Meet-in-the-Filter and Dynamic Counting with Applications to Speck*

Alex Biryukov [1], Luan Cardoso dos Santos [1], Je Sen Teh [1,2], Aleksei Udovenko [1], and Vesselin Velichkov [3]

[1] University of Luxembourg {name.surname}@uni.lu
[2] University Sains Malaysia jesen_teh@usm.my
[3] University of Edinburgh vvelichk@ed.ac.uk

Abstract. We propose a new cryptanalytic tool for differential cryptanalysis, called *meet-in-the-filter* (MiF). It is suitable for ciphers with a slow or incomplete diffusion layer such as the ones based on Addition-Rotation-XOR (ARX). The main idea of the MiF technique is to stop the difference propagation earlier in the cipher, allowing to use differentials with higher probability. This comes at the expense of a deeper analysis phase in the bottom rounds possible due to the slow diffusion of the target cipher. The MiF technique uses a meet-in-the-middle matching to construct *differential trails* connecting the differential's output and the ciphertext difference. The proposed trails are used in the key recovery procedure, reducing time complexity and allowing flexible time-data trade-offs. In addition, we show how to combine MiF with a *dynamic counting* technique for key recovery.

We illustrate MiF in practice by reporting improved attacks on the ARX-based family of block ciphers Speck. We improve the time complexities of the best known attacks up to 15 rounds of Speck32 and 20 rounds of Speck64/128. Notably, our new attack on 11 rounds of Speck32 has practical analysis and data complexities of $2^{24.66}$ and $2^{26.70}$ respectively, and was experimentally verified, recovering the master key in a matter of seconds. It significantly improves the previous deep learning-based attack by Gohr from CRYPTO 2019, which has time complexity 2^{38}. As an important milestone, our conventional cryptanalysis method sets a new high benchmark to beat for cryptanalysis relying on machine learning.

Keywords: Symmetric-key · Differential cryptanalysis · ARX · Speck

1 Introduction

Differential cryptanalysis (DC) is one of the most powerful techniques for analyzing symmetric-key cryptographic algorithms. It has been proposed by Biham and

* The work was supported by the Luxembourg National Research Fund's (FNR) and the German Research Foundation's (DFG) joint project APLICA (C19/IS/13641232) and FNR's project SP2 (PRIDE15/10621687/SPsquared).

Shamir in 1991 [BS91] and since then has been used to successfully attack numerous symmetric-key primitives, including ciphers, hash functions, and MACs. Nowadays, resistance to DC is one of the basic properties that a symmetric-key algorithm must satisfy and new cryptographic designs often come with proofs of such resistance.

In DC, the attacker traces the propagation of differences (most commonly, XOR-differences) between plaintexts through multiple rounds of the analyzed primitive. By analyzing differences rather than plaintexts, the attacker effectively cancels out the action of the unknown round keys (also typically mixed in by an XOR). In this way, a sequence of differences over multiple rounds can be computed, which is called a *differential trail*. The latter typically holds with certain probability $p > 2^{-n}$ for an n-bit state and acts as a *distinguisher* of the analyzed cipher from a random permutation.

A typical DC attack starts with the derivation of a distinguisher on r rounds with probability p. It is then used to attack $r+u$ rounds of the cipher, where u is some number of rounds added after the distinguisher. In the attack, the attacker guesses (at least partially) the last u round keys in order to invert the last u rounds and to compute the output difference after r rounds. If this difference matches the output difference of the distinguisher, then, with some probability, the guess for the last round keys must have been correct. Extra l rounds are often also added at the top of the distinguisher resulting in an attack on $l+r+u$ rounds. DC is a statistical attack, meaning that the described process has to be repeated for many (at least p^{-1}) chosen plaintexts with a given input difference in order to successfully recover the last round keys with a sufficient success probability. Over the years there have been multiple extensions to the basic DC attack.
The contribution of this work is twofold:

Meet-in-the-Filter Tool

First, we propose a new addition to the DC toolkit, which we call *meet-in-the-filter* (MiF). This technique is especially suitable for ciphers with a slow or incomplete diffusion layer such as the ones based on Addition-Rotation-XOR (ARX). The main idea of the MiF technique is to stop the difference propagation earlier in the cipher resulting in a distinguisher on a fewer number of rounds (smaller value of r) with a relatively high probability p. This comes at the expense of a deeper analysis phase in the bottom rounds, i.e., a relatively high value of u. More specifically, in the MiF technique, we split $u = s + t$ into a precomputed *cluster* of differences for s rounds, then perform a Matsui-like search from the ciphertext difference, running backwards for t rounds up to the meeting point with the difference cluster. The filter discards a pair as wrong if the meeting point (the meet-in-the-filter) does not produce a valid $(s+t)$-round trail. For the reverse search, we use the fact that a differential $(\alpha, \beta \to \gamma)$ has the same probability through modular addition and modular subtraction (see Lemma 3). As a result, MiF produces a set of trails that are used in the key-recovery procedure.

To illustrate the practical use of the MiF technique we apply it to the ARX-based family of block ciphers Speck. After obtaining the set of 4-round trails

produced by MiF, an attacker can use a key recovery procedure similar to the one described by Dinur in [Din14a, Din14b][4] by just applying it twice – once for the bottom two rounds and once for the penultimate two rounds. Dinur suggested that although counting techniques could be applied to his procedure, it was not likely to improve the complexity of the attack. However, since MiF proposes trail differences for the full four rounds, we can use an advanced key recovery method with *dynamic counting* to improve time complexity.

Given a set of 4-round trails for SPECK, the dynamic counting procedure returns a set of candidate subkeys that satisfies at least c trails. Enforcing this requirement amplifies the filtering of subkey candidates, which reduces the key recovery time. Further, we describe the recursive implementation of the procedure which reduces the memory overhead of counting. This technique is applied to recover the four bottom subkeys of SPECK, which are sufficient to recover the full master key by applying SPECK's key schedule in reverse. An important distinction of our approach to Dinur's [Din14b] is that the latter analyzes the bottom four rounds of SPECK2n by making 2^{2n} key guesses for the bottom two of the four rounds (since the difference propagation in these rounds is not known) while in our case, the key-recovery procedure runs on all the four rounds.

With the MiF tool, we improve the time complexities of the best attacks reported in the literature on up to 15 rounds of SPECK32/64 and up to 20 rounds of SPECK64/128.

New benchmark for deep learning-based cryptanalysis of Speck32

At CRYPTO 2019 [Goh19], Gohr applied deep learning methods to attack reduced-round variants of SPECK32. These attacks beat the best previously known ones and this direction sparked a lot of interest in the community [BR20, BGPT21, BGL+21, BBP22], trying to understand the source of these speedups and to find further improvements and applications. However, to the best of our knowledge, these results were not yet beaten by conventional cryptanalysis techniques.

Among attacks on SPECK32 obtained with the MiF tool, we present significantly improved attacks on 11- and 12- round versions with practical data and time complexities. Notably, our attack on 11-round version is more than 2^{13} times faster than the deep learning-based (previously best) attack by Gohr [Goh19]. Experimentally, our attack recovers the right key in under a second on a laptop with a success probability of 63% (whereas Gohr's attack has 50% success rate). We provide experimental verification of the estimated complexities for 11- and 12-round attacks[5].

[4] We refer to [Din14b], which is the extended version of [Din14a] and which contains a full description of Dinur's algorithm.

[5] Experimental verification of our 11- and 12-round attacks on SPECK32/64 is available at github.com/1d50f/MiF. Our attack experiments were run on a single core of a laptop with Intel® Core™ i7-1185G7 CPU clocked at 3.00GHz and 32 GiB RAM.

As a result, we set the new high bar for deep learning-based cryptanalysis of SPECK32 to improve. This was one of the inspirations for the MiF tool and its application to SPECK.

Table 1: Notations used throughout this paper

Notation	Definition
FE	Full (SPECK) encryptions (time complexity measure)
n	Word size in bits
SPECK-$2n/(kn)$	SPECK with block size $2n$ and key size kn
$\oplus$, $\wedge$, $\vee$, $\neg$	Bitwise XOR, AND, OR, and NOT
$a \lll b$, $a \ggg b$	Cyclic shift of a by b bits to the left and to the right respectively
ADD , $+$	Addition modulo 2^n
SUB , $-$	Subtraction modulo 2^n
$\lvert S \rvert$	Size of the set S
a_i	i-th bit in the big-endian word a, where a_0 is the LSB
R	Total number of rounds
l, r, u	Number of top, middle, bottom rounds in an $l + r + u$ attack
$u = s + t$	Split of bottom rounds into s and t rounds
k	Number of key recovery rounds/key words
α, β, γ	XOR differences for addition or subtraction modulo 2^n
ΔX	XOR difference
$\Delta_{\text{IN}}, \Delta_{\text{OUT}}$	Input and output XOR difference to a differential (trail)
$\tau_r = (\Delta_{\text{IN}} \overset{r}{\rightarrow} \Delta_{\text{OUT}})$	A differential (trail) on r rounds
$\mathbf{xdp}^+$, $\mathbf{xdp}^-$	XOR differential probability of addition and subtraction
w, weight	Negative $\log_2$ of differential probability, i.e. $\Pr = 2^{-w}$
$\mathcal{S}(s, w_s)$ or $\mathcal{S}$	Cluster of trails on s rounds with $\Pr \geq 2^{-w_s}$
$\mathcal{T}(t, w_t)$ or $\mathcal{T}$	Set of filtered trails on t rounds with $\Pr \geq 2^{-w_t}$
p, q	Trail (single/differential/cumulative) probabilities
D	Number of chosen plaintexts
n_{trails}	Number of trails returned by MiF
c, c'	Target number of right trails in the counting attack its corresponding data multiplier
d	Current depth visited by the dynamic key recovery procedure
$B(k; n, p)$	The binomial distribution, $B(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k}$

The outline of the paper is as follows. Section 2 reviews previous attacks on SPECK, while Section 3 provides basic definitions, theorems, and lemmas used in the paper, as well as some relevant known results. It also includes a high-level description of the SPECK family of block ciphers. Section 4 presents the Meet-in-the-Filter (MiF) technique followed by the improved key-recovery framework based on counting. Attacks on SPECK32 and SPECK64/128 using the MiF tool are presented in Section 5, Section 6 and Section 7. The notations used throughout this paper are given in Table 1.

2 Related Work

2.1 Differential Cryptanalysis Techniques

We emphasize that the main designation of MiF is *aiding the key recovery phase of differential cryptanalysis*. MiF is closely related to *multiple-differential* cryptanalysis, since the concatenation of the main r-round differential and the s-round cluster of differentials can be seen as a multiple-differential *distinguisher*. However, the MiF attack allows to go deeper and also allows to use the s rounds in the cluster for *key recovery*, so that the actual *distinguisher* is simply the r-round differential (in the basic case). Furthermore, we can consider alternative round splits where the cluster covers many more rounds, and the MiF key recovery stage covers several bottom cluster rounds. This setting can be considered as an application of *MiF key recovery* inside a *multiple-differential distinguisher*.

Similarly, MiF can in principle be combined with *truncated differential distinguishers*. We can conclude that MiF-based *key recovery* can be *combined* with various kinds of *differential distinguishers*.

The proposed MiF technique also bears some similarity to other earlier results on meet-in-the-middle attacks, for example, on DES [DSP07], AES [DKS10, DFJ13] and LowMC [RST18, LIM21]. In [DSP07], the authors similarly lower the data complexity of their attack by recovering internal values rather than key bits (in contrast, we recover internal differences). In [DKS10], by enumerating the possible differential input/outputs to active S-boxes, a set of possible differential trails is recovered. The same idea is built upon in some of the results in [DFJ13]. More recently, attacks on LowMC [RST18,LIM21] leverage upon a conceptually similar reconstruction of differential trails but only for probability-one trails.

2.2 Cryptanalysis of Speck

All previous differential attacks on Speck start from a differential (trail) on r rounds to which 1 round is added at the top and u rounds are added at the bottom. In all cases, we can add this additional round at the top due to the fact that the key addition with the first round key is executed at the end of the first round, and so does not influence the attack complexity. Previous attacks on Speck32 and Speck64/128 along with the proposed new attacks are listed in Table 2. Time complexity is measured in the number of full encryptions (FE), data complexity D in the number of chosen plaintexts, and memory is in bytes. A brief summary follows next, which covers classical differential attacks and recently proposed differential-neural approaches.

In SAC 2014, Dinur proposed new attacks on Speck32 for up to 14 rounds, with the latter having time T and data D complexities of $(T, D)_{14R} = (2^{63}, 2^{31})$ [Din14b]. Later, in CRYPTO 2019, Gohr showed that neural networks could be trained to be cryptographic distinguishers [Goh19]. His 11-round attack on Speck32 uses differential-neural distinguishers that consist of 7-round (and a 6-round) neural distinguisher appended to a 2-round classical differential. The attack has

Table 2: Summary of differential attacks on SPECK32/64 and SPECK64/128. **Rounds** R/R' denotes that R out of R' rounds are attacked; **Split** $l + r + k = R$ denotes that to an initial differential (trail) on r rounds, l rounds are added at the top and k rounds are added at the bottom; **Pr diff** is the probability of the differential (trail) on r rounds. **Time**, **Data**, **Mem** are resp. the time, data and memory complexity of the attack; **Ref** is the reference to the publication describing the attack. Highlighted cells indicate the best attack time complexities for a given round.

Variant	Rounds	Split	Pr diff	Time	Data	Mem	Ref
SPECK32/64	11/22	1+6+4	2^{-13}	2^{46}	2^{14}	2^{22}	[Din14b]
SPECK32/64	11/22	1+0+8+2	-	$2^{40.15}$	$2^{14.11}$	$2^{28.97}$	this paper
SPECK32/64	11/22	1+9+1	Neural	2^{38}	$2^{14.5}$	2^{16}	[Goh19]
SPECK32/64	11/22	1+0+8+2	-	$2^{34.87}$	$2^{15.58}$	$2^{24.71}$	this paper
SPECK32/64	11/22	1+0+8+2	-	$2^{24.66}$	$2^{26.70}$	$2^{22.02}$	this paper
SPECK32/64	12/22	1+7+4	2^{-18}	2^{51}	2^{19}	2^{22}	[Din14b]
SPECK32/64	12/22	1+0+9+2	-	$2^{45.91}$	$2^{18.88}$	$2^{32.13}$	this paper
SPECK32/64	12/22	1+10+1	Neural	$2^{44.89}$	2^{22}	2^{16}	[BGL+21]
SPECK32/64	12/22	1+9+1	Neural	$2^{43.40}$	$2^{22.97}$	2^{16}	[Goh19]
SPECK32/64	12/22	1+7+2+2	$2^{-29.85}$	$2^{41.97}$	$2^{22.45}$	$2^{30.46}$	this paper
SPECK32/64	12/22	1+8+1+2	2^{-24}	$2^{33.84}$	$2^{30.42}$	$2^{24.75}$	this paper
SPECK32/64	13/22	1+8+4	2^{-24}	2^{57}	2^{25}	2^{22}	[Din14b]
SPECK32/64	13/22	1+0+10+2	-	$2^{56.41}$	$2^{25.27}$	$2^{36.85}$	this paper
SPECK32/64	13/22	1+8+2+2	$2^{-23.85}$	$2^{50.16}$	$2^{31.13}$	$2^{31.07}$	this paper
SPECK32/64	14/22	1+9+4	2^{-30}	2^{63}	2^{31}	2^{22}	[Din14b]
SPECK32/64	14/22	1+9+4	$2^{-29.47}$	$2^{62.47}$	$2^{30.47}$	2^{22}	[SHY16]
SPECK32/64	14/22	1+9+2+2	$2^{-29.37}$	$2^{61.35}$	$2^{30.64}$	-	this paper
SPECK32/64	14/22	1+9+2+2	$2^{-29.37}$	$2^{60.99}$	$2^{31.75}$	$2^{41.91}$	this paper
SPECK32/64	15/22	1+10+4	$2^{-30.39}$	$2^{63.39}$	$2^{31.39}$	2^{22}	[LKK+18]
SPECK32/64	15/22	1+10+2+2	$2^{-30.39}$	$2^{62.25}$	$2^{31.39}$	-	this paper
SPECK64/128	13/27	1+8+4	2^{-29}	2^{96}	2^{30}	2^{22}	[Din14b]
SPECK64/128	13/27	1+8+2+2	$2^{-28.87}$	$2^{59.53}$	$2^{31.46}$	$2^{39.97}$	this paper
SPECK64/128	13/27	1+8+2+2	$2^{-28.87}$	$2^{52.45}$	$2^{32.14}$	$2^{39.70}$	this paper
SPECK64/128	15/27	1+13+1	$2^{-58.9}$	$2^{61.1}$	2^{61}	2^{32}	[ALLW14]
SPECK64/128	16/27	1+14+1	$2^{-59.02}$	2^{80}	2^{63}	-	[BRV14]
SPECK64/128	19/27	1+14+4	2^{-60}	2^{125}	2^{61}	2^{22}	[Din14b]
SPECK64/128	19/27	1+14+2+2	$2^{-55.69}$	$2^{101.08}$	$2^{61.03}$	$2^{67.30}$	this paper
SPECK64/128	20/27	1+15+4	$2^{-60.56}$	$2^{125.56}$	$2^{61.56}$	2^{22}	[SHY16]
SPECK64/128	20/27	1+15+2+2	$2^{-60.73}$	$2^{122.69}$	$2^{63.96}$	$2^{77.19}$	this paper

a success rate of about 50% to recover the final 2 subkeys[6] with $(T, D)_{11R} = (2^{38}, 2^{14.5})$. Using a similar attack procedure, Gohr also has a 12-round attack with $(T, D)_{12R} = (2^{43.40}, 2^{22.97})$ but only a 40% success rate. Benamira *et al.* later delved into the inner workings of Gohr's approach from the standpoint of classical differential cryptanalysis [BGPT21]. They found that these distinguishers rely not only on the ciphertext pair but also on the difference distributions in the bottom two rounds. Apart from being able to better interpret the behaviour of the neural distinguishers and improving their accuracy, no new attacks on SPECK32 were reported. In [BGL+21], Bao *et al.* use a 10-round differential-neural distinguisher to mount a 12-round key recovery attack on SPECK32 using a similar key recovery framework as Gohr. By using more than one differential prepended to the neural distinguisher, they reported an attack with $(T, D)_{12R} = (2^{44.89}, 2^{22})$ and a higher success rate of 86%. Going back to classical differential cryptanalysis, Song *et al.* [SHY16] and Lee *et al.* [LKK+18] reported attacks on 14 and 15 rounds of SPECK32 with resp. $(T, D)_{14R} = (2^{62.47}, 2^{30.47})$ and $(T, D)_{15R} = (2^{63.39}, 2^{31.39})$ by using differentials rather than single trails as their distinguishers.

Next, we take a look at past attacks on SPECK64/128. In [ALLW14], Abed *et al.* use a differential trail on 13 rounds to which they add one round at the top and at the bottom to mount a $1 + 13 + 1$ attack on SPECK64/128. During the same period, Biryukov *et al.* [BRV14] reported an attack with time and data complexities $(T, D)_{16R} = (2^{80}, 2^{63})$ for SPECK64/128. In [Din14b], Dinur mounts a $1 + 14 + 4$ attack on SPECK64/128 with $(T, D)_{19R} = (2^{125}, 2^{61})$. Song *et al.* [SHY16] attack 20-round SPECK64/128 with $(T, D)_{20R} = (2^{125.56}, 2^{61.56})$. This was a $1 + 15 + 4$ attack that used a differential (rather than a single trail) for 15 rounds with $\Pr = 2^{-60.56}$ (the single trail probability is 2^{62}). The latter results in a slight improvement, the rest being the same as in Dinur's attack. Complexity-wise Song *et al.* attacks are already close to biclique attacks which work almost for any cipher.

2.3 Automatic Trail Search for Speck

Techniques for the automatic search for differential trails for SPECK can be broadly divided into two groups. In the first group the problem is represented in terms of Mixed Integer Linear Programming (MILP) or Satisfiability Modulo Theory (SMT) and off-the-shelf MILP or SAT solvers are employed to execute the search. Some results in this group are by Fu *et al.* [FWG+16] (MILP) and Song *et al.* [SHY16] (SMT) with the latter applying the method proposed by Mouha *et al.* [MP13] to construct a long differential trail from two short ones. The second group is composed of dedicated techniques based on Matsui's search algorithm [Mat94]. Biryukov *et al.* [BVC16] proposed the first adaptation of this algorithm to ARX ciphers and an optimised version using carry-bit-dependent difference distribution tables (CDDT) was later developed by Liu *et al.* [LLJW19]. Huang *et al.* [HW19] further optimized the latter using

[6] The second subkey was allowed to be wrong for at most 2 bits.

combinatorial DDT (cDDT). We note that the differential search algorithms from [BVC16, HW19, LLJW19] are complete, i.e., given enough time, they will return all the differential trails with a given differential probability.

3 Preliminaries

We begin with some preliminaries, necessary to understand the main results presented in subsequent sections. In the following exposition, addition and subtraction modulo 2^n are denoted respectively by ADD and SUB.

3.1 Differential Cryptanalysis

Differential cryptanalysis analyzes pairs of encryptions $P_1 \mapsto C_1, P_2 \mapsto C_2$ by studying the propagation of the input difference $\Delta P = P_1 \oplus P_2$ to the output difference $\Delta C = C_1 \oplus C_2$ through the cipher, which is known as a differential characteristic or trail. A differential trail consists of a sequence of differences:

$$\Delta P \rightarrow \delta_1 \rightarrow \delta_2 \rightarrow ... \rightarrow \delta_{r-1} \rightarrow \Delta C. \tag{1}$$

To perform an attack, an adversary needs a differential trail with sufficiently high differential probability:

$$p = \Pr_P[\Delta P \rightarrow ... \rightarrow \Delta C], \tag{2}$$

which is defined as the probability over all plaintexts. However, for simplicity of the analysis and due to the presence of round keys in ciphers, it is usually approximated by the probability of the trail over assumed-to-be-independent round keys (the so-called Markov assumption [LMM91]). In that case, the probability of the trail can be computed simply as the product of the probabilities of all the individual transitions:

$$p = \Pr[\Delta P \rightarrow \delta_1] \cdot \Pr[\delta_1 \rightarrow \delta_2] \cdot \ldots \cdot \Pr[\delta_{r-1} \rightarrow \Delta C]. \tag{3}$$

A better estimate of the differential probability can be obtained by collecting all differential trails that have the same input and output differences:

$$p = \Pr[\Delta P \rightarrow \Delta C] = \sum_{\delta_1 \ldots \delta_{r-1}} \Pr[\Delta P \rightarrow \delta_1 \rightarrow ... \rightarrow \delta_{r-1} \rightarrow \Delta C]. \tag{4}$$

The *weight* of a differential (trail) is defined as $w = -\log_2(p)$. The following variant of the Markov assumption is used to analyze our attack time complexities.

Assumption 1 *For a (possibly truncated) differential trail $\Delta P \rightarrow \Delta C$ with a weight w, and a uniformly and independently sampled pair of ciphertexts (C_1, C_2), the average* fraction *of subkeys for which the partial decryption of (C_1, C_2) follows the trail is equal to 2^{-w}.*

3.2 The Differential Probability of ADD and SUB

The differential probabilities of addition/subtraction modulo 2^n were studied by Lipmaa and Moriai [LM01].

Definition 1. $\mathbf{xdp}^+$ *and* $\mathbf{xdp}^-$ *are the probabilities with which input XOR differences* α, β *propagate to output XOR difference* γ *through the operations ADD and SUB respectively, computed over all* n*-bit inputs* a, b*:*

$$\mathbf{xdp}^+(\alpha,\beta,\gamma) = 2^{-2n} \cdot |\{(a,b) : ((a \oplus \alpha) + (b \oplus \beta)) \oplus (a+b) = \gamma\}| \ , \tag{5}$$

$$\mathbf{xdp}^-(\alpha,\beta,\gamma) = 2^{-2n} \cdot |\{(a,b) : ((a \oplus \alpha) - (b \oplus \beta)) \oplus (a-b) = \gamma\}| \ . \tag{6}$$

Lemma 1 ([LM01, Lemma 3]). *The probability* $\mathbf{xdp}^+(\alpha,\beta,\gamma)$ *is non-zero if and only if*

$$\alpha_i \oplus \beta_i \oplus \gamma_i = \begin{cases} 0 & \text{if} \quad (i=0) \ , \\ \beta_{i-1} & \text{if} \quad (i \geq 1) \wedge (\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}) \ . \end{cases} \tag{7}$$

If the probability $\mathbf{xdp}^+(\alpha,\beta,\gamma)$ is non-zero (i.e., the differential $(\alpha,\beta \to \gamma)$ is possible), its exact value can be computed with the formula given in Theorem 1.

Theorem 1 ([LM01, Algorithm 2]). *If* $\mathbf{xdp}^+(\alpha,\beta,\gamma) > 0$ *then*

$$\mathbf{xdp}^+(\alpha,\beta,\gamma) = 2^{-n+l+1}, \ \textit{where} \ \ l = |\{i \in \{0,\ldots,n-2\} : \ \alpha_i = \beta_i = \gamma_i\}| \ . \tag{8}$$

Note that the maximum possible transition weight through ADD is $n-1$. From Lemma 1 and Theorem 1, we can deduce that the differential probability of transitions does not depend on the order of the three differences. Furthermore, since the mapping $(a,b) \mapsto (a-b,b)$ is the inverse of $(a,b) \mapsto (a+b,b)$, we can deduce the SUB has exactly the same differential behaviour as ADD. The full proof is given in Appendix F.

Lemma 2. *The probability* $\mathbf{xdp}^+(\alpha,\beta,\gamma)$ *is invariant under any permutation of the inputs* α,β,γ*, i.e.,*

$$\mathbf{xdp}^+(\alpha,\beta,\gamma) = \mathbf{xdp}^+(\alpha,\gamma,\beta) = \mathbf{xdp}^+(\beta,\alpha,\gamma) = \ldots \ . \tag{9}$$

Lemma 3. *The differential* $(\alpha,\beta \to \gamma)$ *has the same probability through modular addition and modular subtraction for any choice of differences* α,β,γ*, i.e.,*

$$\mathbf{xdp}^+(\alpha,\beta,\gamma) = \mathbf{xdp}^-(\alpha,\beta,\gamma) \ . \tag{10}$$

3.3 Distribution of Differential Weights and Probabilities of ADD

In this section, we recall and derive properties of the distribution of weights and/or probabilities of differential transitions through the ADD operation. These properties will be used in the analysis of the MiF tool and complexities of the attacks. All proofs can be easily derived from the following lemma by Lipmaa and Moriai [LM01]. For completeness, they are provided in Appendix F.

Lemma 4 ([LM01, Theorem 2]). *The fraction of all transitions through ADD (including invalid ones) having weight w is given by*

$$\Pr_{\alpha,\beta,\gamma}[\mathbf{xdp}^+(\alpha,\beta,\gamma) = 2^{-w}] = \frac{1}{2}\left(\frac{7}{8}\right)^{n-1} B(w; n-1, \frac{6}{7}) \ . \tag{11}$$

Lemma 5. *Let α, β be chosen independently and uniformly at random. The expected number of differences γ such that the differential transition $(\alpha,\beta) \to \gamma$ is valid (i.e., $\mathbf{xdp}^+(\alpha,\beta,\gamma) > 0$) is given by*

$$\mathop{\mathrm{E}}_{\alpha,\beta}[|\{\gamma : \mathbf{xdp}^+(\alpha,\beta,\gamma) > 0\}|] = \left(\frac{7}{4}\right)^{n-1} = 2^{(n-1)\log_2 \frac{7}{4}} \ . \tag{12}$$

Lemma 6. *Let $(\alpha,\beta) \to \gamma$ be a transition through ADD sampled uniformly at random from all valid transitions through ADD. The average differential transition* probability *p is given by*

$$\mathop{\mathrm{E}}_{\substack{\alpha,\beta,\gamma:\\ \mathbf{xdp}^+(\alpha,\beta,\gamma)>0}}[\mathbf{xdp}^+(\alpha,\beta,\gamma)] = \left(\frac{4}{7}\right)^{n-1} = 2^{(n-1)\log_2 \frac{4}{7}}. \tag{13}$$

Example 1. SPECK32 uses 16-bit additions, for which the differential transitions have average weight approximately $w = 12.86$ and average probability approximately $2^{-12.11}$. SPECK64 uses 32-bit additions, for which the differential transitions have average weight approximately $w = 26.57$ and average probability approximately $2^{-25.03}$.

3.4 Dinur's Attack

Since our work draws parallels to Dinur's attack, we describe it briefly in this section. In its basic version, Dinur's attack uses an r round differential to attack $r+2$ rounds. All internal differences and some values in the bottom two rounds are known from the differential and ciphertexts. To recover the remaining unknown internal values, Dinur applies a guess-and-determine strategy that works bitwise on the bottom two modular additions (cf. *1RProcedure, 2RProcedure* [Din14b, Appendix A]). As a result, the last two round keys are recovered. The basic $r+2$ attack is then trivially extended to $r+4$ rounds for any SPECK variant by recovering two additional round keys through an exhaustive search, which increases the attack complexity by a factor of 2^{2n}. Dinur's attack applies two filtration procedures, called *one-bit* and *multi-bit* filters [Din14b, § 7.2], that exploit the differential properties of modular addition. The one-bit filter provides the main filtration power and is effectively a check for the conditions of Lemma 1. It gives filtration efficiency of $\frac{1}{2} \cdot (\frac{7}{8})^{n-1} \approx 2^{-7}$ for each 32-bit ADD with known α, β, γ differences. The multi-bit filter provides further improvement by a factor of about 2^{-3} for each 32-bit ADD operation.

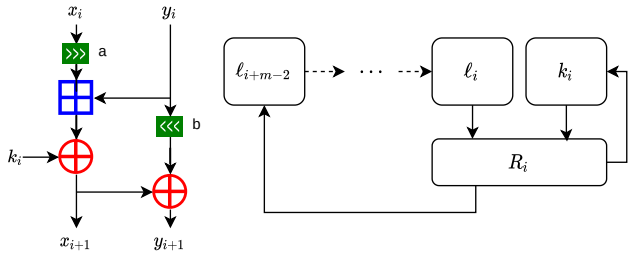

Fig. 1: Speck round function and key schedule.

3.5 Description of the Block Cipher Speck

Speck is a family of lightweight block ciphers proposed by USA National Security Agency in [BSS+13]. It follows an iterative ARX design, supporting block sizes of $32, 48, 64, 96$, and 128 bits and various key sizes. The members of the Speck family have been designed to provide good performance in software with their main target being microcontrollers. With Speck$2n/(kn)$ is denoted the instance of Speck with a block size of $2n$ bits composed of two n-bit words and a key size of kn bits, where k denotes the number of keywords. Speck$2n$ uses three operations over n-bit words: bitwise XOR, addition modulo 2^n and bitwise rotation. The key-dependant round function of Speck$2n$ depicted in Figure 1 is a map $R_K : \{0,1\}^{2n} \to \{0,1\}^{2n}$ defined as

$$R_K(x, y) = ((x \ggg r_a) + y) \oplus K, (y \lll r_b) \oplus (((x \ggg r_a) + y) \oplus K) \ , \qquad (14)$$

where the rotation values are $r_a = 7, r_b = 2$ for $n = 16$, and $r_a = 8, r_b = 3$ for all other block sizes. The decryption of Speck uses modular subtraction on the inverted round function and is naturally derived. The key schedule of Speck$2n$ takes the master key and generates R round-key words $K_0, K_1, \cdots, K_{R-1}$, where R is the number of rounds, using the same round function as used by the encryption. For a detailed description of Speck we refer the reader to [BSS+13].

4 The Meet-in-the-Filter (MiF) Attack

In this section, we describe the Meet-in-the-Filter (MiF) attack, which is divided into two main parts – the MiF tool and the key recovery procedure based on dynamic counting. It is applicable to ciphers with incomplete or relatively slow diffusion such as ARX.

4.1 The MiF Tool

Consider a block cipher with $r + u$ rounds split into r rounds covered by a differential (trail) and u rounds covered by backward search. The goal of the MiF *tool* is to efficiently enumerate trails for the bottom u rounds. We can further

split u into two parts: $u = s+t$, in order to obtain a time-memory trade-off. The s and t rounds of the split are processed separately in search of a meeting point (a matching difference). An illustration of the MiF filter is shown in Figure 2. We

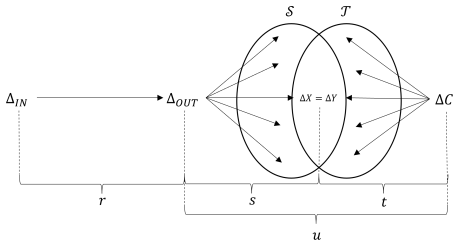

Fig. 2: Illustration of MiF with an $r + u$ and $u = s + t$ split.

start from an r-round differential with probability p denoted as $\Delta_{\mathrm{IN}} \xrightarrow{r} \Delta_{\mathrm{OUT}}$. Next we choose a suitable split of u into s top and t bottom rounds ($u = s + t$) together with corresponding probability thresholds 2^{-w_s} and 2^{-w_t}. In an offline phase, we apply Huang *et al.*'s Matsui-like search [HW19] to prepare *the cluster*.

Definition 2. *The* cluster $\mathcal{S}(s, w_s)$ *is the set of all* s*-round trails* τ_s *starting with the difference* Δ_{OUT} *and having probability at least* 2^{-w_s}*:*

$$\mathcal{S}(s, w_s) = \left\{ \tau_s = (\Delta_{\mathrm{OUT}} \xrightarrow{s} \Delta X) : \ \Pr[\tau_s] \geq 2^{-w_s} \right\} . \tag{15}$$

The functional notation $\mathcal{S}(s, w_s)$ stresses the fact that the set $\mathcal{S}$ is a function of the parameters s and w_s. We use $\mathcal{S}$ as a shorthand for $\mathcal{S}(s, w_s)$ when the parameters are clear from the context. Constructing $\mathcal{S}$ would usually require negligible precomputation time compared to the full differential attacks.

In the online phase, a set of $c' \cdot p^{-1}$ (for some small constant $c' \geq 1$) chosen plaintext pairs $(P_1, P_2 = P_1 \oplus \Delta_{\mathrm{IN}})$ are encrypted for $r + u$ rounds. For each corresponding ciphertext pair (C_1, C_2), a reverse search on t rounds starting with the ciphertext difference $\Delta C = C_1 \oplus C_2$ as input is executed. Since the reverse search is performed on SPECK in decryption mode, the modular addition operation ADD is replaced by modular subtraction SUB which does not change the probability computation due to Lemma 3. For a given observed ciphertext pair, the reverse search produces the filter-set $\mathcal{T}(t, w_t)$.

Definition 3. *The* filter-set $\mathcal{T}$ *consists of all* t*-round trails* τ_t *starting from* ΔC *in the reverse direction and having probability at least* 2^{w_t}*.*

$$\mathcal{T}(t, w_t) = \left\{ \tau_t = (\Delta C \xrightarrow{t} \Delta Y) : \ \Pr[\tau_t] \geq 2^{-w_t} \right\} . \tag{16}$$

Similarly to $\mathcal{S}(s, w_s)$, the set $\mathcal{T}(t, w_t)$ is expressed as a function of the parameters t and w_t. We use $\mathcal{T}$ as a shorthand for $\mathcal{T}(t, w_t)$ when the parameters are clear

from the context. Of all trails τ_t in $\mathcal{T}$, we keep only the ones whose output difference ΔY matches an output difference ΔX of a trail τ_s in $\mathcal{S}$. A match between a given $\tau_s \in \mathcal{S}$ and a given $\tau_t \in \mathcal{T}$ results in a u-round trail τ_u obtained by the following concatenation:

$$\tau_u = (\tau_t || \tau_s) = \Delta C \xrightarrow{u} \Delta_{\mathrm{OUT}} = \left(\Delta \mathrm{C} \xrightarrow{t} (\Delta Y = \Delta X) \xrightarrow{s} \Delta_{\mathrm{OUT}}\right). \tag{17}$$

A ciphertext pair for which a match is found, is recorded as a candidate *right pair*, i.e., a pair whose corresponding plaintexts (P_1, P_2) have followed the differential (trail) $(\Delta_{\mathrm{IN}} \xrightarrow{r} \Delta_{\mathrm{OUT}})$. Each such pair comes with a set of suggested u-round trails $\{\tau_u = \Delta_{\mathrm{OUT}} \xrightarrow{s+t} \Delta C\}$. The latter contains information for the key-recovery phase and is passed on to the key-recovery procedure. The set $\mathcal{S}$ is referred to as *the cluster* while the process of matching the set $\mathcal{T}$ against $\mathcal{S}$ is referred to as *the (backward) filter*. The absolute values of the logarithm base-2 probability thresholds – i.e., the constants w_s and w_t – are called respectively the *cluster weight* and the *filter weight*. Since the split $s + t$ can be seen as one large u-round filter that passes only candidate right pairs, the procedure is called *meet-in-the-filter* or MiF. In general, MiF offers the attacker a reduction in filtration complexity for the u bottom rounds through a time-memory trade-off.

Efficiency of MiF Pairs of plaintexts (P_1, P_2) that follow the differential for the top r rounds are called *right pairs* or *signal* while those that do not are *wrong pairs* or *noise*. After the application of MiF, some of this signal may be lost due to the weight thresholds (w_s and w_t) being applied to the bottom u rounds. Denote the probability that a right pair follows a u-round trail produced (or filtered) by MiF by q. Such a u-round trail is called a *right trail*, i.e., a right u-round trail is one that will be followed by the corresponding right pair after going through the initial r-round differential. We refer to q as the *efficiency* of the MiF filter. The inverse of q is the value by which the attacker needs to multiply the initial data $D = 2 \cdot c' \cdot p^{-1}$ to compensate for the decreased filter efficiency. The constant c' maintains the probability of catching at least c right trails in the set of trails (dataset) produced by MiF (see Section 4.3 and Appendix D), as required by our key recovery technique (see Section 4.3). Thus the overall data complexity of the attack is a function of the efficiency of the MiF filter and is equal to Dq^{-1}.

The efficiency of the MiF filter depends on the choice of the split values s and t, and the corresponding cluster and filter weights, w_s and w_t respectively. To maximize efficiency, the filter weight must be set large enough to allow all possible difference propagations in the backward filter. In particular, based on Lemma 6, the weight of average probability of a random valid t-round trail can be estimated as $-t(n-1)\log_2 4/7$, e.g., for $t = 2$ and $n = 16$, the weight of average probability of a 2-round trail is 24.22. To ensure that no trails will be discarded by the backward filter, the maximum value $w_t = 2(n-1) = 30$ should be set. If no limit is imposed on the backward filter, we can estimate q as the cumulative probability of all trails in $\mathcal{S}$ that comes from one r-round differential:

$$q = \sum_{\tau_s \in \mathcal{S}} \Pr[\tau_s] \ . \tag{18}$$

Typically, most trails suggested by MiF will not be right (i.e., are noise). We will denote the number of trails returned by MiF as n_{trails}. These trails, together with respective ciphertext pairs, are passed on to the key recovery stage, which we will describe and analyze in the following section.

4.2 Key Recovery using Single-Trail Analysis

In this section, we describe a general key recovery procedure based on *single* trail analysis. We recall the general setting – an attacker uses a differential $\Delta_{\text{IN}} \xrightarrow{r} \Delta_{\text{OUT}}$ over r rounds and queries encryption of a plaintext pair with the difference Δ_{IN} over $r + u$ rounds, obtaining a ciphertext pair (C_1, C_2) with a difference ΔC. MiF suggests a set of valid trails of the form $\Delta C \xleftarrow{u} \Delta_{\text{OUT}}$, with a hypothesis that this set contains the right trail.

In single-trail analysis, the attacker analyzes each proposed trail independently of other encryptions and all other trails. The analysis returns a set of *candidate subkeys* for analyzed $k \leq u$ rounds, for which the partial decryption of the ciphertext pair (C_1, C_2) follows the first k rounds of the suggested trail $\Delta C \xleftarrow{u} \Delta_{\text{OUT}}$ (i.e., the subtrail $\Delta C \xleftarrow{k} \Delta Z$ of the trail $\Delta C \xleftarrow{k} \Delta Z \xleftarrow{u-k} \Delta_{\text{OUT}}$). These candidate subkeys can then be used to derive candidates for the master key, to be tested against known encryptions or to follow the expected differential trail. The full key recovery attack simply consists of applying a sufficient number of iterations of the above procedure.

This setting follows the direction of Dinur's work; in fact, the procedure described in this section is simply a generalization of the analysis stage of Dinur's attacks. One of the main advantages of MiF is that this procedure can be applied right from the beginning due to the knowledge of a set of candidate trails. In addition, we pay closer attention to the theoretical analysis of the attack's complexity.

Recursive Single-Trail Procedure The procedure takes as input a ciphertext pair (C_1, C_2) and a trail $\Delta C \xleftarrow{k} \Delta Z$; it outputs all k-round subkeys for which the partial decryption of the pair (C_1, C_2) follows the given trail. The idea is simply to guess the subkeys *in chunks* and *recursively*.

Guessing subkeys *in (small) chunks* allows to quickly filter out wrong guesses, which are those making the partial decryption of the ciphertext pair (C_1, C_2) diverge from the given differential trail. A simple example is guessing subkeys round-by-round: after guessing one full subkey, we may decrypt the pair by one round and check whether the obtained difference follows the trail. Since many ciphers have incomplete diffusion over a small number of rounds, guessing even a small part of the subkey often allows partial decryption and computation of a part of the difference in the previous round, leading to faster discarding of invalid subkeys. For example, guessing even a single subkey bit in SPECK (starting from least significant bits) yields one bit of the difference in the previous round. Smaller chunks allow reducing the unnecessary work, bringing the procedure cost

close to the theoretical lower bound arising from the output size of the procedure – the total number of valid subkey candidates.

Recursive implementation of the procedure aims at minimizing the memory complexity. Indeed, the total number of candidate subkeys can be huge, and keeping all of them in memory at the same time is unnecessarily costly. Recursive guessing of the subkey chunks allows reducing the memory footprint of the procedure to negligible. An alternative formulation of this method is the depth-first traversal of the search tree (as opposed to breadth-first traversal).

Example 2. In our attacks on SPECK variants, we will set the chunk size to be 1 bit. The recursive procedure thus will simply recover the subkeys round-by-round and bit-by-bit, checking the conformance to the differential trail after each subkey bit guess. In SPECK, the n-bit subkey κ is XORed right after the ADD operation. When decrypting, the ADD becomes SUB and this subkey hides one of the inputs. Guessing i least significant bits of the subkey κ allows computing SUB on i least significant bits, leading to the knowledge of i least significant bits of the difference in the previous round[7], which can be used as a filter discarding wrong subkey guesses.

The following definition formalizes the notion of truncated trails, i.e., parts of the analyzed trail that can be tested after guessing some subkey chunks.

Definition 4. *Given a differential trail τ over k rounds and an integer d, by the* ***differential trail τ truncated at the depth*** *d we will understand τ restricted to all bit positions where the difference can be computed from the ciphertext difference and first d chunks of subkeys guessed. The maximum depth $d_{\max}$ is defined as the full number of chunks of subkeys that have to be guessed in the attack.*

Example 3. In SPECK, the maximum depth $d_{\max}$ is simply equal to the number of key recovery rounds (2, 3 or 4) times the word size, i.e., $d_{\max} = k \cdot n$.

4.3 Key Recovery using Multiple-Trail Analysis (Counting)

We propose an advanced method based on the *counting* technique. While Dinur opposed his single-trail analysis of SPECK to the counting method, we will show that counting often allows to significantly reduce the time complexity of the attacks, and becomes more applicable when coupled with the MiF technique. The basic idea of counting is to increase the number of encrypted pairs by a small factor $c > 1$ and target collecting and detecting at least c right trails in the full dataset. This requirement amplifies the filtering of the subkey candidates. For example, if a single-trail attack suggests 2^{26} 32-bit subkeys, a rough estimation shows that in the dataset of *double size* only $(2^{26+1})^2/2/2^{32} = 2^{21}$ 32-bit subkeys would be suggested by at least $c = 2$ trails.

[7] In fact, guessing $i < n$ bits allows to compute $i + 1$ bits of the difference. We will use this fact in Claim 3 to reduce the complexity.

Remark 1. If one wants to keep the same success probability of the attack (e.g., 63% in our case), the actual required multiplier c' to the number of encryptions has to be set slightly larger than c. We list the correct values of c' for small c (the detailed computations based on the Poisson distribution are described in Appendix D):

c	1	2	3	4	5	6	7	8
c' (success rate 63%)	1.00	2.15	3.26	4.35	5.43	6.51	7.58	8.64
$\log_2$	0.00	1.10	1.70	2.12	2.44	2.70	2.92	3.11
c' (success rate 95%)	3.00	4.74	6.30	7.75	9.15	10.51	11.84	13.15
$\log_2$	1.58	2.25	2.65	2.95	3.19	3.39	3.57	3.72

Recursive Multiple-Trail Procedure The most straightforward way to implement *counting* is simply to process each trail on-the-fly separately and maintain a counter for each discovered subkey candidate of *predetermined* size while using a hash table to address the counters. This approach however suffers from large memory complexity, often close to the time complexity of the attack.

We propose an alternative solution, called *dynamic counting*, the memory complexity of which is governed by the number of considered *trails* instead of the number of suggested *subkeys*. Each trail suggests on average a non-negligible amount of candidate subkeys, and this is exactly the savings factor in memory complexity for our solution, compared to naive counting. Furthermore, the memory access patterns in our procedure are sequential, as opposed to random memory accesses of the conventional counter-based method.

The main idea of dynamic counting is to change the order from "trail-then-subkeys" into "subkeys-then-trails". The high-level structure of the procedure is thus a recursive enumeration of subkeys in the depth-first order, as is done in the single-trail recursive procedure for each trail. However, in the new procedure, we keep the list of all trails satisfied by the currently guessed subkey chunks. Each subkey guess works as a "sieve" filtering the list of trails by discarding the trails that do not satisfy the new guess. It also severs subkey search branches having less than c surviving trails (the core of the counting method), hence being *dynamic*.

Note that the original list of trails is kept until the full current subtree is explored, i.e., we will effectively store one list of trails per each depth of the recursion tree (equal to the number of chunks in the involved subkeys). In practice, the size of the stored lists decreases fast when increasing depth, and, compared to the size of the original list, the memory overhead is just a small constant factor (depending only on the minimum partial weights of analyzed trails).

Remark 2. Note that the counting attack requires at least c right *pairs*, instead of c right *trails*. This means that the same subkey suggested from different trails of a given ciphertext pair should be counted only once.

Remark 3. Setting $c = 1$ reduces the algorithm into an alternative implementation of the single-trail procedure. Indeed, the set of visited trail-subkey pairs at

each depth will be the same, only the order differs (breadth-first versus depth-first). Effectively, this means that the time complexities of the two procedures (single-trail and multiple-trail with $c = 1$) are essentially the same, although the multiple-trail procedure requires more memory.

Memory Complexity While the keys-then-trails order improves the memory complexity, it may still be high for cases with large numbers of trails and memory should be allocated carefully. We propose several memory optimization techniques tailored to the SPECK block cipher in Appendix C. Based on the described techniques, we propose the following claim, which we will use for memory complexity estimations of our attacks. We remark that further reduction is possible through careful analysis of on-the-fly filtration efficiency.

Claim 1 *The memory complexity of the multi-trail procedure with optimizations can be estimated as* $2 \cdot n_{\text{trails}}$ *encryption blocks.*

4.4 Distributions of Weights in MiF Trails

The complexity of the MiF attack depends significantly on the chosen differential (especially on its output difference Δ_{OUT}), the round split, the cluster and filter weights w_s and w_t. These parameters affect in particular the properties of the trails suggested by MiF, namely the distribution of weights of *truncated trails* (in the sense of Def. 4), which directly affects the time complexity of the attack. Estimating this distribution purely by using theory from Section 3.3 is not possible as the evolution of the weights of truncated trails with depth is not uniform (we will show it on examples of our attacks). The time complexity of the multiple-trail key recovery is especially sensitive to intermediate weights, as they will often define the dominating stages of the attack.

Definition 5. *Given an integer d, let q_d denote the average probability for the MiF trails truncated at depth d (the trails are sampled uniformly at random from the possible output of MiF).*

By distributions of weights/probabilities in MiF trails we will mean the values $(q_0, q_1, \ldots, q_{d_{\max}})$. In addition, the attack's complexity depends directly on the (expected) number of trails to be suggested by MiF. It is thus necessary to be able to compute these quantities in order to estimate the time complexity of the attacks. The most straightforward way is to compute the full set of possible trails (for all reachable ciphertext differences) and to collect the required statistics from this set. However, in settings with large clusters this approach may not be feasible. To this end, we describe a generic sampling-based method. Its precision depends on the number of samples, which we ensure to be sufficiently large.

Obtaining Distributions via Generic Sampling The most straightforward way to obtain the distributions of weights of truncated trails is to partially simulate the attack and obtain a collection of trails from MiF, to be used further

to compute the necessary distributions. Running a full attack in most cases can be impractical. However, for sampling, the simulation process can be optimized significantly. An important observation is that the high complexity of the attacks stems from the difficulty of catching the *signal* (right pairs/trails), while most of the attack time is actually spent on *noise* (wrong pairs/trails). Since the absence of a few right trails would not change the distributions noticeably, we can restrict sampling to *noise only*. To this end, we propose the following simple procedure:

1. generate a random ciphertext difference ΔC (or, for more genuine results, encrypt a random plaintext pair following the chosen input difference Δ_{IN});
2. run the MiF tool and obtain a set of suggested trails;
3. update the required distributions from the given set;
4. repeat from Step 1 until a sufficient precision is reached.

In our attacks on Speck32 and Speck64 we noticed that sampling provides surprisingly stable and precise results. Our usual sampling goal is 1 million trails[8], or less for very low cluster weights w_s, where a large number of encryptions is needed to pass through the MiF tool. For these low cluster weight/small cluster scenarios, we can in fact avoid sampling and enumerate all reachable trails. For larger cluster weights w_s, one has to ensure that a large number of different ciphertext differences is involved, since a collection of 1 million trails suggested from just a couple of encryptions would not be sufficiently representative. In addition, sampling allows estimating well the average number of trails suggested by MiF per one encrypted pair. This is vital for computing the expected number of trails in a concrete attack, which in turn is needed to compute the time complexities (Claim 2, 3 and 4 below).

4.5 Key Recovery Complexity Analysis

We begin with the complexity analysis of the single-trail case, and we will build the analysis of the multiple-trail case on top of it. Our estimations will be based on the MiF trail weight distributions computed using techniques described in Section 4.4. For simplicity and due to relevance for Speck, we will assume the chunk size of 1 bit. Our key instrument is the following lemma, which connects the distribution q_d of weights/probabilities of truncated trails and the number of surviving trail-subkey pairs per depth.

Lemma 7. *At depth d of the single-trail procedure, across all branches and n_{trails} initial trails, there are on average*

$$v_d = n_{\mathrm{trails}} \cdot 2^d \cdot q_d \tag{19}$$

trail-subkey pairs visited.

[8] Deviations in average trail weights drop below 5% between 100000 to 500000 samples for smaller to larger cluster sizes, respectively.

Proof. Follows as an application of Assumption 1 to the key recovery procedure[9].

The total time complexity T_{cnt} of the key recovery procedure splits into two major parts: the complexity T_{enum} of enumerating (recursively) the subkey candidates and the complexity T_{trials} of checking the candidates by partial trial decryptions:

$$T_{\text{cnt}} = T_{\text{enum}} + T_{\text{trials}}. \tag{20}$$

Estimating T_{trials} The time complexity of the trial decryptions can be easily derived from the number of the final subkey candidates $v_{d_{\max}}$ suggested by the key recovery procedure. Naturally, we also assume that the key schedule can be easily inverted and all rounds' subkeys can be computed from the recovered subkey candidates (this is the case for most modern ciphers), at the cost proportional to the number of involved rounds, namely, $\frac{R-k}{R}$ FE.

In cases when the differential trail is known for at least 1 round longer than the key recovery requires, it can be used to test a subkey candidate at the lower cost of 2 round decryptions (equal to $\frac{2}{R}$ FE). Note that one-round trail extension with even a relatively low weight (say, 5) filters out most of the wrong candidates (31/32) and the consequent rounds add negligible complexity. This was suggested already in Dinur's work [Din14b, Section 6], but since it did not affect the dominating parts of his attacks, it was left only as a suggestion. However, this shortcut might not be available if we have used a differential rather than a single trail.

Claim 2 *Under the above assumptions,*

$$T_{\text{trials}} \leq v_{d_{\max}} \cdot \frac{R'}{R} \quad FE, \tag{21}$$

where $R' = 2$ if the differential trail is known for at least one more round, and $R' = R - k$ otherwise.

Estimating T_{enum} (Single-Trail) In order to estimate T_{enum}, we will assume that the time complexity of the single-trail recursive procedure is overwhelmingly dominated by the partial chunk decryptions. These can be counted by counting all *trail-subkey pairs* at each depth of the recursion. This is explained by the fact that each trail is analyzed independently of all other trails. We emphasize that summing the work done at each depth is needed to obtain an accurate estimate. Furthermore, we will (pessimistically) assume that one partial chunk decryption has cost equivalent to 1 round of the primitive (although it in fact requires just a few logic gates in the case of Speck[10]).

[9] Even though Speck is known not to be a Markov cipher, the theory holds well in practice as confirmed by our experiments.

[10] Bitslice-style optimizations for reducing this crucial constant might significantly improve the attack time complexity further, compared to [Din14b].

Claim 3 *Under the above assumptions,*

$$T_{\text{enum}} \leq \frac{R''}{R} \cdot \sum_{d=0}^{d_{\max}-1} v_d \quad FE, \tag{22}$$

where $R'' = 4$ in the general case. Furthermore, when the key chunks guessed are used for partial decryption of the word addition/subtraction, the complexity can be reduced by a factor of 2. In particular, $R'' = 2$ can be used in the case of SPECK.

Explanation. At depth d, by Assumption 1, each trail suggests on average $2^d q_d$ candidate truncated subkeys, totalling to $n_{\text{trails}} \cdot \sum_{d=0}^{d_{\max}-1} 2^d q_d$ non-final trail-subkey pairs. For each such pair, the partial decryptions are performed for each of the two candidates for the next subkey bit and for each of the two associated state values, leading to the cost of $4/R$ FE per a non-final trail-subkey pair.

The complexity halving in the SPECK case is based on the fact that, by Theorem 1, guessing i least significant bits of the (equivalent) key preceding the addition allows to check the difference for $i+1$ least significant bits. Effectively, this means that we can replace the two checks of the two 1-bit extensions of the current guess by one. Indeed, a direct application of the general estimation would mean that, for a fixed i-bit subkey, the two checks for $(i+1)$-bit subkey candidates would always return the same answer because the most significant bit in (truncated) addition does not affect the difference. Due to our cost estimation of 1 round of the primitive, we may perform decryption of states only after guessing each round's subkey's most significant bit. Since this bit propagates linearly through ADD, the two subkey candidates are related by one bit flip, which has negligible cost.

Estimating T_{enum} (Multiple-Trail) We will model each subkey suggested by trails as sampled independently and uniformly at random. This is formalized by the following assumption.

Assumption 2 *The subkeys suggested by each trail at each depth can be modelled as random uniformly distributed subsets of all possible subkeys, sampled independently from subkeys for trails suggested by another pair.*

The validity of the assumption is not entirely obvious. It is crucial to require independence only *across different pairs* (see Remark 2). Indeed, for one ciphertext pair, there would likely exist multiple trails of the form $\Delta C \to \Delta Z$ with prefixes equal up to some depth $d < d_{\max}$. This means that the keys suggested by these trails would be counted many times until the trails will diverge, even though they belong to a single ciphertext pair. That is why the assumption requires independence only between subkeys suggested by different pairs. In fact, the described intersections of suggested subkey sets related to a single pair of ciphertexts only reduce the number of suggested unique subkeys *per pair* and (slightly) improve the counting efficiency in practice. As we will show (see e.g.

Section 5.3), the analysis relying on this and other used assumptions closely match experimental data.

Claim 4 *For any depth d, $0 \le d \le d_{\max}$, and any integer c, $1 \le c \ll 2^d$, let $\eta_d = n_{\text{trails}} \cdot q_d$. Under the above assumptions,*

$$T_{\text{enum}}^{c>1} \le \frac{R''}{R} \cdot \sum_{d=0}^{d_{\max}-1} 2^d \cdot \eta_d \cdot \left(1 - e^{-\eta_d} \cdot \sum_{i=0}^{c-2} \frac{\eta_d^i}{i!}\right) \quad FE, \tag{23}$$

where R'' is defined as in Claim 3. In particular, $R'' = 2$ in the case of Speck.

Explanation. The high-level structure of this estimation is based on counting the average total number of trail-subkey pairs processed during the procedure, similarly to Claim 3 (estimating T_{enum} for the case $c = 1$).

As was shown in Lemma 7, the average number of trail-subkey pairs at depth d for $c = 1$ is equal to $n_{\text{trails}} \cdot 2^d \cdot q_d = 2^d \cdot \eta_d$. By Assumption 2, we can model them as $2^d \cdot \eta_d$ balls thrown into 2^d bins, with each throw chosen uniformly and independently at random. Our goal is to compute the expected number of balls (trail-subkey pairs) landing in bins (subkeys) with at least c balls in each of them. Solution to this standard problem is given in Proposition 3 in Appendix E, with $\eta = \eta_d$, $N = 2^d$ and $c = c$ in the proposition.

Remark 4. Note that the expression (23) with $c = 1$ reduces exactly to the expression (22) from Claim 3, if we define the sum $\sum_{i=0}^{-1} \ldots$ to be equal to zero.

5 Attacks on 11 Rounds of Speck32

In this section, we estimate the time complexity of the MiF filtering procedure when applied to Speck32 before describing MiF attacks on Speck32 reduced to 11 rounds. Our attacks can be divided into two categories (based on the division of the rounds): Attacks using an $r+s+t$ split, which is a straightforward application of MiF, and attacks with a MiF filter of a different size. We can vary the number of r, s, t rounds depending on the number of rounds being attacked, the availability of a valid differential or to achieve other time-data trade-offs.

5.1 Filtering Speck32 Trails with MiF

We begin by first describing the functionality of MiF when applied to Speck. Recall that we only need subkeys for $k = 4$ rounds to recover the full master key. Thus a straightforward application of MiF appends 4 rounds at the bottom of an r-round differential in the form of a $2 + 2$ MiF filter with $(s, t) = (2, 2)$. The operation of this filter configuration is shown in Figure 3. The elements in green are fixed values from the best r-round differential (trail) used in the attack. The elements in dark yellow come from the pre-computed cluster trails $\tau_s \in \mathcal{S}$. Purple elements correspond to trails $\tau_t \in \mathcal{T}$ generated by the reverse search procedure (the backward filter).

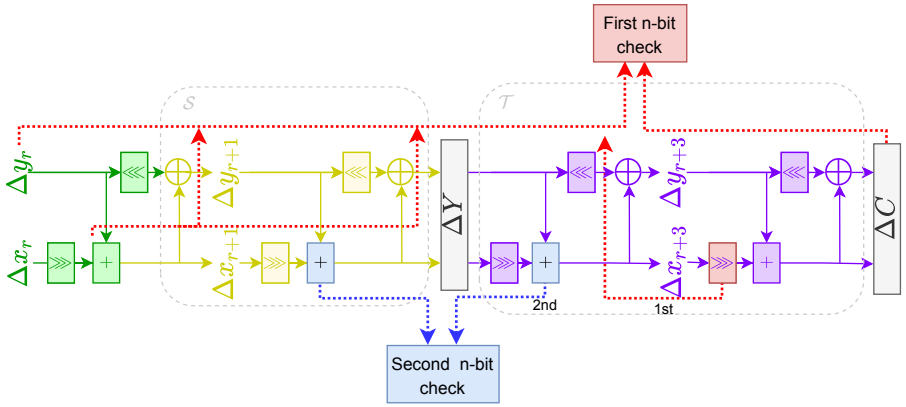

Fig. 3: Operation of a $2+2$ MiF filter on the bottom four rounds of an $r+4$ round attack on Speck2n. The elements in green are fixed values from an r round differential (trail). The elements in dark yellow come from a pre-computed cluster trail $\tau_s \in \mathcal{S}$. Purple elements correspond to trails $\tau_t \in \mathcal{T}$ generated by the reverse search procedure (the backward filter).

Since Speck is a Feistel-like cipher, the match in the middle between the sets $\mathcal{S}$ and $\mathcal{T}$ can be done efficiently n bits at a time. Specifically, the first n-bit check is executed on the right branch of round $r+3$ at the bottom (see Figure 3). It matches the differences generated by the ADD operation in the last round to the differences in the right branch coming from the cluster $\mathcal{S}$. This match is illustrated by the red line in Figure 3 (denoted "First n-bit check"). Only the trails $\tau_t \in \mathcal{T}$ that pass the first check proceed to the second n-bit check. The latter is executed on the left branch at round $r+2$ and is illustrated by a blue line in Figure 3 (denoted "Second n-bit check").

Denote by T_b the time complexity for checking the non-zero probability condition of Lemma 1 for a single ADD differential. Further, let T_a be the time complexity to generate a single output difference γ for fixed input differences α, β for one ADD operation, such that the differential $(\alpha, \beta \to \gamma)$ is of non-zero probability. The parameters $T_\mathrm{a}, T_\mathrm{b}$ are all measured in Speck encryptions. Next, we give the procedure for generating the bottom 4-round trails for the (1+6+2+2)-round attack. For the sake of generality, we omit the additional round at the top in our description since it does not affect the attack complexity.

1. Encrypt $\frac{D}{2}$ chosen plaintext pairs $\{(P_1, P_2 = P_1 \oplus \Delta_{\mathrm{IN}})\}$ for $r+2+2$ rounds and collect the corresponding ciphertexts $\{(C_1, C_2)\}$. Recall that the data complexity is $D = 2 \cdot c' \cdot p^{-1} \cdot q^{-1}$ chosen plaintexts.
2. Each ciphertext pair (C_1, C_2) from Step 1 is expanded into about $2^{12.1}$ ADD differentials for the last round modular addition in time T_a Speck-encryptions (per pair). This is the number of non-zero ADD differentials for Speck32 on average due to Lemma 5.

3. Of the $\frac{D}{2} \cdot 2^{12.1}$ ADD differentials from Step 2, a fraction of $\frac{|\mathcal{S}(s,w_s)|}{2^{16}}$ on average results in a match with an entry from the cluster $\mathcal{S}$. For each match, check the non-zero probability condition of Lemma 1 in time T_b.
4. Each ADD differential from Step 3 has a $p_z = 2^{-3.9}$ chance to be of non-zero probability (cf. Lemma 4). Therefore the total number of possible differentials surviving the $s+t$ MiF filter is $\frac{D}{2} \cdot 2^{12.1} \frac{|\mathcal{S}(s,w_s)|}{2^{16}} 2^{-3.9}$. Each one represents a candidate right pair.
5. For each trail $\tau_k = \tau_s || \tau_t$ from Step 4 execute key-recovery (Section 4.3).

In Step 2, the reverse search procedure in the backward filter of SPECK32 visits at most $2^{12.1}$ ADD differentials per round. Note that starting from ciphertext differences with low Hamming weight will (significantly) reduce this number, while for random differences we shall generally see all $2^{12.1}$ transitions if there is no limit on the permissible transition weight. Since the cluster entry at the point of the match is fixed from the output difference Δ_OUT of the differential (Step 3), the MiF filter checks on average, $\max\left(1, \frac{|\mathcal{S}(s,w_s)|}{2^n}\right)$ elements $(\Delta x_r, \Delta y_r \to \Delta x_{r+1})$ for the non-zero probability condition of Lemma 1. For example, entries in a cluster with $|\mathcal{S}| = 2^{20}$ elements will have $\frac{2^{20}}{2^{16}} = 2^4$ candidates on average, lower than the expected $2^{12.1}$ ADD transitions given in the attack procedure. Therefore the number of operations executed in the above steps is a worst-case estimate.

MiF Complexity The complexity of the MiF filtering procedure, T_mif can be estimated as follows:

$$T_\mathrm{mif} = \underbrace{\frac{D}{2} \cdot 2^{12.1} T_a}_{\text{Steps 1, 2}} + \underbrace{\frac{D}{2} \cdot 2^{12.1} \frac{|\mathcal{S}(s,w_s)|}{2^{16}} T_\mathrm{b}}_{\text{Step 3}} = D \cdot 2^{11.1} (T_a + \frac{|\mathcal{S}(s,w_s)|}{2^{16}} T_\mathrm{b}) \ . \quad (24)$$

The unit of measurement here is FE. For T_a and T_b we assume that each of the three basic arithmetic operations in SPECK (addition, bitwise rotation, XOR) have the same amortized cost of 1 unit operation (UO). Thus one round of SPECK, composed of five basic operations, costs 5 UO.

For SPECK32, we estimate the cost to generate a single output difference γ for fixed input differences α, β for one ADD to be equal to 1 UO on average (3 for SPECK64), since the cDDT is able to generate a new γ at every table access, after the other parts of the word were recursively set (in the cDDT, the (α, β, γ)-differentials are processed in 8-bit chunks). The cost of checking the non-zero probability condition of Lemma 1 is estimated at 11 UO, by counting the number of operations needed to implement. With the given amortized estimations in UO units, the parameters T_a and T_b for SPECK32 reduced to R rounds are computed in terms of R-round FE as: $T_\mathrm{a} = \frac{1}{5R}$ FE and $T_\mathrm{b} = \frac{11}{5R}$ FE. The $5R$ in the denominator comes from the fact that each round has five unit operations, i.e., costs 5 UO. Note that we also assume that the cluster search can be implemented as (hash) table look-ups requiring 1 UO each. In most of our attacks, however,

MiF's time complexity is not the dominating term, especially when larger clusters are in use.

Simplified MiF Filter For clusters that are smaller than $|\mathcal{S}| = 2^{12}$, we can instead opt for a simplified MiF procedure. Knowledge of the output difference ΔX stored in the cluster and the ciphertext difference ΔC allows deriving all XOR differences of the bottom two rounds. This is the same technique used by Dinur in his 2-round attack (cf. [Din14b, Section 7]). We can then verify if the trail is valid by Lemma 1. Thus we do not need to perform the backward filtering procedure (Steps 2–4) that checks (on average) $2^{12.1}$ possible ADD differentials for each ciphertext pair. Given a cluster entry and ciphertext pair, we estimate the cost of checking the validity of a trail to be $T_c = 18$ UOs (three rotations, four XOR s and one Lemma 1 check). The estimated time complexity of the simplified MiF procedure then reduces to:

$$T_{\text{mif}} \quad = \quad \frac{D}{2} \cdot |\mathcal{S}| \cdot T_c \ \text{ UOs} \quad \leq \quad \frac{D}{2} \cdot |\mathcal{S}| \cdot \frac{4}{R} \ \text{ FE.} \tag{25}$$

5.2 Attacks using Splits (1+6+2+2) and (1+0+8+2)

When using MiF, we are not restricted to having $u = s + t$ rounds appended after an r-round differential. Instead, the values of r, s, t can be varied to obtain various trade-offs. One extreme would be to have all r rounds of the differential as the top half of the MiF filter, i.e., a $0+s+t$ split with $s = r$. Note that when attacking the same number of rounds, the latter allows using longer differentials than the $r+s+t$ split. We consider two scenarios that according to our findings produced the best 11-round attacks: a (1+6+2+2) split using 6-round differentials, and a (1+0+8+2) split using 8-round differentials.

The (1+6+2+2) split follows exactly the basic structure of MiF described in Section 4. As for the (1+0+8+2) split, the cluster $\mathcal{S}$ will instead contain 8-round trails obtained by applying the Matsui-like search starting from the input difference Δ_{IN} of the 8-round differentials rather than their output difference Δ_{OUT}. This slightly increases the time required to pre-compute the $\mathcal{S}$ but does not affect the online phase of the attack. We only need to store information about the bottom two rounds of the 8-round trails in $\mathcal{S}$ to reconstruct the 4-round trails required for key recovery during the backward filtering procedure. Apart from using a different round configuration, the rest of the MiF filtering procedure follows the steps described in Section 5.1. Similarly, T_{mif} can be calculated based on Equation (24) or Equation (25) for $|\mathcal{S}| < 2^{12}$.

5.3 Results

Our strategy to find the best 11-round attacks on SPECK32 (and subsequently, other variants of SPECK) is as follows: We first identify the best differentials to be used in our attacks, some of which are listed in Table 6. Starting from a conservative value (usually the weight of the initial differential trail used in the

Table 3: Attacks on 11 round SPECK32: The "Diff. ID" column refers to the IDs of the differentials in Table 6).

No.	Split	w_s	$\lvert\mathcal{S}(s,w_s)\rvert$ ($\log_2$)	pq ($\log_2$)	c	D ($\log_2$)	T_{mif} ($\log_2$)	T_{cnt} ($\log_2$)	T_{att} ($\log_2$)	Diff. ID
1	1+6+2+2	34	19.28	-13.31	2	15.41	27.49	36.84	36.84	1
2		25	3.58	-21.30	3	24	25.13	25.09	26.11	2
3	1+0+8+2	37	21.27	-12.01	2	14.11	29.87	40.15	40.15	4
4		32	17.52	-13.48	2	15.58	25.93	34.87	34.87	5
5		24	0	-24.00	3	26.70	24.24	22.66	24.66	4

attack), we increment the cluster weight w_s and compute the attack complexities for both (1+6+2+2)and (1+0+8+2) splits. Note that we always set w_t to the maximum value of 30 as to not impose any limit on the backward filtering process, thus maximising MiF efficiency. We repeat the process for all possible differentials to identify the attacks with the best time and/or data complexities. The results of our search for the (1+0+8+2) split is shown in Figure 4, which consists of only the best attack time complexities T for varying amounts of data D. Additionally in Table 3, we provide parameters for several other best attacks, including those using the (1+6+2+2) split, on 11-round SPECK32.

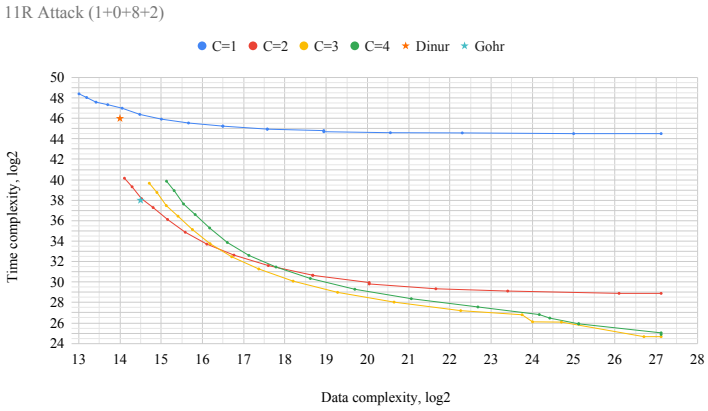

Fig. 4: Time and data complexities of the best 1+0+8+2 attacks on 11-round SPECK32.

For 11-round SPECK32, the figure shows that using either $c = 3$ or $c = 4$ leads to optimal time complexity versus data complexity trade-offs, i.e., by *spending* more data, we get bigger gains in analysis speed. This is in con-

trast to using $c = 1$, which is analogous to adopting Dinur's approach [Din14b], which barely sees any time complexity improvements with more data consumption. When optimized for time complexity, we have an (1+0+8+2) attack with $(T, D)_{11R} = (2^{24.66}, 2^{26.7})$[11] which is $2^{21.34}$ times faster than the 11-round attacks by Dinur [Din14b] and $2^{13.34}$ times faster than Gohr's [Goh19]. Recall also that Gohr's attack successfully recovers 30 bits of key information 50% of the time, while our attacks recover the full master key with a success rate of around 63%.

Generally, we found that using lower cluster weights w_s lead to better attack time complexities since the resulting cluster sizes $|\mathcal{S}|$ are smaller. A smaller cluster produces fewer trails for the key recovery procedure since only a fraction of the trails in the reverse search procedure will find a match in the cluster. Fewer trails in turn reduce the total number of keys that need to be filtered. Also, a smaller cluster size allows using the simplified MiF procedure described in Section 5.1. We can actually push this notion to its limits by setting w_s to the weight of the corresponding trail being used in the attack, thus having $|\mathcal{S}| = 1$. For example, our fastest 11-round attack uses an input difference of (`0x0a20`, `0x4205`) along with an (1+0+8+2) split. This input difference corresponds to an optimal 8-round trail with probability 2^{-24}. Therefore by setting $w_s = 24$, this 8-round trail is the only one being stored in the cluster ($|\mathcal{S}| = 1$).

However, going to these extremes means these attacks require the most data. When optimized for time complexity, our best attack on 11 rounds requires more (albeit still practical, $D < 2^{27}$) data than previous 11-round attacks by Dinur and Gohr, which only require 2^{14} and $2^{14.5}$ chosen plaintexts respectively. This is due to the lower efficiency q of the MiF filter, which has to be compensated by increasing the amount of data used. Thus we can reduce the data complexity for MiF attacks by using larger cluster weights, which increases both $|\mathcal{S}|$ and q. By having $w_s = 37$, we have an 11-round attack which is about 56 times faster than Dinur's attack while using a similar amount of data $(T, D)_{11R} = (2^{40.20}, 2^{14.11})$. By using $w_s = 32$, we still end up using twice as much data as Gohr, but now have an attack that is around 8 times faster $(T, D)_{11R} = (2^{35.06}, 2^{15.58})$ and with better success rate.

Experimental Verification of an 11-round Attack The fastest 11-round attack using the (1+0+8+2) split (Attack #5 from Table 3) was implemented and verified in practice. We provide detailed experiment information for $c = 3$, which is the fastest variant.

The offline MiF phase generates a cluster S with the given parameters: $s = 8, w_s = 24$. Due to low cluster weight, the only trail in the cluster is the best trail (`0x0a20,0x4205`) $\xrightarrow{8}$ (`0x802a,0xd4a8`) of weight 24. Next, $D = 2^{25.7}$ random pairs with difference $\Delta P = \Delta_{\text{IN}} =$ (`0x0a20,0x4205`) are encrypted. For each ciphertext pair, we run the simplified MiF procedure from Section 5.1 to

[11] The time complexity is less than the data complexity since it is measured in full (11-round) SPECK32 encryptions. Most of $2^{25.7}$ collected *pairs* are filtered out by MiF with the complexity of 1-round SPECK encryption.

bridge the difference Δ_{OUT} = (0x802a,0xd4a8) from the cluster with the ciphertext difference ΔC, and checking the resulting 2-round trail for validity (using Lemma 1). Valid trails are recorded together with the associated ciphertext pairs. Our implementation performs this procedure in several seconds. As a result, we collect $2^{14.9}$ trails, which is in line with $2^{-10.8}$ trails/pair obtained using the method from Section 4.4.

We run the multi-trail key recovery procedure (Section 4.3) with $c = 3$ (in fact, using the secret key we could see that 5 right trails were actually suggested by MiF). Our implementation performs this procedure in less than a second, yielding the (only) right secret master key. Our not fully optimized attack demonstrates significant performance improvement over the previous best attack on 11 rounds from Gohr which takes about 500 seconds [Goh19]. A graph illustrating the complexity of key recovery for this attack is provided in Appendix B.

6 Attacks on 12 to 15 Rounds of Speck32

In Table 4 we summarize the best attacks on 12 to 15 rounds of Speck32 along with the attack parameters. For each number of rounds, we list the best attack in terms of time complexity and optimal attacks that use a similar amount of data as previous attacks in the literature. For example, the best 12-round attack using Dinur's approach [Din14b] requires 2^{19} chosen plaintexts and has a time complexity of $T = 2^{51}$. In contrast, we can use slightly less data (Table 4, #1) for an attack that is about 34 times faster. We also have a 12-round attack that is faster than the differential-neural attack by Bao *et al.* [BGL+21] by a factor of 7.6 by only using 1.5 times more data (Table 4, #2). At higher rounds such as 14 and 15, the time-data trade-offs are no longer possible as we are working with almost the full codebook. Due to the restriction in data complexity, we are limited to just using $c = 1$ or 2. However, we still have 14-round and 15-round attacks that are around two to three times faster. In all cases, MiF complexity is not the dominant term and does not affect the overall analysis complexity ($T_{\mathrm{att}} \approx T_{\mathrm{cnt}}$).

Experimental Verification of a 12-round Attack The fastest attack on 12-round Speck32/64 using the round splits $1+8+1+2$ (attack #3 from Table 4) was implemented and verified in practice. Initially, it was executed using the split $1+0+9+2$, however, after the inspection of the generated cluster, it became clear that the split $1+8+1+2$ describes it more precisely (see below). We provide detailed experiment information for $c = 4$, which is the fastest variant.

The offline MiF phase generates a cluster S with the given parameters: $s = 9, w_s = 31, \Delta_{\mathrm{IN}} =$ (0x7458, 0xB0F8). Due to the low cluster weight, the generated cluster contains only 12 trails. Upon a manual inspection, it turned out that the 9-round cluster in fact consists of a single 8-round trail (namely, (0x7458, 0xB0F8) $\xrightarrow{8}$ (0x802A, 0xD4A8) having the best possible trail weight 24), extended by 1 round in 12 different ways. The cluster has efficiency $2^{-27.30}$,

Table 4: Attacks on 12–15 rounds of SPECK32: The "Diff. ID" column refers to the IDs of the differentials in Table 6).

No.	Rounds	Split	w_s	$\lvert\mathcal{S}(s,w_s)\rvert$ ($\log_2$)	pq ($\log_2$)	c	D ($\log_2$)	T_{mif} ($\log_2$)	T_{cnt} ($\log_2$)	T_{att} ($\log_2$)	Diff. ID
1	12	1+0+9+2	38	21.27	-16.17	3	18.88	32.80	45.91	45.91	8
2	12	1+7+2+2	36	15.71	-19.74	3	22.45	30.96	42.02	42.02	3
3	12	1+8+1+2	31	3.58	-27.30	4	30.42	31.42	33.54	33.84	7
4	13	1+0+10+2	43	19.38	-23.16	2	25.27	37.20	56.41	56.41	11
5	13	1+8+2+2	40	11.69	-28.01	4	31.13	36.84	50.16	50.16	6
6	14	1+9+2+2	50	17.84	-29.65	1	30.64	40.95	61.35	61.35	9
7	14	1+9+2+2	50	17.84	-29.65	2	31.75	42.05	60.99	60.99	9
8	15	1+10+2+2	55	18.18	-30.40	1	31.39	41.93	62.25	62.25	10

catching the $2^{-3.30}$ fraction of the signal from the 8-round trail. Next, $D = 2^{29.42}$ random pairs with difference $\Delta P = \Delta_{\text{IN}}$ are encrypted. For each ciphertext pair, we run the simplified MiF procedure from Section 5.1 to bridge one of the differences ΔX from the cluster with the ciphertext difference ΔC. Our implementation performs this procedure in 40 seconds. As a result, we collect $2^{23.52}$ trails, which is in line with $2^{-5.87}$ trails/pair obtained using the method from Section 4.4.

We run the multi-trail key recovery procedure (Section 4.3) with $c = 4$. Using the secret key we could see that 7 right trails were actually suggested by MiF. The increased number of right trails was persistent across several executions. We explain this by probability increase for the cluster due to the *differential effect* of the underlying trails. This means that we could, in principle, use higher c with the same data complexity D while maintaining the target success rate above 63%. Our implementation performed this procedure in 13 minutes for $c = 4$ or in 6 minutes for $c = 7$ (on the same data set), yielding only the correct secret master key.

Our attack demonstrates significant performance improvement over the previous best attack on 12 rounds by Gohr (12 hours) [Goh19]. The illustration of the time complexity evolution of the attack for different values of c (and data complexity adapted to maintain the success rate of 63%) can be found in Figure 5.

7 Attacks on Speck64/128

In Table 5 we highlight some of our best attacks on 13, 19 and 20 rounds of SPECK64/128, all of which adopt a $1 + r + 2 + 2$ split. Contrary to intuition, our results show that using suboptimal differentials can sometimes produce better attacks due to having better trail weight distributions for key recovery e.g. 4-round trails with heavier weights in the top two rounds would suggest fewer keys than those with less. The time-data trade-offs that are possible with MiF can be clearly observed in the 13-round and 19-round attacks which both have data

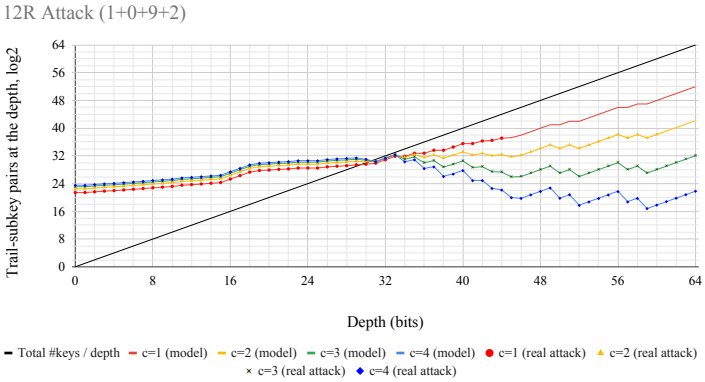

Fig. 5: Time complexity analysis of an attack family on 12-round SPECK32 (see Table 4, #3). Lines plotted are the predicted numbers of trail-subkey pairs visited per each depth $0 \ldots 64$ for attacks with $c = 1, 2, 3, 4$; data points mark values collected from real attack runs, one run per each c (full attacks for $c = 3, 4$ and partial samples up to feasible depths for $c = 1, 2$).

complexities that are well within the codebook. When using 2.5 times the data, we have a 13-round MiF attack that is around $2^{34.66}$ times faster than Dinur's approach [Din14b]. By further doubling the amount of data, analysis speed is further improved by a factor of $2^{8.89}$.

Table 5: Attacks on SPECK64: The "Diff. ID" column refers to the IDs of the differentials in Table 6).

No.	Rounds	Split	w_s	$\lvert\mathcal{S}(s, w_s)\rvert$ ($\log_2$)	pq ($\log_2$)	c	D ($\log_2$)	T_{mif} ($\log_2$)	T_{cnt} ($\log_2$)	T_{att} ($\log_2$)	Diff. ID
1	13	1+8+2+2	59	26.13	-29.18	2	31.28	50.96	61.34	61.34	12
2	13	1+8+2+2	56	23.71	-29.35	2	31.46	51.06	59.53	59.53	12
3	13	1+8+2+2	55	22.86	-29.44	3	32.14	51.75	51.07	52.45	12
4	19	1+14+2+2	86	26.97	-56.46	2	58.56	77.76	114.65	114.65	13
5	19	1+14+2+2	81	22.02	-57.32	6	61.03	79.80	101.08	101.08	13
6	20	1+15+2+2	92	27.98	-61.86	2	63.96	83.22	122.69	122.69	14

When using around the same amount of data $D \approx 2^{61}$ as the best attacks in literature, our attack has an analysis complexity of $2^{101.08}$, which is around 2^{24} faster. Compared to SPECK32, we clearly see bigger gains when using MiF on SPECK64 because more noise (wrong trails) can be quickly discarded using the counting technique. This is due to these trails having a lower differential

transition probability (Lemma 6). When it comes to 20 rounds of Speck64/128, we face restrictions in terms of data complexity as we have almost exhausted the codebook. Thus, we are limited to using $c = 2$ in our best attack, which is still 7.3 times faster than the best 20-round attack proposed by Song *et al.* [SHY16].

Acknowledgement

We thank Daniel Feher and Giuseppe Vitto for implementation of Dinur's filtering algorithm and early study of key-recovery strategies in MiF.

References

ALLW14. Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential cryptanalysis of round-reduced Simon and Speck. In *FSE 2014*, volume 8540 of *LNCS*, pages 525–545. Springer, 2014. 6, 7

BBP22. Nicoleta-Norica Bacuieti, Lejla Batina, and Stjepan Picek. Deep neural networks aiding cryptanalysis: A case study of the Speck distinguisher. Cryptology ePrint Archive, Report 2022/341, 2022. 3

BGL+21. Zhenzhen Bao, Jian Guo, Meicheng Liu, Li Ma, and Yi Tu. Conditional differential-neural cryptanalysis. Cryptology ePrint Archive, Report 2021/719, 2021. 3, 6, 7, 27

BGPT21. Adrien Benamira, David Gérault, Thomas Peyrin, and Quan Quan Tan. A deeper look at machine learning-based cryptanalysis. In *EUROCRYPT 2021*, volume 12696 of *LNCS*, pages 805–835. Springer, 2021. 3, 7

BR20. Emanuele Bellini and Matteo Rossi. Performance comparison between deep learning-based and conventional cryptographic distinguishers. Cryptology ePrint Archive, Report 2020/953, 2020. 3

BRV14. Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential analysis of block ciphers SIMON and SPECK. In *FSE 2014*, volume 8540 of *LNCS*, pages 546–570. Springer, 2014. 6, 7

BS91. Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991. 2

BSS+13. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. 11

BVC16. Alex Biryukov, Vesselin Velichkov, and Yann Le Corre. Automatic search for the best trails in ARX: Application to block cipher Speck. In *FSE 2016*, volume 9783 of *LNCS*, pages 289–310. Springer, 2016. 7, 8

DFJ13. Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved key recovery attacks on reduced-round AES in the single-key setting. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 371–387. Springer, 2013. 5

Din14a. Itai Dinur. Improved differential cryptanalysis of round-reduced Speck. In *SAC 2014*, volume 8781 of *LNCS*, pages 147–164. Springer, 2014. 3

Din14b. Itai Dinur. Improved differential cryptanalysis of round-reduced Speck. Cryptology ePrint Archive, Report 2014/320, 2014. 3, 5, 6, 7, 10, 19, 24, 26, 27, 29

DKS10. Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved single-key attacks on 8-round AES-192 and AES-256. In *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 158–176. Springer, 2010. 5

DSP07. Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved meet-in-the-middle attacks on reduced-round DES. In *INDOCRYPT 2007*, volume 4859 of *LNCS*, pages 86–100. Springer, 2007. 5

FWG+16. Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. MILP-based automatic search algorithms for differential and linear trails for Speck. In *FSE 2016*, volume 9783 of *LNCS*, pages 268–288. Springer, 2016. 7

Goh19. Aron Gohr. Improving attacks on round-reduced Speck32/64 using deep learning. In *CRYPTO 2019*, volume 11693 of *LNCS*, pages 150–179. Springer, 2019. 3, 5, 6, 26, 27, 28

HW19. Mingjiang Huang and Liming Wang. Automatic tool for searching for differential characteristics in ARX ciphers and applications. In *INDOCRYPT 2019*, volume 11898 of *LNCS*, pages 115–138. Springer, 2019. 7, 8, 12

LIM21. Fukang Liu, Takanori Isobe, and Willi Meier. Cryptanalysis of full lowmc and lowmc-m with algebraic techniques. In *CRYPTO 2021*, volume 12827 of *LNCS*, pages 368–401. Springer, 2021. 5

LKK+18. HoChang Lee, Seojin Kim, HyungChul Kang, Deukjo Hong, Jaechul Sung, and Seokhie Hong. Calculating the approximate probability of differentials for ARX-based cipher using SAT solver. *Journal of the Korea Institute of Information Security & Cryptology*, 28(1):15–24, 2018. 6, 7, 32

LLJW19. Zhengbin Liu, Yongqiang Li, Lin Jiao, and Mingsheng Wang. A new method for searching optimal differential and linear trails in ARX ciphers. Cryptology ePrint Archive, Report 2019/1438, 2019. 7, 8

LM01. Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. In *FSE*, volume 2355 of *LNCS*, pages 336–350. Springer, 2001. 9, 10, 38

LMM91. Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *EUROCRYPT*, volume 547 of *LNCS*, pages 17–38. Springer, 1991. 8

Mat94. Mitsuru Matsui. On correlation between the order of S-boxes and the strength of DES. In *EUROCRYPT 1994*, volume 950 of *LNCS*, pages 366–375. Springer, 1994. 7

MP13. Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Report 2013/328, 2013. 7

RST18. Christian Rechberger, Hadi Soleimany, and Tyge Tiessen. Cryptanalysis of low-data instances of full LowMCv2. *IACR Trans. Symmetric Cryptol.*, 2018(3):163–181, 2018. 5

SHY16. Ling Song, Zhangjie Huang, and Qianqian Yang. Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In *ACISP 2016*, volume 9723 of *LNCS*, pages 379–394. Springer, 2016. 6, 7, 30

Supplementary Material

A Differentials

Table 6: Differentials used in this paper. Where existing differentials were not available we used a SAT solver to compute them. **Pr T** is the probability of the best trail, and **Pr D** is the probability of the differential, and both are expressed as $-\log_2(Pr)$.

	ID	r	Δ_{in}	Δ_{out}	Pr T	Pr D	Ref.
Speck32	1	6	0x0211,0x0A04	0x850A,0x9520	13	-	-
	2	6	0x0A20,0x4205	0x8000,0x840A	14	-	-
	3	7	0x0A20,0x4205	0x850A,0x9520	18	17.94	-
	4	8	0x0A20,0x4205	0x802A,0xD4A8	24	23.84	-
	5	8	0x0A60,0x4205	0x802A,0xD4A8	24	23.84	-
	6	8	0x7448,0xB0F8	0x850A,0x9520	24	23.95	-
	7	8	0x7458,0xB0F8	0x802A,0xD4A8	24	23.95	-
	8	9	0x0A20,0x4205	0x01A8,0x530B	31	30.37	-
	9	9	0x8054,0xA900	0x0040,0x0542	30	29.37	-
	10	10	0x2800,0x0010	0x0004,0x0014	35	30.39	[LKK+18]
	11	10	0x7448,0xB0F8	0x00a8,0x520B	37	36.30	-
Speck64	12	8	0x00820200,0x00001202	0x20200000,0x01206008	29	28.87	-
	13	14	0x04092400,0x20040104	0x80008004,0x84008020	56	55.69	-
	14	15	0x04092400,0x20040104	0x808080a0,0xA08481A4	62	60.73	-

B Key Recovery Complexity Graphs

In this appendix, we provide workload graphs for various best MiF attack families on different Speck instances. These graphs show the prediction of the total number of *trail-subkey* pairs visited at each depth (accumulated over all visited branches). Each graphs presents a *family* of attacks for varying values of $c = 1 \ldots 6$. This allows us to clearly illustrate the effect of the counting technique coupled with MiF. We outline briefly the most interesting data on these graphs:

– The starting point for each curve defines the predicted initial number of trails to be suggested by MiF. It is adapted for each c based on the factor c' (see Appendix D).

– The total number of trail-subkey pairs across all (integral) depths except the last one (equal to, roughly, the area under the curve) defines the time complexity T_{enum} of the recursive procedure, up to a complexity coefficient (see Claim 3, Claim 4).
– The final value of the curve defines the number of recovered subkey groups to be tested either using conformance to the trail or by full trial decryptions. This induces the time complexity T_{trials}, again, up to a complexity coefficient (see Claim 2).

Note that the coefficients of T_{enum} and T_{trials} are slightly different, therefore the dominating term is not always clear from these graphs. However, they both contribute to the attack's complexity T_{att}.

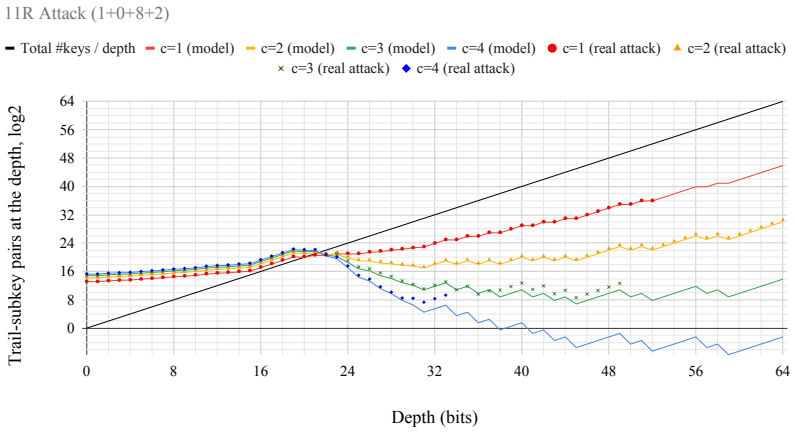

Fig. 6: Time complexity analysis of the fastest attack family on 11-round SPECK32 (see Table 3, attack #5 with $c = 3$). Lines plotted are the predicted numbers of trail-subkey pairs visited per each depth $0 \dots 64$ for attacks with $c = 1, 2, 3, 4$; data points mark data collected from a real experiment with bad trails only (in the case $c = 1$, missing points were not obtained due to complexity limitations; in the cases $c = 3, 4$ missing points signify the absence of survived wrong candidates).

C Memory Optimizations for the Multi-Trail Key Recovery Procedure

– *On-the-fly quick filtering.* In SPECK, due to a round subkey being added only to one branch, a large fraction of suggested trails does not have valid keys for decryption of the associated ciphertext pairs in accordance with the

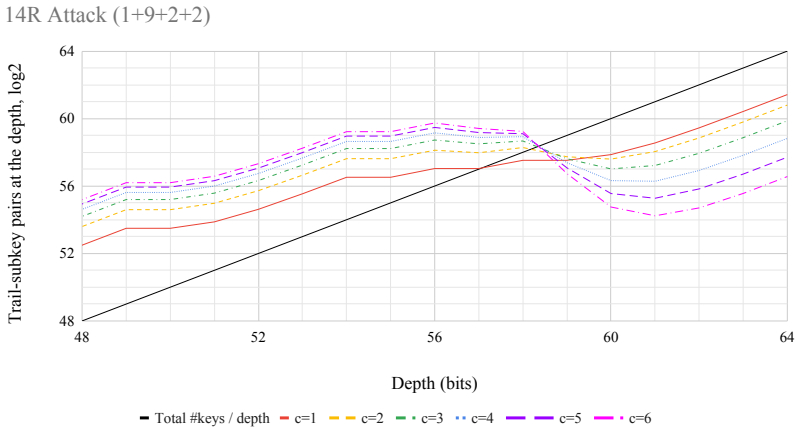

Fig. 7: Time complexity analysis of an attack family on 14-round SPECK32 (see Table 4, #6,#7), zoomed into the last 16 bits of key recovery (the dominating part). Lines plotted are the predicted numbers of trail-subkey pairs visited per each depth $0 \ldots 64$ for attacks with $c = 1, 2, 3, 4, 5, 6$. Note that attacks with $c \geq 3$ require an infeasible amount of data.

trails. Part of this filter can be implemented very efficiently using Dinur's multi-bit filters. For example, in SPECK32, 6-bit filters applied to the last round's transition keep only about 0.25 of all trails.

– *On-the-fly deep filtering.* In our attacks, the MiF backwards filter covers 2 rounds, and these 2 rounds have very high-weight transitions on average. Therefore, checking the existence of 2-round keys would allow filtering out more trails. This can be implemented by running the single-trail recursive procedure up to 2 rounds. Note that this method has negligible time overhead, in contrast to seemingly similar Dinur's initial 2- round subkey guessing. This is due to the availability of the full trail from MiF, allowing search tree cutoffs on each bit level.

– *Larger first recursion step.* The multi-trail procedure keeps a list of trails per each depth level in the recursion. These lists have quickly decreasing sizes (according to Lemma 6, the expected factor per bit of a random differential transition through ADD is $\sqrt[n]{(4/7)^{n-1}} \leq 2^{-0.75}$ for $n \geq 16$. Therefore, the total storage size expansion (compared to the size of the input list of trails) is below the sum of this geometric progression, equal to $1/(1-2^{-0.75}) \approx 2.47$. It can be effectively reduced to 1 by increasing the first recursion step's guess to several bits. This would chop off the heaviest lists of trails on the recursion path. For example, guessing 8 bits instead of 1 would replace the factor

$$1 + 2^{-0.75} + 2^{-1.5} + 2^{-2.25} + \ldots + 2^{-6} + 2^{-6.75} + \ldots \approx 2.47 \tag{26}$$

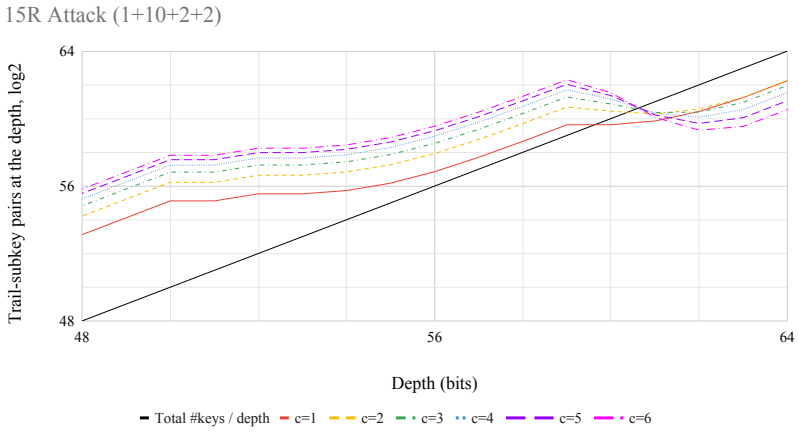

Fig. 8: Time complexity analysis of an attack family on 15-round SPECK32 (see Table 4, #8), zoomed into the last 16 bits of key recovery (the dominating part). Lines plotted are the predicted numbers of trail-subkey pairs visited per each depth $0 \ldots 64$ for attacks with $c = 1, 2, 3, 4, 5, 6$. Note that attacks with $c \geq 2$ require an infeasible amount of data.

by

$$1 + 2^{-6} + 2^{-6.75} + \ldots \approx 1.039. \tag{27}$$

We remark that this step is very similar to Dinur's initial 2-round subkey guessing. However, by guessing a smaller number of bits (which is possible due to the availability of the trail) we can minimize the memory overhead without visibly affecting the time complexity.

– *Compact storage.* In our attacks on SPECK, the backwards filter covers 2 rounds. Due to the Feistel-like structure, input and output differences of 2 rounds of SPECK completely determine the intermediate differences, i.e., the full 2-round trail. Therefore, instead of storing full 4-round trails as required for the key recovery, we could initially store trails in a compressed form: the ciphertext difference ΔC and the cluster difference ΔX. The last 2 rounds of the trail can be recovered due to the aforementioned property of the Feistel structure, and the preceding rounds can be recovered from the cluster.

  Note that the (de)compression overhead on time complexity would be negligible on first depths. At a particular depth, when the size of the list of trails is sufficiently small, all the necessary auxiliary information required to minimize the time complexity can be computed and stored for subsequent computations, causing only a negligible memory overhead.

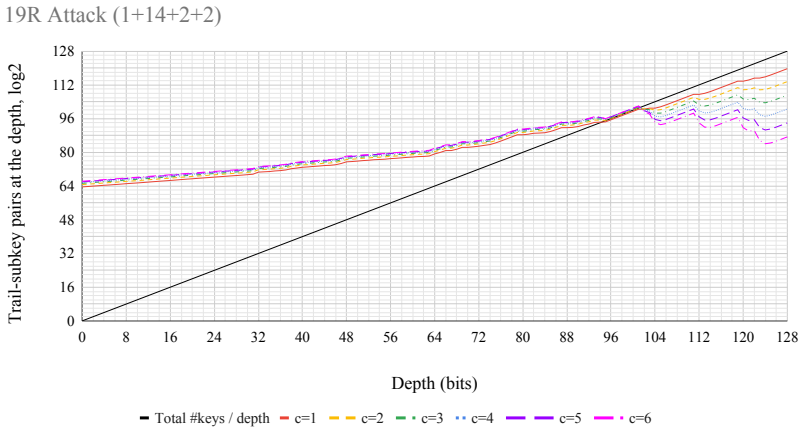

Fig. 9: Time complexity analysis of an attack family on 19-round SPECK64 (see Table 5, #5). Lines plotted are the predicted numbers of trail-subkey pairs visited per each depth $0 \ldots 128$ for attacks with $c = 1, 2, 3, 4, 5, 6$.

D Computing the Required Multiplier for Counting

The binomial distribution converges to the Poisson distribution when the number of trials goes to infinity. The following proposition is essentially given by the Poisson distribution. We derive it explicitly to highlight the approximations used so that the approximation error can be bounded if necessary.

Proposition 1. *Let $q \in \mathbb{R}_+, q \ll 1$ and $c \in \mathbb{Z}, 1 \le c \ll 1/q$. Consider c'/q independent experiments each with the probability of a positive outcome equal to q. The probability to succeed at least c times is equal to (up to a negligible error)*

$$1 - e^{-c'} \sum_{i=0}^{c-1} \frac{(c')^i}{i!}. \tag{28}$$

Proof. The exact probability can be computed by subtracting from 1 the probabilities to succeed strictly less than c times:

$$\Pr[\#\text{successes} \ge c] = 1 - \sum_{i=0}^{c-1} \binom{c'/q}{i} q^i (1-q)^{(c'/q)-i}. \tag{29}$$

Since $i \le c \ll 1/q \le c'/q$, we can use the approximation $\binom{n}{k} \approx \frac{n^k}{k!}$. Since $q \ll 1$, we can approximate $(1-q)^{(c'/q)-i}$ as $e^{-c'}/(1-q)^i \approx e^{-c'}$. After cancelling q^i and moving $e^{-c'}$ outside, the proposition follows.

We now consider the problem of finding the right c' given the target success rate $\tilde{q}$ of at least c positive outcomes. Note that this value is practically independent of q when q is sufficiently large.

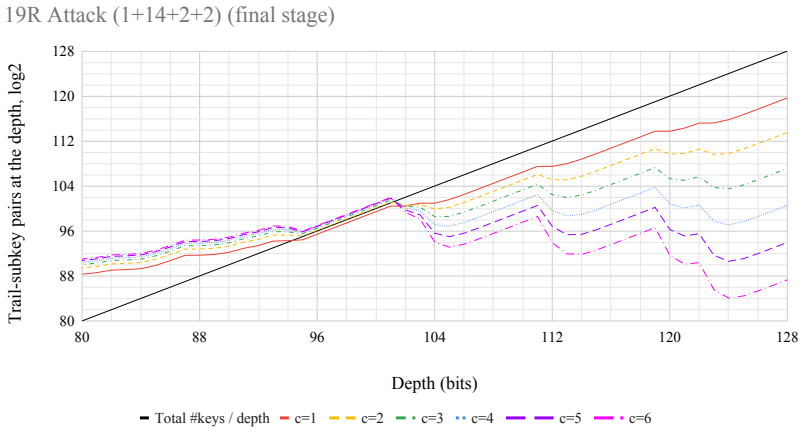

Fig. 10: Time complexity analysis of an attack family on 19-round SPECK64 (see Table 5, #5), zoomed in to the last 48 bits of key recovery (the dominating part). Lines plotted are the predicted numbers of trail-subkey pairs visited per each depth $0 \ldots 128$ for attacks with $c = 1, 2, 3, 4, 5, 6$.

Proposition 2. *Given the target success rate $\tilde{q} \in \mathbb{R}_+$, the required number c'/q of experiments is characterized by the following equation:*

$$\sum_{i=0}^{c-1} \frac{(c')^i}{i!} = e^{c'+b}, \tag{30}$$

where $b = \ln(1 - \tilde{q})$.

Proof. Follows from Proposition 1 by equating (28) to $\tilde{q}$.

Since c' affects the overall success probability in a monotone way, its value can be computed using binary search on the error of the equation (i.e., the difference between the left-hand and the right-hand sides, which is decreasing with increasing c').

E Throwing Balls into Bins

Proposition 3. *Let $\eta \in \mathbb{R}_+, N \in \mathbb{Z}_+, c \in \mathbb{Z}_+$. Consider ηN balls be thrown into N bins, each throw destination chosen independently and uniformly at random. Then, the expected number T_c of balls to have landed into bins with at least c balls in each of the bins is well approximated by:*

$$T_c = \eta N \cdot \left(1 - e^{-\eta} \sum_{i=0}^{c-2} \frac{\eta^i}{i!}\right). \tag{31}$$

Proof. Consider any arbitrary bin. The number of balls in it follows the binomial distribution with $B(M, 1/N)$. In particular, the probability of the bin having exactly c balls is equal to

$$\binom{\eta N}{c} \cdot (1/N)^c \cdot (1-1/N)^{\eta N - c}. \tag{32}$$

For small c and large $N, \eta N$, the probability is very closely approximated by

$$\frac{(\eta N)^c}{c!} \cdot (1/N)^c \cdot (1-1/N)^{\eta N - c} \approx \frac{\eta^c}{c!} \cdot e^{-\eta}. \tag{33}$$

To get the expected number of balls landing in bins with at least c balls in each of them can be computed by subtracting from the total number of balls ηN the amounts of balls landing in bins with $1, 2, \ldots, c-1$ balls (here we use the linearity of expectation, which does not require independence):

$$T_c = \eta N - N \cdot \sum_{i=1}^{c-1} i \cdot e^{-\eta} \cdot \frac{\eta^i}{i!} = \eta N \cdot \left(1 - e^{-\eta} \sum_{i=0}^{c-2} \frac{\eta^i}{i!}\right). \tag{34}$$

F Proofs for Differential Properties of Addition

First, we recall the main results from Lipmaa and Moriai.

Lemma 1 ([LM01, Lemma 3]). *The probability* $\mathbf{xdp}^+(\alpha, \beta, \gamma)$ *is non-zero if and only if*

$$\alpha_i \oplus \beta_i \oplus \gamma_i = \begin{cases} 0 & \text{if} \quad (i = 0)\ , \\ \beta_{i-1} & \text{if} \quad (i \geq 1) \wedge (\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1})\ . \end{cases} \tag{35}$$

Theorem 1 ([LM01, Algorithm 2]). *If* $\mathbf{xdp}^+(\alpha, \beta, \gamma) > 0$ *then*

$$\mathbf{xdp}^+(\alpha, \beta, \gamma) = 2^{-n+l+1}, \ \textit{where} \ \ l = |\{i \in \{0, \ldots, n-2\} : \ \alpha_i = \beta_i = \gamma_i\}|\ . \tag{36}$$

Lemma 4 ([LM01, Theorem 2]). *The fraction of all transitions through ADD (including invalid ones) having weight* w *is given by*

$$\Pr_{\alpha,\beta,\gamma}[\mathbf{xdp}^+(\alpha, \beta, \gamma) = 2^{-w}] = \frac{1}{2}\left(\frac{7}{8}\right)^{n-1} B(w; n-1, \frac{6}{7})\ . \tag{37}$$

We can now derive proofs for the results about $\mathbf{xdp}^+/\mathbf{xdp}^-$ used in the paper.

Lemma 2. *The probability* $\mathbf{xdp}^+(\alpha, \beta, \gamma)$ *is invariant under any permutation of the inputs* α, β, γ*, i.e.,*

$$\mathbf{xdp}^+(\alpha, \beta, \gamma) = \mathbf{xdp}^+(\alpha, \gamma, \beta) = \mathbf{xdp}^+(\beta, \alpha, \gamma) = \ldots\ . \tag{38}$$

Proof. Follows from Lemma 1 and Theorem 1 after observing that the latter two remain valid for any permutation of the bits $\alpha_i, \beta_i, \gamma_i$, $0 \leq i \leq n-1$.

Lemma 3. *The differential $(\alpha, \beta \to \gamma)$ has the same probability through modular addition and modular subtraction for any choice of differences α, β, γ, i.e.,*

$$\mathbf{xdp}^+(\alpha, \beta, \gamma) = \mathbf{xdp}^-(\alpha, \beta, \gamma) \ . \tag{39}$$

Proof. By definition of $\mathbf{xdp}^+/\mathbf{xdp}^-$, we get

$$\begin{aligned}
\mathbf{xdp}^+(\alpha, \beta, \gamma) &= 2^{-2n} \cdot |\{(a, b) : ((a \oplus \alpha) + (b \oplus \beta)) \oplus (a + b) = \gamma\}| \\
&= 2^{-2n} \cdot |\{(c, b) : (((c - b) \oplus \alpha) + (b \oplus \beta)) \oplus c = \gamma\}| \\
&= 2^{-2n} \cdot |\{(c, b) : ((c - b) \oplus \alpha) + (b \oplus \beta) = c \oplus \gamma\}| \\
&= 2^{-2n} \cdot |\{(c, b) : ((c \oplus \gamma) - (b \oplus \beta)) \oplus (c - b) = \alpha\}| \\
&= \mathbf{xdp}^-(\gamma, \beta, \alpha) = \mathbf{xdp}^-(\alpha, \beta, \gamma),
\end{aligned}$$

by using $c = a + b$ and the previous lemma.

Lemma 9. *The total number of valid differential transitions through ADD is given by*

$$|\{\alpha, \beta, \gamma : \mathbf{xdp}^+(\alpha, \beta, \gamma) > 0\}| = 4 \cdot 7^{n-1}. \tag{40}$$

Proof. Follows from Lemma 4 by summing it over all weights $w \le n - 1$ and over all 2^{3n} possible values for (α, β, γ).

Lemma 5. *Let α, β be chosen independently and uniformly at random. The expected number of differences γ such that the differential transition $(\alpha, \beta) \to \gamma$ is valid (i.e., $\mathbf{xdp}^+(\alpha, \beta, \gamma) > 0$) is given by*

$$\mathop{\mathrm{E}}_{\alpha,\beta}[|\{\gamma : \mathbf{xdp}^+(\alpha, \beta, \gamma) > 0\}|] = \left(\frac{7}{4}\right)^{n-1} = 2^{(n-1)\log_2 \frac{7}{4}} \ . \tag{41}$$

Proof. Follows from Lemma 9 by dividing the total number $4 \cdot 7^{n-1}$ of valid transitions by the total number 2^{2n} of input pairs (α, β).

Lemma 6. *Let $(\alpha, \beta) \to \gamma$ be a transition through ADD sampled uniformly at random from all valid transitions through ADD. The average differential transition* probability *p is given by*

$$\mathop{\mathrm{E}}_{\substack{\alpha,\beta,\gamma: \\ \mathbf{xdp}^+(\alpha,\beta,\gamma)>0}} [\mathbf{xdp}^+(\alpha, \beta, \gamma)] = \left(\frac{4}{7}\right)^{n-1} = 2^{(n-1)\log_2 \frac{4}{7}}. \tag{42}$$

Proof. Follows from Lemma 9 by dividing the total number 2^{2n} of input pairs by the total number $4 \cdot 7^{n-1}$ of valid transitions.