Automated Truncation of Differential Trails and Trail Clustering in ARX

Automated Truncation of Differential Trails and Trail Clustering in ARX*

Alex Biryukov[1], Luan Cardoso dos Santos[1], Daniel Feher[1], Vesselin Velichkov[2], and Giuseppe Vitto[1]

[1] University of Luxembourg
{alex.biryukov,luan.cardoso,daniel.feher,giuseppe.vitto}@uni.lu
[2] University of Edinburgh
vvelichk@ed.ac.uk

Abstract. We propose a tool for automated truncation of differential trails in ciphers using modular addition, bitwise rotation, and XOR (ARX). The tool takes as input a differential trail and produces as output a set of truncated differential trails. The set represents all possible truncations of the input trail according to certain predefined rules. A linear-time algorithm for the exact computation of the differential probability of a truncated trail that follows the truncation rules is proposed. We further describe a method to merge the set of truncated trails into a compact set of non-overlapping truncated trails with associated probability and we demonstrate the application of the tool on block cipher Speck64.
We have also investigated the effect of clustering of differential trails around a fixed input trail. The best cluster that we have found for 15 rounds has probability $2^{-55.03}$ (consisting of 389 unique output differences) which allows us to build a distinguisher using 128 times less data than the one based on just the single best trail, which has probability 2^{-62}. Moreover, we show examples for Speck64 where a cluster of trails around a suboptimal (in terms of probability) input trail results in higher overall probability compared to a cluster obtained around the best differential trail.

Keywords: Symmetric-key · Block ciphers · Differential cryptanalysis · Truncated Differentials · ARX · Speck.

1 Introduction

Truncated differential cryptanalysis (TC) is a technique for analysing symmetric-key cryptosystems proposed in [10]. It is a variant of differential cryptanalysis (DC) [3] and has been used successfully against a number of cryptographic algorithms such as IDEA, Skipjack and Salsa20 among others. Similarly to differential cryptanalysis, truncated cryptanalysis traces the propagation of differences through multiple rounds of a cipher. In contrast to DC, TC does not

* This work was supported by the Luxembourg National Research Fund (FNR) projects FinCrypt (C17/IS/11684537) and SP^2 (PRIDE15/10621687/SPsquared).

analyse full but *truncated* differences. A truncated difference is one in which only some of the bits are specified i.e. fixed to given value 0 or 1, while the rest are truncated i.e. not specified. A truncated bit is typically denoted by a $*$ symbol implying that it may take any value.

In differential cryptanalysis a sequence of differences through several rounds of a cipher is called a *differential trail* (or differential characteristic). When only the input and output differences (and not the intermediate differences) of a differential trail are specified the resulting object is called a *differential*. The analogous concepts in truncated differential cryptanalysis are *truncated differential trails* and *truncated differentials*, both being composed of *truncated* differences.

As in DC, the objective of TC is to find a truncated differential (trail) with a sufficiently high probability p over R rounds. The latter is called a *distinguisher* as it distinguishes the cipher from a random permutation, which has probability lower than p. In its most general form, the attack principle of TC is the same as in DC. Namely, the distinguisher is used to attack $R + r$ rounds for some value of r, by guessing the last r round keys, inverting the permutation and checking if the output truncated difference after R rounds matches the one computed after the inversion under the guessed key/s. The success and complexity of a TC attack crucially depends on the ability to find high probability truncated trails and differentials.

ARX (standing for Addition-Rotation-XOR) is a class of cryptographic algorithms designed using three simple arithmetic operations: modular addition, bitwise rotation and XOR. These algorithms are typically easy to describe and implement and are very efficient, especially in software. At the same time they have been notoriously difficult to analyse due to intricate dependencies between the various operations [11]. As a result a significant body of research has been dedicated to the development of tools and techniques for the automated analysis of ARX.

One of the first automated techniques for constructing differential trails for ARX-based designs is due to De Cannière *et al.* [7]. It uses the idea of generalized bit conditions to find collisions in the hash function SHA1. A few related automated techniques have been subsequently proposed by Leurent [12], Stevens [20] and Mendel *et al.* [17]. Similarly, all of them have been applied to hash functions. Dedicated tools for searching for differential paths in (pure) ARX ciphers have been proposed by Liu *et al.* [16], Huang *et al.* [9] and Biryukov *et al.* [5, 6]. Finally, several authors have modelled the differential search problem in terms of Boolean satisfiability or mixed-integer linear programming and have proposed the use of off-the-shelf SAT or MILP solvers to find solutions in an automated way. Some results in this direction are by Mouha *et al.* [18], Fu *et al.* [8], Sun *et al.* [21, 22, 22] and Song *et al.* [19]. The problem of clustering of differential characteristics has been researched in [1, 19, 4], where the authors apply SMT solvers or dedicated tools to enumerate characteristics belonging to a given differential.

In this paper we extend the set of existing tools for analysis of ARX. More specifically, we propose a new automated tool for constructing truncated differ-

ential trails for ARX from existing non-truncated ones and computing their exact probability. The main idea is to truncate every bit from the input non-truncated trail (i.e. transforming all 0 and 1 bits into a $*$), according to certain predefined propagation rules. The rules ensure that the truncated $*$ bit will propagate until the last round of the input trail so that the resulting truncated trail will remain valid and of non-zero probability for *any* assignment of the $*$. As a result, from an input trail we obtain a set of trails represented by a single truncated trail that has probability at least as high as the probability of the initial trail. In addition, we propose a method to construct a cluster of non-overlapping truncated trails composed of all possible truncations of the input (up to the propagation rules) together with its associated probability. In contrast to [1, 19, 4] the trails in the constructed clusters do not necessarily belong to the same differential. They have compact representation due to which the analyst is able to trace the propagation of multiple trails at the same time.

We propose two sets of truncation rules: simple rules (Section 3) and relaxed rules (Section 6). The simple rules do not consider dependencies between consecutive bits within the same round (i.e. within the same modular addition operation). Consequently, with the simple rules, truncated bits with different labels are independent from each other and can take values 0 and 1 with equal probability. In contrast, the relaxed rules are a generalization of the simple rules that is applicable also in cases in which bits within the same round are dependent on each other. In that case truncated bits with different labels are dependent on each other (often in complex ways) and may take values 0 and 1 with different probability.

Both for the simple and for the relaxed truncation rules the only assumption we rely on is the Markov assumption i.e. treating rounds as independent. In particular, we do *not* assume that individual non-truncated trails belonging to the same truncated trail have equal probability. Indeed, in general they don't and this is taken care of by the proposed tool.

The tool is useful for constructing truncated differential distinguishers which have lower data complexity than the traditional ones based on the best non-truncated trail. Its application is demonstrated on block cipher Speck64, for which we report clusters of truncated trails produced from the optimal non-truncated trails on up to 15 rounds. The latter is the highest number of rounds covered by a single trail with probability 2^{-62} higher than random 2^{-64}. For 15 rounds in particular, we report a set of 24 truncated trails, encoding 135 non-truncated trails, the top 22 of which have probability $\geq 2^{-64}$ and cumulative probability $2^{-59.05}$. The latter improves the probability of the single optimal trail by a factor of about 8 at the expense of considering multiple trails. A summary of those results is given in Table 1.

In the context of the existing tools mentioned earlier, the proposed tool bears similarity to the generalized conditions idea introduced in [7] and extended in [12]. Indeed the set of truncated and fixed bits is a subset of the full set of (extended) generalized conditions. Several features set our tool apart from [7, 12]. First, by limiting ourselves to just a very small subset of the generalized

conditions we are able to compute the exact probability of a single truncated trail in linear time in its length. Second, due to the same reason we are also able to transform a set of overlapping truncated trails into a set of disjoint truncated trails. The latter is critical for being able to compute the probability of a cluster of truncated trails, which on its turn is critical in estimating the data complexity of an attack. Finally, ours is a dedicated tool for finding truncated trails, while the mentioned tools have been applied in the context of collision search in hash functions.

Table 1. Truncation of optimal trails for SPECK64. Legend: R number of rounds; Δ_{in} (#) input differences to the ADD at first round (# number of trails with such input); $\#T_{\text{tr}}$ number of truncated trails produced by the tool; $\#T_{\text{ntr}}$ number of non-truncated trails in the truncated cluster T_{tr} (in brackets are the number of trails with $\Pr \geq 2^{-64}$); $P_{\min}$ and $P_{\max}$ resp. minimum and maximum trail Pr in the set T_{ntr} ($\log_2$ scale); P_{tr} total Pr of the truncated cluster ($\log_2$ scale); $\log_2(S/N) = 64 - |\log_2(P_{\text{tr}})| - \log_2(\#T_{\text{ntr}})$; Numbers in brackets in col. 7, 8 based on top trails in T_{ntr} with $\Pr \geq 2^{-64}$. The columns mat.S/N and mat.P_{tr} are the signal-noise and probabilities of the optimal truncation, approximated with a Matsui-search tool, whose probability limit was chosen in such a way as to make computation time feasible on a small scale server PC with a few hours of computation.

R	Δ_{in} (#)	$\#T_{\text{tr}}$	$\#T_{\text{ntr}}$	$P_{\min}$ ($\log_2$)	$P_{\max}$ ($\log_2$)	P_{tr} ($\log_2$)	S/N ($\log_2$)	mat.S/N ($\log_2$)	mat.P_{tr} ($\log_2$)
5	02000012 02000002 (1)	3	20	−15	−10	−7.58	52.10	33.13	−3.70
6	00008202 00001202 (1)	6	48	−23	−15	−12.02	46.40	32.31	−6.46
6	00401042 00400240 (1)	3	20	−20	−15	−12.58	47.10	33.37	−8.92
7	92400040 10420040 (1)	3	40	−27	−21	−18.00	40.68	24.42	−13.10
7	40924000 40104200 (1)	6	48	−29	−21	−18.02	40.40	24.47	−13.05
7	C0924000 40104200 (1)	6	48	−29	−21	−18.02	40.40	24.49	−13.06
8	00008202 00001202 (2)	6	144	−42	−29	−25.00	31.83	20.94	−16.63
8	92400040 10420040 (3)	28	576	−40	−29	−23.37	31.46	21.28	−16.23
8	40924000 40104200 (3)	25	544	−41	−29	−23.40	31.51	21.28	−16.23
9	00008202 00001202 (1)	3	48	−44	−34	−30.65	27.76	20.25	−23.33
9	80240000 00040080 (1)	3	20	−39	−34	−31.58	28.10	20.87	−27.39
9	80208080 00048080 (1)	2	12	−43	−34	−32.24	28.18	20.93	−28.85
9	00802400 80000400 (1)	6	30	−46	−34	−31.58	27.51	20.86	−27.33
10	80208080 00048080 (1)	6	30	−50	−38	−35.58	23.51	20.35	−32.69
11	00000090 00000010 (1)	3	5	−45	−42	−40.75	20.93	20.19	−40.00
12	00000090 00000010 (1)	5	24	−53	−46	−43.60	15.81	11.07	−40.12
12	00008202 00001202 (1)	3	5	−49	−46	−44.75	16.93	11.59	−42.35
13	00008202 00001202 (1)	5	24	−57	−50	−47.60	11.81	10.22	−45.57
14	20200008 20200001 (1)	6	48	−64	−56	−53.02	5.40	5.06	−51.07
14	00008202 00001202 (1)	15	112 (99)	−67	−56	(−52.41)	(4.79)	4.70	−50.68
14	92400040 10420040 (1)	6	24	−63	−56	−53.60	5.81	5.70	−51.37
14	40924000 40104200 (1)	5	24	−63	−56	−53.60	5.81	4.97	−52.06
15	92400040 10420040 (1)	24	135 (22)	−74	−62	(−59.05)	(0.49)	0.37	−58.54
15	40924000 40104200 (1)	15	112 (22)	−73	−62	(−59.05)	(0.49)	0.37	−58.54
15	00040924 20040104 (1)	6	48 (16)	−70	−62	(−59.42)	(0.58)	0.50	−58.79

The outline of the paper is as follows. We begin with preliminaries in Sect. 2, followed by exposition of the rules for truncation in Sect. 3. In Sect. 4 and Sect. 5 is presented respectively a tool for automated truncation of differential trails in ARX and a tool for merging a set of truncated trails into a set of non-overlapping truncated trails. A set of relaxed truncation rules is described in Sect. 6. Results from the application of those tools to block cipher SPECK64 are given in Sect. 7. Statistical analysis of the distinguishing advantage using truncated distinguishers is given in Sect. 8. In Sect. 9 we discuss an improved truncated distinguisher for 15 rounds of SPECK64. The exposition concludes with Sect. 10. Notations and abbreviations are listed in Table 2.

Table 2. Symbols and notation.

Symbol	Meaning
n	Word size in bits
$\boxplus$ or ADD	Addition modulo 2^n
$\lll$, $\ggg$	Left, right bitwise rotation
$\wedge$, $\vee$	Logical AND, OR
$\overline{x}$ or $\neg x$	Logical NOT
$\oplus$	Binary exclusive-OR (XOR)
α, β, γ	n-bit XOR or truncated differences
α_i	The i-th bit of α (α_0 is LSB, α_{n-1} is MSB)
$(\alpha\beta\gamma)_i$	The i-th bits of α, β, γ as 3-bit string
$*$	Truncated bit (can be both 0 and 1)
$\tilde{*}$	Dependent truncated bit (can be both 0 and 1)
$\cdot$	Fixed bit (can be either 0 or 1)
T, τ	Truncated trail
$\mathfrak{T}, \mathfrak{t}$	Sets of truncated trails
$\#\mathfrak{T}$ or $\lvert\mathfrak{T}\rvert$	Size of the set $\mathfrak{T}$
Pr	Probability
DP	Differential probability
S/N	Signal-to-Noise Ratio

2 Preliminaries

In this section we present notations, definitions and theorems that are relevant to the subsequent parts of the paper.

By $\mathbf{xdp}^+$ is denoted the XOR differential probability (DP) of ADD and is defined below.

Definition 1. $\mathbf{xdp}^+$ *is the probability with which input XOR differences* α, β *propagate to output XOR difference* γ *through the operation ADD, computed over*

all n-bit inputs a, b:

$$\mathbf{xdp}^+(\alpha, \beta, \gamma) = 2^{-2n} \ \#\{(a, b) : ((a \oplus \alpha) + (b \oplus \beta)) \oplus (a + b) = \gamma\} \ . \quad (1)$$

The following lemma provides the condition under which $\mathbf{xdp}^+$ is non-zero:

Lemma 1 (Lemma 3 [15]). *The probability $\mathbf{xdp}^+(\alpha, \beta, \gamma)$ is non-zero if:*

$$\alpha_i \oplus \beta_i \oplus \gamma_i = \begin{cases} 0 & \text{if} \quad (i = 0) \ , \\ \alpha_{i-1} & \text{if} \quad (\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}) \wedge (i > 0) \end{cases} \ . \quad (2)$$

Proof. Lemma 3 [15].

The next theorem provides a formula for the computation of $\mathbf{xdp}^+$.

Theorem 1 (Algorithm 2 [15]). *If $\mathbf{xdp}^+(\alpha, \beta, \gamma)$ is non-zero then its exact value is computed according to the following formula:*

$$\mathrm{xdp}^+(\alpha, \beta, \gamma) = 2^{-n+k+1} : \ k = \#\{i \geq 1 : \ (\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1})\} \ . \quad (3)$$

Proof. Algorithm 2 [15].

Theorem 1 essentially states that the probability $\mathbf{xdp}^+$ decreases by a factor of $1/2$ for every bit position i at which the three bits of the differences α_i, β_i and γ_i are *not* equal, excluding the most significant bit (MSB) (hence the $+1$ in the power).

A bit in a truncated differential trail can either be *fixed*, denoted by the dot symbol $\cdot$ or *truncated*, denoted by the star symbol $*$. A fixed bit has value either 0 or 1. A truncated bit can take on both values 0 and 1. More precisely, if a bit in a truncated differential trail is truncated, then the trail is valid (i.e. of non-zero probability) for both assignments of this bit.

3 Rules for Truncation

Truncation is performed according to three simple rules. They make truncation feasible over multiple rounds of a cipher, where the ARX operations are sequentially applied one after another. We describe those rules next, together with the rationale behind them.

Rule 1. Let (α, β, γ) be a differential through ADD. Allow at most one truncated bit in $(\alpha\beta\gamma)_i$ at all bit positions i except the least significant bit (LSB) and allow no truncated bits at the LSB:

$$(\alpha\beta\gamma)_i \in \begin{cases} \{(\cdot\,\cdot\,\cdot)\} & , \ i = 0 \ , \\ \{(\cdot\,\cdot\,\cdot), (\cdot\,\cdot\,*), (\cdot\,*\,\cdot), (*\,\cdot\,\cdot)\} & , \ n > i > 0 \end{cases} \ . \quad (4)$$

The rationale behind Rule 1 is to make truncation feasible over multiple ADD operations iterated in sequence as in an ARX algorithm. If we allow more than one truncated bit per bit position in Rule 1 then the number of $*$ bits quickly explodes in the number of rounds. Consequently it becomes infeasible to keep track of the truncated bits across multiple rounds i.e. to maintain information as to which $*$ bit at round r is related to which $*$ bit/s at round $r-1$. Note that the final goal is to end up with a truncated differential trail which results in non-zero probability non-truncated trail for *any* assignment of the $*$ bits. Finally and most importantly due to Rule 1 it is possible to efficiently (in linear time) compute the differential probability of a truncated differential through a single ADD.

Rule 2. Let (α, β, γ) be input/output differences through XOR so that $\alpha \oplus \beta = \gamma$. Allow at most one truncated bit in $(\alpha\beta)_i$ at all bit positions:

$$(\alpha\beta)_i \in \{(\cdot\cdot), (\cdot *), (* \cdot)\} : \ n > i \geq 0 \ . \tag{5}$$

Similarly to Rule 1, the rationale behind Rule 2 is to make it feasible to keep track of the dependency between $*$ bits over sequences of XOR operations. For example if $\alpha_i = \cdot$ and $\beta_i = *$ then the output of XOR is $\alpha_i \oplus * = *$. Thus the output star $*$ is either equal to the input star $*$ or to its negation depending on the value of α_i which is fixed. In contrast, if both input bits are truncated i.e. $\alpha_i = *$ and $\beta_i = *$ then the output is a $*$ bit that is dependent on the inputs in a (relatively) complex way.

Rule 3. Let (α, β, γ) be a truncated differential through ADD respecting Rule 1. If, at position $i-1$, two bits are fixed and equal while the third is truncated or all three bits are fixed and equal to each other, then all bits at position i must be fixed:

$$\begin{aligned} &((\alpha_{i-1} = \beta_{i-1} = \cdot) \wedge (\gamma_{i-1} = *)) \vee \\ &((\beta_{i-1} = \gamma_{i-1} = \cdot) \wedge (\alpha_{i-1} = *)) \vee \\ &((\alpha_{i-1} = \gamma_{i-1} = \cdot) \wedge (\beta_{i-1} = *)) \vee \\ &(\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1} = \cdot) \implies (\alpha\beta\gamma)_i = (\cdot\cdot\cdot) \ . \end{aligned} \tag{6}$$

Rule 3 is a consequence of the $\mathbf{xdp}^+$ non-zero condition (Lemma 1). It ensures that a non-zero probability differential (trail) remains of non-zero probability for all assignments of the $*$ bits after truncation. More specifically, if e.g. $\alpha_{i-1} = \beta_{i-1} = \cdot$ and $\gamma_{i-1} = *$ then we treat the $*$ value of γ_{i-1} as being equal to α_{i-1} in order to check that this is a valid truncation i.e. that the differential remains of non-zero Pr for both assignments of γ_{i-1}. This is the case only if $\alpha_i \oplus \beta_i \oplus \gamma_i \oplus \alpha_{i-1} = 0$, otherwise the truncation is invalid (cf. Lemma 1).

The described rules allow stars in all bit positions (even several stars per round) and in all rounds, except in the input differences. In practice however a

star at a given position, for example round j, bit i, might violate one of the rules as it propagates to the last round. If that is the case, then bit (j, i) remains fixed. As a result the number of $*$ bits is relatively small. Another related consequence is that more stars appear in the last rounds since at those positions there is smaller chance to break any of the rules.

Rule 1, Rule 2 and Rule 3, when used in combination, make it possible to compute the DP of a truncated differential trail in linear time in the length of the trail.

We note that the proposed rules can be relaxed in several directions. In particular, one may relax Rule 2 by allowing two $*$ bits to enter the XOR operation. Rule 3 can be relaxed to allow a $*$ bit at a position that follows a position with equal fixed bits. Indeed we describe such a set of relaxed rules in Section 6. Such relaxations naturally allow to capture more signal (larger cluster of differential trails) at the expense of added complexity for keeping track of $*$ dependencies across rounds.

4 Differential Trail Truncation

The truncation algorithm takes a non-truncated trail as input and produces as output all its truncated variants that comply to Rules 1, 2 and 3. The input trail can be found by using one of the existing tools mentioned earlier e.g. [16, 9, 6].

Denote the input trail by τ. The i-th bit at round j is denoted τ_i^j for $0 \leq i < n$, $0 \leq j < R$ and it can either be truncated or not. The algorithm explores both possibilities recursively in a depth-first search manner. Once a bit is truncated i.e. $\tau_i^j \leftarrow *$, it is propagated to the last round of the trail. The propagation through the ADD and XOR operations is performed according to Rules 1, 2, 3. Propagation through the bitwise rotation operation is done by simply rotating the $*$ bit by the corresponding rotation amount. If propagation fails for a given bit (i.e. a rule is violated), the algorithm backtracks and explores the next possibility or the next bit position. A pseudocode description of this procedure is given in Appendix A, Algorithm 1.

The differential probability (DP) of a truncated differential trail that follows Rules 1, 2, 3 through a single ADD operation can be computed in linear time. The procedure represents a slight modification of the one for $\mathbf{xdp}^+$ (Theorem 1) and is outlined next.

Let (α, β, γ) be a differential through ADD. In the non-truncated case the probability p of this differential decreases by a factor of $1/2$ for every bit position i at which $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}$ does *not* hold (cf. Theorem 1). The modification of this rule concerns the cases in which there is a $*$ at some bit positions. Let $i-1$ be such a position other than the MSB i.e. $(\alpha\beta\gamma)_{i-1} \in \{(\cdot\cdot *), (\cdot * \cdot), (* \cdot\cdot)\}$ and $i \neq n$. Without loss of generality assume that $(\alpha\beta\gamma)_{i-1} = (* \cdot\cdot)$ i.e. $\alpha_{i-1} = *$. By Rule 3 it is ensured that the bits at the next position are all fixed i.e. $(\alpha\beta\gamma)_i = (\cdots)$. Two cases are possible. In the first case $\beta_{i-1} = \gamma_{i-1}$ and the probability is multiplied by 1 if $\alpha_{i-1} = \beta_{i-1}$ and by $1/2$ if $\alpha_{i-1} \neq \beta_{i-1}$ Therefore the total probability p is multiplied by $1 + 1/2 = 3/2$ in this case. In the second case $\beta_{i-1} \neq \gamma_{i-1}$ and the

probability decreases by 1/2 for both values of α_{i-1}. Thus the total probability p is multiplied by $1/2 + 1/2 = 1$ (i.e. p remains unchanged) in this case. When $i = n$ and there is a $*$ at $i-1$ (MSB), the probability p is multiplied by 2 as the value at the MSB does not change the (non-truncated) DP (cf. Theorem 1).

The differential probability of a truncated trail that follows Rules 1, 2, 3 can be computed in linear time in the length of the trail. Consider a conceptual ARX cipher with round function composed of a single ADD operation followed by a linear part composed of XOR-s and bitwise rotations. Let $\tau = (\tau^0 \dots \tau^{R-1})$ be a truncated differential trail over R rounds such that the differential transition at round $0 \leq j < R$ represents input/output differences to ADD i.e. $\tau^j = (\alpha^j, \beta^j, \gamma^j)$. The DP of a single non-truncated differential $(\alpha^j, \beta^j, \gamma^j)$ at round j can be computed bitwise with bit i conditioned on bit $i-1$ using Theorem 1 as follows:

$$\mathbf{xdp}^+(\alpha^j, \beta^j, \gamma^j) = p_0^j \prod_{i=1}^{n-1} p_i^j \,:\, p_i^j = \mathsf{DP}[(\alpha\beta\gamma)_i^j \mid (\alpha\beta\gamma)_{i-1}^j] \,, \tag{7}$$

where $p_0^j = \mathsf{DP}[(\alpha\beta\gamma)_0^j]$. Therefore, under the Markov assumption, the probability of the trail τ is computed as $\mathsf{DP}[\tau] = \prod_{j=0}^{R-1} p_0^j \prod_{i=1}^{n-1} p_i^j$.

Notice that the probabilities p_i^j in the expression for $\mathsf{DP}[\tau]$ can be computed in any order (for a fixed τ). When computing the DP of a truncated trail τ, we are ordering the terms p_i^j by the dependency of the $*$ bits in consecutive rounds. Then we compute each term for both possible values 0 and 1 of the $*$ bit denoted resp. $(p_i^j)_0$ and $(p_i^j)_1$ and we sum the two products.

For example suppose that $\tau_r^k = *$ for some round k and bit r and that this $*$ bit propagates to subsequent rounds, up to the final one, at positions τ_s^{k+1}, …, τ_t^{R-1}. Suppose also that these are the only $*$ bits in τ. The truncated DP of τ then is computed as:

$$\prod_{\substack{(i,j) \notin \\ \{(k,r),(k+1,s)\dots(R-1,t)\}}} p_i^j \Big((p_r^k)_0 (p_s^{k+1})_0 \dots (p_t^{R-1})_0 + (p_r^k)_1 (p_s^{k+1})_1 \dots (p_t^{R-1})_1 \Big) \tag{8}$$

In equation (8), we are essentially splitting the trail τ into bitwise subtrails, where each subtrail contains $*$ bits that are directly dependent on each other. Note that Rules 1, 2, 3 ensure that there are no dependencies between the $*$ bits belonging to different subtrails. In other words, if a $*$ bit is part of a given subtrail, then it can not be part of other subtrails.

An example 6 round truncated trail on SPECK32 generated with Algorithm 1 is shown in Appendix D, Table 4. It has probability $2^{-11.16}$ and has been produced from an optimal 6 round trail on SPECK32 with probability 2^{-13}. The shown truncated trail has 4 independent $*$ bits and therefore encodes 16 non-truncated trails. The dependency between the $*$ bits is shown in the equivalent label representation in the third column of the table. The 4 independent truncated bits are denoted by the labels `a,b,c,d`. Another example of a strongly truncated trail produced from a suboptimal trail for SPECK32 and encoding 512 non-truncated trails is shown in Appendix D Table 5.

5 Merging of Truncated Trails

In the general case an input non-truncated trail may have more than one possible truncation. Therefore the set of all possible (up to Rules 1, 2, 3) truncated trails produced from a given non-truncated trail by Algorithm 1 may contain duplicate non-truncated trails. In this section we describe a method to transform this set into a set of truncated trails that are disjoint i.e. do not contain any duplicates.

As described earlier, a truncated difference (TD) α represents a set of non-truncated differences defined by the $*$ bit positions in its truncated representation. We say that two TD α and α' are *disjoint*, denoted as $\alpha \cap \alpha' = \emptyset$, if their corresponding sets are disjoint i.e. if they do not have any common non-truncated differences. Note that α and α' are disjoint if there is at least one bit that is fixed and of opposite value in each TD i.e. $\exists i: \ 0 \leq i < n: \ (\alpha_i = \cdot) \wedge (\alpha'_i = \cdot) \wedge (\alpha_i \neq \alpha'_i)$.

If the set represented by α is fully contained in the set represented by α' then we say that α is a *subset* of α' denoted as $\alpha \subset \alpha'$. If α and α' are not subsets of each other and are not disjoint i.e. if $(\alpha \not\subset \alpha') \wedge (\alpha \not\supset \alpha')$ and $\alpha \cap \alpha' \neq \emptyset$ then we say that α and α' are *partially overlapping* (PO). The latter implies that some, but strictly not all, differences that are in α are also in α' and vice versa. Note that if α and α' are PO then there exists at least one bit position i at which $(\alpha_i = *) \wedge (\alpha'_i = \cdot)$ and there exists at least one bit position j at which $(\alpha_j = \cdot) \wedge (\alpha'_j = *)$, where clearly $i \neq j$.

The terms *disjoint*, *subset* and *partially overlapping* have analogous meaning for the cases of truncated differentials through ADD (α, β, γ) and of truncated trails $\tau = ((\alpha^0, \beta^0, \gamma^0), (\alpha^1, \beta^1, \gamma^1) \ldots)$ composed of ADD truncated differentials.

The merging algorithm takes as input a set $\mathfrak{T}$ of truncated trails T that are all pairwise disjoint and a truncated trail τ to be merged with $\mathfrak{T}$. The output is an updated set $\mathfrak{T}$ composed of disjoint truncated trails and containing all (non-truncated) trails from τ that were not initially in $\mathfrak{T}$. For each truncated trail T in $\mathfrak{T}$, the algorithm checks three cases. If τ is already in T i.e. $\tau \subset \mathrm{T}$ then output $\mathfrak{T}$ and terminate. If τ and T are disjoint i.e. $\tau \cap \mathrm{T} = \emptyset$ then move on to the next trail in $\mathfrak{T}$ or add τ to $\mathfrak{T}$ if all trails in $\mathfrak{T}$ have been processed. Finally, if T is a subset of τ i.e. $\mathrm{T} \subset \tau$ or if T and τ are partially overlapping, then split τ into a set of truncated trails $\mathfrak{t}$ (explained below). The set $\mathfrak{t}$ is such that all its elements are pairwise disjoint and each trail from $\mathfrak{t}$ is disjoint to T. With this the procedure is finished for the trail T and moves on to the next trail in $\mathfrak{T}$ where it performs the same steps for each trail in the set $\mathfrak{t}$. The process terminates either when the set $\mathfrak{t}$ becomes empty (i.e. the initial trail τ has been fully absorbed into $\mathfrak{T}$) or when all trails from $\mathfrak{T}$ have been processed, in which case the set $\mathfrak{t}$ is added to $\mathfrak{T}$. In both cases the updated set $\mathfrak{T}$ is returned. Pseudocode description of this procedure is given in Algorithm 2.

A step that needs clarification in the described procedure is how the trail τ is split into pairwise disjoint trails $\mathfrak{t}$ that are also disjoint to T. Let $\mathrm{T} = ((\alpha^0, \beta^0, \gamma^0), (\alpha^1, \beta^1, \gamma^1) \ldots)$ and $\tau = ((a^0, b^0, c^0), (a^1, b^1, c^1) \ldots)$. By design we know that either $\mathrm{T} \subset \tau$ or T, τ: PO. In either case there must be at least one bit position i and round j for which the bit in T is fixed and the same bit in τ

is truncated. Let α_i^j be one such bit i.e. $(\alpha_i^j = \cdot) \wedge (a_i^j = *)$. We construct a new trail τ' by setting a_i^j to the opposite value of α_i^j i.e. $a_i^j = 1 \oplus \alpha_i^j$. Note that this makes τ' disjoint from T since it differs in one fixed bit. We add τ' to $\mathfrak{t}$ and we discard the original trail τ. By doing so we don't lose any information since τ for $a_i^j = \alpha_i^j$ is already in T and τ for the opposite value $a_i^j = 1 \oplus \alpha_i^j$ is in $\mathfrak{t}$. If T and τ happen to differ also in another bit, say $(\beta_l^k = \cdot) \wedge (b_l^k = *)$ then we set a_i^j to the value in T: $a_i^j = \alpha_i^j$ and we set b_l^k to the negated value in T: $b_l^k = 1 \oplus \beta_l^k$. We add this new trail τ'' to $\mathfrak{t}$. Now $\mathfrak{t}$ contains τ' and τ'' which are pairwise disjoint since they differ in a_i^j. At the same time they are also disjoint from T since they differ from T respectively in a_i^j and b_l^k. This procedure is executed iteratively for all positions in which τ is fixed and T is truncated.

In Algorithm 2 there are two nested for-loops – the outer over $\mathfrak{T}$, the inner over $\mathfrak{t}$, where the size of $\mathfrak{t}$ is at most the number of non-truncated trails in τ. Therefore the complexity of the algorithm is quadratic in $\max(\#\mathfrak{T}, \#\mathfrak{t})$, where $\#\mathfrak{T}$ is the number of truncated trails in $\mathfrak{T}$ and $\#\mathfrak{t}$ is the number of non-truncated trails in τ.

6 Relaxed Rules

In this Section we will generalize the truncation rules provided in Section 3 by allowing truncations with dependent truncated bits $\tilde{*}$, i.e. bits whose value depends (non-linearly) on previous bits' assignments and for which Lipmaa-Moriai conditions are automatically satisfied.

Rule 1 naturally generalizes to this setting by allowing at most one dependent truncated bit per bit position except for the LSB. Other truncation rules are as follows.

Rule 4. Let (α, β, γ) be input/output differences through XOR so that $\alpha \oplus \beta = \gamma$. Then

$$\begin{aligned} \alpha\beta_i = (\cdot\cdot) &\implies (\gamma_i = \cdot) \\ \alpha\beta_i = \{(\cdot *), (* \cdot), (**), (\tilde{**}), (*\tilde{*}), (\tilde{**})\} &\implies (\gamma_i = \tilde{*}) \end{aligned}$$

where, in the latter case the dependent truncate bit $\gamma_i = \tilde{*}$ is equal to $\alpha_i \oplus \beta_i$.

Rule 5. Let (α, β, γ) be a truncated differential through ADD respecting Rule 1. If, at position $i-1$ and i two bits are fixed and not equal, then we can freely truncate the remaining third bit at position i:

$$\begin{aligned} \big((\alpha_{i-1} \neq \beta_{i-1} = \cdot) \ \vee \ (\alpha_{i-1} \neq \gamma_{i-1} = \cdot) \ \vee \ (\beta_{i-1} \neq \gamma_{i-1} = \cdot) \big) \ \wedge \\ \big((\alpha_i \neq \beta_i = \cdot) \ \vee \ (\alpha_i \neq \gamma_i = \cdot) \ \vee \ (\beta_i \neq \gamma_i = \cdot) \big) \wedge \big(\gamma_i = \cdot \big) \\ \implies (\alpha\beta\gamma)_i = \{(\cdot\,\cdot *), (\cdot * \cdot), (* \cdot\,\cdot)\} \ . \end{aligned}$$

Rule 5 is a consequence of the $\mathbf{xdp}^+$ non-zero condition (Lemma 1), i.e. there are no conflicts at position i if at position $i-1$ and i two bits are not equal.

Rule 6. Let (α, β, γ) be a truncated differential through ADD respecting Rule 1. If, at position $i-1$, two bits are fixed and equal while the third is truncated, then at position i we allow the truncation of bit γ_i:

$$((\alpha_{i-1} = \beta_{i-1} = \cdot) \wedge (\gamma_{i-1} = \{*, \tilde{*}\})) \vee ((\beta_{i-1} = \gamma_{i-1} = \cdot) \wedge (\alpha_{i-1} = \{*, \tilde{*}\})) \vee$$
$$((\alpha_{i-1} = \gamma_{i-1} = \cdot) \wedge (\beta_{i-1} = \{*, \tilde{*}\})) \implies (\alpha\beta\gamma)_i = (\cdot\,\cdot\,\tilde{*})$$

Rule 6 is a consequence of the $\mathbf{xdp}^+$ non-zero condition (Lemma 1). To make explicit the dependence relation we assume, without loss of generality that $((\alpha_{i-1} = \beta_{i-1} = \cdot) \wedge (\gamma_{i-1} = *))$ and $(\alpha\beta\gamma)_i = \{(\cdot\,\cdot\,\tilde{*})$. Hence the truncated bit γ_i depends on the value of the truncated bit γ_{i-1} as

$$\gamma_i = \ * \cdot (\alpha_{i-1} \oplus \gamma_{i-1}) \oplus \alpha_i \oplus \beta_i \oplus \alpha_{i-1} \tag{9}$$

where $*$ represents a (new) independent truncated bit. Recalling that a transition is impossible if and only if $(\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}) \ \wedge \ (\alpha_i \oplus \beta_i \oplus \gamma_i \oplus \alpha_{i-1}) \ = \ 1$, two alternatives are possible when we expand equation 9:

- $\gamma_{i-1} = \alpha_{i-1}$: then $(\alpha_{i-1} \oplus \gamma_{i-1}) = 0$ and $(\alpha_i \oplus \beta_i \oplus \gamma_i \oplus \alpha_{i-1}) = 0$;
- $\gamma_{i-1} \neq \alpha_{i-1}$: then $(\alpha_{i-1} \oplus \gamma_{i-1}) = 1$ and $\gamma_i = *$, in accordance to Rule 5.

Fixed bits and truncated bit assignments can be modelled as outputs of multivariate Boolean functions $f(x_0, .., x_m) \in \mathbb{F}_2[x_0, ..x_m]$. More precisely, fixed bits $\cdot$ are represented by constant functions $f(x_0, .., x_m) = 0, 1$, an independent $*$ bit can be seen as the output of a degree-one monomial $f(x_0, .., x_m) = x_i$, while $\tilde{*}$ bits correspond in general to non-linear functions, e.g. $f(x_0, .., x_m) = x_0x_1x_2 + x_3 + 1$. Within this model, relaxed truncation rules allow to compute Boolean function $f_{\alpha_i}, f_{\beta_i}, f_{\gamma_i}$ representations of the (fixed or truncated) bits $\alpha_i, \beta_i, \gamma_i$.

However there are some technicalities: i) the Boolean polynomial ring $\mathbb{F}_2[x_0, ..x_m]$ in which truncated bit functions are defined should have enough variables to truncate (independently) all bits in the input trail, hence, in general, we assume $m = \frac{(\#\texttt{of rounds}) \times (\texttt{block-size})}{2} - 1$; ii) when a bit is truncated with a $*$ or we need a new independent $*$ bit like in Rule 6, to ensure independence of their assignments, the corresponding Boolean function representing such truncated bit is set to be equal to a never-used independent variable x_i with $0 \leq i \leq m$.

From now on we will refer to bits by meaning the corresponding Boolean multivariate function representing its assignment. Thus, XOR and AND operations used in above rules naturally map to addition and multiplications of Boolean functions.

An example of truncation using relaxed rules and multivariate Boolean functions for a 6 round trail can be found in Appendix G, Table G.

Differential Probability Computation. Since different bit assignments for truncated bits are represented by unique solutions to a multivariate system of equations, in order to correctly compute the overall accumulated weight of non-truncated trails obtained by expanding a truncated one, we need to compute

the truth tables of each Boolean functions and filter out duplicate solutions. Then, we can compute the weight as usual by using the Lipmaa-Moriai algorithm. This results in an $O(n \cdot 2^{m_0})$ space-time algorithm for computing the accumulated weight of all expanded non-truncated trails given a truncated one where n is the number of unique Boolean functions appearing in the truncated representation and m_0 is the number of independent variables appearing in all such functions (in fact, the working polynomial ring can, without loss of generality, be the smaller $\mathbb{F}_2[x_0, ..x_{m_0}]$).

Implementation and Experimental Results. We implemented the relaxed rules in C++ and we were able to truncate in few seconds all trails reported in Table 1 except 2 cases with 15 rounds due to memory limitations. For some of these input trails, we expanded all obtained truncations and we computed, using the above algorithm, the accumulated weight of each corresponding non-truncated trails cluster. We then identified the best and the worst truncation in terms of its corresponding cluster weight and we computed the accumulated probability of all unique trails given by expanding all truncations available (we note that different truncated trails, when expanded, may overlap in some non-truncated trails). Some experimental results can be found in Table 6.

Table 3. Trail clustering obtained by truncating trails for SPECK64 using relaxed rules. R denotes the number of rounds; Δ_{in} denotes the input difference of the truncated optimal weight trail seed; $\#T_{\text{tr}}$ denotes the number of different truncations obtained; $\#T_{ntr}$ denotes the number of unique non-truncated trails obtained by expanding all $\#T_{\text{tr}}$ truncations; $\#B_{\text{ntr}}$ denotes the number of unique non-truncated trails obtained by expanding the best truncation in terms of cumulative weight; $P_{\min}$ and $P_{\max}$ are the cumulative probability of the worst and best truncated trail among all $\#T_{\text{tr}}$ truncations (here $P_{\max}$ correspond to the cumulative weight of all; P_{ntr} denotes the cumulative weight of all $\#T_{\text{ntr}}$ non-truncated trails; Finally, the last column is the S/N ratio of the best truncated trail, i.e. $S/N = 64 - \log_2(\#B_{\text{ntr}}) + P_{\max}$.

R	Δ_{in}	$\#T_{\text{tr}}$	$\#T_{\text{ntr}}$	$\#B_{\text{ntr}}$	$P_{\min}$ ($\log_2$)	$P_{\max}$ ($\log_2$)	$P_{B_{\text{ntr}}}$ ($\log_2$)	S/N ($\log_2$)
6	00008202 00001202	8823	7437	5589	−15	−10	−9.8	42.5
6	00401042 00400240	3102	2772	2310	−15	−11	−10.9	41.8
9	80240000 00040080	1281	1246	1127	−34	−30	−29.9	23.8
10	80208080 00048080	2745	2670	2415	−38	−34	−33.9	18.7

Comparison with Simple Rules (Sect. 3). The simple truncation rules described in Sect. 3 allow for linear-time computation of the DP of a truncated trail. This comes at the expense of missing much of the signal (many trails are not captured by the truncation). In comparison, the relaxed rules capture a significantly larger portion of the signal, but the computation of the DP in this case has exponential

complexity in the number of truncated bits. Therefore one may use either the simple or the advanced rules depending on the available computational resources.

7 Application to Speck64

SPECK is a family of lightweight block ciphers proposed in [2]. The family has five members corresponding resp. to the block sizes $32, 48, 64, 96$ and 128 bits and denoted by SPECKN, where $N/2$ is the word size in bits. In the remaining part of this exposition we shall be concerned with SPECK64 i.e. the variant with 32-bit words. SPECK64 has two variants: 96-bit key and 26 rounds, and 128-bit key and 27 rounds.

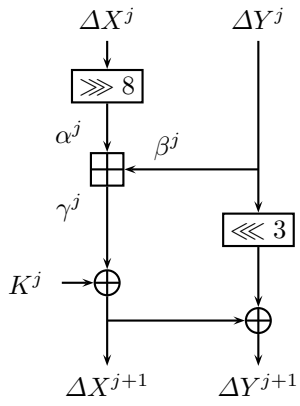

Fig. 1. The round function of SPECK64 with differential inputs.

Denote by X^j and Y^j the left and right 32-bit input words to the j-th round of SPECK64 ($0 \le j \le R$) and by K^j the 32-bit round key applied at round j ($0 \le j < R$). The output X^{j+1}, Y^{j+1} from round j is computed as follows:

$$X^{j+1} = ((X^j \ggg 8) \boxplus Y^j) \oplus K^j \ , \tag{10}$$

$$Y^{j+1} = (Y^j \lll 3) \oplus X^{j+1} \ , \tag{11}$$

where $\boxplus$ denotes addition modulo 2^n for $n = N/2 = 32$. The round function of SPECK64 with differential inputs is shown in Fig. 1.

We have applied the tool for automated truncation (Algorithm 1) and merging of truncated trails (Algorithm 2) to the optimal (non-truncated) differential trails of SPECK64 for up to 15 rounds. The results are shown in Table 1. Explanation and analysis of the data in the table follows.

The first column of Table 1 gives the number of rounds R. The second column shows the input difference Δ_{in} of the input optimal trail followed by the number of such trails with this input difference (in brackets). From the input trail/s [3], a set of $\#T_{\text{tr}}$ non-overlapping truncated trails (column 3) is computed by applying all possible truncations (up to Rules 1, 2, 3) and merging them. The set T_{tr} contains $\#T_{\text{ntr}}$ number of distinct non-truncated trails (column 4) with probabilities ranging from $P_{\max}$ and $P_{\min}$ (columns 5, 6). The total probability of the truncated set T_{tr} is P_{tr} (column 7). The last column of Table 1 shows the $\log_2$ of the signal-to-noise ratio (S/N) computed as $\log_2(S/N) = 64 - |\log_2(P_{\text{tr}})| - \log_2(\#T_{\text{ntr}})$ (we elaborate further on this parameter below).

[3] There can be more than one input trail, provided that they share the same input difference

Numbers in brackets in the $\#T_{\text{ntr}}$ column show the number of trails in the set T_{ntr} that have $\Pr \geq 2^{-64}$ (the probability of a random output difference). Correspondingly, the numbers in brackets in the last two columns are based on this subset of trails of T_{ntr} (as opposed to the full set T_{ntr}).

The S/N ratio shown in the last column of Table 1 is the ratio between the probability of the truncated set distinguisher P_{tr} and the probability of choosing at random a ciphertext difference that belongs to the set T_{ntr}: $\#T_{\text{ntr}} \cdot 2^{-64}$. Note that all ciphertext differences composing the distinguisher are unique. To ensure this, trails with the same output difference are "merged" in one and their probabilities are summed. The S/N ratio is an indicator of the strength of the truncated differential set distinguisher. In particular, when $S/N > 1$ the distinguisher can be used to distinguish the cipher from a random permutation.

The data in Table 1 indicates that the probability of the truncated differential set P_{tr} is strictly higher than the probability of the underlying optimal non-truncated trail $P_{\max}$. Consequently a truncated differential set distinguisher built around the optimal non-truncated trail is better than just single optimal trail in most cases (see next) in terms of data complexity.

The above conclusion has to be applied with caution. In particular, one has to be careful when the probability of the truncated distinguisher approaches the probability of the random event i.e. when $\Pr_{\text{tr}} \approx \#T_{\text{ntr}} \cdot 2^{-64}$ as then the S/N can easily drop below 1. This indeed happens in the case of the 15 round truncated distinguishers for SPECK64 (see Table 1). If the full truncated sets are used as distinguishers in those cases, the corresponding three S/N ratios are $2^{-1.42}$, $2^{-1.21}$ and $2^{-0.60}$ all of which are below 1. To increase them, one has to consider only those non-truncated trails from the sets that have probability $\geq 2^{-64}$. For the three 15 round distinguishers from the table, these are the top 22, 22 and 16 trails respectively (as indicated by the numbers in brackets in the T_{ntr} column). By discarding all trails with $\Pr < 2^{-64}$ in those cases, the S/N ratios are increased respectively to $2^{0.49}$, $2^{0.49}$ and $2^{0.59}$ as shown in the table.

Another observation from the data in Table 1 is that some input trails have higher truncation rate (more number of truncated bits) than others. The reason for this is the specific structure of the trails with respect to Rules 1, 2 and 3. More specifically, for some trails the rules are contradicted in smaller number of bit positions (higher truncation rate) than in others.

For example from the input trail on 11 rounds, 3 truncated trails are produced containing (only) 5 non-truncated ones. At the same time the first input trail on 12 rounds (starting with the same input difference as the 11 round one) is truncated into a set of 5 truncated trails containing 24 non-truncated. Upon inspection we could see that the trail on 11 rounds has a very *thick* (i.e. low probability) transition at round 5 (counting from 0) that costs 2^{-13}, followed by *thin* (i.e. high probability) transitions until the end. So one explanation of the mentioned effect is that the thick transition breaks all rules up to round 5, while the following thin transitions don't offer many options for truncation. Interestingly the trail on 12 rounds is an extension of the one on 11 and the better truncation rate there is due to the extra round added at the end.

In the following section we provide a more detailed statistical analysis of the distinguishing advantage of distinguishers built from clusters of differential trails.

8 Distinguishing Advantage

Distinguishing from Random In this section we provide a probabilistic model for distinguishers for SPECK built from clusters of trails. In this setting, the attacker does some pre-processing by analysing the cipher (i.e. collects trails, computes their differential weights and clusters them by weights) and then queries an oracle black-box, that can either be a `speck-box`, which returns SPECK encryptions for a uniformly chosen key, or a `random-box`, which returns random values.

Assume that in the pre-processing phase the attacker has collected **disjoint** clusters of trails $\{C_i\}_{i=0,\dots,l}$ where C_i has weight w_i (i.e. probability 2^{-w_i}) so that a random trail belongs to it with probability p_{C_i}.

Thus, if we're in

- `random-box` then $p_{C_i} = \frac{|C_i|}{2^{64}}$;
- `speck-box` then $p_{C_i} = |C_i| \cdot 2^{-w_i} = |C_i| \cdot 2^{-(w_0+i)}, i = 0, 1, \dots, l$, where w_0 is the weight of the best differential trail.[4] We consider trails with weight increasing from the optimal one. Thus we express trail weights w_i as $w_i = w_0 + i$ where w_0 is the optimal weight and i ranges from 0 to a given bound.

The probability p to hit at least 1 ciphertext in a collection of clusters after 2^N queries to the oracle is then equal to

$$p = 1 - \Pr(\text{none of the ciphertexts is in any cluster}) = 1 - (1 - \sum_{i=0}^{l} p_{C_i})^{2^N}$$

By approximating $\left(1 - \frac{1}{x}\right)^n \approx e^{-n/x}$, we can rewrite this probability as

$$1 - \left(1 - \sum_{i=0}^{l} p_{C_i}\right)^{2^N} = 1 - \left(\frac{1}{e}\right)^{2^{N+\log_2\left(\sum_{i=0}^{l} p_{C_i}\right)}} = 1 - \left(\frac{1}{e}\right)^{2^{N+k}}$$

Where for the `speck-box` we have

$$k_{speck} \doteq \log_2\left(\sum_{i=0}^{l} p_{C_i}\right) = -w_0 + \log_2\left(\sum_{i=0}^{l} |C_i| \cdot 2^{-i}\right)$$

while for the `random-box` we have

$$k_{rand} \doteq \log_2\left(\sum_{i=0}^{l} p_{C_i}\right) = \log_2\left(\sum_{i=0}^{l} |C_i|\right) - 64$$

[4] In practical attacks the differential effect would increase these probabilities and make the distinguisher better.

It follows that if the attacker wants to hit with probability p a ciphertext in any cluster, he then needs to make 2^N queries, where N is equal to

$$N = \log_2(-\log(1-p)) - k$$

and k is either k_{speck} or k_{rand} depending on his guess for the oracle box.

Note that both models `random-box` and `speck-box` are similar and differ only in their terms k_*: in fact, the more these two values differ, the easier would be to distinguish points belonging to one model or the other. We then define

$$k_{speck} - k_{rand} \;=\; -w_0 + \log_2\left(\sum_{i=0}^{l} |C_i| \cdot 2^{-i}\right) - \log_2\left(\sum_{i=0}^{l} |C_i|\right) + 64 = S/N$$

(which corresponds to the S/N definition we introduced in previous Sections) and the higher this value is, the better we distinguish the two boxes. We assume that there exists at least one trail of weight less than 64 for the reduced Speck64 (i.e. $w_0 < 64$) and in order for the distinguisher to work we require $S/N > 1$ thus:

$$S/N_l \;\doteq\; 64 - w_0 + \log_2\left(\sum_{i=0}^{l} |C_i| \cdot 2^{-i}\right) - \log_2\left(\sum_{i=0}^{l} |C_i|\right) > 1$$

From this inequality and given the histogram of cluster sizes $|C_i|$ we can derive the optimal l up to which we can grow our collection of signal ciphertexts. The main criteria for the attacker is to minimize the amount of data for the distinguisher, i.e. minimizing $N = \log_2(-\log(1-p)) - k_{speck}$ (which is equivalent to maximizing the collection weight l), while keeping $S/N = k_{\text{speck}} - k_{\text{rand}} > 1$. The larger the gap $64 - w_0$ the higher l the attacker can afford.

Statistical Distinguisher. Here we provide a statistical test to distinguish with a certain confidence level α if the queried box is the `random-box` or the `speck-box`. We can model our experiments using geometric distribution of parameter p, i.e. $X_{rand}, X_{speck} \approx Geo(p)$. [5]

The two statistical alternative hypothesis can be then formulated as follows:

- H_0: $p = p_{speck} = \sum_i |C_i| \cdot 2^{-w_i}$, i.e. encryptions come from the `speck-box`.
- H_1: $p = p_{rand} = \sum_i |C_i| \cdot 2^{-64}$, i.e. encryptions come from the `random-box`.

Given a certain confidence level α (e.g. $\alpha = 0.05$) we want to compute a threshold t_α so that if the first matching ciphertext is found after $X_* = 2^N$ encryptions, we accept H_0 if $2^N \leq t_\alpha$, otherwise if $2^N > t_\alpha$ we accept H_1. We then have the following

$$\begin{aligned}\Pr(\text{Reject } H_0 \,|\, H_0) \;&=\; \Pr(X > t_\alpha \,|\, p = p_{speck}) \;=\; \sum_{k=t_\alpha+1}^{\infty} (1-p_{speck})^{k-1} p_{speck} \\ &= (1-p_{speck})^{t_\alpha} p_{speck} \cdot \sum_{l=0}^{\infty} (1-p_{speck})^{l} \;=\; (1-p_{speck})^{t_\alpha}\end{aligned}$$

[5] For Speck, this is a consequence of the assumed Markov assumption.

By requiring $\Pr(\text{Reject } H_0 \,|\, H_0) \leq \alpha$, we have at least

$$t_\alpha = \left\lceil \frac{\ln \alpha}{\ln (1 - p_{speck)}} \right\rceil$$

Thus, given such threshold t_α, the probability of accepting H_0 while being in H_1 would then be equal to

$$\Pr(\text{Accept } H_0 \,|\, H_1) \;=\; \Pr(X \leq t_\alpha \,|\, p = p_{rand}) \;=\; 1 - (1 - p_{rand})^{t_\alpha}$$

Best distinguishing confidence level α. We are interested in achieving the highest distinguishing power possible within the statistical model outlined above. In practice, for a given distinguisher with probabilities p_{rand}, p_{speck}, we would like to choose a confidence level α which maximizes

$$f(\alpha) = \Pr(\text{Accept } H_0 \,|\, H_0) - \Pr(\text{Accept } H_0 \,|\, H_1)$$

where we assume at least $f(\alpha) > 0$. By expanding this definition, we get

$$f(\alpha) = (1 - p_{rand})^{t_\alpha} - (1 - p_{speck})^{t_\alpha} = (1 - p_{rand})^{\frac{\ln \alpha}{\ln (1 - p_{speck})}} - \alpha = \alpha^c - \alpha$$

where $c = \frac{\ln(1-p_{rand})}{\ln(1-p_{speck})}$. So, a solution for $f'(\alpha) = 0$, would then be $\alpha = (\frac{1}{c})^{\frac{1}{c-1}}$ and this is a local maximum if $f''(\alpha) = c \cdot (c-1)\alpha^{c-2} < 0$. Thus, since $c > 0$, $\alpha = c^{\frac{1}{1-c}}$ is a local maximum when $c < 1$ or, equivalently, when $p_{speck} > p_{rand}$.

Experimental verification. We have experimentally verified the above probabilistic and statistical model. More precisely, we have run a distinguishing attack on Speck32 reduced to 9 rounds. We have collected a cluster of differentials with the input difference (`0211, 0a04`) with a cumulative probability of at least $2^{-25.4}$. These were gathered by first finding optimal full trails until weight 32, of which there were 30 unique input/output pairs, and then calculating all the possible trails until weight 40 on these input/output pairs to accommodate for the differential effect. This resulted in a S/N ratio of 1.7. Then, using the formula from the previous section we can calculate the highest distinguishing α, which in this case is $\alpha = 0.1825$, which results in the threshold $t_\alpha = 2^{26.16}$. Using t_α we can also calculate the probability of false positives for the random permutation box given t_α samples, i.e. $\Pr(\text{Accept } H_0 \,|\, H_1)$, which is equal to 0.408. The distinguishing gap is $0.8175 - 0.408 = 0.4095$ which is clearly significant.

In our experiment we have used two boxes (Speck32 with 9 rounds and the random permutation). For the test with the Speck box, we encrypt t_α random input pairs with the fixed input difference, and record a success if any of the output differences is equal to one specified by our differentials. For the random box we use Speckey32 with 40 rounds (as that should emulate a random permutation), and similarly we encrypt t_α pairs of samples with the same input difference, and record a success if any of the two differences are in our set of output differences.

We ran both the Speck32 and the random experiment 1000 times, and received 892 successes for Speck32 (hinting at an even higher real differential probability), and 404 successes for the random variant, verifying our statistical model.

9 Best Distinguisher for Speck64

In this Section we will discuss the best distinguisher we have found for 15 rounds of SPECK64. Aiming at finding the most suitable one, we considered 4 optimal trails of weight -62 found with Matsui's search with input differences $\Delta_0 = (\Delta x_0, \Delta y_0)$ equal to (`40004092, 10420040`), (`04092400, 20040104`), (`92400040, 40104200`), (`924000c0, 40104200`), respectively.

Given an optimal 15 rounds trail, we split it in $n + k$ rounds; by iteratively setting $k = 3, 4, 5$ we compute the best feasible approximation of the differential probability for the first n rounds while maximizing the S/N ratio obtained from freely varying the difference transitions in the last k rounds.

Since computing the differential probability over $n = 12, 11, 10$ rounds (depending on the value of k set) quickly becomes prohibitive as the minimum trail weight limit decreases, we split the first n rounds in two chunks of j and $n - j$ rounds, respectively. Hence, by iteratively setting $j = 3, \ldots, n - 3$ we independently compute the two differential probabilities of these two chunks and we select the best index j so that $\Pr(\Delta_0 \to \Delta_j) + \Pr(\Delta_j \to \Delta_n)$ is minimum.

In order to approximate the differential probability of the two sub-trails $\Delta_0 \to \Delta_j$ and $\Delta_j \to \Delta_n$, we use an SMT solver to find all trails with such input/output differences and weight exceeding at most -25 with respect to their optimal weight.

The trail that performed better within this framework is the one we report in Appendix F Table 7 with parameters $k = 3$, $n = 12$ and $j = 3$. More precisely, for the differential $\Delta_0 \to \Delta_3$ we found 6 trails of total probability -10.954, while for $\Delta_3 \to \Delta_{12}$ we found 21022 trails of total probability -37.418. Thus

$$\Pr(\Delta_0 \to \Delta_{12}) \geq 2^{-48.372}$$

We then proceed by collecting all possible $k = 3$ rounds trails with input difference equal to Δ_{12} and weight less equal -12, as long as the total S/N remains greater than 0: for $\Delta_{12} = (\texttt{00080000}, \texttt{00080000})$ we obtained 389 unique Δ_{15} of weight at least -16 with total weight -6.731 and $S/N = 0.361$. We further slightly improve the total weight to -6.657 by computing the differential probability of each 3 round trail found $\Delta_{12} \to \Delta_{15}$.

This, gives us a distinguisher of probability $p_{speck} = 2^{-48.372-6.657} = 2^{-55.029}$ consisting of 389 unique ciphertexts.

Given that $p_{rand} = \frac{389}{2^{64}} = 2^{-55.396}$, in light of the previous section, we obtain the best confidence level $\alpha = c^{\frac{1}{1-c}} = 0.322$ with $c = \frac{\ln(1-p_{rand})}{\ln(1-p_{speck})} = 0.775$ which in turn correspond to a distinguishing threshold $t_\alpha = 2^{55.576}$ and distinguishing gap of 0.094, small but non-negligible.

Best cluster around sub-optimal trail. It is natural to use the best trail as a starting point for a trail cluster. However one may wonder if it always produces the cluster with the highest total probability for a given S/N ratio. Interestingly the answer is "no". In some cases the cluster around sub-optimal trail will have higher distinguishing power than the one starting from the best trail. In Appendix C we show this behaviour for clusters collected for SPECK64 reduced to 11 and 14 rounds. The plots showing this behaviour are in Figures C and 3. For 11 rounds there is one best trail and there are numerous sub-optimal trails with better clusters. For 14 rounds there are three best trails and there are two sub-optimal trails which are better than two of them and very close to the very best cluster. For 15 rounds sub-optimal trails always have weaker clusters but the four available best trails differ significantly. This fact has helped us to build the best distinguisher for 15 rounds SPECK64 described above.

These results were obtained by exploring sub-optimal trails up to certain weight bound beyond the best trail. This analysis shows that when deciding on a number of rounds of a cipher it might be important to consider not only the best differential, but the best differential cluster.

10 Conclusion

In this paper we described a new tool for the automated truncation of differential trails in ARX. The tool generates all possible truncations of an input non-truncated trail (up to certain pre-defined rules) and outputs a set of non-overlapping truncated trails with associated probability. The latter is strictly greater than the probability of the input trail. The proposed tool is useful for constructing truncated differential distinguishers which have lower data complexity than the traditional ones based on the best non-truncated differential. Interestingly, in some cases differential cluster around sub-optimal trail gives better resulting distinguisher then when starting from the best trail.

The application of the tool was demonstrated on block cipher SPECK64. More specifically, truncated differential set distinguishers based on the optimal trail/s on up to 15 (out of 24) rounds were reported. A natural future direction is the application of the tool to other ARX algorithms. Beside other ciphers, the tool could potentially be used in the area of ARX-based hash functions and sponge permutations. In particular, it may be worth exploring its use in an initial preprocessing phase that would facilitate the subsequent application of advanced collision search tools such as e.g. [20, 17, 13, 14].

11 Acknowledgements

The authors thank the anonymous reviewers for their time and for the insightful comments and corrections. Luan Cardoso dos Santos was supported by the Luxembourg National Research Fund project SP^2 (PRIDE15/10621687/SPsquared). Daniel Feher and Giuseppe Vitto were supported by the Luxembourg National Research Fund project FinCrypt (C17/IS/11684537).

Alex Biryukov, Luan Cardoso dos Santos and Vesselin Velichkov have significant contribution to the sections on the simple rules for truncation (Sect. 3), probability computation using simple rules (Sect. 4) and merging of differential trails produced with simple rules (Sect. 5). Daniel Feher and Giuseppe Vitto have significant contribution to the sections on relaxed rules for truncation (Sect. 6), the statistical analysis of the distinguishing advantage (Sect. 8) and on the best distinguisher for Speck64 (Sect. 9).

References

1. Ankele, R., Kölbl, S.: Mind the gap - A closer look at the security of block ciphers against differential cryptanalysis. In: Cid, C., Jr., M.J.J. (eds.) Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11349, pp. 163–190. Springer (2018). https://doi.org/10.1007/978-3-030-10970-7_8, https://doi.org/10.1007/978-3-030-10970-7_8
2. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404
3. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology **4** (1991)
4. Biryukov, A., Roy, A., Velichkov, V.: Differential Analysis of Block Ciphers SIMON and SPECK. In: Cid, C., Rechberger, C. (eds.) FSE 2014. Lecture Notes in Computer Science, vol. 8540, pp. 546–570. Springer (2014). https://doi.org/10.1007/978-3-662-46706-0_28, http://dx.doi.org/10.1007/978-3-662-46706-0_28
5. Biryukov, A., Velichkov, V.: Automatic search for differential trails in ARX ciphers. In: CT-RSA (2014)
6. Biryukov, A., Velichkov, V., Corre, Y.L.: Automatic search for the best trails in ARX: application to block cipher speck. In: FSE (2016)
7. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: ASIACRYPT (2006)
8. Fu, K., Wang, M., Guo, Y., Sun, S., Hu, L.: MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck. FSE (2016)
9. Huang, M., Wang, L.: Automatic tool for searching for differential characteristics in ARX ciphers and applications. In: INDOCRYPT (2019)
10. Knudsen, L.R.: Truncated and higher order differentials. In: FSE (1994)
11. Leurent, G.: Analysis of Differential Attacks in ARX Constructions. In: ASIACRYPT (2012)
12. Leurent, G.: Construction of Differential Characteristics in ARX Designs - Application to Skein. IACR Cryptology ePrint Archive **2012**, 668 (2012)
13. Leurent, G.: Construction of Differential Characteristics in ARX Designs Application to Skein. In: CRYPTO (2013)
14. Leurent, G., Peyrin, T.: SHA-1 is a shambles: First chosen-prefix collision on SHA-1 and application to the PGP web of trust. In: USENIX (2020)
15. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In: FSE (2001)
16. Liu, Z., Li, Y., Jiao, L., Wang, M.: A new method for Searching Optimal Differential and Linear Trails in ARX Ciphers. Cryptology ePrint Archive, Report 2019/1438

17. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In: ASIACRYPT (2011)
18. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Inscrypt (2011)
19. Song, L., Huang, Z., Yang, Q.: Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In: ACISP (2) (2016)
20. Stevens, M.: New collision attacks on SHA-1 based on optimal joint local-collision analysis. In: EUROCRYPT (2013)
21. Sun, S., Hu, L., Wang, M., Wang, P., Qiao, K., Ma, X., Shi, D., Song, L., Fu, K.: Towards finding the best characteristics of some bit-oriented block ciphers and automatic enumeration of (related-key) differential and linear characteristics with predefined properties. Cryptology ePrint Archive, Report 2014/747
22. Sun, S., Hu, L., Wang, M., Yang, Q., Qiao, K., Ma, X., Song, L., Shan, J.: Extending the applicability of the mixed-integer programming technique in automatic differential cryptanalysis. In: ISC (2015)

A Differential Trail Truncation Algorithm

Algorithm 1 Truncation of Differential Trails in ARX

Input:
$i: \ 0 \leq i < n$: bit position; $j: \ 1 \leq j < R$: round position
τ: non-truncated trail on R rounds, where τ_i^j is the i-th bit at round j
Output:
$\{\tau\}$: all truncations of τ that follow Rule 1, Rule 2 and Rule 3

procedure truncate_trail
if $j < R$ **then**
  for `truncate` $=$ `true`, `false` **do**
    // truncate bit τ_i^j
    if `truncate` $=$ `true` and $\tau_i^j \neq *$ **then**
      Truncate bit $\tau_i^j \leftarrow *$ and propagate to rounds $j+1, \ldots, R-1$
      if Rules 1,2 and 3 are not violated for any round **then**
        Update τ with $\tau^j, \tau^{j+1} \ldots \tau_j^{R-1}$
        Call **truncate_trail** for next bit $i+1$ or next round $j+1$
    // do not truncate τ_i^j: move to next bit
    if `truncate` $=$ `false` **then**
      Call **truncate_trail** for next bit $i+1$ or next round $j+1$
else
  // Last round: return a truncated version of τ
  return τ

B Trail Absorption Algorithm

Algorithm 2 Absorb a new TD trail $\boldsymbol{\tau}$ into exisiting set of trails $\mathfrak{T}$

Input:
$\mathfrak{T}$: set of disjoint TD trails; $\boldsymbol{\tau}$: new TD trail (possibly $\boldsymbol{\tau} \in \mathfrak{T}$)
Output:
$\mathfrak{T}'$: updated set of disjoint TD trails that contains all new (non-truncated) trails from $\boldsymbol{\tau}$ (possibly $\mathfrak{T} = \mathfrak{T}'$)

procedure tdiff_absorb_new_trail($\mathfrak{T}, \boldsymbol{\tau}$)
// initialize a set $\mathbf{t}$ of TD trails with the input trail $\boldsymbol{\tau}$
$\mathbf{t} \leftarrow \emptyset$; add $\boldsymbol{\tau}$ to $\mathbf{t}$
for all $\mathbf{T} \in \mathfrak{T}$ **do**
    if $\mathbf{t} = \emptyset$ **then**
        // all trails in $\mathbf{t}$ have been fully absorbed; return
        return $\mathfrak{T}$
    // absorb $\mathbf{t}$ into $\mathbf{T}$ and store the remainder in $\mathbf{t}'$
    $\mathbf{t}' \leftarrow \emptyset$
    for all $\boldsymbol{\tau} \in \mathbf{t}$ **do**
        // if $\boldsymbol{\tau}$ contains trails not already in $\mathbf{T}$, then split $\boldsymbol{\tau}$ into TD trail subsets to exclude duplicates using
        if $(\mathbf{T} \subset \boldsymbol{\tau}) \vee (\mathbf{T}, \boldsymbol{\tau} : \mathsf{PO})$ **then**
            $\mathbf{t}_{\text{temp}} \leftarrow$ **tdiff_madd_trails_make_disjoint($\mathbf{T}, \boldsymbol{\tau}$)**
            add $\mathbf{t}_{\text{temp}}$ to $\mathbf{t}'$
        // if $\boldsymbol{\tau}, \mathbf{T}$: disjoint, then all trails in $\boldsymbol{\tau}$ are new, so add it
        if $(\boldsymbol{\tau}, \mathbf{T})$: disjoint **then**
            add $\boldsymbol{\tau}$ to $\mathbf{t}'$
        // if $\boldsymbol{\tau}$ is a subset of $\mathbf{T} \implies$ it contains no new trails, so do nothing
        if $(\boldsymbol{\tau} \subset \mathbf{T})$ **then**
            continue
    // overwrite $\mathbf{t}$ with the part of it that was not absorbed i.e. $\mathbf{t}'$
    $\mathbf{t} \leftarrow \mathbf{t}'$
// $\mathbf{t}$ contains all trails not absorbed in $\mathfrak{T}$ – add them to $\mathfrak{T}$ and return
$\mathfrak{T}' \leftarrow \mathfrak{T} \cup \mathbf{t}$
return $\mathfrak{T}'$

C Clustering Around Sub-optimal Trails

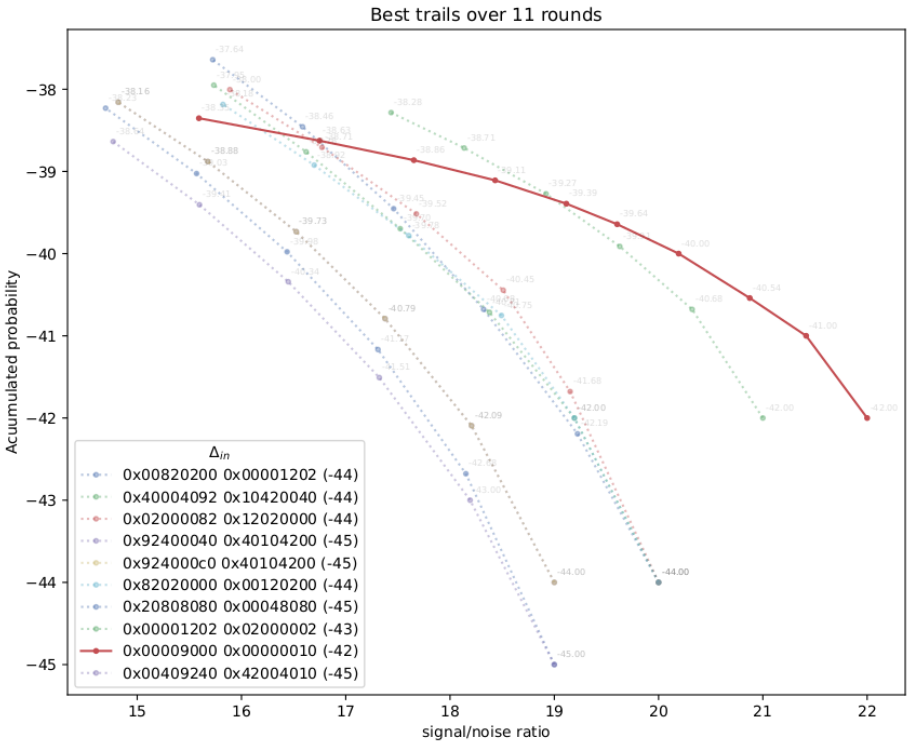

Fig. 2. Trail probability and signal to noise ratio of clustering around 11 round seed trails. The solid-lines are those of the best seed trails, while the dotted ones are sub-optimal. The number near each vertex is the weight of the cluster up to this point. Notice that some trails, after clustering around them, result in better probabilities than the clustering around the best trail (for example, -42 near the start of the dotted green line shows that there are two trails of weight -43 at this point). The legend indicates the input differences to the cluster, in hexadecimal, with the probability of the seed trail in parenthesis.

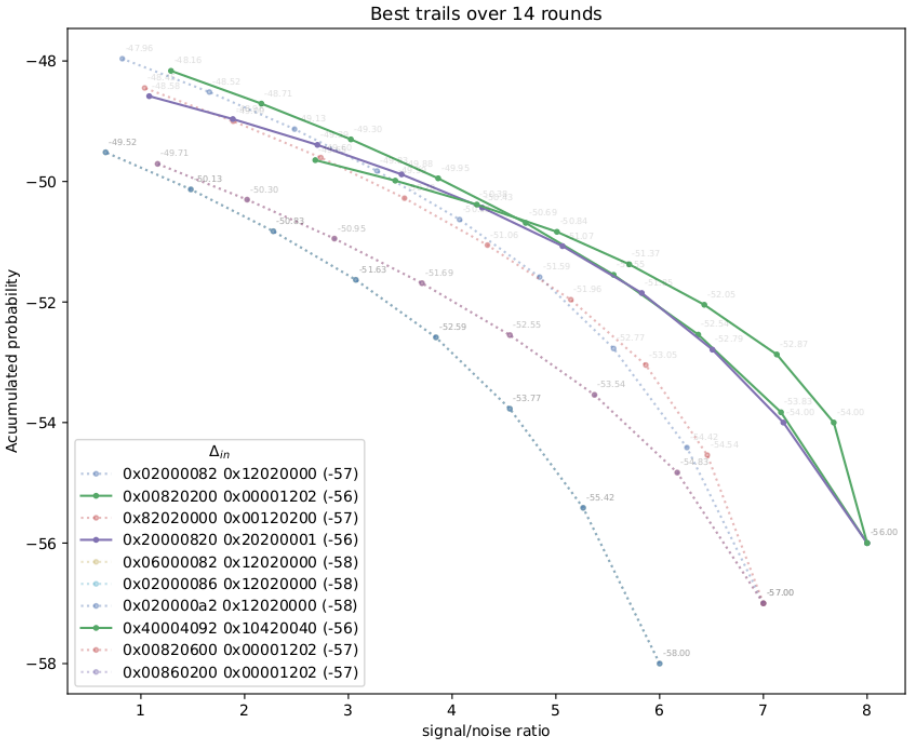

Fig. 3. Trail probability and signal to noise ratio of clustering around 11 round seed trails. The solid-lines are those of the best seed trails, while the dotted ones are sub-optimal. The legend indicates the input differences to the cluster, in hexadecimal, with the probability of the seed trail in parenthesis ($\log_2$).

D Examples of 6 Round Truncated Trails for Speck32

Table 4. Example of a 6 round truncated trail for SPECK32 with $\Pr = 2^{-11.16}$ obtained from the optimal non-truncated trail with $\Pr = 2^{-13}$. The labelled representation (right) indicates the dependency between the stars in the star representation (left). Numbers in brackets are the cumulative probabilities up to the corresponding round ($\log_2$ scale). The trail is expressed in terms of a sequence of three 16-bit values representing the two inputs and one output of the modular addition operation at each round.

R	Star representation	Label representation	Pr
0	0010001000000100	0010001000000100	(−4.00)
	0000101000000100	0000101000000100	
	0010100000000000	0010100000000000	
1	0000000001010000	0000000001010000	(−6.00)
	0000000000010000	0000000000010000	
	0000000001000000	0000000001000000	
2	1000000000000000	1000000000000000	(−6.00)
	0000000000000000	0000000000000000	
	1000000000000000	1000000000000000	
3	0000000100000000	0000000100000000	(−6.42)
	1000000000000000	1000000000000000	
	100000*100000000	100000a100000000	
4	0000000100000*10	0000000100000a10	(−8.83)
	100000*100000010	100000a100000010	
	1000000000000000	1000000000000000	
5	0000000100000000	0000000100000000	(−11.16)
	1000*10000001010	1000a10000001010	
	100001*1000*1*10	100001d1000c1b10	

In Table 5 is shown an example of a strongly truncated trail (i.e. relatively many truncated bits) on 6 rounds SPECK32 produced from an initial suboptimal trail. Capital letter labels represent the negated value of the corresponding small letter label e.g. $\mathtt{A} = 1 \oplus \mathtt{a}$.

Table 5. Example of a 6 round truncated trail for SPECK32. The labelled representation (right) indicates the dependency between the stars in the star representation (left). Numbers in brackets are the cumulative probabilities up to the corresponding round. Capital letter labels represent the negated value of the corresponding small letter label e.g. $\mathtt{A} = 1 \oplus \mathtt{a}$. The trail is expressed in terms of a sequence of three 16-bit values representing the two inputs and one output of the modular addition operation at each round.

R	Star representation	Label representation	Pr
0	0000000000000000	0000000000000000	(0.00)
	1000000000000000	1000000000000000	
	1000000000000000	1000000000000000	
1	0000000100000000	0000000100000000	(−1.42)
	1000000000000010	1000000000000010	
	1000000100000*10	1000000100000a10	
2	0000*10100000010	0000a10100000010	(−6.09)
	1000000100001*00	1000000100001a00	
	1000010000*1*110	1000010000c1b110	
3	0*1*110100001000	0c1b110100001000	(−14.09)
	1000000000***100	1000000000CAb100	
	*110110100010100	d110110100010100	
4	0010100*11011010	0010100d11011010	(−22.51)
	*1101101***00110	d1101101CAb00110	
	00111011001***00	00111011001gfe00	
5	01***00001110110	01gfe00001110110	(−33.34)
	100011***01****1	100011cab01GFed1	
	**11010110111001	ih11010110111001	

E Best 15 Round Distinguisher for Speck64 using Simple Truncation Rules

For completeness we list the output differences that correspond to the 22 non-truncated trails with $\Pr \geq 2^{-64}$ that compose the 15 round truncated set distinguisher for SPECK64 with probability $2^{-59.05}$ (Table 1) in Table 6.

Table 6. Output differences (in hexadecimal) corresponding to the 22 non-truncated trails for 15 rounds of SPECK64 with input difference (to the first round ADD) $\alpha^0 =$ `92400040` $\beta^0 =$ `10420040`. These trails are a subset of the 135 trails in the set T_{ntr} in the third to last line of Table 1.

i	$\Delta_{\text{OUT}} = (\gamma^{15} \parallel \beta^{15})$	$\log_2 \Pr_i$	$\sum_i \log_2 \Pr_i$
0	`0A080808 02084008`	-62	-62.00
1	`0A088808 02084008`	-63	-61.42
2	`0E080808 02084008`	-63	-61.00
3	`0A180808 02084008`	-63	-60.69
4	`1A080808 02084008`	-63	-60.42
5	`0A081808 02084008`	-63	-60.19
6	`0A080818 02084008`	-63	-60.00
7	`1A088808 02084008`	-64	-59.91
8	`0A089808 02084008`	-64	-59.83
9	`0A181808 02084008`	-64	-59.75
10	`0A081818 02084008`	-64	-59.68
11	`1E080808 02084008`	-64	-59.61
12	`0A188808 02084008`	-64	-59.54
13	`1A081808 02084008`	-64	-59.48
14	`0E081808 02084008`	-64	-59.42
15	`0A180818 02084008`	-64	-59.36
16	`1A080818 02084008`	-64	-59.30
17	`0A088818 02084008`	-64	-59.25
18	`1A180808 02084008`	-64	-59.19
19	`0E180808 02084008`	-64	-59.14
20	`0E080818 02084008`	-64	-59.09
21	`0E088808 02084008`	-64	-59.05

F Optimal 15 rounds Trail for best Speck64 Distinguisher

In Table 7 we report the optimal trail for SPECK64 reduced to 15 rounds we used in Section 9 to build a distinguisher with $S/N > 1$ and data complexity equal to $2^{-55.03}$.

Table 7. Optimal 15 round trail for SPECK64.

i	Δx_i	Δy_i	$\Pr_i$	$\sum_i \Pr_i$
0	40004092	10420040		
1	82020000	00120200	-5	-5
2	00900000	00001000	-4	-9
3	00008000	00000000	-2	-11
4	00000080	00000080	-1	-12
5	80000080	80000480	-1	-13
6	00800480	00802084	-3	-16
7	80806080	848164a0	-6	-22
8	040f2400	20040104	-13	-35
9	20000820	20200001	-8	-43
10	00000009	01000000	-4	-47
11	08000000	00000000	-2	-49
12	00080000	00080000	-1	-50
13	00080800	00480800	-2	-52
14	00480008	02084008	-4	-56
15	0a080808	1a4a0848	-6	-62

G Example of 6 rounds Truncated Trail for Speck64 using Relaxed Rules

R	Truncated Trail Representation	Truncated bit representations
0	00000000010000000001000001000010	$\mathtt{a} = x_0$
	00000000010000000000001001000000	$\mathtt{b} = x_1$
	00000000000000000001001000000010	$\tilde{\mathtt{c}} = x_1 \cdot x_2$
		$\tilde{\mathtt{d}} = x_1 \cdot x_2 \cdot x_3$
1	00000010000000000000000000010010	$\tilde{\mathtt{e}} = x_1 \cdot x_2 \cdot x_3 \cdot x_4$
	00000010000000000000000000000010	$\tilde{\mathtt{f}} = x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5$
	00000000000000000000000000010000	$\tilde{\mathtt{g}} = x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6$
		$\tilde{\mathtt{h}} = x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_7$
2	00010000000000000000000000000000	$\tilde{\mathtt{i}} = x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_7 \cdot x_8$
	00010000000000000000000000000000	$\tilde{\mathtt{j}} = x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_7 \cdot x_8 \cdot x_9$
	00000000000000000000000000000000	$\tilde{\mathtt{k}} = x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_7 \cdot x_8 \cdot x_9 \cdot x_{10} + 1$
		$\tilde{\mathtt{l}} = x_0 \cdot x_{11} + x_0$
3	00000000000000000000000000000000	$\tilde{\mathtt{m}} = x_0 \cdot x_{11} \cdot x_{12} + x_1 \cdot x_{12}$
	10000000000000000000000000000000	$\tilde{\mathtt{n}} = x_0 \cdot x_{11} \cdot x_{12} \cdot x_{13} + x_1 \cdot x_{12} \cdot x_{13}$
	10000000000000000000000000000000	$\tilde{\mathtt{o}} = x_0 \cdot x_{11} \cdot x_{12} \cdot x_{13} \cdot x_{14} + x_0 \cdot x_{12} \cdot x_{13} \cdot x_{14}$
		$\tilde{\mathtt{p}} = x_1 \cdot x_{11} \cdot x_{12} \cdot x_{13} \cdot x_{14} \cdot x_{15} +$
4	00000000100000000000000000000000	$\quad x_0 \cdot x_{12} \cdot x_{13} \cdot x_{14} \cdot x_{15}$
	10000000000000000000000000000100	$\tilde{\mathtt{q}} = x_0 \cdot x_{11} \cdot x_{12} \cdot x_{13} \cdot x_{14} \cdot x_{15} \cdot x_{16} +$
	1000000a10000000000000000000100	$\quad x_1 \cdot x_{12} \cdot x_{13} \cdot x_{14} \cdot x_{15} \cdot x_{16}$
		$\mathtt{r} = x_{17}$
5	000001001000000a1000000000000000	$\tilde{\mathtt{s}} = x_{17} \cdot x_{17}$
	1000000a10000000000000000100000	$\tilde{\mathtt{t}} = x_{17} \cdot x_{18} \cdot x_{19}$
	r1000qponmlkjihgfedcb100000	$\tilde{\mathtt{u}} = x_{17} \cdot x_{18} \cdot x_{19} \cdot x_{20}$
		$\tilde{\mathtt{v}} = x_{17} \cdot x_{18} \cdot x_{19} \cdot x_{20} \cdot x_{21}$

Table 8. Example of a 6 round truncated trail for SPECK64 using relaxed rules. The labelled representation (right) indicates the dependency between the stars in the truncated representation (left). The trail is expressed in terms of a sequence of three 16-bit values representing the two inputs and one output of the modular addition operation at each round.