consequently, our attacks remain valid in terms of time complexity, but the memory complexity of our attacks should be re-evaluated [BL24]. To incorporate the updated setting by Bonnetain and Lallemand, we need to modify our CP model slightly: instead of reducing $c_b$ by $|\Delta_t|$, we should increase $c_b$ by $\Delta_t$. Interestingly, this change has the same effect on terms $T_0$ in Equation 4 and only changes the term $T_1$ in Equation 2. However, in almost all of our discoveries, $T_1$ was not the most dominant term in the time complexity. As a result, our reported attacks remain optimal in terms of time complexity.

B 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 tweakey sizes, $t \in \{n, 2n, 3n\}$. SKINNY-$n$-$t$ represents SKINNY with $n$-bit block size and $t$-bit tweakey size. The internal state of SKINNY is a $4 \times 4$ array of cells arranged in a row-major order. The tweakey state is composed of $z \times 4 \times 4$ array of cells, where $z = \frac{t}{n} \in \{1, 2, 3\}$. We use $TK1$, $TK2$, and $TK3$ to denote the tweakey arrays. The cell size is 4 (or 8) bits when $n = 64$ (resp. $n = 128$).

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 subtweakey are XORed to the corresponding cells in the internals state. SR applies a permutation $P$
on the position of the state cells, where \( P = [0, 1, 2, 3, 4, 5, 6, 7, \ldots, 10, 11, 12, 13, 14, 15, 16] \).
MC multiplies each column of the internal state by a non-MDS matrix \( M \). \( M \) and its inverse are as follows:

\[
M = \begin{pmatrix}
1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
\end{pmatrix},
M^{-1} = \begin{pmatrix}
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
\end{pmatrix}.
\]

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 \( \Delta X_r \) and the linear mask as \( \Gamma X_r \). 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 subtweakey after \( r \) rounds, and \( ETK_r \), referred to as the equivalent subtweakey in round \( r \), is calculated as \( ETK_r = MC \odot SR(STK_r) \). Figure 11 shows the relation between \( STK_r \) and \( ETK_r \).

The tweakey schedule of SKINNY splits the master tweakey into \( z \) tweakey arrays \( (TK_1, \ldots, TK_z) \), each consisting of \( n \) bits. Here, \( z \) can be one of three values: 1, 2, or 3. After this division, each tweakey array follows its own separate schedule. The subtweakey for the \( r \)th round is generated as follows:

\[
\begin{align*}
STK_r &= TK1_r & \text{if } t = n \\
STK_r &= TK1_r \oplus TK2_r & \text{if } t = 2n \\
STK_r &= TK1_r \oplus TK2_r \oplus TK3_r & \text{if } t = 3n,
\end{align*}
\] (8)

where \( TK1_r, TK2_r, \) and \( TK3_r \) represent the tweakey arrays in round \( r \), and they’re generated as follows: First, we use a permutation \( h \) on each tweakey 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 \( TK2 \) and \( TK3 \). For more in-depth information about SKINNY, you can refer to [BJK+16].

## C  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 \( X,Y \in \mathbb{F}_2^{24} \), and \( M \) represents the matrix of SKINNY. Additionally, we use a compact representation for the constraint encoding the branching operation \( \text{Copy}(Y[k], X[i], X[j]) \) as \( \text{Branch}(AX[i], DX[i], AX[j], DX[j], AY[k], DY[k]) \).
The following CP constraints describe valid transitions for deterministic truncated linear trails through $M$:

$$M_{\text{diff}}(AX, DX, AY, DY) := \begin{cases} 
AY[1] = AX[0] \land DY[1] = DX[0] \land \\
\text{XOR}(AX[1], DX[1], AX[2], DX[2], AY[2], DY[2]) \land \\
\text{XOR}(AX[0], DX[0], AX[2], DX[2], AY[3], DY[3]) \land \\
\text{XOR}(AY[3], DY[3], AX[3], DX[3], AY[0], DY[0])
\end{cases}$$

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

$$\text{Minvdiff}(AX, DX, AY, DY) := \begin{cases} 
AY[0] = AX[1] \land DY[0] = DX[1] \land \\
\text{XOR}(AX[1], DX[1], AX[3], DX[3], AY[2], DY[2]) \land \\
\text{XOR}(AX[0], DX[0], AX[3], DX[3], AY[3], DY[3]) \land \\
\text{XOR}(AY[2], DY[2], AX[2], DX[2], AY[1], DY[1])
\end{cases}$$
D Integral Attacks on SKINNY

D.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 $4 \cdot 2^{2(16–4+2)} = 2^{58}$ and a brute-force complexity of $2^{120–4} \cdot 2^{104}$. 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 data complexity of $4 \cdot 2^{2(16–4+2)} = 2^{114}$ and a brute-force complexity of $2^{240–8} = 2^{208}$. Using partial sums, the memory complexity is $2^{208}$ and the summation time complexity $4 \cdot (2^{210.08} + 2^{202.96}) \approx 2^{112.09}$ encryption equivalents plus merging, less than $2^{213}$ in total.

![Figure 12: ZC-based integral attack on 22 rounds of SKINNY-n-2n.](image-url)
Figure 13: Complexity of partial-sum key-recovery of $X_{15}[3]$ for 22 rounds of SKINNY-\textit{n}-2\textit{n}.
Improved Search for Integral, Impossible Differential and Zero-Correlation Attacks

Figure 14: Complexity of partial-sum key-recovery of \(X_{15}[15]\) for 22-round SKINNY-\(n\)-2n.
D.2 Integral Key-Recovery Attack on 26-Round SKINNY-\(n\)-3\(n\)

For SKINNY-64-192 with 180-bit keys, we repeat the attack 4 times for a total data complexity of \(4 \cdot 2^{8 \cdot (16-4+3)} = 2^{62}\) 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 \(4 \cdot 2^{8 \cdot (16-4+3)} = 2^{122}\) 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-\(n\)-3\(n\).
Figure 16: Complexity of partial-sum key-recovery of $X_{17}[0]$ for 26 rounds of SKINNY-$n$-$3n$. 

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed</th>
<th>$K \times D$</th>
<th>Mem Time</th>
<th>Stored Texts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STK_{23} [1, 2, 4, 5, 6]; STK_{24} [3, 6]; STK_{25} [1, 2, 4, 6, 7]; STK_{26} [0, 1, 2, 4, 6, 7]; $2^{116} \times 2^{160} = 2^{256}$</td>
<td>$2^{60} - 4.9$</td>
<td>$Z_{23}[0-6]; Z_{25}[8, 9, 10, 12-15]; Z_{24}[3, 6]; X_{24}[9]; STK_{23}[3] Z_{25}[0, 3]; Z_{26}[12, 15]; Z_{24}[0, 1, 2, 4, 5, 7]; X_{24}[8, 9, 11, 14, 15]; Z_{23}[2, 5]; X_{23}[8, 12]; STK_{24}[5]$</td>
<td>$2^{152} - 4.6$</td>
</tr>
</tbody>
</table>
Figure 17: Complexity of partial-sum key-recovery of $X_{17}[12]$ for 26-round SKINNY-$n$-3n.
E 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 $r_i$. To represent the number of rounds in the $C_0$ and $C_1$ branches (see Figure 18), we use $r_0$ and $r_1$, respectively. Depending on the underlying SKINNY variant and the parameters $(r_i, r_0, r_1)$, 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 $r_i - x$ rounds before the fork, and $r_0 - x = r_1 - x$ 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].](image)

Table 4: Different variants of ForkSKINNY.

<table>
<thead>
<tr>
<th>Primitive</th>
<th>block</th>
<th>tweak</th>
<th>tweakey</th>
<th>$r_i$</th>
<th>$r_0$</th>
<th>$r_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ForkSKINNY-64-192</td>
<td>64</td>
<td>64</td>
<td>192</td>
<td>17</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>ForkSKINNY-128-192</td>
<td>128</td>
<td>64</td>
<td>192</td>
<td>21</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>ForkSKINNY-128-256</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>21</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td><strong>ForkSKINNY-128-288</strong></td>
<td>128</td>
<td>128</td>
<td>288</td>
<td>25</td>
<td>31</td>
<td>31</td>
</tr>
</tbody>
</table>
F Integral Attacks on ForkSKINNY

F.1 Integral Key-Recovery Attack on 23-Round ForkSKINNY-128-256 with \( r_1 = 9 \) and \( r_0 = 27 \)

We follow the same approach as for SKINNY above. For our attacks, we assume that the tweakey is similarly flexible as in SKINNY, i.e., we choose each tweakey 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 \( 4 \cdot 2^8 \cdot (16-16+4+4) = 2^{114} \) and a brute-force complexity of \( 2^{240-8-4} = 2^{208} \). 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.](image-url)
Improved Search for Integral, Impossible Differential and Zero-Correlation Attacks

Figure 20: Complexity of partial-sum key-recovery of $X_{16}[1]$ for 23 rounds of ForkSKINNY-128-256.
Figure 21: Complexity of partial-sum key-recovery of $X_{10}[13]$ for 23 rounds of ForkSkinny-128-256.
F.2 Integral Key-Recovery Attack on 27-Round ForkSKINNY-64-192 with \( r_1 = 9 \) and \( r_0 = 23 \)

For ForkSKINNY-64-192 with 180-bit keys, we repeat the attack 4 times for a total data complexity of \( 4 \cdot 2^{16} \cdot 4 \cdot 2^2 = 2^{58} \) 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.
Figure 23: Complexity of partial-sum key-recovery of $X_{18}[3]$ for 27 rounds of ForkSkinny-64-192.
Figure 24: Complexity of partial-sum key-recovery of $X_{15}[15]$ for 27 rounds of ForkSKINNY-64-192.
G 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 (fixed), (nonzero), and (any) in forward and backward propagations, respectively. In addition, we represent the active tweak cell in our integral distinguishers by (active tweak).

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-tweak 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 $r_1 = 6$, $r_0 = 27$, and $r_1 = 9$.](image-url)
Figure 26: Integral and ZC distinguishers for 17 rounds of ForkSKINNY-64-192, where $r_1 = 6$, $r_0 = 23$, 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 $r_1 = 8$. 
fixed  nonzero  any

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 $r_1 = 14$. 
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 (namely a lower bound). 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.

H.1 28-round Related-Tweakey ID Attack on ForkSKINNY-64-192-192

$(r_1 = 11, r_0 = 17)$

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 $r_1 = 17$ – X rounds before the fork, and $r_0 = r_1 = 23 – X$ 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 $\Delta Y_4 = 00\delta_00000000000000000000000$, and the output difference $\Delta Z_{23} = 000000000\delta_00000000000000000000000$, where $\delta_0$ and $\delta_0'$ can take any nonzero differences ($1 \leq \delta_0, \delta_0' \leq 15$) and $\delta_i$ (resp. $\delta_i'$) is determined after a linear transformation implemented for $i$ times with the LFSRs (in $TK_2$, and $TK_3$) on $\delta_0$ (resp. $\delta_0'$).

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 $2^t$ structures $S_t, 0 \leq t \leq 2^t - 1$ 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 $(C_u = E_K(P_u), C'_u = E_K \oplus \Delta (P_u))$, where

$$\Delta = \Delta ETK_0 = 000\Delta_1000\Delta_100\Delta_20000\Delta_1, \quad \Delta_1, \Delta_2 \neq 0.$$ 

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

Guess-and-Filter

Step 1. For each guess of $ETK_0[0–3, 8–11] = K_0$, encrypt all N pairs. For each guess, create a set $L_1$ containing pairs that satisfy both $\Delta Y_1$ and $Y_1$, as illustrated in Figure 29. Due to the $MC^{-1}$ operation on active cells in $X_2$, we have $\Delta Y_1[4] = \Delta Y_1[11], \Delta Y_1[2] \oplus \Delta Y_1[8] = \Delta Y_1[15], \Delta Y_1[3] \oplus \Delta Y_1[6] = \Delta Y_1[12]$, and $\Delta Y_1[6] = \Delta Y_1[9]$. Consequently, a 20-bit filter is established. On average, the size of a single $L_1$ set is approximately $N2^{-20}$. The time complexity of this step is approximately $N2^{32}$.

Step 2. For each guess of $K_0$, and $STK_1[0–7] = K_1$, encrypt all pairs within the corresponding $L_1$ set associated with $K_0$. Additionally, construct a subset $L_2 \subseteq L_1$ that contains all the pairs satisfying $\Delta Y_2$ and $Y_2$ as shown in Figure 29. Due to

Suppose 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 $\delta_i = d_1 \oplus LFSR_{i_d}(d_2) \oplus LFSR_{i_d}(d_3)$.
In this case we have
\[ r = \frac{1}{12} \times 15 \times 2 \times 15 \times 2 \times 15 \times 2 = 56 \]
MC$^{-1}$ operation on the active cells in the $X_3$, we will have a 16-bit filter on $\Delta Y_2$. On average, the size of a single $L_2$ 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 $K_0$, $K_1$, and $STK_0[0-2,6]$, encrypt all pairs within the corresponding $L_2$ set associated with $K_0||K_1$. Additionally, construct a subset $L_3 \subseteq L_2$ that contains all the pairs satisfying $\Delta W_3$ 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 $\Delta W_3$. Therefore, we can determine $\Delta Y_4[1,5]$. On average, the size of a single $L_3$ 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 $\Delta Y_4[1] = STK_4[1]$, and $\Delta Y_4[5] = STK_4[5]$ 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 $15 \times 15$ distinct nonzero values for the pair $(STK_4[1], STK_4[5]) = (\delta_0', \delta_0'').$ To handle this, drawing inspiration from [SMB18], we utilize a hash table with 225 rows, labeled as $h(\delta_0, \delta_0')$, where $\delta_0$ ranges from 1 to 15 and $\delta_0'$ ranges from 1 to 15. This hash table facilitates the partitioning of the remaining pairs within the $L_3$ 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 $(\delta_0, \delta_0')$ 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 $K_0$, $K_1$, $K_2,3$, and $STK_2[0,1,3,6,7] = K_{27}$ decrypt all pairs within the corresponding $L_3$ set associated with $K_0||K_1||K_2,3$. Additionally, construct a subset $L_4 \subseteq L_3$ that contains all the pairs satisfying $\Delta Z_{27}$ and $Z_{27}$ as shown in Figure 29. There is no requirement for any tweakey 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 $STK_2[4]$ (due to $STK_2[4] = Z_{27}[4] \oplus Y_{27}[7]$). 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 $STK_2[2,5]$. We compute $Z_{26}$ and $\Delta Z_{26}$ 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 $STK_2[6]$. 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 $STK_2[2,6]$. On average, the size of a single $L_4$ 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 $K_0$, $K_1$, $K_2,3$, $K_{27}$, and $STK_2[5] = K_{26}$ decrypt all pairs within the corresponding $L_4$ set associated with $K_0||K_1||K_2,3||K_{27}$. Additionally, construct a subset $L_5 \subseteq L_4$ that contains all the pairs satisfying $\Delta W_{25}[5]$ as shown in Figure 29. For each pair in the row $(\delta_0', \delta_0'')$ of hash table $h$ of set $L_5$, 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 $L_5$ set is approximately $N2^{-48-4} = N2^{-52}$. The time complexity of this step is approximately $N2^{-48+32+32+20+20} = N2^{60}$.

**Step 6.** For each guess of $K_0$, $K_1$, $K_2,3$, $K_{27}$, $K_{26}$, and $STK_2[1,4,7] = K'_{26}$ decrypt all pairs within the corresponding $L_5$ set associated with $K_0||K_1||K_2,3||K_{27}||K_{26}$. 

**Step 7.** For each guess of $K_0$, $K_1$, $K_2,3$, $K_{27}$, $K_{26}$, and $STK_2[5] = K_{26}$ decrypt all pairs within the corresponding $L_5$ set associated with $K_0||K_1||K_2,3||K_{27}||K_{26}$.
Additionally, construct a subset $L_6 \subseteq L_5$ that contains all the pairs satisfying $Z_{25}$ and $\Delta Z_{25}$ 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 $STK_{25}[0]$ 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 $L_6$ set is approximately $N_2^{-52-4} = N_2^{-56}$. The time complexity of this step is approximately $N_2^{-52+32+32+20+20+4+12} = N_2^{68}$.

**Step 7.** For each guess of $K_0$, $K_1$, $K_{2,3}$, $K_{27}$, $K_{26}$, and $K'_{26}$, and $STK_{25}[7] = K_{25}$ decrypt all pairs within the corresponding $L_6$ set associated with $K_0 || |K_1| || |K_{2,3}| || |K_{27}| || |K_{26}| || |K'_{26}|$.

Additionally, construct a subset $L_7 \subseteq L_6$ that contains all the pairs satisfying $Z_{24}$ and $\Delta Z_{24}$ 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 tweakey 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 $Z_{24}$ and $\Delta Z_{24}$ as shown in Figure 29. For each pair in the row $(\delta_0, \delta_0')$ of hash table $h$ of $L_7$ 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 $L_6$ set is approximately $N_2^{-56-8} = N_2^{-64}$. The time complexity of this step is approximately $N_2^{-56+32+32+20+20+4+12+4} = N_2^{68}$.

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:

$$\Delta Y_4 = \delta_0 \cdot 000000000 \cdots 0, \Delta Z_{23} = 000\delta_18000000\delta_18'0 \cdots 0.$$ 

After the above steps, each of the rows in each of $L_7$ sets which is nonempty suggests the wrong keys. The probability of a given pair surviving the filtering step satisfying $\Delta Y_4$, and $\Delta Z_{23}$ under a random key guess is about $(1 - 2^{-2^{100}})^{2^{x+52}} = e^{-2^{-x-4}}$. Therefore, the total number of remaining tweakeys is $2^{x \cdot |L_7|} \times e^{-2^{-x-4}} = 2^{160} \times e^{-2^{-x-4}}$. 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:

$$2^{x+53} + \frac{1}{25} \times (N2^{32} + N2^{44} + N2^{68} \times 2 + N2^{60} \times 2 + N2^{68} \times 2) + 2^{192} \times e^{-2^{-x-4}}$$

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 $2^{194}$, respectively.

We can also find the specification 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 procedure in their respective figures and tables.
Table 5: Key-recovery procedure for ID attack on 28 rounds of ForkSKINNY-64-192-128, where $r_i = 17 - X$, $r_0 = r_1 = 23 - X$ for $X = 6$.

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed keys (#)</th>
<th>Condition(s) (# filters)</th>
<th># Remaining pairs(#L_1)</th>
<th>Time</th>
<th>Deduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>$ETK_0[0-3, 8-11] = K_0$ {32}</td>
<td>$\Delta Y_1[1 = 4 = 11]$ $\Delta Y_1[2 = 8 = 15]$</td>
<td>$N_2^{20}$</td>
<td>$N_2^{32}$</td>
<td>$\Delta Y_1, Y_1$</td>
</tr>
<tr>
<td>i=2</td>
<td>$STK_i[0-7] = K_i$ {32}</td>
<td>$\Delta Y_2[0 = 7 = 10]$ $\Delta Y_2[5 = 8 = 15]$</td>
<td>$N_2^{36}$</td>
<td>$N_2^{44}$</td>
<td>$\Delta Y_2, Y_2$</td>
</tr>
<tr>
<td>i=3</td>
<td>$K_0, K_1, K_2, K_3$ $STK_{27}[1, 3, 6, 7] = K_{27}$ {20}</td>
<td>$\Delta W_3[1 = 5 = 9]$</td>
<td>$N_2^{44}$</td>
<td>$N_2^{48}$</td>
<td>$\Delta W_3, W_3$</td>
</tr>
<tr>
<td>i=4</td>
<td>$K_0, K_1, K_2, K_3, K_{27}$ $STK_{26}[0, 5] = K_{26}$ (4)</td>
<td>$\Delta W_5[i = 5]$</td>
<td>$N_2^{44}$</td>
<td>$N_2^{60}$</td>
<td>$STK_{27}[4]$ $STK_{27}[5]$</td>
</tr>
<tr>
<td>i=5</td>
<td>$K_0, K_1, K_2, K_3, K_{27}$ $STK_{26}[5] = K_{26}$ (4)</td>
<td>$\Delta W_5[5] = \delta_{19}$</td>
<td>$N_2^{52}$</td>
<td>$N_2^{60}$</td>
<td></td>
</tr>
<tr>
<td>i=6</td>
<td>$K_0, K_1, K_2, K_3, K_{27}, K_{26}$ $STK_{26}[1, 4, 7] = K_{26}$ (ETK) {8, 9}</td>
<td>$(STK_{27}[1, 6])$ $(STK_{27}[0, 5])$</td>
<td>$N_2^{36}$</td>
<td>$N_2^{68}$</td>
<td>$Z_{25}, \Delta Z_{25}$</td>
</tr>
<tr>
<td>i=7</td>
<td>$K_0, K_1, K_2, K_3, K_{27}, K_{26}$, $STK_{25}[7] = K_{25}$ (4)</td>
<td>$\Delta X_{25}[6 \oplus 10 = 14]$</td>
<td>$N_2^{64}$</td>
<td>$N_2^{68}$</td>
<td>$(STK_{25}[1, 6])$</td>
</tr>
</tbody>
</table>

**Complexity analysis**

<table>
<thead>
<tr>
<th>z</th>
<th>Data</th>
<th>Memory</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$2^{60}$</td>
<td>$2^{104}$</td>
<td>$2^{61} + N_2^{64.19} + 2^{192} e^{-2^{r-4}} = 2^{168.60}$</td>
</tr>
</tbody>
</table>

**H.2 28-round Related-Tweakey ID Attack on ForkSKINNY-64-192-128**

$(r_i = 11, r_0 = 17)$

The 28-round related-tweakey ID Attack on ForkSKINNY-64-192-128 occurs in a limited setting, where there are $r_i = 17 - X$ rounds before the fork, and $r_0 = r_1 = 23 - X$ rounds in each branch after the fork, with $X = 6$. We give the pattern of the attack in Figure 30.

Since the attack in this section has the same structure as in the 30-round related-tweakey ID Attack on ForkSKINNY-64-192-128 (Subsection H.4, Figure 32), we can reuse the key recovery part of that attack. Consequently, we proceed to target a modified version of ForkSKINNY-64-192-128 with 28 rounds in a constrained scenario, with identical computational complexity as the attack of Subsection H.4. Therefore, the attack’s data, memory, and time complexities are $2^{62}$, $2^{86}$, and $2^{123.73}$, respectively.
Figure 30: Related-Tweakey ID attack on 28-round ForSKINNY-64-192-128 in the limited setting (r1 = 11, r0 = 117). |k0 ∪ k2| = 112, c0 = 24 (c0 = 20), c1 = 64, Δ0 = 24, Δy = 64.
H.3 32-round Related-Tweakey ID Attack on ForkSKINNY-64-192-192
\((r_1 = 11, r_0 = 15)\)

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.

Table 6: Key-recovery procedure for ID attack on 32 rounds of ForkSKINNY-64-192-192, where \(r_1 = 11, r_0 = 15\), and \(r_1 = 21\).

<table>
<thead>
<tr>
<th>Step</th>
<th># Plaintexts</th>
<th># Pairs</th>
<th>Pair Generation</th>
<th>N</th>
<th># Enc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>(2^{t+52})</td>
<td>(2^{t+104})</td>
<td>(2^{t+53})</td>
<td>(2^{t+104})</td>
<td></td>
</tr>
<tr>
<td>i=2</td>
<td>(K_{31})</td>
<td>(STK_{30}[0..7] = K_{30})</td>
<td>(\Delta X_{31} [r = 8 = 12])</td>
<td>(N_{2^{16}})</td>
<td>(N_{2^{22}})</td>
</tr>
<tr>
<td>i=3</td>
<td>(K_{31})</td>
<td>(STK_{30}[0..7] = K_{30})</td>
<td>(\Delta X_{31} [r = 9 = 13])</td>
<td>(N_{2^{32}})</td>
<td>(N_{2^{46}})</td>
</tr>
<tr>
<td>i=4</td>
<td>(K_{31}, K_{30})</td>
<td>(STK_{29}[1..7] = K_{28})</td>
<td>(\Delta X_{28} [r = 6 = 14])</td>
<td>(N_{2^{44}})</td>
<td>(N_{2^{60}})</td>
</tr>
<tr>
<td>i=5</td>
<td>(K_{31}, K_{30}, K_{29})</td>
<td>(STK_{28}[1..7] = K_{28})</td>
<td>(\Delta X_{27} [r = 6 = 14])</td>
<td>(N_{2^{44}})</td>
<td>(N_{2^{60}})</td>
</tr>
<tr>
<td>i=6</td>
<td>(K_{31}, K_{30}, K_{29}, K_{28})</td>
<td>(STK_{27}[0..2] = K_{27})</td>
<td>(\Delta X_{26} [r = 13 = 7])</td>
<td>(N_{2^{48}})</td>
<td>(N_{2^{68}})</td>
</tr>
<tr>
<td>i=7</td>
<td>(K_{31}, K_{30}, K_{29}, K_{28})</td>
<td>(STK_{26}[0..3] = K_{26})</td>
<td>(\Delta X_{25} [r = 11 = 12])</td>
<td>(N_{2^{56}})</td>
<td>(N_{2^{72}})</td>
</tr>
<tr>
<td>i=8</td>
<td>(K_{31}, K_{30}, K_{29}, K_{28})</td>
<td>(STK_{25}[1..7] = K_{25})</td>
<td>(\Delta X_{24} [r = 6 = 12])</td>
<td>(N_{2^{64}})</td>
<td>(N_{2^{86}})</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total -</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(N_{2^{57.17}})</td>
<td>32-round Enc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>Data</th>
<th>Memory</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(2^{62})</td>
<td>(2^{414})</td>
<td>(2^{436.27})</td>
</tr>
</tbody>
</table>
Figure 31: Related-Tweak ID attack on 32 rounds of ForkSKINNY-64-192-192 in the arbitrary setting \( r_1 = 11, r_2 = 15 \). \(|k_0 \cup k_1| = 176, c'_0 = 52 (c_0 = 48), c_0 = 64, \Delta_0 = 52, \Delta_0 = 64 \). In this case we have 15 related-key impossible differences.
H.4 30-round Related-Tweakey ID Attack on ForkSKINNY-64-192-128 (\(r_1 = 9, r_0 = 15\))

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.

![Diagram](image)

Figure 32: Related-Tweakey ID attack on 30-round ForkSKINNY-64-192-128 in the arbitrary setting \((r_1 = 9, r_0 = 15)\). \(|k_n \cup k_r| = 112, c'_n = 24 (c_n = 20), c_f = 64, \Delta_n = 24, \Delta_f = 64\).
Table 7: Key-recovery procedure for ID attack on 30 rounds of ForkSKINNY-64-192-128, where \( r_1 = 9 \), \( r_0 = 15 \), and \( r_1 = 21 \).

| Step | Guessed keys (#) | Condition(s) (# filters) | # Remaining pairs(\(|\mathcal{L}|\)) | Time Deduced |
|------|-----------------|--------------------------|-----------------------------------|--------------|
| \( i = 1 \) | \( ETK_0[2] || STK_{20}[0-7] = K_{0,20} \) | \( \Delta Y_r[2] = \delta_{-1}(\Delta STK_0[2]) \) | \( N2^{-16} \) | \( N2^{30} \) |
| \( i = 2 \) | \( K_{0,20} \) | \( \Delta X_{27}[0] = 4 = 12 \) | \( Z_{27}, \Delta Z_{27} \) | \( STK_{27}[2] \) |
| \( i = 3 \) | \( STK_{27}[5, 6] = K_{27} \) | \( \Delta X_{16}[0] = 8 \) | \( Z_{26}, \Delta Z_{26} \) | \( STK_{26}[0] \) |
| \( i = 4 \) | \( K_{0,20}, K_{28}, STK_{27}[4] = K_{26} \) | \( \Delta Z_{24}[5] = \delta_{15}(\Delta STK_{24}[5]) \) | \( Z_{24}, \Delta Z_{24} \) | \( STK_{24}[5] \) |
| \( i = 5 \) | \( K_{0,20}, K_{28}, K_{27}, ETK_0[9] = K_0 \) | \( \Delta Y_1[3] = 6 = 9 \) | \( \Delta Y_1, Y_1 \) | \( STK_{5}[5] \) |
| Total | - | - | \( N2^{37.11} \) | \( 30\text{-round} \text{ Enc.} \) |

**Complexity analysis**

| \( z \) | Data Memory Total time |
|------|----------------------|---------------------|
| 38   | \( 2^{62} \)         | \( 2^{65} \)        | \( 2^{63} + N2^{37.11} + 2^{128} e^{-2z-36} = 2^{123.73} \) |
H.5 26-round Related-Tweakey ID Attack on ForkSKINNY-128-256-256 ($r_1 = 9$, $r_0 = 17$)

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 ForkSKINNY-128-256-256 in the arbitrary setting ($r_1 = 9$, $r_0 = 17$). $|k_0 \cup k_f| = 216$, $c_f = 104$ ($c_0 = 96$), $e_f = 72$, $\Delta_k = 104$, $\Delta_f = 72$. 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 \( r_1 = 9, r_0 = 17, \) and \( r_1 = 17. \)

| Step | Guessed keys {#} | Condition(s) | # Remaining pairs(\(|L_i|\)) | Time | Deduced |
|------|------------------|--------------|-------------------------------|------|---------|
| 1    | \( STK_{25}[1, 4, 7] = K_{25} \) | \( \Delta X_{25}[5] = \delta_{19}(\Delta STK_{24}[1]) \) \((15 \times 2^{-6}\) filters\) | \{4\} | \( N^2 \rightarrow 20 \) | \( N^2 \rightarrow 24 \) |
|      |                  | \( \Delta X_{25}[8 = 12] \) | \( 8 \) | \( \Delta X_{25}[0 = 12] \) |
|      |                  | \( \Delta X_{25}[5 = 13] \) | \( 8 \) | \( \Delta X_{25}[2 = 6 = 14] \) |
|      |                  | \( \Delta X_{24}[10 = 14] \) | \( 8 \) | \( \Delta X_{24}[2 = 10] \) |
| 2    | \( STK_{24}[6] = K_{24} \) | \( \Delta Z_{22}[7] = \delta_{18}(\Delta STK_{22}[7]) \) | \{8\} | \( N^2 \rightarrow 28 \) | \( N^2 \rightarrow 12 \) |
| 3    | \( STK_{25}, K_{24} \) | \( \Delta Y_{3}[0 = 7 = 10] \) | \( 8 \) | \( \Delta Y_{3}[1 = 11] \) |
|      | \( ETK_{0}[3, 9] \) | \( \Delta Y_{2}[5 = 8 = 2 \oplus 15] \) | \( 8 \) | \( \Delta Y_{2}[1 = 7 = 13] \) |
|      | \( STK_{3}[0, 1, 2, 4 = 7] \) | \( \Delta Y_{1}[3 = 6 = 9] \) | \( 8 \) | \( \Delta Y_{1}[3 = 6 = 9] \) |
|      | \( ETK_{0}[0, 3, 8 = 11] \) | \( \Delta Y_{0}[2 = 10] \) | \( 8 \) | \( \Delta Y_{0}[2 = 10] \) |
| 4    | \( STK_{25}, K_{24}, K_{0, 1} \) | \( \Delta X_{2}[1 = 14 = 4 = 11] \) | \( 8 \) | \( \Delta X_{2}[1 = 14 = 4 = 11] \) |
|      | \( STK_{5}[0] = K_{1} \) | \( \Delta X_{2}[11] = W_{7} \oplus 11 \) | \( 8 \) | \( \Delta X_{2}[11] = W_{7} \oplus 11 \) |
|      |                  | \( \Delta X_{2}[14] = W_{7} \oplus 10 \) | \( 8 \) | \( \Delta X_{2}[14] = W_{7} \oplus 10 \) |
|      |                  | \( \Delta X_{2}[3 = 6 = 9] \) | \( 8 \) | \( \Delta X_{2}[3 = 6 = 9] \) |
|      |                  | \( \Delta X_{2}[3 = 6 = 9] \) | \( 8 \) | \( \Delta X_{2}[3 = 6 = 9] \) |
| 5    | \( STK_{25}, K_{24}, K_{0, 1}, K_{3} \) | \( \Delta Y_{5}[3 = 6 = 9] \) | \( 8 \) | \( \Delta Y_{5}[3 = 6 = 9] \) |
|      | \( STK_{5}[5][1] STK_{5}[3, 4] = \) | \( \Delta Y_{3}, Y_{5} \) | \( 8 \) | \( \Delta Y_{3}, Y_{5} \) |
|      | \( K_{25} \) | \( \Delta Y_{3}[7 = 13] \) | \( 8 \) | \( \Delta Y_{3}[7 = 13] \) |
|      | \( \Delta Y_{3}[8 = 17] \) | \( 8 \) | \( \Delta Y_{3}[8 = 17] \) |
|      | \( ETK_{0}[0, 3, 8 = 11] \) | \( \Delta Y_{3}[4, 3, 9] \) | \( 8 \) | \( \Delta Y_{3}[4, 3, 9] \) |
|      | \( STK_{25}[6], STK_{1}[1] \) | \( \Delta Y_{3}[3, 6 = 9] \) | \( 8 \) | \( \Delta Y_{3}[3, 6 = 9] \) |

Total Complexity analysis

<table>
<thead>
<tr>
<th>( x )</th>
<th>Data</th>
<th>Memory</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>( 2^{127} )</td>
<td>( 2^{175} )</td>
<td>( 2^{123} + \frac{N^2 \rightarrow 20}{2^{123} + 2^{256}e^{-2^{256}} = 2^{244.45}} )</td>
</tr>
<tr>
<td>23.6</td>
<td>( 2^{127.6} )</td>
<td>( 2^{175.6} )</td>
<td>( 2^{123.6} + \frac{N^2 \rightarrow 20}{2^{123.6} + 2^{256}e^{-2^{256}} = 2^{238.5}} )</td>
</tr>
</tbody>
</table>
H.6 24-round Related-Tweakey ID Attack on ForkSKINNY-128-256-128 ($r_1 = 17, r_0 = 27$)

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.

![Diagram of related-tweakey ID attack](image-url)  

Figure 34: Related-Tweakey ID attack on 24 rounds of ForkSKINNY-128-256-128 in the arbitrary setting ($r_1 = 17, r_0 = 27$). $|k_0 \cup k_1| = 104, c'_0 = 56$ ($c_0 = 48$), $c_0 = 64$, $\Delta_0 = 64$, $\Delta_0 = 64$. 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 \( r_1 = 17, r_0 = 27, \) and \( r_1 = 7. \)

| Step | Guessed keys (#) | Condition(s) | # Remaining pairs \( (|L_i|) \) | Time | Deduced |
|------|------------------|--------------|-----------------|------|---------|
| 1    | \( STK_{23}[5] = K_{23} \) \( \{8\} \) | \( \Delta X_{23}[9 = 13] \) \( \{8\} \) \( \Delta X_{23}[1 = 9] \) \( \Delta X_{23}[3 = 7 = 15] \) | \( N^2 \to 20 \) | \( N^2 \) | \( ETK_0[11] \) |
|      | \( \Delta X_{23}[11 = 15] \) \( \{8\} \) \( \Delta Y_1[1 = 4 = 11] \) \( \Delta Y_1[1 \oplus 3 = 6 = 9] \) | \( (STK_{23}[7]) \) \( (STK_{23}[3, 7]) \) \( ET_{K_0}[6, 9] \) \( Y_1, \Delta Y_1 \) \( Z_{22}, \Delta Z_{22} \) | | | |
| 2    | \( STK_{22} \{8\} \) | \( \Delta Z_{20}[4 = 8] \) \( \{STK_{22}[7]\} \) \( \Delta Y_2[8 = 5] \) \( X_2[5] \) \( \Delta Y_2[2 \oplus 3 = 6 = 9] \) \( \Delta Y_2[1 \oplus 3 = 6 = 9] \) \( \Delta Z_{20}[4 = 8] \) \( \{STK_{22}[7]\} \) \( \Delta Z_{20}[4 = 8] \) \( \{STK_{22}[7]\} \) \( \Delta Y_2[8 = 5] \) \( X_2[5] \) \( \Delta Y_2[2 \oplus 3 = 6 = 9] \) \( \Delta Y_2[1 \oplus 3 = 6 = 9] \) | \( N^2 \to 28 \) | \( N^2 \to 4 \) | \( \Delta Z_{20}[4] \) |

Total - - \( N^2 \to 41 \) 24-round Enc. |

<table>
<thead>
<tr>
<th>( x )</th>
<th>Data</th>
<th>Complexity analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( 2^{118.4} )</td>
<td>( 2^{118.4} )</td>
</tr>
</tbody>
</table>

Total time \( 2^{123.17} \)
H.7 24-round Related-Tweakey ID Attack on ForkSKINNY-128-256-256 ($r_1 = 9, r_0 = 15$)

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 $r_1 = 21 - X$ 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 ($r_1 = 9, r_0 = 15$). $|k_0 \cup k_1| = 224$, $c_0 = 64$ ($c_0 = 56$), $c_p = 96$, $\Delta_{n} = 64$, $\Delta_{o} = 96$. In this case we have 15 related-key impossible differences.](image-url)
Table 10: Key-recovery procedure for ID attack on 24 rounds of ForkSkinny-128-256-256, where \( r_1 = 9 \), and \( r_0 = r_1 = 15 \).

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed keys (#)</th>
<th>Condition(s)</th>
<th># Remaining pairs (( L_i ))</th>
<th>Time</th>
<th>Deduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>( STK_{23}[1, 3, 7] = K_{23} ) [24]</td>
<td>( \Delta X_{23}[0 = 4] = 12 ) ( \Delta X_{23}[5] = 13 ) ( \Delta X_{23}[2 = 6 \oplus 10] = 14 )</td>
<td>( N^2 - 8 )</td>
<td>( N^2^{24} )</td>
<td>( STK_{23}[0, 4] ) ( STK_{23}[5] ) ( STK_{23}[2, 6] ) ( Z_{22} \Delta Z_{22} )</td>
</tr>
<tr>
<td>i=2</td>
<td>( K_{23} = STK_{22}[1, 4, 7][1, 3, 7] = K_{22, 21} ) [32]</td>
<td>( \Delta X_{22}[0 = 8] ) ( \Delta X_{22}[2 = 6] = 14 ) ( \Delta X_{21}[10 = 14] ) ( \Delta X_{21}[2] = 14 ) ( \Delta Z_{19}[7] = \delta_{15}(\Delta STK_{19}[7]) ) ( (15 \times 2^{-8} \text{ filters}) \rightarrow (4) )</td>
<td>( N^2 - 16 )</td>
<td>( N^2^{48} )</td>
<td>( STK_{22}[0] ) ( STK_{22}[2, 6] ) ( Z_{21} \Delta Z_{21} ) ( STK_{21}[2] ) ( \Delta Z_{19} ) ( \Delta Z_{19} )</td>
</tr>
<tr>
<td>i=3</td>
<td>( K_{23, 22, 21, 0} = STK_{1}[3] = K_{0, 1} ) [24] ( (STK_{23}[4, 5]) ) ( (STK_{21}[2, 6]) ) ( (STK_{1}) )</td>
<td>( \Delta Y_1[1 = 4 = 11] ) ( \Delta Y_1[3 \oplus 12 = 6 = 9] ) ( \Delta Y_1[6] = \delta_{-1}(\Delta STK_{2}[7]) ) ( {8} ) ( \Delta Y_2[15 = 5 = 8] ) ( X_2[5] ) ( X_2[5] ) ( W_1[1] ) ( W_1[4 \oplus 8] )</td>
<td>( N^2 - 28 )</td>
<td>( N^2^{64} )</td>
<td>( ETK_0[1, 2, 6, 14] ) ( ETK_0[4, 11, 12] ) ( ETK_0[5, 9] ) ( Y_1 ) ( \Delta Y_1 ) ( \Delta Y_2[6 = 15] ) ( {8} ) ( {8} ) ( STK_1[1] ) ( STK_1[7] )</td>
</tr>
<tr>
<td>i=4</td>
<td>( K_{23, 22, 21, 0, 1} = STK_{1}[2, 4, 6] = K_1 ) [24] ( (STK_{23}[2, 3, 6]) ) ( (ETK_0[4, 8, 9]) ) ( (STK_{22}[0], STK_{1}[7]) )</td>
<td>( \Delta Y_1 = \delta_{0}(\Delta STK_1[1]) ) ( {8} )</td>
<td>( N^2 - 36 )</td>
<td>( N^2^{76} )</td>
<td>( Y_2 ) ( \Delta Y_2 ) ( {8} ) ( {8} ) ( {8} ) ( {8} )</td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>-</td>
<td>( N^2^{72, 41} )</td>
<td>24-round Enc.</td>
<td>-</td>
</tr>
</tbody>
</table>

**Complexity analysis**

<table>
<thead>
<tr>
<th>( x )</th>
<th>Data</th>
<th>Memory</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>62.7</td>
<td>( 2^{126.7} )</td>
<td>( 2^{158.7} )</td>
<td>( 2^{127.7} + N^2^{72, 41} + 2^{556} e^{-2^x - 60} = 2^{246.62} )</td>
</tr>
</tbody>
</table>
H.8 20-round Related-Tweakey ID Attack on ForkSKINNY-128-256-128 ($r_1 = 7$, $r_0 = 13$)

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 $r_1 = 21 - X$ 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 ForkSKINNY-128-256-128 in the limited setting ($r_1 = 7$, $r_0 = 13$). $|k_h \cup k_f| = 96$, $c_h' = 40$ ($c_h = 32$), $c_f = 56$, $\Delta_h = 40$, $\Delta_f = 80$. 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 $r_1 = 7$, and $r_0 = r_1 = 13$.

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed keys ( # )</th>
<th>Condition(s)</th>
<th># Remaining pairs([L_1])</th>
<th>Time</th>
<th>Deduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i=1$</td>
<td>$STK_{19}[0] = K_{19}$</td>
<td>$\Delta X_{19}[0 = 4 = 12]$</td>
<td>$N_2^{-8}$</td>
<td>$N_2^8$</td>
<td>$STK_{19}[0, 4]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta X_{19}[1 = 13]$</td>
<td></td>
<td></td>
<td>$STK_{19}[1]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta X_{19}[2 = 10]$</td>
<td></td>
<td></td>
<td>$STK_{19}[2]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta X_{19}[10 = 14]$</td>
<td></td>
<td></td>
<td>$STK_{19}[3, 5]$</td>
</tr>
<tr>
<td></td>
<td>$(STK_{19}[0])$</td>
<td>$\Delta Y_1[0 = 7 = 10]$</td>
<td></td>
<td></td>
<td>$Z_{18}, \Delta Z_{18}$</td>
</tr>
<tr>
<td>$i=2$</td>
<td>$STK_{1}[0] = K_{1}$</td>
<td>$\Delta Y_2[4 = 1 = 11]$</td>
<td>$N_2^{-16}$</td>
<td>$N_2^8$</td>
<td>$STK_{19}[0, 4]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta X_{18}[0 = 12]$</td>
<td></td>
<td></td>
<td>$STK_{1}[0]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$X_2[1] = W_1[1 \oplus 9 \oplus 13]$</td>
<td></td>
<td></td>
<td>$STK_{1}[1]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$Y_2, \Delta Y_2$</td>
<td></td>
<td></td>
<td>$STK_{19}[1, 2, 5]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$Z_{17}, \Delta Z_{17}$</td>
<td></td>
<td></td>
<td>$STK_{19}[0, 6, 7]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$X_{17}, \Delta X_{17}$</td>
<td></td>
<td></td>
<td>$STK_{19}[0, 6, 7]$</td>
</tr>
<tr>
<td>$i=2$</td>
<td>$K_{19}, K_{1}$</td>
<td>$STK_{2}[1] = K_{2}$</td>
<td>$N_2^{-16}$</td>
<td>$N_2^8$</td>
<td>$\Delta Y_3[5]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta Y_3[5] = \delta_0(\Delta STK_3[5])$</td>
<td></td>
<td></td>
<td>$STK_{19}[1, 2, 5]$</td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>-</td>
<td>$N_2^6$</td>
<td>$2^{20}$</td>
<td>20-round Enc.</td>
</tr>
</tbody>
</table>

Complexity analysis

<table>
<thead>
<tr>
<th>x</th>
<th>Data</th>
<th>Memory</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>$2^{101}$</td>
<td>$2^{93}$</td>
<td>$2^{102} + N^2 + 2^{26} + 2^{128} = 2^{102} + 2^{26}$</td>
</tr>
</tbody>
</table>

H.9 31-round Related-Tweakey ID Attack on ForkSKINNY-128-288-288 ($r_1 = 10, r_0 = 25$)

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 $r_1 = 10$ rounds before the fork, and $r_0 = 25/r_1 = 21$ 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 ForkSkinny-128-288-288 in the arbitrary setting ($r_1 = 10, r_2 = 25$). $|k_1 \cup k_2| = 272$, $c_B = 64$ ($c_0 = 56$), $c_F = 128$, $\Delta_B = 64$, $\Delta_F = 128$. 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 \( r_1 = 10 \), \( r_0 = 25 \), and \( r_1 = 21 \).

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed keys {#}</th>
<th>Condition(s) {# filters}</th>
<th># Remaining pairs {L, i}</th>
<th>Time Deduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( STK_{30}^{[0, \ldots, 7]} = K_{30} ) {64}</td>
<td>( \Delta W_{29}^{[5 = 8 = 15 = 0]} ) {24}</td>
<td>( N^{2^{-24}} )</td>
<td>( N^{2^{54}} )</td>
</tr>
<tr>
<td>2</td>
<td>( STK_{29}^{[0, 2, 4]} = K_{29} ) {24}</td>
<td>( \Delta X_{29}^{[1 = 13 = 5 \oplus 9]} ) ( \Delta X_{29}^{[6 = 14]} ) ( \Delta X_{29}^{[3 = 15 = 7 \oplus 11]} )</td>
<td>( Z_{29}, \Delta Z_{29} )</td>
<td>( N^{2^{-36}} )</td>
</tr>
<tr>
<td>3</td>
<td>( K_{30}, K_{29} ) ( STK_{28}^{[2, 4, 5]} = K_{28} ) {24}</td>
<td>( \Delta X_{28}^{[1 = 9]} ) ( \Delta X_{28}^{[3 = 7 = 15]} )</td>
<td>( N^{2^{-44}} )</td>
<td>( N^{2^{76}} )</td>
</tr>
<tr>
<td>4</td>
<td>( K_{30}, K_{29}, K_{28} ) ( STK_{27}^{[7]} = K_{27} ) {8}</td>
<td>( \Delta X_{27}^{[3 = 11]} ) ( \Delta Z_{27}^{[3]} )</td>
<td>( N^{2^{-52}} )</td>
<td>( N^{2^{76}} )</td>
</tr>
<tr>
<td>5</td>
<td>( K_{30}, K_{29}, K_{28}, K_{27} ) ( STK_{26}^{[1, 2, 8]} = K_{1} ) {24} ( STK_{25}^{[1, 7]} ) ( STK_{24}^{[5, 7]} )</td>
<td>( \Delta Y_{1}^{[0 = 7 = 10]} ) ( \Delta Y_{1}^{[3 \oplus 12 = 9]} )</td>
<td>( N^{2^{-60}} )</td>
<td>( N^{2^{32}} )</td>
</tr>
<tr>
<td>6</td>
<td>( K_{30}, K_{29}, K_{28}, K_{27}, K_{26} ) ( STK_{25}^{[7]} = K_{1} ) ( STK_{24}^{[2]} ) ( STK_{23}^{[8]} )</td>
<td>( \Delta Y_{2}^{[1 = 4]} ) ( \Delta Y_{2}^{[1 = 11]} ) ( X_{2}^{[11]} ) ( STK_{1}^{[2]} )</td>
<td>( N^{2^{-68}} )</td>
<td>( N^{2^{32}} )</td>
</tr>
</tbody>
</table>

Total \( \sim \) \( N^{2^{68.63}} \) 31-round Enc.

<table>
<thead>
<tr>
<th>( x )</th>
<th>Data</th>
<th>Complexity analysis</th>
</tr>
</thead>
</table>
| 62.5 | \( 2^{126.5} \) | \( 2^{196.5} \) \( 2^{127.5} 
+ N^{2^{68.63}} + 2^{288}e^{-2^{-68}} = 2^{260.52} \)

\[ x \] Data   Complexity analysis
\[ 62.5 \] \( 2^{126.5} \) \( 2^{196.5} \) \( 2^{127.5} + N^{2^{68.63}} + 2^{288}e^{-2^{-68}} = 2^{260.52} \)
H.10 28-round Related-Tweakey ID Attack on ForkSKINNY-128-288-128 \((r_1 = 14, r_0 = 15)\)

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 ForkSKINNY-128-288-128](image)

The arbitrary setting \((r_1 = 14, r_0 = 15)\). \(|k_0 \cup k_1| = 112, c_0' = 64 (c_0 = 56), c_p = 40, \Delta_0 = 64, \Delta_p = 40\). 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 \( r_1 = 14, r_0 = 15, \) and \( r_1 = 14. \)

<table>
<thead>
<tr>
<th>Step</th>
<th># Plaintexts</th>
<th>Pair Generation</th>
<th># Pairs</th>
<th>N</th>
<th># Enc.</th>
<th>( 2^{x+64} )</th>
<th>( 2^{x+128} )</th>
<th>( 2^{x+40} )</th>
<th>( 2^{x+65} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Guess-and-Filter</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td># Guessed keys</td>
<td>Length of Condition(s)</td>
<td>( # ) Remaining</td>
<td>Time</td>
<td>Deduced</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( STK_{27}[1, 7] = K_{27} )</td>
<td>( 16 )</td>
<td>( \Delta X_{27}[2 = 6 = 14] )</td>
<td>( N^2 ) ( 12 )</td>
<td>( N^2 ) ( 16 )</td>
<td>( STK_{27}[2, 6] )</td>
<td>( Z_{26}, \Delta Z_{26} )</td>
<td>( STK_{26}[2, 6] )</td>
<td>( \Delta Z_{25}[2] )</td>
<td></td>
</tr>
<tr>
<td>( i = 1 )</td>
<td>( STK_{27}[1, 2, 6, 7] )</td>
<td>( \Delta Y_i[6 = 9] )</td>
<td>( (15 \times 2^{-8}) \rightarrow ) ( {4} )</td>
<td>( (ETK_0[6, 8, 9, 10]) )</td>
<td>( (STK_2[7]) )</td>
<td>( (STK_1[1]) )</td>
<td>( (STK_3[1]) )</td>
<td>( X_2[8] )</td>
<td></td>
</tr>
<tr>
<td>( i = 2 )</td>
<td>( STK_{26}[2, 6] )</td>
<td>( \Delta Y_2[7 = 7] = \delta_{-1}(\Delta STK_2[7]) )</td>
<td>( {8} )</td>
<td>( {8} )</td>
<td>( {8} )</td>
<td>( {8} )</td>
<td>( {8} )</td>
<td>( {8} )</td>
<td></td>
</tr>
<tr>
<td>( K_{27}, K_0 )</td>
<td>( 24 )</td>
<td>( STK_4[2, 4, 6] = K_1 )</td>
<td>( N^2 ) ( 36 )</td>
<td>( N^2 ) ( 40 )</td>
<td>( Y_2, \Delta Y_2 )</td>
<td>( (STK_4[2, 6]) )</td>
<td>( (STK_1[7]) )</td>
<td>( \Delta Y_2[7] )</td>
<td></td>
</tr>
<tr>
<td>( i = 2 )</td>
<td>( ETK_0[4, 8, 9] )</td>
<td>( (ETK_0[4, 8, 9]) )</td>
<td>( (STK_4[2, 6]) )</td>
<td>( (STK_1[7]) )</td>
<td>( \Delta Y_4[1] = \delta_{0}(\Delta STK_4[1]) )</td>
<td>( {8} )</td>
<td>( {8} )</td>
<td>( {8} )</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>( N^2 )</td>
<td>( 16, 80 )</td>
<td>( 28 )-round Enc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>Data</th>
<th>Memory</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>60.8</td>
<td>( 2^{124.80} )</td>
<td>( 2^{106.69} )</td>
<td>( 2^{125.8 + N^2 16.8 + 2^{128.8 - 2^{-0.00}} = 2^{126.68}} )</td>
</tr>
</tbody>
</table>

**H.11 28-round Related-Tweakey ID Attack on ForkSKINNY-128-288-288 (\( r_1 = 11, r_0 = 17 \))**

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 \( r_1 = 25 - X \) rounds before the fork, and \( r_0 = r_1 = 31 - X \) 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 ForkSKINNY-128-288-288 in the limited setting \((r_1 = 11, r_2 = 17)\). \(|k_0 \cup k_2| = 248\), \(c'_0 = 56\) \((c_0 = 48)\), \(c_F = 104\), \(\Delta_a = 56\), \(\Delta_F = 104\). 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 r₁ = 11, and r₀ = r₁ = 17.

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed keys (#)</th>
<th>Condition(s) (# filters)</th>
<th># Remaining pairs(LLL)</th>
<th>Time</th>
<th>Deduced</th>
</tr>
</thead>
</table>

Total - - N^2^⁻⁶⁸ 2⁸⁻⁶⁸ 2⁸⁻⁶⁸ = 2⁷⁷. 23

<table>
<thead>
<tr>
<th>x Data</th>
<th>Memory</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>70.9 2¹₂₆. 9</td>
<td>2¹⁵₈. 9</td>
<td>2¹₂₇. 9 + N^2^⁻⁶⁸ + 2⁸⁻⁶⁸ = 2²⁶⁷. 23</td>
</tr>
</tbody>
</table>
H.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 \( r_0 = r_i = 31 - X \) 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 ForkSKINNY-128-288-128 in the limited setting \( (r_i = 10, r_0 = 16) \). \( |k_0 \cup k_1| = 88, c'_0 = 8(c_0 = 0), c_0 = 64, \Delta_0 = 8, \Delta_f = 64 \). In this case we have 15 related-key impossible differences.](image-url)
Table 15: Key-recovery procedure for ID attack on 26 rounds of ForkSKINNY-128-288-128, where $r_1 = 10$, and $r_0 = r_1 = 16$.

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed keys (#)</th>
<th>Condition(s) (# filters)</th>
<th># Remaining pairs($\mathcal{L}_i$)</th>
<th>Time</th>
<th>Deduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i=1$</td>
<td>$STK_{25}[2, 4, 5] = K_{25}$ $\left(24\right)$</td>
<td>$\Delta X_{23}[3 = 9]$</td>
<td>$N_2^{-16}$</td>
<td>$N_2^{24}$</td>
<td>$STK_{25}[1]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta X_{25}[9 = 13]$ $\left(8\right)$</td>
<td></td>
<td></td>
<td>$STK_{25}[3, 7]$ $\mathcal{Z}<em>{24}$, $\Delta Z</em>{24}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta X_{25}[3 = 7 = 15]$</td>
<td></td>
<td></td>
<td>$STK_{24}[8]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta Z_{24}[6] = \delta_{10}(\Delta STK_{24}[6])$ $\left(15 \times 2^{-8}\right) \rightarrow \left(4\right)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta X_{24}[11 = 15]$ $\left(8\right)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta X_{24}[3 = 15]$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i=2$</td>
<td>$K_{25}$ $STK_{24}[7] = K_{24}$ $\left(8\right)$</td>
<td>$\Delta Z_{22}[4] = \delta_{18}(\Delta STK_{22}[4])$ $\left(8\right)$</td>
<td>$N_2^{-28}$</td>
<td>$N_2^{24}$</td>
<td>$\Delta Z_{22}[4]$</td>
</tr>
<tr>
<td></td>
<td>$K_{25}, K_{24}$ $ETK_0[0, 10, 13] = K_0$ $\left(24\right)$ $(STK_{25}[5])$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i=3$</td>
<td></td>
<td></td>
<td>$N_2^{-36}$</td>
<td>$N_2^{28}$</td>
<td>$\Delta Y_1, Y_1$ $(STK_1[0])$ $\Delta Y_2[0]$ $\left(8\right)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta Y_2[0] = \delta_0(= STK_2[0])$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>-</td>
<td>$N_2^{23.46}$</td>
<td>26-round Enc.</td>
<td></td>
</tr>
</tbody>
</table>

$x$ Data Memory Complexity analysis
| 116.6 | $2^{124.5}$ | $2^{68.5}$ | $2^{125.5} + N_2^{23.46} + 2^{128} e^{-2^{x-116}} = 2^{126.74}$ |
1  Related-Tweakey ID Attack on SKINNY

1.1 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.

![Related-Tweakey ID Attack diagram](image)

**Figure 41:** Related-Tweakey ID attack on 26 rounds of SKINNY-128-288-288. $|k_{in} \cup k_{et}| = 264$, $c_{in} = 72$ ($c_{et} = 64$), $c_{i} = 128$, $\Delta_{in} = 72$, $\Delta_{et} = 128$. 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

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed keys { #</th>
<th>Condition(s) { # filters</th>
<th># Remaining pairs([C_i])</th>
<th>Time</th>
<th>Deduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>SKK_{25}[0, 7] = K_{25} (64)</td>
<td>ΔW_{24}[5 = 8 = 15] = 0</td>
<td>N^{2 - 24}</td>
<td>N^{2^{64}}</td>
<td>Z_{24}, ΔZ_{24}</td>
</tr>
<tr>
<td>i=2</td>
<td>STK_{24}[0, 2, 4] = K_{24} (24)</td>
<td>ΔX_{24}[1 = 13 = 5 ⊕ 9]</td>
<td>N^{2 - 32}</td>
<td>N^{2^{64}}</td>
<td>STK_{24}[1, 5]</td>
</tr>
<tr>
<td>i=3</td>
<td>STK_{23}[2, 4, 5] = K_{23} (24)</td>
<td>ΔX_{23}[3 = 7 = 11]</td>
<td>N^{2 - 40}</td>
<td>N^{2^{60}}</td>
<td>STK_{23}[3, 7]</td>
</tr>
<tr>
<td>i=4</td>
<td>STK_{22}[7] = K_{22} (8)</td>
<td>ΔX_{22}[1 = 9]</td>
<td>N^{2 - 48}</td>
<td>N^{2^{60}}</td>
<td>STK_{22}[1]</td>
</tr>
<tr>
<td>i=5</td>
<td>ETK_0[1] = K_0 (8)</td>
<td>ΔY_{1}[1 ⊕ 11 = 14]</td>
<td>N^{2 - 56}</td>
<td>N^{2^{80}}</td>
<td>ETK_0[9, 11]</td>
</tr>
<tr>
<td>i=6</td>
<td>STK_{21}[3, 4, 6, 7]) (STK_{21}[0, 1, 5])</td>
<td>ΔY_{2}[2 = 9]</td>
<td>N^{2 - 64}</td>
<td>N^{2^{86}}</td>
<td>ETK_0[14]</td>
</tr>
<tr>
<td>i=7</td>
<td>STK_{0}[0] = K_0 (8)</td>
<td>ΔY_{5}[3 = 6]</td>
<td>N^{2 - 80}</td>
<td>N^{2^{86}}</td>
<td>ETK_0[8]</td>
</tr>
<tr>
<td>i=8</td>
<td>STK_{0}[0] = K_0 (8)</td>
<td>ΔY_{3}[0 = 7 = 10]</td>
<td>N^{2 - 104}</td>
<td>N^{2^{88}}</td>
<td>STK_{5}[0, 3, 5]</td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>-</td>
<td>N^{2^{92, 30}} 26-round Enc.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>Data</th>
<th>Memory</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>2^{122}</td>
<td>2^{104}</td>
<td>2^{123} + N^{2^{92, 30}} + 2^{288}e^{-2^{48}} = 2^{286.38}</td>
</tr>
</tbody>
</table>
I.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. $|k_b \cup k_f| = 120$, $c'_b = 56$ ($c_b = 48$), $c_f = 40$, $\Delta_b = 56$, $\Delta_f = 40$. 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.

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed keys (#)</th>
<th>Condition(s) {# filters}</th>
<th># Remaining pairs(|L_i|)</th>
<th>Time</th>
<th>Deduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair Generation</td>
<td>2^{2+56}</td>
<td>2^{2+112}</td>
<td>2^{2+24}</td>
<td>2^{2+57}</td>
<td></td>
</tr>
<tr>
<td>Guess-and-Filter</td>
<td>STK_{22}[1, 7]</td>
<td></td>
<td>ETK_0[9] = K_{22, 0} {24}</td>
<td>\Delta X_{22}[2 = 6 = 14]</td>
<td>STK_{22}[2, 6]</td>
</tr>
<tr>
<td>i=1</td>
<td>\Delta Y_i[7 = 10]</td>
<td>\Delta Y_i[7 = 10]</td>
<td>\Delta Y_i[7 = 10]</td>
<td>\Delta Y_i[7 = 10]</td>
<td>\Delta Y_i[7 = 10]</td>
</tr>
<tr>
<td></td>
<td>\Delta Y_i[6 = 9 = 12]</td>
<td>\Delta Y_i[6 = 9 = 12]</td>
<td>\Delta Y_i[6 = 9 = 12]</td>
<td>\Delta Y_i[6 = 9 = 12]</td>
<td>\Delta Y_i[6 = 9 = 12]</td>
</tr>
<tr>
<td></td>
<td>\Delta Y_i[0] = \delta_{-1}(\Delta STK_2[0])</td>
<td>\Delta Y_i[0] = \delta_{-1}(\Delta STK_2[0])</td>
<td>\Delta Y_i[0] = \delta_{-1}(\Delta STK_2[0])</td>
<td>\Delta Y_i[0] = \delta_{-1}(\Delta STK_2[0])</td>
<td>\Delta Y_i[0] = \delta_{-1}(\Delta STK_2[0])</td>
</tr>
<tr>
<td>K_{22, 0}</td>
<td>STK_{1}[3, 5, 7] = K_1 {24}</td>
<td>\Delta Y_2, Y_2</td>
<td>\Delta Y_2, Y_2</td>
<td>\Delta Y_2, Y_2</td>
<td>\Delta Y_2, Y_2</td>
</tr>
<tr>
<td>i=3</td>
<td>\Delta Y_i[2] = \delta_{0}(\Delta STK_4[2])</td>
<td>\Delta Y_i[2] = \delta_{0}(\Delta STK_4[2])</td>
<td>\Delta Y_i[2] = \delta_{0}(\Delta STK_4[2])</td>
<td>\Delta Y_i[2] = \delta_{0}(\Delta STK_4[2])</td>
<td>\Delta Y_i[2] = \delta_{0}(\Delta STK_4[2])</td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>-</td>
<td>N^{2^{57.06}}</td>
<td>23-round Enc.</td>
<td>N^{2^{57.06}}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>Data</th>
<th>Memory</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>64.8</td>
<td>2^{120.80}</td>
<td>2^{88.80}</td>
<td>2^{121.80} + N^{2^{57.06}} + 2^{128} e^{-2^x} = 2^{126.73}</td>
</tr>
</tbody>
</table>

I.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. $|k_0 \cup k_1| = 240, c'_5 = 48$ ($c_5 = 44$), $c_p = 64$, $\Delta_n = 48$, $\Delta_p = 64$. In this case, we have 15 related-key impossible differences.
Table 18: Key-recovery procedure for ID attack on 31 rounds of SKINNYe-v2.

<table>
<thead>
<tr>
<th>Step</th>
<th>Guessed keys (#)</th>
<th>Condition(s) (# filters)</th>
<th># Remaining pairs (#L_i)</th>
<th>Time</th>
<th>Deduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>STK_{31, 30, 29}[0-7]</td>
<td>N^{2^{32}}</td>
<td>N_{2^{96}}</td>
<td>Z_{27}, Z_{27}</td>
<td></td>
</tr>
<tr>
<td>i=2</td>
<td>K_{31, 30, 29} STK_{27}[0-7] = K_{27} (32)</td>
<td>N^{2^{32}}</td>
<td>N_{2^{116}}</td>
<td>Z_{26}, Z_{26}</td>
<td></td>
</tr>
<tr>
<td>i=3</td>
<td>K_{31, 30, 29}, K_{27} STK_{26}[1-5, 7] = K_{26} (24)</td>
<td>N^{2^{32}}</td>
<td>N_{2^{220}}</td>
<td>Z_{25}, Z_{25}</td>
<td></td>
</tr>
<tr>
<td>i=4</td>
<td>K_{31, 30, 29}, K_{27}, K_{26} STK_{3}[3, 7] = K_{25} (8)</td>
<td>N^{2^{32}}</td>
<td>N_{2^{224}}</td>
<td>Z_{23}[4]</td>
<td></td>
</tr>
<tr>
<td>i=5</td>
<td>K_{31, 30, 29}, K_{27}, K_{26}, K_{3}, ETK_{0}[0, 3, 8][1]</td>
<td>N^{2^{32}}</td>
<td>N_{2^{232}}</td>
<td>ETK_{0}[4]</td>
<td></td>
</tr>
<tr>
<td>i=6</td>
<td>K_{31, 30, 29}, K_{27}, K_{26}, K_{25}, K_{0, 1} STK_{2}[5] = K_{2} (4)</td>
<td>N^{2^{32}}</td>
<td>N_{2^{236}}</td>
<td>ETK_{0}[9], Y_3, Y_3</td>
<td></td>
</tr>
<tr>
<td>i=7</td>
<td>K_{31, 30, 29}, K_{27}, K_{26}, K_{25}, K_{0, 1}, K_{2} STK_{2}[2, 3, 7] = K_{3} (12)</td>
<td>N^{2^{32}}</td>
<td>N_{2^{236}}</td>
<td>ETK_{0}[2]</td>
<td></td>
</tr>
</tbody>
</table>

Total: N^{2^{140.04}} 31-round Enc.

<table>
<thead>
<tr>
<th>x</th>
<th>Data</th>
<th>Memory</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>2^{62}</td>
<td>2^{110}</td>
<td>2^{63} + N^{2^{140.04}} + 2^{256} e^{-2^{5.12}} = 2^{251.14}</td>
</tr>
</tbody>
</table>
J Integral Distinguishers of QARMAv2

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

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

(c) Integral distinguisher for 9 rounds of QARMAv2-64 (T = 1). Data complexity: 2^{14}.

Figure 44: ZC-based integral distinguishers for QARMAv2-64 (T = 1).
(a) Integral distinguisher for 8 rounds of QARMAv2-64 ($\mathcal{F} = 2$). Data complexity: $2^8$.

(b) Integral distinguisher for 9 rounds of QARMAv2-64 ($\mathcal{F} = 2$). Data complexity: $2^{16}$.

(c) Integral distinguisher for 10 rounds of QARMAv2-64 ($\mathcal{F} = 2$). Data complexity: $2^{14}$.

Figure 45: ZC-based integral distinguishers for QARMAv2-64 ($\mathcal{F} = 2$). 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.
Figure 46: ZC-based integral distinguisher for 12 rounds of QARMAv2-128 ($\mathcal{T} = 2$). Data complexity: $2^{14}$. 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 ($\mathcal{F} = 2$). 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.
K  Integral Distinguishers of MANTIS

(a) Integral distinguisher for 7 rounds of MANTIS. Data complexity: $2^8$.

(b) Integral distinguisher for 8 rounds of MANTIS. Data complexity: $2^{16}$.

(c) Integral distinguisher for 9 rounds of MANTIS. Data complexity: $2^{23}$.

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.
L  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.
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 distinguishers 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 bits (in terms of difference value or mask value) in the forward and backward propagations are represented by \( \square \) and \( \blacksquare \) respectively. In addition, the bit difference (or linear mask) 1 is represented by \( \blacktriangledown \) and \( \blacktriangleleft \) 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, CCJ16], 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.
Improved Search for Integral, Impossible Differential and Zero-Correlation Attacks

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.
The S-box Analyzer is a SageMath [Sag22] module designed to efficiently encode the S-boxes’ differential, linear, and integral properties in MILP, and SMT/SAT models. The 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.

```
# Import the S-box Analyzer and define the S-box
from sboxanalyzer import *
from sage.crypto.sboxes import MANTIS as sb
sa = SboxAnalyzer(sb)

# Encode deterministic differential behaviour of S-box
diff = sa.encode_deterministic_differential_behavior()
cpdiff = sa.generate_cp_constraints(diff)
Input: a0 || a1 || a2 || a3; a0: msb
Output: b0 || b1 || b2 || b3; b0: msb
print(cpdiff)
if (a0 == 0 \ a1 == 0 \ a2 == 0 \ a3 == 0) then (b0 = 0 \ b1 = 0 \ b2 = 0 \ b3 = 0)
elseif (a0 == 0 \ a1 == 0 \ a2 == 1 \ a3 == 0) then (b0 = -1 \ b1 = -1 \ b2 = 0 \ b3 = -1)
elseif (a0 == 0 \ a1 == 0 \ a2 == -1 \ a3 == 0) then (b0 = -1 \ b1 = -1 \ b2 = 0 \ b3 = -1)
elseif (a0 == 1 \ a1 == 0 \ a2 == 0 \ a3 == 1) then (b0 = 1 \ b1 = -1 \ b2 = 1 \ b3 = -1)
elseif (a0 == 1 \ a1 == 1 \ a2 == 0 \ a3 == 1) then (b0 = -1 \ b1 = -1 \ b2 = 1 \ b3 = -1)
elseif (a0 == 1 \ a1 == 1 \ a2 == 1 \ a3 == 1) then (b0 = -1 \ b1 = 1 \ b2 = 1 \ b3 = -1)
else (b0 = -1 \ b1 = -1 \ b2 = -1 \ b3 = -1)
endif
```

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