The Window Heuristic: Automating Differential Trail Search in ARX Ciphers with Partial Linearization Trade-offs

Emanuele Bellini[1], David Gerault[1], Juan Grados[1], and Thomas Peyrin[2]

[1] Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, UAE
{emanuele.bellini,david.gerault,juan.grados,rusydi.makarim}@tii.ae
[2] Nanyang Technological University, Singapore
thomas.peyrin@ntu.edu.sg

Abstract. The search for optimal differential trails for ARX ciphers is known to be difficult and scale poorly as the word size (and the branching through the carries of modular additions) increases. To overcome this problem, one may approximate the modular addition with the XOR operation, a process called linearization. The immediate drawback of this approach is that many valid and good trails are discarded.

In this work, we explore different partial linearization trade-offs to model the modular addition through the *window heuristic*, which restricts carry propagation to windows of w_s consecutive positions. This strategy enables the exploration of full linearization ($w_s = 0$), normal modelling ($w_s = n$), and all the different trade-offs between completeness and speed in between.

We give the corresponding SAT and MILP model and their parallel versions, and apply them to `ChaCha`, `SPECK`, `LEA`, and `HIGHT`. Our method greatly outperforms all previous modeling of modular addition. In particular, we find the first differential path for 4 rounds of `ChaCha` with a probability greater than 2^{-256}, and a corresponding 6 rounds boomerang distinguisher. This indicates that purely differential-based attacks have the potential to become competitive with differential-linear attacks, currently, the best-known attacks against `ChaCha` and other ARX ciphers.

On `SPECK`, we find many improvements over the state-of-the-art in terms of speed of the differential trail search in the single-key setting and new upper bound probabilities in the related-key setting.

Finally, we exhibit an improved key recovery attack on reduced `LEA`.

Keywords: differential cryptanalysis · modular addition · ARX · MILP · SAT

1 Introduction

Addition-Rotation-XOR (ARX) ciphers have become popular primitives in the community for their good performances in software, as they are constructed upon CPU-friendly components. Despite their relative simplicity, they can provide good resistance against well-known cryptanalysis. A famous example is the cipher ChaCha20 [9], which is widely used by several protocols such as SSH and in the more recent version of the TLS protocol, TLSv1.3.

Together with linear cryptanalysis [36], differential cryptanalysis [10] is among the most famous and powerful attack techniques against symmetric-key ciphers, including ARX ones. The main preparation of a differential attack is to find a differential path that will occur with a relatively high probability. Such differential paths can, in turn, be used to build a more advanced distinguisher, or a global key-recovery attack (for example, using probabilistic neutral bits against ChaCha [2]).

In the past decade, we have seen a strong push by the community towards the usage of automated solvers, based on Boolean Satisfiability (SAT), its generalization to Modulo Theory (SMT), Mixed Integer Linear Programming (MILP), or Constraint Programming (CP), to look for good differential paths [37, 26, 43]. The task of building a differential path search algorithm is essentially reduced to the task of efficiently producing accurate modeling of the path search problem, and many recent works have been devoted to improving such modeling [41, 51, 46, 35].

Compared to Substitution-Permutation Network (SPN) ciphers, in ARX ciphers the study of differential properties has been shown to be more complex. This is mainly because of the usage of modular addition. In particular, the number of possible input/output differential pairs, i.e. the number of entries in the Differential Distribution Table (DDT), of a nonlinear vectorial Boolean function (such as an S-Box or a modular addition) is exponential in the input/output bit length. A consequence of this is the exponential explosion of the tree of the possible differential paths. To overcome this explosion, researchers try to discard in advance paths that lead to low probability. This is achieved in essentially two ways: (1) considering entries of the DDT that only have a probability above a certain threshold [11, 14, 16]; (2) approximating the modular addition with the XOR operation, a process called *linearization* [18, 50, 49, 25, 2]. The main drawback of both approaches is that good trails might be discarded. In the first case, this can happen because a globally optimal path (i.e. with optimal probability) might have local probabilities that are not optimal. In the second case, local high probabilities might simply be missed.

In this paper, we focus on the second approach, and explore various levels of linearization of the modular addition through a heuristic termed the *window heuristic*. This heuristic provides a good and adaptable balance between the retention of the non-linearity of the modular addition and the size of the search tree. The idea is to let the modular addition behave non-linearly only for a fixed number of consecutive bits. Throughout this paper, we refer to this limit as the window size, denoted as w_s. Applying our technique to a selection of notable ARX ciphers, we obtain improvements over the state-of-the-art, by rediscovering known differential trails faster and by finding new differential trails.

1.1 Related works on automated differential path search tools for ARX ciphers

The automation of differential characteristic search in cryptanalysis is crucial, reducing both time and error for crypyanalysts. However, it remains challenging to find efficient differential trails for ARX ciphers with large word sizes, such as ChaCha, which has seen limited analysis against pure differential cryptanalysis [1, 7]. Early methods relying on local constraint propagation [20, 31] were restricted by engineering limitations, unlike later approaches using solvers like SAT, SMT, MILP, and CP. Biryukov *et al.* [11, 14, 16] advanced the search by approximating modular addition as a large S-Box and employing a partial difference distribution table (pDDT), significantly enhancing the search process.

Subsequent research further integrated solver-based techniques. Mouha *et al.* [37] used SAT solvers for ARX ciphers such as Salsa20, building on Lipmaa and Moriai's XOR-differential probability model [34]. Fu *et al.* [26] and Song *et al.* [43] used MILP methods to uncover improved differential and linear trails for SPECK, while Monte Carlo Tree Search techniques [24, 6] refined differential path discovery for LEA. More recently, Sadeghi *et al.* [42] validated differential trails using MILP, and Liu *et al.* [35] split modular additions into smaller parts to identify differential and linear trails for SPECK and HIGHT. Notably, SAT-based improvements such as sequential encoding for Matsui's bounding condition [46, 17] and Laurent *et al.* 's clustering technique [32] further optimized differential cryptanalysis for ciphers like SIMON and Simeck.

Other approaches tried to tackle the modular addition component by linearizing it, essentially by replacing it with an XOR. This strategy was extensively used for the cryptanalysis of the `MD-SHA` family of ARX hash functions [18, 50, 49]. Specifically, in those works the authors explore differential cryptanalysis methods for MD5, SHA-0, SHA-1 focusing on the linearization of modular addition. They induce modular addition behave as a XOR linear operation under certain conditions. This is achieved by strategically applying perturbations and corrections to the input, ensuring that no carry propagation occurs during modular addition.

In [25], Fischer *et al.* used this technique to improve the search for differential trails in `Salsa20` and `TSC-4`. Similarly, [2] Aumasson *et al.* showed that approximating the modular addition as an XOR improves the time complexity of searching for collisions in `Rumba`. Interestingly, in both papers, the authors remarked that this technique worked well only when applied to the initial rounds. Indeed, the approximation does not remain valuable after a few rounds, after the diffusion property of the modular addition starts to kick in.

1.2 Our contributions

– **The Window Heuristic** We propose the *window heuristic* to partially linearize modular additions in differential trails search, and the corresponding SAT and MILP models and their parallel versions (`https://anonymous.4open.science/r/Improved-MILP-model-for-Modular-Addition-o` n-ARX-ciphers-FA81). In the window heuristic, the carry propagation is bounded to w_s consecutive positions, with $w_s = -1$ denoting no heuristic used, $w_s = 0$ corresponding to linearization, $w_s = n$ to regular modular addition modelling, and the values in between being explored for the first time. In other words, if the window heuristic is not used, it is equivalent to the regular modular addition modeling.

– **Differential Cryptanalysis of** `ChaCha` We demonstrate the power of window heuristic by proposing the first 4-round single-key differential trail with probability higher than 2^{-256}, improving the state of the art from 2^{316} to 2^{217}; for 3 rounds, we improve the known differential bounds from 2^{147} to 2^{120} for 3 rounds(Table 4). In addition, we propose a boomerang distinguisher for 6 rounds (Table 7);
– **Single-Key Differential Cryptanalysis of** `SPECK` Using the parallelized version of the window heuristic, we retrieve known results with speedups of up to 5x (and an outlier of 152x), using 5 cores, compared to the state of the art (Table 1);
– **Related-Key Differential Cryptanalysis of** `SPECK` Using the window heuristic, we improve the known bounds for related-key differential trails (Table 3); in particular, we improve by a factor 2^{12} for SPECK-32/64, propose the first 16-round trail for SPECK-48/96, and improve the best 19-round trail for SPECK-128/256 by a factor 2^{19};
– **Single-Key Differential Cryptanalysis of** `LEA` Using the window heuristic, we improve the known bound for 13 rounds of LEA-128 by a factor 2^{11}; this differential trail is used to build a key recovery for 14 rounds, with time and data complexity $2^{111.97}$; these results are summarized in Table 10.
– **Single-key Differential Cryptanalysis of** `HIGHT` Using the window heuristic, we improve the search speed for trails up to 11 rounds. On the other hand, for 12 rounds, our tool with window sizes 0, 1, 2, and 3 do not retrieve the state-of-the-art trails; in fact, the best known 12-round trail for `HIGHT` contains a window size of 5, which is out of the capacities of our tool at the moment.

In the main contributions cited above, our work is improving differential probabilities by utilizing a window size of 0 or 1 in our heuristic approach. We emphasize that this exploration into the value of 1 was previously unconsidered. It enabled us to uncover more efficient differential trails in ARX ciphers, enhancing both the field's understanding and the efficiency benchmarks of differential cryptanalysis on ARX ciphers. For instance, using the window size 1, we improve the best-known differential trail probability for `ChaCha` by an impressive factor 2^{98}. While extending the window size beyond 1, our research observed comparable results to those of previous works, yet notably, our methodology achieved these outcomes faster in most instances. This advancement not only prompts a reassessment of existing differential search models for ARX ciphers but also provides a strong basis for future research.

Finally, we would like to highlight that, when possible, the differential trails found in this work were verified using the tool presented in [42].

1.3 Outline

In Section 2, we first provide some preliminaries. We introduce our new strategy, the window heuristic, in Section 3 and explain how to encode it in SAT and MILP. We apply this new method to find XOR-differential trails and various attacks on selected ARX ciphers in Section 4 and provide our conclusions and future work in Section 5.

2 Background

In this section, we provide a brief description of differential and boomerang distinguishers, and of existing SAT and MILP encoding techniques for searching XOR-differential trails on ARX ciphers. The studied ciphers, `ChaCha`, `SPECK`, `LEA` are very well-known, and described in Section B.

2.1 Differential and Boomerang distinguishers

Differential cryptanalysis is a well-established cryptanalytic technique introduced by Biham and Shamir in the early 1990s [10]. It examines how differences in pairs of plaintexts propagate through the encryption process to produce differences in ciphertexts. The core concept involves analyzing an input difference, denoted as $\Delta X = X \oplus X'$, where X and X' are two different plaintexts. By observing how this input difference evolves into an output difference $\Delta Y = Y \oplus Y'$, cryptanalysts aim to find high-probability differential characteristics over multiple rounds of the cipher. The probability of a differential characteristic $(\Delta X \to \Delta Y)$ is defined as:

$$\Pr(\Delta X \to \Delta Y) = \frac{\#\text{pairs}(X, X') \text{ such that } E(X) \oplus E(X') = \Delta Y}{\#\text{all possible pairs}(X, X')}$$

where E represents the encryption function for the cipher. The goal is to identify differentials with a higher-than-random probability.

In 1999, Wagner *et al.* developed a technique called the boomerang attack [47], for ciphers where the differential probability is very small for long differential trails, but high for shorter ones. The technique considers a cipher E as two parts, the top part and the bottom part, denoted here as E_0 and E_1, with respectively r_0 and r_1 rounds. Given a differential $(\Delta_{in}, \Delta_{out})$ for E_0 with probability p, and another differential $(\nabla_{in}, \nabla_{out})$ for E_1 with probability q, the corresponding boomerang distinguisher for r_0+r_1 rounds has probability of p^2q^2, assuming that E_0 and E_1 are independent.

In 2011, Murphy found that the aforementioned assumption causes some invalid boomerang distinguishers [38]. Many studies have been done to evaluate the interaction between the differential of the top and bottom parts [12, 13, 23, 30], with the majority of insights being integrated into the so-called sandwich attack framework introduced in [23]. Within this perspective, a cipher is divided into three parts: $E = E_1 \circ E_m \circ E_0$. Here, E_m can be seen as a small boomerang distinguisher of probability p_m. For ciphers based on S-Boxes, and if E_m covers only a single round, the computation of this probability is reduced to analyzing a sole S-Box. This idea was elucidated in [22], where the switching effect was represented using a table termed the Boomerang Connectivity Table (BCT), with dimensions $2^n \times 2^n$. For components like modular addition, the table's size can grow considerably. If one interprets a modular addition component as an S-Box $M(x_0, x_1) = x_0 + x_1 \parallel x_1$ of dimension m, this table can expand to a size of 2^{128}, as observed in the case of `ChaCha`. In [48], a suite of methods is presented to tackle this challenge. Specifically, the authors have devised a technique that minimizes the computation of this table by employing another table named the partial boomerang connectivity table, inspired by [15]. Additionally, the authors integrated their techniques into an automated tool designed to identify boomerang distinguishers in ARX ciphers. In this paper, we used these techniques through their automated tool to construct boomerang distinguishers against `ChaCha` in Subsection 4.2.

2.2 Finding XOR-differential trails with MILP and SAT solvers

In this section, we review the existing MILP and SAT general modeling techniques we will use to find the differential trails for ARX ciphers: [26] for MILP and [46] for SAT.

MILP-based automatic search for differential trails As described in [26], one can build a model for ARX ciphers using linear inequalities. In the following paragraphs, we describe the inequalities used to model each component of the ARX ciphers.

Constraints from a XOR *operation.* For every n-bit XOR operation with input differences $\mathbf{a} \in \mathbb{F}_2^n$ and $\mathbf{b} \in \mathbb{F}_2^n$ and output difference $\mathbf{c} \in \mathbb{F}_2^n$, the constraints at bit level for j in $\{0, \ldots, n-1\}$ are

$$\begin{array}{ccc} d_{\oplus}[j] \geq a[j], & d_{\oplus}[j] \geq b[j], & d_{\oplus}[j] \geq c[j], \\ a[j] + b[j] + c[j] \geq 2d_{\oplus}[j], & a[j] + b[j] + c[j] \leq 2. & \end{array} \tag{1}$$

where $d_{\oplus}[j]$ is a dummy variable used to verify that there are at least two active terms in $a[j] \oplus b[j] = c[j]$, every time that $a[j] \neq 0$, $b[j] \neq 0$, or $c[j] \neq 0$.

Constraints from a modular addition. To construct the constraints of the n-bit modular addition, we again follow [26]. We use $13 \times (n-1) + 5$ inequalities to model the modular addition operation modulo 2^n with input differences $\mathbf{a} \in \mathbb{F}_2^n$ and $\mathbf{b} \in \mathbb{F}_2^n$, and output difference $\mathbf{c} \in \mathbb{F}_2^n$. These constraints are derived from the differential propagation rules of modular addition [33]; each output bit is the XOR of two input bit, and a carry difference bit. The carry difference bit at position i can be: (1) 0 with probability 1 if $a[i-1] = b[j-1] = c[j-1] = 0$; (2) 1 with probability 1 if $a[i-1] = b[j-1] = c[j-1] = 1$; (3) 0 or 1 with probability $\frac{1}{2}$ otherwise. For j in $\{0, \ldots, n-2\}$ these inequalities are described in Section A. Besides that, the following inequalities are also needed

$$\begin{aligned} d_+ &\geq a[0], & d_+ &\geq b[0], \quad d_+ \geq c[0], \\ a[0] + b[0] + c[0] &\geq 2d_+, & a[0] + b[0] + c[0] &\leq 2. \end{aligned} \tag{2}$$

where d_+ is a dummy variable and $d[j]$ in Equation (6) represents the probability weight variable [26].

Constraints from a rotation operation. Assume that $Rot(\cdot, \alpha)$ is the rotation operation whose rotation amount is α. Then, for every rotation operation where the input difference is $\mathbf{a} \in \mathbb{F}_2^n$ and the output difference is $\mathbf{b} \in \mathbb{F}_2^n$, the constraints at bit level are $Rot\,(\mathbf{a}, \alpha)_j = \mathbf{b}[j]$ for j in $\{0, \ldots, n-1\}$.

Objective function. Let R be the number of rounds that we are modeling. Also, let $d[r][j]$ be the variable representing the probability weight price at round r and bit j if there is a difference in the carry bit. Then according to [26], we need to minimize $\sum_{r=1}^{R} \sum_{i=0}^{n-2} d[r][j]$.

Finding XOR-differential trails with SAT solvers In this section, we recall the methodology to model ARX ciphers with CNF clauses for the search of differential trails from [46].

For every n-bit modular addition operation with input differences $\mathbf{a} \in \mathbb{F}_2^n$ and $\mathbf{b} \in \mathbb{F}_2^n$ and output difference $\mathbf{c} \in \mathbb{F}_2^n$, we define the CNF clauses at the bit level. For each bit position $j \in \{0, \ldots, n-2\}$, the relationship between the input bit differences and the output bit differences is described using Boolean expressions. These expressions, which include clauses representing valid possible combinations of input and output differences, ensure that the modular addition is correctly modeled using SAT techniques.

The full set of CNF clauses for this operation is presented in Appendix Section A, where the individual conditions are detailed for clarity.

From the set of formulas described in Section A, $-\sum_{j=0}^{n-2} w[j]$ represents the exponent of the differential probability weight .

CNF clauses for the XOR *operations.* For every n-bit XOR operation with input differences $\mathbf{a} \in \mathbb{F}_2^n$ and $\mathbf{b} \in \mathbb{F}_2^n$ and output difference $\mathbf{c} \in \mathbb{F}_2^n$, the CNF clauses at bit level for j in $\{0, \ldots, n-1\}$ are:

$$\begin{aligned} a[j] \vee b[i] \vee \neg c[i] &= 1, & a[j] \vee \neg b[i] \vee c[i] &= 1, \\ \neg a[j] \vee b[i] \vee c[i] &= 1, & \neg a[j] \vee \neg b[i] \vee \neg c[i] &= 1. \end{aligned} \tag{3}$$

CNF clauses for the modular addition operations. For every n-bit modular addition operation with input differences $\mathbf{a} \in \mathbb{F}_2^n$ and $\mathbf{b} \in \mathbb{F}_2^n$ and output difference $\mathbf{c} \in \mathbb{F}_2^n$, the CNF clauses at bit level for j in $\{0, \ldots, n-2\}$ are described in Section A. From that set of formulas it is important to mention that $\sum_{i}^{n-2} w[i]$ is the exponent of the differential probability multiplied by -1.

In contrast to MILP modeling, SAT modeling lacks a built-in mechanism to identify differential trails with minimal probability weight. An incremental approach is taken instead. This approach involves assessing the satisfiability of a problem representing the search for differential trails, with the probability weight $\sum_i w[i]^{n-2}$ initially set to 0. If the result is satisfiable, the search ends; if not, the search increments the probability weight to 1 and continues iteratively until satisfiability is achieved. Following the method illustrated in [46], we utilize the sequential encoding technique to depict $\sum_i w[i]^{n-2} \leq k$, where k encompasses the potential values from the incremental procedure. This leads to an additional $2 \times k \times n + n - 3 \times k - 1$ clauses.

2.3 Splicing and extending heuristics and linearisation technique

Within the existing literature, two primary heuristics emerge in the search for differential trails using SAT or MILP solvers on ARX ciphers: the *splicing* heuristic and the *extending* heuristic, as well as their combined approach. These heuristics are elaborated upon in [26] and [43]. The motivation for these heuristics comes from the fact that fixing some differential states reduces the complexity of the search for differential trails. The word heuristic is used because the proposed method in the following section, and others described in this section, only explores part of the search space and may therefore miss optimal solutions.

The splicing heuristic divides the problem of finding a long differential characteristic into the (easier) problem of finding two short matching differential characteristic. In this technique, two differentials $\delta_a \to \delta_b$ and $\delta_b \to \delta_c$ (where the output difference of the first differential matches the output difference of the second one) are concatenated into $\delta_a \to \delta_c$.

The extending heuristic is characterized by the expansion of a known differential trail either m rounds upward or downward. These heuristics are frequently cited in the literature and have proven to be efficient in determining upper-bound probabilities of differential trails. For instance, to the best of our knowledge from the literature, the best bound for the probability of the differential trail of `SPECK-128-128` reduced to a 19-round was identified in [26] using both heuristics.

In addition to these heuristics, the *linearisation technique* provides a powerful method to simplify the analysis of non-linear operations such as modular addition. The core idea of this technique is to transform modular addition, which typically involves complex carry propagation, into a form that behaves similarly to the XOR operation under certain conditions. By introducing restrictions on the input differences (e.g., controlling the Hamming weight of the input differences or ensuring no carry propagation), modular addition can be linearized, allowing it to behave like XOR. This linearization reduces the complexity of modeling modular addition in differential cryptanalysis.

3 The window heuristic

In this section, we introduce the window heuristic and describe its benefits. The corresponding SAT and MILP models are derived from [26] for SAT and [46] for MILP, which are described in Subsection 2.2.

3.1 The window heuristic using MILP and SAT constraints

We conducted an analysis of the differential trails obtained using the model delineated in Subsection 2.2. Our observations indicated that, for high-probability differential trails, the carry difference stops propagating beyond a certain point. Specifically, for most of the analyzed differential trails, the number of carry-bit differences involved in the propagation is usually less than or equal to 3. For instance, this pattern is evident in all differential trails outlined in [26], with the exception of `SPECK-96-96`. A comparable pattern was noted in the trails for `LEA-128` from [43]. Notably, the upper-bound probabilities in the trails from [26] have remained unchanged for over six years.

Based on these insights, we propose the window heuristic. This involves creating a few new constraints to control the propagation of differences in the carry. We describe these new constraints below, but first, we formalize the terms *carry difference* and *carry propagation* in Definition 1 and Definition 2, respectively.

Definition 1 **(i^{th} carry difference).** *Let* $\mathbf{a} \in \mathbb{F}_2^n$ *and* $\mathbf{b} \in \mathbb{F}_2^n$ *be input differences propagating to the output difference* $\mathbf{c} \in \mathbb{F}_2^n$ *of the modular addition* 2^n *with certain probability. Then the* i^{th} *carry of this modular addition is given by* $\mathcal{C}[i] = a[i] \oplus b[i] \oplus c[i]$*, for* $i = 0, \cdots, n-2$.

Definition 2 **(m-window carry propagation).** *With* $\mathbf{a}, \mathbf{b}, \mathbf{c}$ *as above, we say that the carry difference* $\mathcal{C}[i]$ *propagates to the next* m *bits if* $\mathcal{C}[i+j] = 1$ *for* $j = 1, ..., m$.

Using a window heuristic with smaller window sizes, w_s, prioritizes speed rather than precision. For words of size n, the carry differences can propagates up to $k = \lceil (n/(w_s+1)) \rceil$ times, which reduces the search area exponentially by a factor of k. When it comes to precision, a window of w_s consecutive carries offers, at most, a probability bounded by $2^{-w_s/2}$. Given that the most favorable transitions involve few carries (as outlined in Theorem 4 of [33]), cutting out lengthy sequences of carries is a practical strategy, albeit not the most accurate one for precision.

Let us see an example. Consider a modular addition component with input differences of `0x080000001e4a0848` and `0x08000000000e0808`, where the Most Significant Bit (MSB) is on the left. Assume its resulting output difference is `0x00000000f2400040`. This leads to carry differences (with background yellow) of `0xec040000`=11101100000001000000000000000000 (i.e., the last active carry is at position 31). Observations from this example reveal a 1-window carry propagation at the 18th-bit position, a 2-window carry difference propagation at the

26th-bit position, and a 3-window carry difference propagation at the 29th-bit position.

As we said, based on the observation that a carry difference $\mathcal{C}[i]$ does not propagate to a large number of their subsequent differences, we added constraints in the SAT and MILP models with the aim of fixing the propagation of $\mathcal{C}[i]$ to their m subsequent carry differences.

The window heuristic using MILP techniques. For every modular addition with input differences **a** and **b** and output difference **c**, the constraints for the m-window carry propagation are

$$\sum_{i=1}^{m+1} \mathcal{C}[j-i] \leq m \text{ for all } j \text{ in } m+1, \cdots, n-1. \tag{4}$$

In other words, we are adding $n-1-m$ new constraints for every modular addition. Adding the new constraints to those described in Section 2.2 gives us a total of $R \times ((14 \times n - 9 - m) \times \mathtt{a}_r + 5 \times n \times \mathtt{x}_r + n \times \mathtt{r}_r)$ constraints.

The window heuristic using CNF formulas. For every modular addition with input differences **a** and **b** and output difference **c**, the clauses for the m-window carry propagation presented in Definition 2 are derived from

$$\bigwedge_{j=m+1}^{n-1} \bigvee_{i=1}^{m+1} \neg(a[j-i] \oplus b[j-i] \oplus c[j-i]) \tag{5}$$

For every j the expression $\bigvee_{i=1}^{m+1} \neg(a[j-i] \oplus b[j-i] \oplus c[j-i])$ generates $2^{2(m+1)}$ clauses. In other words, we are adding $2^{2(m+1)}(n-1-m)$ new clauses for every modular addition. Adding these new clauses to those described in Section 2.2 gives us a total of $R(\mathtt{x}_r(4(k-1)n)) + (13(n-1) + 2^{2(m+1)}(n-1-m))R\mathtt{a}_r$ clauses.

A different model for Equation 5, using the xor-exclusive CNF clauses proposed by Cryptominisat [44], was implemented as well; however, it did not yield a significant advantage, so that we only report the more generic CNF model.

We explored the possibility of constructing more heuristics centered around carry differences with the aim of uncovering novel upper-bound probabilities for the differential trail search of ARX ciphers. Regrettably, these heuristics did not yield the anticipated results. For instance, one of these heuristics was conceived after noting the clustering pattern in the carry difference variables. With clustering pattern we mean the number of sequences that contains consecutive active carry differences for example: 10101010111 and 10000011111 have 7 clusters. We observed the cluster generated by the active differences. We noticed that the number of these clusters in high-probability differential trails was relatively small. Consequently, we incorporated constraints into our SAT and MILP models to limit the number of clusters present in the carry difference variables. One other attempt was to devise a heuristic that restricted the total number of active differences within the carry bits. This attempt, too, was unsuccessful.

3.2 Model-level Parallelism Through the Window Heuristic

Solvers supporting parallel resolution sometimes allow to solve problems significantly faster; for instance, the optimal trail for 10 rounds of `SPECK-128-128` was found using the parallel SAT solver ParKissat [5]. However, the speedup obtained by solver-level parallelization techniques is often weaker than that obtained through informed decomposition into sub-problems, as in [27]; such decompositions have however rarely been used for ARX ciphers.

Our window heuristic enables straightforward model-level parallelism, as each window size w_s can be solved as an independent sub-problem: an instance of the solver is started for each window size w_s between zero and an upper bound provided by the user. An instance with a special window size $w_s = -1$, denoting no window heuristic, can be started in parallel.

We combined this model-level parallelism with solver-level, using 16 cores for each window size, from -1 to 3, in the corresponding MILP model, resulting in 80 cores being used in total.

Furthermore, more subproblems can be built by fixing the number of "full windows", *i.e.,* windows that actually contain exactly w_s carries. For instance, in the 9-round differential trail for `SPECK-32-64` presented in Table 16 the carries for every modular addition in each round, represented as binary numbers for each round are as follows: 0, 11000000, 1000001000000110, 1000000000000, 1100000, 0, 0, 0, 0. Thus, there are three "full windows" with $w_s = 2$, which are highlighted in yellow for illustration.

4 Applications of the window heuristic

The models were solved on a machine running Ubuntu 20.10 and using 80 parallel Intel(R) Xeon(R) Platinum 8280 CPUs clocked at 2.70GHz. The MILP models were implemented in MiniZinc [39] and solved with OR-Tools [40][3]; the SAT models were solved with CadiCal. Unless stated otherwise in the text, all our experiments were run with a 24 hours timeout, so that the solutions for each instance may not be optimal.

In the following sections, we distinguish between "Timing Results" (the cumulative time it takes to complete a specific experiment across for all considered window sizes) and "Individual Timing Results.", denoting the time our tool requires to solve a particular experiment using a single, specific window size value.

4.1 Application to `SPECK`

Search for differential trails in the single-key scenario. In the single-key scenario, optimal (or close to optimal) bounds for many versions of `SPECK` are already known (*e.g.,* [43, 26, 35] and Table 10 of [46]).

[3] The more popular Gurobi did not perform as well on our problems, though we did not run a detailed comparison as it was out of scope.

Our parallel SAT implementation retrieved these results, albeit significantly faster. For instance, compared to [46], our tool is 152 times faster for `SPECK-128-128` reduced to 8 rounds (Table 1). Our parallel MILP model achieves similar results, and the detailed timings are given in Tables 16, 18, 20, 22 and 24 in Section D.

Number of rounds	[46]	**5 cores**	**Pr**
	Time (s)		
1	0.02	0.02	0
2	0.03	0.03	1
3	0.05	0.05	3
4	0.27	0.27	6
5	1.82	0.95	10
6	8.65	1.19	15
7	10.49	10.49	21
8	264.55	1.73	30
9	6685.22	1876.24	39

Table 1: Timing comparison between the tool developed in [46] and a parallelized version of the tool presented in this paper for `SPECK-128-128`, with a window size ranging from -1 to 3 using SAT techniques. In the parallel version, the search is divided into five parts, each solved by a different core, using distinct window size. Probability is denoted as $-\log_2(\cdot)$. For rounds 1 to 4 and round 7, the results are nearly identical, with no significant timing differences (this pattern is very similar also to the results obtained with MILP techniques, see Table 24). Thus, the results we obtained with the tool developed in [46] for these rounds were copied in the corresponding cells. For rounds 5, 6, 8, and 9, the timings diverge, especially in rounds 8 and 9.

Window heuristic and splicing heuristic. For more difficult problems, where the known bounds still diverge significantly from the block sizes, the basic parallel tool was not sufficient, even increasing the timeout to over two week. However, by combining it with splicing heuristics, we were able to match known results considerably faster. In particular, to recreate the 19-round differential trail cited in [43], we used the splicing heuristic described in [26] and [43]. Specifically, we set all bits to 0, with the exception of the bit at position 71 in the internal state at round 15. Relying solely on the splicing heuristic, we determined a probability weight of 119 for `SPECK-128-128` reduced to 19 rounds, accomplishing this in 45 minutes. When we combined a window size of $w_s = 3$ with the splicing heuristic in the same 15th round and using the parallel SAT version, we achieved the identical probability weight in a mere 300 seconds. While one might argue that we were already privy to the intermediate value at round 15, where the single-bit difference should be placed, the time taken to pinpoint this intermediate value is indeed the same for both methods.

For differential trail searches, utilizing both the window and splicing heuristics, coupled with the parallel MILP version, the timings were comparable.

Window Heuristic constraining the number of "full windows". Inspection of the obtained trails shows that the length of carry propagations rarely reaches the allowed window size. For instance, the 13 round trail on `SPECK-128-128` given in Section C has window size 3, but only one series of consecutive carries reaches this length through the entire trail.

Therefore, the number of *full windows*, *i.e.,* the number of times the maximum allowed length of carry propagation appears, can be used to divide into additional sub-problems, where the number of full windows is set to be lower than, equal to or greater than a fixed value k.

In these additional experiments, we searched for differential trails using four configurations and two SAT solvers: a sequential SAT solver and a parallel SAT solver (16 cores). The configurations were: 1) window size without "full windows"; 2) window size with "full windows" and the "exactly" constraint; 3) window size with "full windows" and the "at least" constraint; and 4) window size with "full windows" and the "at most" constraint. The corresponding timings are given in Section C).

Constraining the number of "full windows" significantly helps in the search for trails (Figure 2 of Section C); in particular, we successfully find a new 19-round differential trail for `SPECK-128-128`, available in the repository accompanying this paper. Interestingly, we did not use the splicing heuristic to perform this search like in previous papers. This was achieved in approximately six hours using a window size of 3 and constraining the number of "full windows" to be at most 1. This result is notable because our previous attempts using ParKissat or CadiCal and similar window sizes ran for weeks without success.

However, the results using sequential SAT solver, shown in Figure 3, were less consistent. Out of 11 versions of SPECK, the sequential solver was beneficial in only 5 cases. It appears that the sequential solver is effective when there is more than one full window, but less so when there is only one. This phenomenon requires further investigation to fully understand.

Search for differential trails in the related-key scenario. We conducted several experiments applying the window size heuristic to search for differential trails in the related-key scenario. Given that the key schedule round function of `SPECK` mirrors its permutation round function, the diffusion characteristics of both processes align closely. Consequently, drawing parallels with studies such as [42] and [41], our search for differential trails will lead to differential distinguishers in the weak-key setting. This implies that, in line with earlier research, the differential distinguishers generated from our differential trails are effective for only a subset of the key space.

In Table 3, we present our related-key results compared with the state-of-the-art; the full trails and timing informations are given in `https://anonymous.4open.science/r/Improved-MILP-model-for-Modular-Addition-on-ARX-ciphers-FA81`. As shown in Table 3, the results were obtained with $ws = 0$. Similar results can be achieved for other values of ws, but they are not included here because the time required to discover these trails with larger ws was greater than for $ws = 0$. In Table 3, the $\log_2(P_D)$ represents the probability weight for the internal permutation, and $\log_2(P_K)$ represents the probability weight for the key schedule process. Similar to previous works, we set the objective function in the MILP and in the SAT models as the minimum of $\log_2(P_{DK}) = \log_2(P_D) + \log_2(P_K)$; our results improve the upper-bound probabilities for all the versions of SPECK in the related-key scenario.

On related-key boomerang distinguishers and hash distinguishers on SPECK*.* Using our newly found related-key differential trails for SPECK to build related-key (boomerang) distinguishers covering a larger number of rounds might seem promising at first sight. However, as SPECK has a quite strong key schedule compared to many other lightweight block ciphers (basically a repetition of the internal cipher round function), this is not quite the case. One can observe from Tables 16, 18, 20, 22 and 24 in Section D that for a given number of rounds, a related-key differential path for SPECK is not really better than a single-key one. It can actually be much worse, especially for SPECK versions for which the key size is the same as the block size, for larger versions of SPECK, and when the number of rounds increases. Therefore, except for small SPECK versions on a small number of rounds, the related-key (boomerang) distinguishers have little chance to improve over single-key (boomerang) distinguishers, while requiring a much less realistic attack setting in practice.

However, another interesting scenario is the chosen-key model, or when SPECK is used as an internal component to building a block-cipher-based compression/hash function, as proposed for example in [45] with the classical Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO), and Miyaguchi-Preneel (MP) configurations. In this setting, the adversary can attack the internal state and the key schedule parts independently, and the cost of a related-key differential trail reduces from $P_D \times P_K$ to $P_D + P_K$. We have conducted experiments to search for differential trails in this setting. Specifically, we looked for related-key differential trails that minimizes $\max\{P_D, P_K\}$ instead of P_{DK} and we get improvements compared to single-key ones.

Table 2 shows the differential trail probabilities found in the related-key scenario for SPECK minimizing $\max\{P_D, P_K\}$ and that can be used to mount distinguishers in the chosen-key model. We started by selecting the lowest probabilities between P_D and P_K for every specific version of SPECK in the related-key scenario, i.e., those presented in Table 3. Using these values as lower bounds, we launch experiments to search for related-key differential trails, but instead of minimizing $P_D \times P_K$, we minimize $P_D + P_K$. This way, we discovered differential trails for SPECK-48/96 and SPECK-64/128. In this setting, different from the results obtained for Table 3, where all trails were found with $w_s = 0$, we manage to find a better differential trail for SPECK-48/96 reduced to 15 rounds only by using $w_s = 1$. In this setting, for other versions SPECK we did not manage to find better trails than those that we already presented in Table 3.

The differential trails obtained in Table 2 can lead to simple distinguishers in the chosen-key model. However, these results can be enhanced using more sophisticated strategies such as the start-from-the-middle technique and the application of truncated differences at the extremities to identify limited-birthday distinguishers [28, 29]. Implementing this approach requires a different minimization function for the MILP encoding and a unique counting method for the SAT encoding.

We employ the start-from-the-middle strategy to derive new distinguishers on `SPECK`. In this strategy, the attacker gains access to certain rounds in the "middle" of the cipher, specifically within the key-schedule process and the permutation process. These rounds are referred to as *controlled* rounds. This strategy is based on the observation that higher probability weights are generally concentrated in a sequence of cipher rounds. By identifying these rounds, the attacker can leverage the degrees of freedom within them, allowing the extremities to be verified probabilistically.

Different configurations of the window heuristic can be applied to each round of ARX ciphers. Using the start-from-the-middle approach, we find distinguishers by enabling the attacker to search for differential trails with the controlled rounds set to $ws = -1$, permitting higher values in these rounds while restricting the window size to be less than or equal to 0, 1, 2, or 3 for the other rounds. This choice is based on the observation that the best probability weights for differential trails in `SPECK` occur with window sizes not exceeding 3. We create a window for the controlled rounds and move it across the rounds to determine where the best distinguishers can be found.

As the controlled rounds are manipulated by the attacker, the probability weight of the distinguisher is calculated only from the rounds outside the controlled rounds. To select the number of controlled rounds, we consider the parameter m of the `SPECK` cipher (refer to Section 2). Since the attacker can select m words, each assigned to a different round, we choose a number of controlled rounds close to m. Specifically, we select the following pairs for the key-schedule and permutation controlled rounds: (4, 1), (5, 2), and (6, 3).

We encode all the described constraints and configurations into SAT CNF clauses and apply them to `SPECK`-48/96. We select this version of `SPECK` due to its higher m value and because it provides results for constructing distinguishers in the chosen-key model without considering the start-from-the-middle approach (see Table 2). We discovered a distinguisher for 19 rounds with a probability weight of 43, with $ws = 0$, and 5 controlled rounds in the key-schedule and 2 in the permutation. Namely, we are able to produce a pair of plaintexts/ciphertexts/keys (P, C, K) and (P', C', K') such that the exact differences predicted by the differential path $(P \oplus P', C \oplus C', K \oplus K')$ is present. As described in [28, 29], in the generic case the attacker can't use the birthday search strategy (since all differences are fixed) and will require to use 2^{48} cipher calls to achieve the same goal. One can find this distinguisher in the repository accompanying this paper.

Interpretation of some "timing jumps" in the experiments. Throughout our experiments searching for differential trails in `SPECK` within the single-key scenario, a notable spike in timings can be observed starting around round 6 to round 8. This is evident in Table 1 as well as in Tables 16, 18, 20, 22 and 24 in Section D. We attribute this timing anomaly to the diffusion properties inherent

Table 2: Differential trail probabilities in the related-key scenario for SPECK found minimizing $\max\{P_D, P_K\}$ and that can be used to mount distinguishers in the chosen-key model.

Version	**Rounds**	$\min\{\log_2(P_D), \log_2(P_K)\}$	$\log_2(P_D)$	$\log_2(P_K)$	w_s
SPECK-48/96	14	-33	-33	-32	0
SPECK-48/96	15	-38	-38	-37	1
SPECK-64/128	15	-40	-40	-37	0

Version	**Rounds**	$\log_2(P_{DK})$	$\log_2(P_D)$	$\log_2(P_K)$	**Ref.**
SPECK-32/64	15	-94.0	-32.0	-62.0	[42]
	15	-85.0	-32.0	-53.0	[41]
	15	-73.0	-31.0	-42.0	$w_s = 0$
SPECK-48/96	14	-68.0	-43.0	-25.0	[42]
	14	-66.6	-43.0	-23.6	[41]
	14	-65.0	-34.0	-31.0	$w_s = 0$
	15	-89.0	-46.0	-43.0	[42]
	15	-83.5	-42.0	-41.5	[41]
	15	-75.0	-41.0	-34.0	$w_s = 0$
	16	-86.0	-43.0	-43.0	$w_s = 0$
SPECK-64/128	14	-88.0	-37.0	-51.0	[42]
	14	-72.0	-35.0	-37.0	[41]
	14	-66.0	-32.0	-34.0	$w_s = 0$
	15	-105.0	-45.0	-60.0	[42]
	15	-89.0	-42.0	-47.0	[41]
	15	-76.0	-33.0	-43.0	$w_s = 0$
	16	-103.0	-60.0	-43.0	[42]
	16	-85.0	-39.0	-46.0	$w_s = 0$
	17	-112.0	-62.0	-50.0	[42]
SPECK-128/256	16	-121.0	-76.0	-45.0	[42]
	16	-96.0	-52.0	-44.0	$w_s = 0$
	19	-190.0	-111.0	-79.0	[42]
	19	-171.0	-102.0	-69.0	$w_s = 0$

Table 3: Upper-bound comparison on the differential trails probability in the related-key scenario for SPECK variants. w_s denotes the window size. P_D represents the probability of the differential trail for data encryption. P_K represents the probability of the differential trail for the key-schedule process, while $P_{DK} = P_D \times P_K$.

to SPECK. Taking SPECK-128-128 as an example, its Avalanche Factor (AF) becomes prominent around rounds 7 and 8, as detailed in Table 1 of [19]. This diffusion characteristic of SPECK likely accounts for the observed jump in the timings. Furthermore, when juxtaposing our timings with those from two other studies, a consistent pattern emerges. This congruent behavior can be seen, for instance, in Tables 18-20 of [46].

4.2 Application to ChaCha

Search for differential trails in the single-key scenario. Differential cryptanalysis of ChaCha is notoriously difficult; not only is the choice of input difference positions restricted to one row out of four in the single-key scenario, but the high number of modular addition results in a combinatorial explosion that renders the search of long trails difficult. Therefore, ChaCha is a valuable benchmark to assess the strength of our heuristic. The main point of comparison is the single-key trails for ChaCha reduced to 1, 2, 3, and 4 rounds documented [7].

The summarized results are presented in Table 4. We managed to identify differential trails spanning up to four rounds with $w_s = 1$. Remarkably, our 3-round distinguisher outperforms the best known pure differential distinguisher by a factor 2^{27} (Table 26). Even more remarkably, we outperformed by a factor 2^{98} the 4-round distinguisher presented in [7].

Number of rounds	**MILP + window heur.**		**SAT + window heuristic + "full window size"**	**MILP** [7]	**S-function** [7]
	Pr	w_s	Pr	Pr	Pr
1	3	3	3	3	3
2	37	3	37	37	37
3	120	1	120	147	157
4	218	1	217	316	349

Table 4: Comparative results of differential trail weight probabilities with [7] for ChaCha in a single-key scenario. Probability is denoted in $-\log_2(\cdot)$.

These trails were found with a combination of sequential and parallel SAT and MILP implementations of the heuristic. More specifically, for 1 and 2 rounds, we allocated 16 cores to MILP (with $w_s = 3$) and 1 to SAT. For 3 and 4 rounds, we did could not find new bounds with the MILP model, even though the search used 80 cores (16 for each w_s value, from -1 to 3). The same bounds were found for ChaCha reduced to 3 rounds by using the parallel implementation of our tool for SAT. Surprisingly, the bound 2^{-218} for ChaCha reduced to 4 rounds was found using the parallel MILP implementation of our heuristic, but was not found in either the sequential or the parallel SAT versions of our tool. This might be a sign that our encoding method of the proposed heuristic still has room for improvement. We leave this as future research since exploring deeply how internal algorithms of MILP, or SAT, solvers are not in the scope of this paper. Individual timings and results for MILP and SAT can be found in Table 5 and Table 6 respectively. In Table 6, the symbol (-) indicates that we were unable to determine the probability weights listed in the fifth column even after one day.

r	Single-key			
	no-condition		conditions	
	Time	Pr	Time	Pr
1	2.17s	3	1.62s	3
2	13.2s	37	3.30s	37(0)
3	13.6s	147	319s	120(0)
4	36728s	316	48745s	218(1)

Table 5: Individual timing results for `ChaCha-512-256` using MILP. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

r	Single-key			
	no-condition		conditions	
	Time	Pr	Time	Pr
1	0.62s	3	0.3s	3
2	108s	37	1.81s	37(0)
3	-	-	95.0s	120(0)
4	-	-	-	218(1)

Table 6: Individual timing results for `ChaCha-512-256` using SAT. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

Window Heuristic constraining the number of "full windows". We conducted experiments similar to those described in Subsection 4.1, utilizing the window heuristic with constraints on the number of "full window sizes". This time, we constrained the number of "full window sizes" to be at most 7 to search for differential trails on `ChaCha` reduced to four rounds. Using ParKissat (with 16 cores), we found a trail in approximately 4.5 hours with a probability weight of 217 (see Table 27), which is better than the one found using MILP. The details of this trail are available in the repository accompanying this paper.

Boomerang distinguishers. In order to showcase the potential of the window heuristic, we applied it to the search of *boomerang distinguishers*, as defined in Subsection 2.1. In particular, we extended the ARX-oriented automatic search tool of [48], which adapts the notion of BCT table and its derivatives to ARX, to include the window heuristic. More specifically, we build boomerang distinguishers for 2 to 6 rounds of `ChaCha`. In the single-key scenario, the input difference is constrained to the last row of the state; in contrast, the difference in intermediate rounds is unconstrained, leaving more freedom in the bottom part of the distinguisher. Therefore, the choice of the middle round is crucial to the quality of the distinguisher.

We wanted to see how the probabilities shifted with different round numbers after cutting the cipher into top and bottom parts. If the probabilities were balanced, it meant a good split point. Our tests showed that the best places to split `ChaCha` are as follows: after the first round for 2 and 3 rounds, and after the second for 4, 5, and 6 rounds distinguishers. To make these distinguishers even better, we applied the BCT methods from [48] to the first modular addition in the bottom part. In Figure 1, we show an example of where we applied the BCT method for rounds 2, 3.

A summary of our results by using these techniques and the window heuristic is presented in Table 7. The details and differentials used in that table can be found in `https://anonymous.4open.science/r/Improved-MILP-model-for-Modular-Addition-on-ARX-ciphers-FA81`.

Table 7: New boomerang distinguishers for ChaCha.

Number of rounds	Theoretical probability	Experimental probability
2	2^{-6} ($w_s = -1$, bottom part starting round 2)	$2^{-0.05}$
3	2^{-10} ($w_s = -1$, bottom part starting round 2)	$2^{-1.41}$
4	2^{-90} ($w_s = 0$, bottom part starting round 3)	-
5	2^{-154} ($w_s = 1$, bottom part starting round 3)	-
6	2^{-228} ($w_s = 1$, bottom part starting round 3)	-

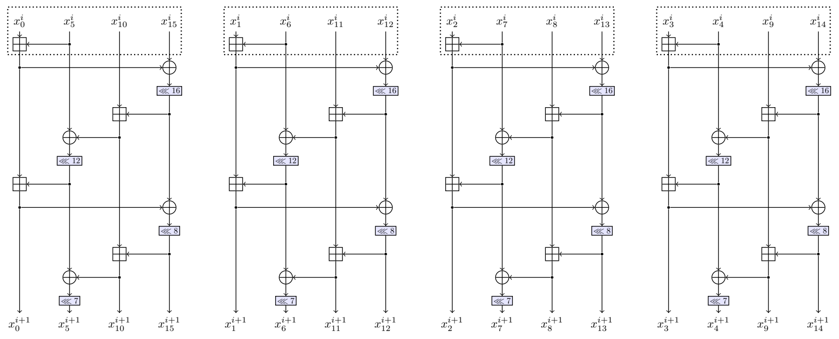

Fig. 1: The dotted lines highlight the BCT utilized to create the boomerang distinguishers in ChaCha for an odd round.

4.3 Application to LEA

We use the window heuristic to search for differential trails on LEA-128, improving the state-of-the-art differential characteristic for LEA-128 reduced to 13 rounds, by a factor of 2^{11} without taking into consideration the differential effect, and by a factor of 2^4 taking into account the differential effect. Individual timings and results for MILP can be found in Table 8

We used the aforementioned trail for LEA-128 and the strategy proposed in [21, 43], to mount a key recovery attack against LEA-128 reduced to 14 rounds. Specifically, given a r-round differential characteristic of LEA-128 with probability $p > 2 \times 2^{-128}$, the attack recovers the key LEA-128 reduced to $(r+1)$ rounds with $2 \times p^{-1}$ plaintexts, in expected time complexity of $2 \times p^{-1}$ encryptions. Using the probability $p = 2^{-123}$ of our distinguisher and $r = 13$, we have a distinguisher with an expected time complexity of $2^{111.97}$ after taking into account the differential effect similar to how it was done in [43] (see Table 9). This complexity outperforms the previously pure differential known key-recovery attack for LEA-128 reduced to 14 rounds by a factor 2^{11}. A summary, of our distinguishers and key-recovery attack against LEA can be found in Table 10.

r	Window size		Others			
	Pr	w_s	Pr	Ref.	Pr	Ref.
12	107	9	112	[43]	107	[3]
13	123	1	134	[43]	127	[3]

r	Single-key			
	no-condition		conditions	
	Time	Pr	Time	Pr
12	100709s	107/(-1)	73405s	107/(9)
13	609.53s	143/(-1)	7985s	123/(1)

Table 8: Differential trails weight probabilities results and comparisons for LEA-128 (top) and individual timing results using MILP (bottom). Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

weight	#sol.	Pr.	$\boldsymbol{\Sigma}_{\mathrm{acc}}$
123	22	-118.54	-118.54
124	167	-116.62	-116.28
125	720	-115.51	-114.84
126	2783	-114.56	-113.69
127	5926	-114.47	-113.03
128	30785	-113.09	-112.06
129	79668	-112.72	-111.35
130	124094	-113.08	-110.97

Table 9: Differential effect of the 13-round differential trail of LEA-128. Pr. and $\boldsymbol{\Sigma}_{\mathrm{acc}}$ are denoted in $\log_2(\cdot)$.

Distinguisher			Key recovery		
Rounds	Pr	Ref.	Rounds	Complexity	Ref.
13	$2^{-123.79}$	[43]	14	$2^{124.79}$	[43]
13	$2^{-110.97}$	Ours ($w_s = 1$)	14	$2^{111.97}$	Ours ($w_s = 1$)

Table 10: Upper-bound probabilities and key-recovery complexities comparison on LEA using pure differentials. w_s denotes the window size.

4.4 Application to HIGHT

We applied the window heuristic to HIGHT-64-128; the corresponding results are given in Table 11, and the individual timings in Table 12, where (-) denotes no results within 24 hours.

A window size of 7 was used to identify the differential trail 11 rounds with the MILP model. Interestingly, this window size significantly deviates from the typical values we have adopted when seeking enhanced trails for other ciphers discussed in this paper. Yet, with a window size of three and through the application of SAT techniques, we the best documented trail in the existing literature.

5 Discussions, conclusions and future work

In this work, we proposed a generalization of the linearization technique used to find differential trails in ARX ciphers. Our technique allows us to significantly decrease the space size during a tree-like search for good differential trails, by focusing on those trails that have a small number of consecutive carry-bit differences. As observed in the literature and in our experiments, and with few

Number of rounds	**Single-key**		
	ours (MILP)	**ours (SAT)**	**heuristic**
11	45/(7)	45/(3)	45/([35])
12	54/(7)	-	53/(splicing [3])([35])

Table 11: Differential paths probabilities results (denoted in $-\log_2(\cdot)$) and comparisons for `HIGHT-64-128`. Between parenthesis is given the window size for this result.

Number of rounds	**Single-key (MILP)**				**Single-key (SAT)**			
	no-condition		**conditions**		**no-condition**		**conditions**	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
11	1635s	46	24408s	45(7)	703s	45	374s	45(3)
12	42s	55	13s	54(7)	-	-	-	-

Table 12: Individual timing results for `HIGHT-64-128` using MILP. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

exceptions, this property seems to be enough to quickly spot trails with the best (often optimal) probability. We demonstrate this claim by showcasing novel results in the case of `ChaCha`, `SPECK`, and `LEA`.

We would like to emphasize that our method is different from [32]. While both use the word "window", this article's strategy is very different and has a different goal from ours. The technique presented in [32], forces the differences to only appear in a small window of the state words of the differential trail to exploit the strong clustering effect in `SIMON` and `SPECK` (because of the structure of these ciphers). In our case, the differences in the state can appear anywhere. We are restricting the consecutive carry-bit differences in the modular additions. Our technique is relevant for any ARX cipher (it would actually not apply to `SIMON` as it is not ARX).

Looking ahead, our techniques open several possibilities for future research in automated cryptanalysis tools. In particular, one could try different heuristics, such as limiting the total number of carries, not necessarily consecutive or by limiting the total number of window heuristic sizes summing to n. Another research direction could be to investigate the effect of the window heuristic on the search for other types of trails, such as linear trails or truncated differential trails. Regarding the applications, interestingly, we could push the boundaries of the basic boomerang distinguishers against `ChaCha`, now competing with known differential-linear distinguishers. Another application we explore in this work is to use SAT encoding to search for good differential paths on ARX ciphers to be used in the chosen-key scenario. This require taking into account the possibility for the attacker to start from the middle of the path. The window heuristic's flexibility enabled us to discover distinguishers in this setting. A future research in this direction is the usage of truncated differences on both extremities of the differential path (to be used as limited-birthday distinguishers [28, 29]). This would basically boil down to building a new and much more involved minimization function in the SAT/MILP encoding.

References

1. Aaraj, N., Caullery, F., Manzano, M.: MILP-aided Cryptanalysis of Round Reduced ChaCha. Cryptology ePrint Archive, Report 2017/1163 (2017), https://ia.cr/2017/1163
2. Aumasson, J.P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In: Nyberg, K. (ed.) Fast Software Encryption. pp. 470–488. Springer (2008)
3. Bagherzadeh, E., Ahmadian, Z.: MILP-Based Automatic Differential Searches for LEA and HIGHT. Cryptology ePrint Archive, Report 2018/948 (2018), https://ia.cr/2018/948
4. Beaulieu, R., Treatman-Clark, S., Shors, D., Weeks, B., Smith, J., Wingers, L.: The SIMON and SPECK lightweight block ciphers. In: DAC 2015. pp. 1–6. IEEE (2015)
5. Bellini, E., Gerault, D., Grados, J., Huang, Y.J., Rachidi, M., Tiwari, S., Makarim, R.H.: CLAASP: a Cryptographic Library for the Automated Analysis of Symmetric Primitives. Cryptology ePrint Archive, Paper 2023/622 (2023), https://eprint.iacr.org/2023/622
6. Bellini, E., Gérault, D., Protopapa, M., Rossi, M.: Monte Carlo Tree Search for Automatic Differential Characteristics Search: Application to SPECK. In: Isobe, T., Sarkar, S. (eds.) INDOCRYPT 2022. LNCS, vol. 13774, pp. 373–397. Springer (2022)
7. Bellini, E., Vasquez, J.D.C.G., Makarim, R., Sanna, C.: Finding differential trails on ChaCha by means of state functions. International Journal of Applied Cryptography (IJACT) (2023)
8. Bernstein, D.J.: Salsa20 specification. eSTREAM Project algorithm description, http://www.ecrypt.eu.org/stream/salsa20pf.html (2005)
9. Bernstein, D.J.: ChaCha, a variant of Salsa20. In: Workshop Record of SASC. vol. 8, pp. 3–5 (2008)
10. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology **4**(1), 3–72 (1991)
11. Biryukov, A., Velichkov, V.: Automatic search for differential trails in ARX ciphers. In: CT-RSA 2014. pp. 227–250. Springer (2014)
12. Biryukov, A., Cannière, C.D., Dellkrantz, G.: Cryptanalysis of SAFER++. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 195–211. Springer (2003)
13. Biryukov, A., Khovratovich, D.: Related-Key Cryptanalysis of the Full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 1–18. Springer (2009), https://doi.org/10.1007/978-3-642-10366-7_1
14. Biryukov, A., Roy, A., Velichkov, V.: Differential analysis of block ciphers SIMON and SPECK. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 546–570. Springer, Heidelberg (Mar 2015)
15. Biryukov, A., Velichkov, V.: Automatic Search for Differential Trails in ARX Ciphers. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 227–250. Springer (2014)
16. Biryukov, A., Velichkov, V., Corre, Y.L.: Automatic Search for the Best Trails in Arx: Application to Block Cipher Speck. Cryptology ePrint Archive, Report 2016/409 (2016), https://ia.cr/2016/409

17. Bittner, P.M., Thüm, T., Schaefer, I.: Sat encodings of the at-most-k constraint: A case study on configuring university courses. In: SEFM 2019. pp. 127–144. Springer (2019)
18. Chabaud, F., Joux, A.: Differential collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 56–71. Springer, Heidelberg (Aug 1998). https://doi.org/10.1007/BFb0055720
19. Coutinho, M., de Sousa Júnior, R.T., Borges, F.: Continuous Diffusion Analysis. IEEE Access **8**, 123735–123745 (2020), https://doi.org/10.1109/ACCESS.2020.3005504
20. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: General results and applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (Dec 2006)
21. Dinur, I.: Improved Differential Cryptanalysis of Round-Reduced Speck. IACR Cryptol. ePrint Arch. p. 320 (2014), http://eprint.iacr.org/2014/320
22. Dunkelman, O.: Efficient construction of the boomerang connection table. IACR Cryptol. ePrint Arch. p. 631 (2018), https://eprint.iacr.org/2018/631
23. Dunkelman, O., Keller, N., Shamir, A.: A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 393–410. Springer (2010)
24. Dwivedi, A.D., Srivastava, G.: Differential Cryptanalysis of Round-Reduced LEA. IEEE Access **6**, 79105–79113 (2018)
25. Fischer, S., Meier, W., Berbain, C., Biasse, J.F., Robshaw, M.J.B.: Non-randomness in eSTREAM Candidates Salsa20 and TSC-4. In: Barua, R., Lange, T. (eds.) Progress in Cryptology - INDOCRYPT 2006. pp. 2–16. Springer (2006)
26. Fu, K., Wang, M., Guo, Y., Sun, S., Hu, L.: MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 268–288. Springer (2016), https://doi.org/10.1007/978-3-662-52993-5_14
27. Gérault, D., Lafourcade, P., Minier, M., Solnon, C.: Revisiting AES related-key differential attacks with constraint programming. Inf. Process. Lett. **139**, 24–29 (2018)
28. Gilbert, H., Peyrin, T.: Super-sbox cryptanalysis: Improved attacks for AES-like permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (Feb 2010)
29. Iwamoto, M., Peyrin, T., Sasaki, Y.: Limited-birthday distinguishers for hash functions - collisions beyond the birthday bound can be meaningful. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 504–523. Springer, Heidelberg (Dec 2013)
30. Leurent, G.: Analysis of Differential Attacks in ARX Constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 226–243. Springer (2012)
31. Leurent, G.: Construction of differential characteristics in ARX designs application to Skein. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 241–258. Springer, Heidelberg (Aug 2013)
32. Leurent, G., Pernot, C., Schrottenloher, A.: Clustering Effect in Simon and Simeck. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13090, pp. 272–302. Springer (2021), https://doi.org/10.1007/978-3-030-92062-3_10
33. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In: FSE 2001, LNCS. vol. 2355, pp. 336–350. Springer (2001)
34. Lipmaa, H., Moriai, S.: Efficient algorithms for computing differential properties of addition. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 336–350. Springer, Heidelberg (Apr 2002)

35. Liu, Z., Li, Y., Jiao, L., Mingsheng, W.: A New Method for Searching Optimal Differential and Linear Trails in ARX Ciphers. IEEE Transactions on Information Theory **67**(2), 1054–1068 (2021). https://doi.org/10.1109/TIT.2020.3040543
36. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT'93. LNCS, vol. 765, pp. 386–397. Springer (1993)
37. Mouha, N., Preneel, B.: Towards finding optimal differential characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Report 2013/328 (2013), https://eprint.iacr.org/2013/328
38. Murphy, S.: The Return of the Cryptographic Boomerang. IEEE Trans. Inf. Theory **57**(4), 2517–2521 (2011), https://doi.org/10.1109/TIT.2011.2111091
39. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a Standard CP Modelling Language. In: Bessière, C. (ed.) CP 2007. pp. 529–543. Springer (2007)
40. Perron, L., Furnon, V.: OR-Tools, https://developers.google.com/optimization/
41. Qin, H., Wu, B.: Towards non-independence of modular additions in searching differential trails of ARX ciphers: new automatic methods with application to SPECK and Chaskey. CoRR **abs/2203.09741** (2022), https://doi.org/10.48550/arXiv.2203.09741
42. Sadeghi, S., Rijmen, V., Bagheri, N.: Proposing an MILP-based method for the experimental verification of difference-based trails: application to SPECK, SIMECK. Des. Codes Cryptogr. **89**(9), 2113–2155 (2021)
43. Song, L., Huang, Z., Yang, Q.: Automatic Differential Analysis of ARX Block Ciphers with Application to SPECK and LEA. In: Liu, J.K., Steinfeld, R. (eds.) ACISP 2016. LNCS, vol. 9723, pp. 379–394. Springer (2016)
44. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT Solvers to Cryptographic Problems. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 244–257. Springer (2009)
45. Su, Y., Gao, Y., Kavehei, O., Ranasinghe, D.C.: Hash Functions and Benchmarks for Resource Constrained Passive Devices: A Preliminary Study. In: 2019 IEEE PerCom Workshop. pp. 1020–1025 (2019)
46. Sun, L., Wang, W., Wang, M.: Accelerating the Search of Differential and Linear Characteristics with the SAT Method. IACR Transactions on Symmetric Cryptology **2021**(1), 269–315 (Mar 2021), https://tosc.iacr.org/index.php/ToSC/article/view/8840
47. Wagner, D.A.: The Boomerang Attack. In: Knudsen, L.R. (ed.) FSE '99. LNCS, vol. 1636, pp. 156–170. Springer (1999)
48. Wang, D., Wang, B., Sun, S.: SAT-aided Automatic Search of Boomerang Distinguishers for ARX Ciphers (Long Paper). IACR Cryptol. ePrint Arch. p. 202 (2023), https://eprint.iacr.org/2023/202
49. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (Aug 2005)
50. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (May 2005)
51. Zhang, Y., Sun, S., Cai, J., Hu, L.: Speeding up milp aided differential characteristic search with matsui's strategy. In: Chen, L., Manulis, M., Schneider, S. (eds.) Information Security. pp. 101–115. Springer International Publishing, Cham (2018)

Supplementary material

This supplementary section provides additional materials that support the main text. It includes extended analyses, detailed data tables, additional figures, or other relevant information that complements the results and discussions presented in the main part of this paper.

A Finding differential trails using SAT and MILP techniques

For each bit position $j \in \{0, \ldots, n-2\}$, the inequalities used to model the propagation of differences in a modular addition with input differences a and b and output difference c are given as follows:

$$\begin{aligned}
a[j+1] + b[j+1] - c[j+1] + a[j] + b[j] + c[j] + d[j] &\geq -0 \\
a[j+1] - b[j+1] + c[j+1] + a[j] + b[j] + c[j] + d[j] &\geq -0 \\
-a[j+1] + b[j+1] + c[j+1] + a[j] + b[j] + c[j] + d[j] &\geq -0 \\
-a[j+1] - b[j+1] - c[j+1] + a[j] + b[j] + c[j] - d[j] &\geq -3 \\
a[j+1] + b[j+1] + c[j+1] + a[j] + b[j] + c[j] - d[j] &\geq -0 \\
a[j+1] + b[j+1] + c[j+1] - a[j] - b[j] - c[j] + d[j] &\geq -6 \\
-a[j+1] - b[j+1] - c[j+1] + a[j] - b[j] - c[j] + d[j] &\geq -6 \\
-a[j+1] - b[j+1] - c[j+1] - a[j] + b[j] - c[j] + d[j] &\geq -6 \\
-a[j+1] - b[j+1] - c[j+1] - a[j] - b[j] + c[j] + d[j] &\geq -6 \\
a[j+1] + b[j+1] + c[j+1] + a[j] + b[j] - c[j] + d[j] &\geq -0 \\
a[j+1] + b[j+1] + c[j+1] + a[j] - b[j] + c[j] + d[j] &\geq -0 \\
a[j+1] + b[j+1] + c[j+1] - a[j] + b[j] + c[j] + d[j] &\geq -0 \\
a[j+1] + b[j+1] - c[j+1] + a[j] + b[j] + c[j] + d[j] &\geq -0.
\end{aligned} \tag{6}$$

Here $d[j]$ is used to model the probability weight variable of the differential (see Section 2).

For each bit position $j \in \{0, \ldots, n-2\}$, the CNF clauses to model the propagation of differences in a modular addition with input differences a and b and output difference c are given as follows:

$$
\begin{aligned}
a[j] \vee b[j] \vee \neg c[j] \vee a[j+1] \vee b[j+1] \vee c[j+1] &= 1,\\
a[j] \vee \neg b[j] \vee c[j] \vee a[j+1] \vee b[j+1] \vee c[j+1] &= 1,\\
\neg a[j] \vee b[j] \vee c[j] \vee a[j+1] \vee b[j+1] \vee c[j+1] &= 1,\\
\neg a[j] \vee \neg b[j] \vee \neg c[j] \vee a[j+1] \vee b[j+1] \vee c[j+1] &= 1,\\
a[j] \vee b[j] \vee c[j] \vee \neg a[j+1] \vee \neg b[j+1] \vee \neg c[j+1] &= 1,\\
a[j] \vee \neg b[j] \vee \neg c[j] \vee \neg a[j+1] \vee \neg b[j+1] \vee \neg c[j+1] &= 1,\\
\neg a[j] \vee b[j] \vee \neg c[j] \vee \neg a[j+1] \vee \neg b[j+1] \vee \neg c[j+1] &= 1,\\
\neg a[j] \vee \neg b[j] \vee c[j] \vee \neg a[j+1] \vee \neg b[j+1] \vee \neg c[j+1] &= 1,\\
\neg a[j+1] \vee c[j+1] \vee w[j] &= 1,\\
b[j+1] \vee \neg c[j+1] \vee w[j] &= 1,\\
a[j+1] \vee \neg b[j+1] \vee w[j] &= 1,\\
a[j+1] \vee b[j+1] \vee c[j+1] \vee \neg w[j] &= 1,\\
\neg a[j+1] \vee \neg b[j+1] \vee \neg c[j+1] \vee \neg w[j] &= 1.
\end{aligned}
$$

Here $w[j]$ is used to model the probability weight variable of the differential (see Section 2).

B Description of `ChaCha`, `SPECK` and `LEA`

B.1 Description of ChaCha

In [9], Bernstein presented `ChaCha`, now one of the most famous ARX ciphers. This stream cipher works with a 128-bit or 256-bit key and was constructed to improve the diffusion properties of `Salsa` [8] while using about the same amount of operations. `ChaCha` works on a (4×4) matrix of 32-bit words (denoted $x_0, \dots, x_{15}$ as seen in Table 13b), and its initial state comprises four constants words $c_1 =$ `0x61707865`, $c_2 =$ `0x3320646e`, $c_3 =$ `0x79622d32`, $c_4 =$ `0x6b206574`, one counter word, three nonce words, and eight key words for the 256-bit key. The distribution of these words is presented in Table 13a.

c_1	c_2	c_3	c_4
key 1	key 2	key 3	key 4
key 5	key 6	key 7	key 8
counter	nonce 1	nonce 2	nonce 3

(a) Initial state of `ChaCha`.

x_0^i	x_1^i	x_2^i	x_3^i
x_4^i	x_5^i	x_6^i	x_7^i
x_8^i	x_9^i	x_{10}^i	x_{11}^i
x_{12}^i	x_{13}^i	x_{14}^i	x_{15}^i

(b) x^i state of `ChaCha`.

ChaCha is an iterative stream cipher with 20 rounds, where each round updates the i^{th} state presented in Table 13b using the following equations, called quarter round (QR).

$$\begin{array}{llll} a = a + b, & d = d \oplus a, & d = d \lll 16, & c = c + d; \\ b = b \oplus c, & b = b \lll 12, & a = a + b, & d = d \oplus a; \\ d = d \lll 8, & c = c + d, & b = b \oplus c, & b = b \lll 7; \end{array}$$

If the round number is even, then the QR equations are applied to each column of the (4×4) word matrix. Thus, the state x^{i+1} can be written as:

$$\begin{aligned} (x_0^{i+1}, x_4^{i+1}, x_8^{i+1}, x_{12}^{i+1}) &= QR(x_0^i, x_4^i, x_8^i, x_{12}^i), \\ (x_1^{i+1}, x_5^{i+1}, x_9^{i+1}, x_{13}^{i+1}) &= QR(x_1^i, x_5^i, x_9^i, x_{13}^i), \\ (x_2^{i+1}, x_6^{i+1}, x_{10}^{i+1}, x_{14}^{i+1}) &= QR(x_2^i, x_6^i, x_{10}^i, x_{14}^i), \\ (x_3^{i+1}, x_7^{i+1}, x_{11}^{i+1}, x_{15}^{i+1}) &= QR(x_3^i, x_7^i, x_{11}^i, x_{15}^i) \end{aligned}$$

If the round number is odd, then the QR equations are applied to each diagonal of the (4×4) word matrix. Thus, in that case the state x^{i+1} can be written as:

$$\begin{aligned} (x_0^{i+1}, x_5^{i+1}, x_{10}^{i+1}, x_{15}^{i+1}) &= QR(x_0^i, x_5^i, x_{10}^i, x_{15}^i), \\ (x_1^{i+1}, x_6^{i+1}, x_{11}^{i+1}, x_{12}^{i+1}) &= QR(x_1^i, x_6^i, x_{11}^i, x_{12}^i), \\ (x_2^{i+1}, x_7^{i+1}, x_8^{i+1}, x_{13}^{i+1}) &= QR(x_2^i, x_7^i, x_8^i, x_{13}^i), \\ (x_3^{i+1}, x_4^{i+1}, x_9^{i+1}, x_{14}^{i+1}) &= QR(x_3^i, x_4^i, x_9^i, x_{14}^i) \end{aligned}$$

B.2 Description of SPECK

SPECK and SIMON are two families of lightweight block ciphers proposed by the National Security Agency in 2013 [4], with the former being of ARX type in order to obtain good performances in constrained software architectures. The SPECK family has several instances, depending on the block and key sizes. If the block size is $2n$ and the key size is mn, the cipher is denoted as SPECK-$2n/mn$ (see Table 14 for the various versions).

SPECK is a Feistel cipher, where the round function works with two halves as follows. Let (L^{i-1}, R^{i-1}) be the input of the i^{th} round, k_i be the i^{th} round subkey, the output of the i^{th} round (L^{i-1}, R^{i-1}) is then computed as follows:

$$L_i = F(L^{i-1}, R^{i-1}) \oplus k_i, \quad R_i = (R_{i-1} \lll \beta) \oplus L_i,$$

where $F(x, y) = (x \ggg \alpha) + y$, $\alpha = 7$ and $\beta = 2$ if the block size is 32-bit and $\alpha = 8$ and $\beta = 3$ otherwise. The key schedule part follows a similar process, and we refer you to [?] to see the details.

B.3 Description of LEA

LEA is a block cipher designed by Hong *et al.* [?], for high-speed encryption. This cipher has a block size of 128 bits and the key size could be 128, 192, or 256 bits.

Version	**block size** $(2n)$	**key size** (mn)	**rounds** (T)
SPECK-32/64	32	64	22
SPECK-48/72	48	72	22
SPECK-48/96	48	96	23
SPECK-64/96	64	96	26
SPECK-64/128	64	128	27
SPECK-96/92	96	96	28
SPECK-96/144	96	144	29
SPECK-128/128	128	128	32
SPECK-128/192	128	192	33
SPECK-128/256	128	256	34

Table 14: Variants of the `SPECK` family of block ciphers.

In this paper, we denote these instances by `LEA-128`, `LEA-192`, and `LEA-256`, respectively. The i^{th} state of this cipher comprises four 32-bit words. Its round function is composed of modular additions (modulus 2^{32}), XORs, and rotations. Let x_0^i, x_1^i, x_2^i, x_3^i be the inputs words at round i, then the output words can be described as:

$$x_0^{i+1} = ((x_0^i \oplus k_0^i) + (x_1^i \oplus k_1^i)) \lll 9,\ x_1^{i+1} = ((x_1^i \oplus k_2^i) + (x_2^i \oplus k_3^i)) \ggg 5,$$
$$x_2^{i+1} = ((x_2^i \oplus k_4^i) + (x_3^i \oplus k_5^i)) \lll 3, x_3^{i+1} = x_0^i.$$

where k_0^i, k_1^i, k_2^i, k_3^i, k_4^i, k_5^i are the subkey words of the round i. We refer to [**?**] for details on the key schedule process. Regarding the number of rounds, it is 24 for `LEA-128`, 28 for `LEA-192`, and 32 for `LEA-256`.

C Window size heuristic constraining the number "Full window"

Table 15: Differential Trail Results and number of "full window sizes"

Rounds	**Number of Full Window Size (k)**	**Block Size**	**Key Size**	**Window Size**	Pr **Weight**
9	3	32	64	2	30
10	1	32	64	3	34
11	3	48	96	2	45
12	3	48	96	2	49
11	1	64	128	3	42
12	1	64	128	3	46
13	1	64	128	3	50
16	1	64	128	3	71
11	1	96	96	3	58
12	1	128	128	3	68
13	1	128	128	3	78

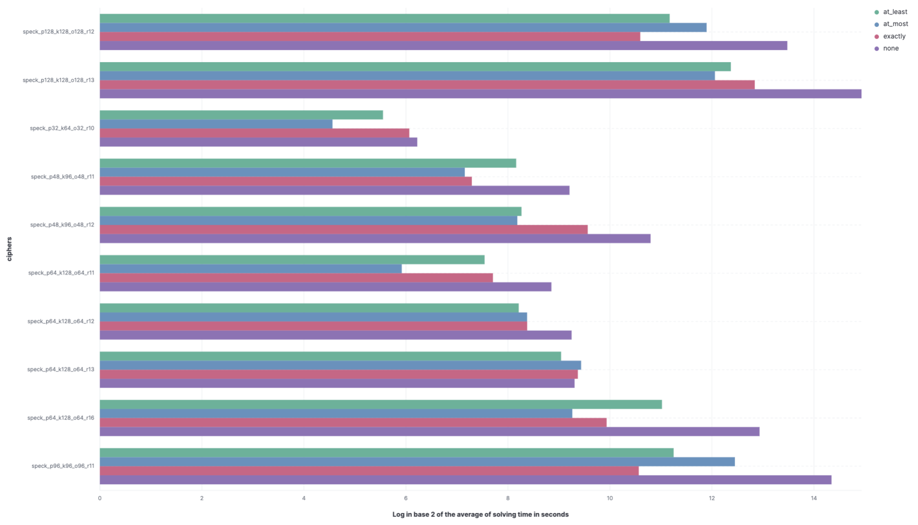

Fig. 2: Comparison of the average of the solving time in seconds (on a $\log_2$ scale) to search for differential trails on different versions of `SPECK` using the window heuristic constraining the number of "full window size" and using ParKiSat SAT solver.

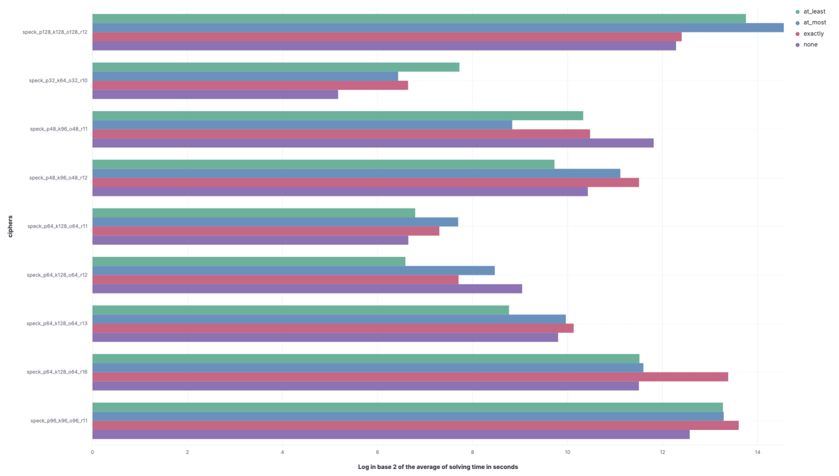

Fig. 3: Comparison of the average of the solving time in seconds (on a $\log_2$ scale) to search for differential trails on different versions of `SPECK` using the window heuristic constraining the number of "full window size" and using CaDiCal SAT solver.

D Result tables for `SPECK`

The following tables show the timing results we get for

Number of rounds	**Single-key**				**Related-key**			
	no-condition		**conditions**		**no-condition**		**conditions**	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
1	0.11s	0	0.11s	0(0)	0.10s	0	0.09s	0(0)
2	0.23s	1	0.25s	1(0)	0.17s	0	0.18s	0(0)
3	0.35s	3	0.40s	3(0)	0.24s	0	0.24s	0(0)
4	0.48s	5	0.55s	5(0)	0.31s	0	0.33s	0(0)
5	0.69s	9	0.72s	9(0)	0.37s	1	0.40s	1(0)
6	0.80s	13	0.80s	13(0)	0.83s	4	0.56s	4(0)
7	1.46s	18	0.91s	18(0)	0.90s	8	0.70s	8(0)
8	1.05s	24	0.75s	24(0)	5.65s	12	1.59s	12(0)
9	9.05s	30	172879s	30(2)	7.33s	17	0.93s	17(0)
10	420s	34	259589s	34(3)	6.81s	24	1.10s	24(0)
11	-	-	-	-	1217s	34	1.21s	35(0)

Table 16: Timing results for `SPECK-32-64`. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

D.1 Differential trails for `ChaCha`

Number of rounds	Single-key				Related-key			
	no-condition		conditions		no-condition		conditions	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
1	0.11s	0	0.11s	0(0)	0.10s	0	0.09s	0(0)
2	0.23s	1	0.25s	1(0)	0.17s	0	0.18s	0(0)
3	0.35s	3	0.40s	3(0)	0.24s	0	0.24s	0(0)
4	0.48s	5	0.55s	5(0)	0.31s	0	0.33s	0(0)
5	0.69s	9	0.72s	9(0)	0.37s	1	0.40s	1(0)
6	0.80s	13	0.80s	13(0)	0.83s	4	0.56s	4(0)
7	1.46s	18	0.91s	18(0)	0.90s	8	0.70s	8(0)
8	1.05s	24	0.75s	24(0)	5.65s	12	1.59s	12(0)
9	9.05s	30	70.3s	30(2)	7.33s	17	0.93s	17(0)
10	420s	34	115s	34(3)	6.81s	24	1.10s	24(0)
11	-	-	-	-	1217s	34	1.21s	35(0)

Table 17: Individual timing results for `SPECK-32-64`. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

Number of rounds	Single-key				Related-key			
	no-condition		conditions		no-condition		conditions	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
1	0.13s	0	0.13s	0(0)	0.12s	0	0.11s	0(0)
2	0.29s	1	0.35s	1(0)	0.21s	0	0.23s	0(0)
3	0.52s	3	0.52s	3(0)	0.33s	0	0.34s	0(0)
4	0.70s	6	0.84s	6(0)	0.44s	0	0.47s	0(0)
5	1.14s	10	0.99s	10(0)	0.55s	1	0.62s	1(0)
6	1.32s	14	1.02s	14(0)	0.79s	4	0.74s	4(0)
7	1.44s	19	1.15s	19(0)	1.36s	8	1.22s	8(0)
8	24.7s	26	1.26s	26(0)	8.52s	12	1.36s	12(0)
9	16.6s	33	86404s	33(1)	115s	18	86425s	18(1)
10	584s	40	86410s	40(1)	67.2s	25	4.85s	25(0)
11	5442s	45	1828875s	45(2)	-	-	-	-
12	30164s	49	197512s	49(2)	-	-	-	-

Table 18: Timing results for `SPECK-48-96`. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

Number of rounds	Single-key				Related-key			
	no-condition		conditions		no-condition		conditions	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
1	0.13s	0	0.13s	0(0)	0.12s	0	0.11s	0(0)
2	0.29s	1	0.35s	1(0)	0.21s	0	0.23s	0(0)
3	0.52s	3	0.52s	3(0)	0.33s	0	0.34s	0(0)
4	0.70s	6	0.84s	6(0)	0.44s	0	0.47s	0(0)
5	1.14s	10	0.99s	10(0)	0.55s	1	0.62s	1(0)
6	1.32s	14	1.02s	14(0)	0.79s	4	0.74s	4(0)
7	1.44s	19	1.15s	19(0)	1.36s	8	1.22s	8(0)
8	24.7s	26	1.26s	26(0)	8.52s	12	1.36s	12(0)
9	16.6s	33	2.77s	33(1)	115s	18	19.3s	18(1)
10	584s	40	6.36s	40(1)	67.2s	25	4.85s	25(0)
11	5442s	45	10069s	45(2)	198s	36	5.00s	36(0)
12	30164s	49	24591s	49(2)	-	-	-	-

Table 19: Individual timing results for `SPECK-48-96`. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

Number of rounds	Single-key				Related-key			
	no-condition		conditions		no-condition		conditions	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
1	0.18s	0	0.16s	0(0)	0.38s	0	0.39s	0(0)
2	0.43s	1	0.32s	1(0)	0.82s	0	0.97s	0(0)
3	0.72s	3	0.50s	3(0)	1.56s	0	1.58s	0(0)
4	0.96s	6	1.04s	6(0)	2.01s	0	2.18s	0(0)
5	1.35s	10	1.35s	10(0)	2.67s	1	2.78s	1(0)
6	1.77s	15	1.79s	15(0)	3.24s	4	3.33s	4(0)
7	3.56s	21	2.16s	21(0)	3.77s	8	4.10s	8(0)
8	5.71s	29	2.02s	29(0)	5.62s	12	6.39s	12(0)
9	24.4s	34	2.82s	34(0)	60.2s	18	7.62s	18(0)
10	10229s	38	5.93s	38(0)	236s	26	105s	26(0)
11	81.6s	42	312312s	42(3)	454s	38	26.7s	38(0)
12	7938s	46	262350s	46(3)	137s	52	2477s	49(0)
13	29566s	50	280784s	50(3)	35160s	57	240s	57(0)
14	59787s	56	264460s	56(4)	27021s	81	612s	66(0)
15	13931s	62	286405s	62(4)	-	-	-	-

Table 20: Timing results for `SPECK-64-128`. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

Number of rounds	**Single-key**				**Related-key**			
	no-condition		**conditions**		**no-condition**		**conditions**	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
1	0.18s	0	0.16s	0(0)	0.38s	0	0.39s	0(0)
2	0.43s	1	0.32s	1(0)	0.82s	0	0.97s	0(0)
3	0.72s	3	0.50s	3(0)	1.56s	0	1.58s	0(0)
4	0.96s	6	1.04s	6(0)	2.01s	0	2.18s	0(0)
5	1.35s	10	1.35s	10(0)	2.67s	1	2.78s	1(0)
6	1.77s	15	1.79s	15(0)	3.24s	4	3.33s	4(0)
7	3.56s	21	2.16s	21(0)	3.77s	8	4.10s	8(0)
8	5.71s	29	2.02s	29(0)	5.62s	12	6.39s	12(0)
9	24.4s	34	2.82s	34(0)	60.2s	18	7.62s	18(0)
10	10229s	38	5.93s	38(0)	236s	26	105s	26(0)
11	81.6s	42	1046s	42(3)	454s	38	26.7s	38(0)
12	7938s	46	2903s	46(3)	137s	52	2477s	49(0)
13	29566s	50	8409s	50(3)	35160s	57	240s	57(0)
14	59787s	56	5260s	56(4)	27021s	81	612s	66(0)
15	13931s	62	27205s	62(4)	-	-	-	-
16	25905s	72	285105s	71(3)		-	-	-

Table 21: Individual timing results for `SPECK-64-128`. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

Number of rounds	**Single-key**				**Related-key**			
	no-condition		**conditions**		**no-condition**		**conditions**	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
1	0.16s	0	0.17s	0(0)	0.21s	0	0.18s	0(0)
2	0.40s	1	0.40s	1(0)	0.35s	0	0.49s	0(0)
3	0.77s	3	0.66s	3(0)	0.59s	2	0.84s	2(0)
4	1.06s	6	0.98s	6(0)	1.02s	5	1.28s	5(0)
5	1.46s	10	1.22s	10(0)	4.77s	9	1.83s	9(0)
6	13.3s	15	1.66s	15(0)	9.22s	15	2.05s	15(0)
7	49.5s	21	1.66s	15(0)	247s	24	2.80s	24(0)
8	46.2s	30	2.39s	30(0)	1644s	34	10.2s	34(0)
9	512s	39	2.93s	39(0)	11897s	46	7.78s	46(0)
10	3978s	49	2.94s	49(0)	2423s	62	6.46s	62(0)
11	98416s	58	261915s	58(3)	44827s	78	435s	76(0)
12	83754s	62	-	-	831s	97	37s	92(0)
13	26181s	67	-	-	1927s	114	1439s	108(0)
14	107671s	75	-	-	-	-	-	-
15	107671s	81	86088s	80(7)	-	-	-	-
16	65553s	90	running	87(11)	-	-	-	-

Table 22: Timing results for `SPECK-96-96`. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

Number of rounds	Single-key				Related-key			
	no-condition		conditions		no-condition		conditions	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
1	0.16s	0	0.17s	0(0)	0.21s	0	0.18s	0(0)
2	0.40s	1	0.40s	1(0)	0.35s	0	0.49s	0(0)
3	0.77s	3	0.66s	3(0)	0.59s	2	0.84s	2(0)
4	1.06s	6	0.98s	6(0)	1.02s	5	1.28s	5(0)
5	1.46s	10	1.22s	10(0)	4.77s	9	1.83s	9(0)
6	13.3s	15	1.66s	15(0)	9.22s	15	2.05s	15(0)
7	49.5s	21	1.66s	15(0)	247s	24	2.80s	24(0)
8	46.2s	30	2.39s	30(0)	1644s	34	10.2s	34(0)
9	512s	39	2.93s	39(0)	11897s	46	7.78s	46(0)
10	3978s	49	2.94s	49(0)	2423s	62	6.46s	62(0)
11	98416s	58	2715s	58(3)	44827s	78	435s	76(0)
12	83754s	62	-	-	831s	97	37.0s	92(0)
13	26181s	67	-	-	1927s	114	1439s	108(0)
14	107671s	75	86088s	74(7)	-	-	-	-
15	107671s	81	86088s	80(7)	-	-	-	-

Table 23: Individual Timing results for SPECK-96-96. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

Number of rounds	Single-key				Related-key			
	no-condition		conditions		no-condition		conditions	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
1	0.37s	0	0.41s	0(0)	0.67s	0	0.70s	0(0)
2	0.87s	1	0.86s	1(0)	1.76s	0	1.83s	0(0)
3	1.38s	3	1.47s	3(0)	2.96s	2	3.12s	2(0)
4	2.34s	6	2.17s	6(0)	91.9s	24	7.18s	5(0)
5	3.73s	10	2.39s	10(0)	20.8s	9	8.70s	9(0)
6	6.5s	15	2.81s	15(0)	27.1s	15	12.5s	15(0)
7	506s	21	5.07s	21(0)	1206s	24	19.9s	24(0)
8	742s	30	11.4s	30(0)	2673s	38	720s	34(0)
9	4253s	39	5.72s	39(0)	5335s	58	499s	46(0)
10	55704s	49	23.9s	49(0)	420s	77	140s	62(0)
11	42204s	59	7.19s	59(0)	133s	146	108s	76(0)
12	3360s	68	286811s	66(3)	3818s	164	235s	92(0)
13	39539s	78	300421s	77(3)	1127s	112	13847s	109(0)
14	37.7s	99	20190s	96(2)	48003s	197	48698s	131(0)
15	709s	97	75.8s	120(0)	8988s	140	-	-
16	87493s	108	87472s	108(1)	95072s	324	74354s	181(0)
17	449s	134	89482s	114(1)	353s	503	1750s	206(0)
18	46794s	146	97867s	121(1)	-	-	-	-
19	20827s	154	90757s	128(1)	-	-	-	-

Table 24: Timing results for SPECK-128-128. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

Number of rounds	Single-key				Related-key			
	no-condition		conditions		no-condition		conditions	
	Time	Pr	Time	Pr	Time	Pr	Time	Pr
1	0.37s	0	0.41s	0(0)	0.67s	0	0.70s	0(0)
2	0.87s	1	0.86s	1(0)	1.76s	0	1.83s	0(0)
3	1.38s	3	1.47s	3(0)	2.96s	2	3.12s	2(0)
4	2.34s	6	2.17s	6(0)	91.9s	24	7.18s	5(0)
5	3.73s	10	2.39s	10(0)	20.8s	9	8.70s	9(0)
6	6.50s	15	2.81s	15(0)	27.1s	15	12.5s	15(0)
7	506s	21	5.07s	21(0)	1206s	24	19.9s	24(0)
8	742s	30	11.4s	30(0)	2672s	38	720s	34(0)
9	4253s	39	5.72s	39(0)	5335s	58	499s	46(0)
10	55704s	49	23.9s	49(0)	420s	77	140s	62(0)
11	42204s	59	7.19s	59(0)	133s	146	108s	76(0)
12	3360s	68	26988s	66(3)	3818s	164	235s	92(0)
13	39539s	78	16162s	77(3)	1127s	112	13847s	109(0)
14	37.7s	99	20190s	96(2)	48003s	197	48698s	131(0)
15	710s	97	75.8s	120(0)	8988s	140	-	-
16	87493s	108	200s	108(1)	95072s	324	87169s	178(1)
17	449s	134	2458s	114(1)	353s	503	1750s	206(0)
18	46794s	146	11467s	121(1)	-	-	-	-
19	20827s	154	43571s	128(1)	-	-	-	-

Table 25: Individual timing results for `SPECK-128-128`. Probability is denoted in $-\log_2(\cdot)$. Between parenthesis is given the window size for this result.

Round	Difference
-	`0x00000000, 0x00000000, 0x00000000, 0x00000000,` `0x00000000, 0x00000000, 0x00000000, 0x00000000,` `0x00000000, 0x00000000, 0x00000000, 0x00000000,` `0x80088008, 0x00000000, 0x80088008, 0x00000000`
1	`0x88008800, 0x00000000, 0x88008800, 0x00000000,` `0x00000000, 0x00000000, 0x00000000, 0x00000000,` `0x88008800, 0x00000000, 0x88008800, 0x00000000,` `0x08080808, 0x00000000, 0x08080808, 0x00000000`
2	`0x88008800, 0x80808080, 0x88008800, 0x80808080,` `0x00000000, 0x00000000, 0x00000000, 0x00000000,` `0x00000000, 0x80808080, 0x00000000, 0x80808080,` `0x88888888, 0x00000000, 0x88888888, 0x00000000`
3	`0x08080808, 0x80808080, 0x08080808, 0x80808080,` `0x44004400, 0x00000000, 0x44004400, 0x00000000,` `0x80808080, 0x00000000, 0x80808080, 0x00000000,` `0x80088008, 0x00000000, 0x80088008, 0x00000000`

Table 26: `ChaCha` differential trail over 3 rounds with a probability weight of 217. The trail was derived using the $w_s = 0$ and MILP techniques.

Round	Difference
-	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x80088008, 0x00000000, 0x80088008, 0x00000000
-	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x88008800, 0x00000000, 0x88008800, 0x00000000, 0x88008800, 0x00000000, 0x88008800, 0x00000000, 0x80088008, 0x00000000, 0x80088008, 0x00000000
1	0x88008800, 0x00000000, 0x88008800, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x88008800, 0x00000000, 0x88008800, 0x00000000, 0x08080808, 0x00000000, 0x08080808, 0x00000000
-	0x88008800, 0x00000000, 0x88008800, 0x00000000, 0x80808080, 0x00000000, 0x80808080, 0x00000000, 0x00000000, 0x08080808, 0x00000000, 0x08080808, 0x08080808, 0x88008800, 0x08080808, 0x88008800
2	0x88008800, 0x80808080, 0x88008800, 0x80808080, 0x00000000, 0x00000000, 0x00008000, 0x00000000, 0x81808080, 0x00000000, 0x80808180, 0x00000000, 0x88888888, 0x00000000, 0x88888888, 0x00000000
-	0x08008800, 0x80808080, 0x08008800, 0x80808080, 0x88088808, 0x00000000, 0x88088808, 0x00000000, 0x80088008, 0x00000000, 0x80088008, 0x00000000, 0x88808880, 0x80808080, 0x88008880, 0x80808080
3	0x80080008, 0x81808080, 0x80008800, 0x80808080, 0xc4000400, 0x00008000, 0x04004400, 0x00008000, 0x80090008, 0x00000001, 0x00088008, 0x00000000, 0x08080808, 0x00000001, 0x08080808, 0x00000000
-	0x80080088, 0x818480c4, 0x00880088, 0x04480840, 0x00c100c0, 0x00800080, 0x01400040, 0x00000000, 0x00000000, 0x000400c5, 0x00800000, 0x04401040, 0x04401048, 0x00900080, 0x0400c400, 0x88800800
4	0x80000080, 0x809081c0, 0x00880088, 0x4090804c, 0x2a24a204, 0x44444444, 0x6862c06a, 0xc0004000, 0x01800080, 0x44944458, 0x8008880d, 0x4c4c584d, 0x480c4800, 0x18000804, 0x49044808, 0x88808880

Table 27: `ChaCha` differential trail over 4 rounds with a probability weight of 217. The trail was derived using the "full window size heuristic" described in Section 3. The table contains 9 rows, as it also includes the half-round state of `ChaCha`.