Improved Differential and Linear Trail Bounds for ASCON

Solane El Hirch, Silvia Mella, Alireza Mehrdad and Joan Daemen

Radboud University, Nijmegen, The Netherlands
solane.elhirch@ru.nl,silvia.mella@ru.nl,alireza.mehrdad@ru.nl,joan@cs.ru.nl

Abstract. Ascon is a family of cryptographic primitives for authenticated encryption and hashing introduced in 2015. It is selected as one of the ten finalists in the NIST Lightweight Cryptography competition. Since its introduction, Ascon has been extensively cryptanalyzed, and the results of these analyses can indicate the good resistance of this family of cryptographic primitives against known attacks, like differential and linear cryptanalysis.

Proving upper bounds for the differential probability of differential trails and for the squared correlation of linear trails is a standard requirement to evaluate the security of cryptographic primitives. It can be done analytically for some primitives like AES. For other primitives, computer assistance is required to prove strong upper bounds for differential and linear trails. Computer-aided tools can be classified into two categories: tools based on general-purpose solvers and dedicated tools. General-purpose solvers such as SAT and MILP are widely used to prove these bounds, however they seem to have lower capabilities and thus yield less powerful bounds compared to dedicated tools.

In this work, we present a dedicated tool for trail search in Ascon. We arrange 2-round trails in a tree and traverse this tree in an efficient way using a number of new techniques we introduce. Then we extend these trails to more rounds, where we also use the tree traversal technique to do it efficiently. This allows us to scan much larger spaces of trails faster than the previous methods using general-purpose solvers. As a result, we prove tight bounds for 3-rounds linear trails, and for both differential and linear trails, we improve the existing upper bounds for other number of rounds. In particular, for the first time, we prove bounds beyond 2^{-128} for 6 rounds and beyond 2^{-256} for 12 rounds of both differential and linear trails.

Keywords: Differential Trail Search · Linear Trail Search · Trail Weight Bounds · ASCON

1 Introduction

Ascon is a family of cryptographic algorithms for authenticated encryption (AE) and hashing [DEMS21a]. It is currently one of the ten finalists in the NIST lightweight cryptography (LWC) competition for lightweight AE [TMC+21] and was selected in the final portfolio of the CAESAR competition [com14] as primary choice for lightweight AE [DEMS16]. The AE schemes are based on the duplex construction [BDPV11a], while the hashing functions are based on the sponge construction [BDPV07,BDPV08]. All family members are based on the Ascon permutation, which is also used in Isap [DEM+20], another finalist in the NIST LWC competition.

The Ascon permutation has been extensively cryptanalyzed since its introduction, giving confidence on the security of the schemes based on it. However a thorough effort to prove bounds on the differential probability (DP) and squared correlation (C^2) of its trails was conducted only recently [GPT21, EME22, MR22]. Before that, only bounds for 3-round trails were proved in [DEMS15] and for more rounds, the authors performed heuristic searches showing small DP and small C^2.

Proving bounds for trails is an important task in the evaluation of the security of a permutation. The cost of a differential attack based on a given trail is inversely proportional to its DP. Similarly, the cost of a linear attack is inversely proportional to the C^2. Therefore, the smaller the DP or C^2 is, the higher the cost of the attack is. Bounds on the DP or C^2 of trails are usually proven by bounding the number of active S-boxes of the trails or its weight. Roughly speaking, the weight w of a differential trail relates to its DP as $\mathrm{DP} \approx 2^{-\mathrm{w}}$. Similarly, the weight of a linear trail relates to its C^2 as $C^2 \approx 2^{-\mathrm{w}}$. Therefore, the higher the number of active S-boxes or the higher the weight is, the more costly the attack is.

For some primitives, bounds can be proved analytically. An example is the AES with its simple proof that a 4-round differential trail has weight at least 150 [DR20]. For other primitives they are obtained by computer-aided proofs. In this case, a program scans the space of all r-round trails satisfying a given requirement. The requirement is usually that the number of active S-boxes in the trail is below a given threshold, or that the weight of the trail is below a given threshold. Large state size and weak alignment contribute in making the search space very large and thus the cost of scanning it very costly. It follows that the bounds that one can prove are limited by the capability of the tool for scanning such spaces.

Automated tools that are often used to prove bounds on the number of active S-boxes are based on general-purpose solvers like Boolean satisfiability (SAT) [MP13, EME22], (mixed) integer linear programming ((M)ILP) [SHW+14, BPP+17, BJK+16, WH19] or Satisfiability Modulo Theories (SMT) [DEMS15]. Dedicated tools were used to prove lower bounds on the weight of trails in NOEKEON [DPAR00], KECCAK-p [DV12, MDV17], XOODOO [DHVV18b], and SUBTERRANEAN [MMGD22]. Such dedicated programs allow to better exploit the structural properties of the primitive and usually allow to scan larger spaces, leading to better results than tools based on general-purpose solvers. Before 2022, the best result obtained with tools based on general-purpose solvers that we are aware of is the work of Mouha and Preneel, who used a SAT-based method to scan the space of all 3-round characteristics up to weight 26 in the ARX primitive Salsa20, which implies a weight per round below 9 [MP13]. The dedicated search for Noekeon in [DPAR00] and KECCAK-f[1600] in [DV12] both reached a weight per round of 12, while the improvements of [MDV17] allowed to reach a weight per round of 15. The dedicated search on SUBTERRANEAN reached a weight per round beyond 14 [MMGD22]. In the last months, better results have been achieved with both solvers-based tools and dedicated tools. In [EME22] Erlacher et al. reached a weight per round of 17 with their SAT-based method to scan the space of trails in ASCON. While the most recent improvements to the dedicated tool for XOODOO allowed to reach a weight of 21 per round [DMA22].

Inspired by the previous works on dedicated tools and their results compared to automated tools based on general-purpose solvers, in this work we introduce a dedicated tool for ASCON. We present a number of techniques that deeply exploit the properties of the linear and non-linear layer of ASCON to generate trails very efficiently. Such techniques allow us to scan larger spaces of trails at a smaller computational cost compared to previous work, that results in improved bounds. In particular, we reach a weight per round of 21.

Related work. Exact values for the DP and C^2 of trails over 1 and 2 rounds of ASCON can be derived by the fact that the S-box has maximum DP of 2^{-2} and maximum C^2 of 2^{-2}, and that the linear layer has branch number $\mathcal{B} = 4$. For more rounds, lower bounds were proven in [DEMS15] and [EME22]. Both works are based on SAT solvers and prove bounds on the number of active S-boxes. Directly bounding the probability would require a more expensive model for the SAT solver compared to bounding the number of active S-boxes, which already requires a major computational effort.

In [DEMS15], Dobraunig et al. presented an SMT model and used it to prove that a 3-round differential trail has a minimum of 15 active S-boxes and a 3-round linear trail has a minimum of 13 active S-boxes. These bounds automatically give bounds of 2^{-30} and 2^{-26} for the DP and the C^2 of 3-round trails, respectively. Bounds for more rounds were proven later in [EME22], where Erlacher et al. presented a SAT model and used it to prove bounds on the number of active S-boxes for 4 and 6 rounds, from which they derived bounds for 8 and 12 rounds. In addition, by using these results and the bound on 1 round, we can derive bounds for 5, 7, 9, 10, and 11 rounds. We summarize such bounds in the second column of Table 1.

To overcome the computational limitation of SAT solvers, the authors of [EME22] aim at reducing the search space as much as possible and split it in sub-spaces that can be scanned in parallel. To this end, they introduced a number of techniques similar to those usually used in dedicated tools, like starting from shorter trails with minimum number of active S-boxes, building long trails from short trails in an incremental way, and taking advantage of the translation symmetry of the primitive [DV12].

A significant effort has been also performed to find trails with the highest DP or C^2. Such searches are based on heuristic tools and provide upper bounds. In [DEMS15], the authors used a dedicated guess-and-determine tool (`nldtool`) to find differential trails up to 5 rounds, while a heuristic tool (`lineartrails`) to find linear trails for 4 and 5 rounds was introduced in [DEM15]. In [GPT21], the authors used constrained programming (CP) to find best differential trails for 5 and 6 rounds. The authors in [MR22] presented an MILP-based approach that allowed them to find a new 5-round linear trail with best known C^2 and proved tight bound for differential trails over 3 rounds. We report the best known trails found by these tools in the first column of Table 1.

In dedicated tools, bounds on the weight of trails are derived, instead of evaluating the number of active S-boxes. The first dedicated tool for trail search was introduced as early as 2000 for Noekeon [DPAR00]. It was later improved and refined in [DV12] and [MDV17] for Keccak-p and then adapted to Xoodoo in [DHVV18b] and Subterranean in [MMGD22]. In each of these works, the authors presented a number of techniques specific for the permutation under analysis that deeply exploit the structure of its linear and non-linear layers. However, the approach underlying these works is the same and is generic, so it can in principle be applied to other ciphers. In a few words, the goal of such approach is to reduce as much as possible the search space and define methods to scan it efficiently. To this end, trails are split into classes where the weight of trails in the same class can be easily bounded by generating only one representative trail per class, called *trail core*. By exploiting the symmetry properties of the permutation, trail cores can be further split into classes where each trail core in a class is the translated version of another trail in the class and trail cores in the same class have the same weight. Therefore, only one representative is generated, that is called *canonical* (or *necklace* to use the terminology of [EME22]). Trail cores over multiple rounds are built by first generating the shortest possible trail cores, that are those over 2 rounds, and by extending them one round at the time each time checking if the weight is below the expected limit. In [MDV17] a generic method is introduced to generate such 2-round trail cores efficiently as a tree search.

Our contribution. In this work we present a dedicated tool for trail search in Ascon, based on the tree-based approach introduced in [MDV17]. To obtain an efficient instantiation of the tree-based approach, we introduce a number of techniques that deeply exploit the structure of the linear and non-linear layers in Ascon. We also introduce methods to efficiently extend trails over multiple rounds. We implemented such techniques in a

Table 1: Previous and new bounds for the differential probability (DP) of differential trails and squared correlation (C^2) of linear trails in Ascon. R denotes the number of rounds; min #S denotes the minimum number of active S-boxes.

(a) Differential trails

R	best known probability			previous lower bound			new bound
	DP	method	reference	DP	method	reference	DP
1	2^{-2}	DDT		2^{-2}	DDT		
2	2^{-8}	DDT+$\mathcal{B}$		2^{-8}	DDT+$\mathcal{B}$		
3	2^{-40}	`nldtool`	[DEMS15]	2^{-40}	MILP	[MR22]	
4	2^{-107}	`nldtool`	[DEMS15]	$\leq 2^{-72}$	SAT+min #S	[EME22]	$\leq 2^{-86}$
5	2^{-190}	CP	[DEMS15, GPT21]	$\leq 2^{-74}$	combine 1R+4R		$\leq 2^{-100}$
6	2^{-305}	CP	[GPT21]	$\leq 2^{-108}$	SAT+min #S	[EME22]	$\leq 2^{-129}$
7				$\leq 2^{-110}$	combine 1R+6R		$\leq 2^{-131}$
8				$\leq 2^{-144}$	SAT+min #S	[EME22]	$\leq 2^{-172}$
9				$\leq 2^{-146}$	combine 1R+8R		$\leq 2^{-186}$
10				$\leq 2^{-180}$	combine 4R+6R		$\leq 2^{-215}$
11				$\leq 2^{-182}$	combine 1R+10R		$\leq 2^{-229}$
12				$\leq 2^{-216}$	SAT+min #S	[EME22]	$\leq 2^{-258}$

(b) Linear trails

R	best known squared correlation			previous lower bound			new bound
	C^2	method	reference	C^2	method	reference	C^2
1	2^{-2}	LAT		2^{-2}	DDT		
2	2^{-8}	LAT+$\mathcal{B}$		2^{-8}	DDT+$\mathcal{B}$		
3	2^{-28}	`lineartrails`	[DEM15]	$\leq 2^{-26}$	SMT+min #S	[DEMS15]	2^{-28}
4	2^{-98}	`lineartrails`	[DEM15]	$\leq 2^{-72}$	SAT+min #S	[EME22]	$\leq 2^{-88}$
5	2^{-184}	MILP	[MR22]	$\leq 2^{-74}$	combine 1R+4R		$\leq 2^{-96}$
6				$\leq 2^{-108}$	SAT+min #S	[EME22]	$\leq 2^{-132}$
7				$\leq 2^{-110}$	combine 1R+6R		$\leq 2^{-134}$
8				$\leq 2^{-144}$	SAT+min #S	[EME22]	$\leq 2^{-176}$
9				$\leq 2^{-146}$	combine 1R+8R		$\leq 2^{-184}$
10				$\leq 2^{-180}$	combine 4R+6R		$\leq 2^{-220}$
11				$\leq 2^{-182}$	combine 1R+10R		$\leq 2^{-228}$
12				$\leq 2^{-216}$	SAT+min #S	[EME22]	$\leq 2^{-264}$

dedicated tool, called `AsconTrailTool`, that we used to prove bounds for differential and linear trails for different number of rounds. Though a comparison of the computational costs of our method and the method of [EME22] is not straightforward, due to the different machines employed in the two works, our techniques allowed us to scan a larger space at a lower cost. The most direct consequence is that we can improve over known bounds. We report our improved bounds in the third column of Table 1. Notably, for linear trails, we prove tight bound for 3 rounds, closing the gap between the lower bound and the best known trail. For 4 rounds, we can prove the bound of 2^{-86} for differential trails in 13 CPU days, and of 2^{-88} for linear trails in 110 CPU days. Our method is more efficient in comparison to the previous methods where the cost estimation for proving the bound of 2^{-80} is 6688 CPU days in [EME22] and 3898 CPU days in [MR22].

Given the aforementioned 4-round trails, proving bounds for 6 rounds required us 6 additional CPU days to prove the bound of 2^{-129} for differential trails and 21 additional CPU days to prove the bound of 2^{-132} for linear trails. Our method performs better than the one in [EME22] where the authors indicated that it required 2 additional CPU months to prove the bound of 2^{-108}. For 12 rounds, we can prove for the first time bounds beyond 2^{-256}. We also prove better bounds for other numbers of rounds, which can be useful information for designers when they have to choose the number of rounds to use in the different phases of a given construction.

Organization of the paper. In Section 2 we first recall some concepts about trails and trail cores, then we recall the strategy used in previous dedicated tools to prove trail bounds and the generic tree-based method. Then, in Section 3 we present the specification of Ascon round function and propagation properties through it. In Section 4, we introduce the tree-based method applied to Ascon to generate 2-round trail cores and provide new techniques to traverse the tree in a more efficient way. After that, we explain how we efficiently perform trail core extension using the techniques introduced in Section 5. Finally, we present our practical results and improved bounds in Section 6 and in Section 7 we provide some final remarks.

2 Trails and trail search strategy

In this section we first recall some concepts related to differential and linear cryptanalysis. Then we explain the general strategy for performing trail search using the tree-based approach.

2.1 Trails and trail cores

We start by defining differential trails and trail cores over iterative cryptographic primitives. Then, we do the same for linear trails and we introduce a unified notation for both cases.

2.1.1 Differentials and differential trails

Let x_1 and x_2 be two inputs to a transformation α over $\mathbb{F}_2^n$, and $y_1 = \alpha(x_1)$ and $y_2 = \alpha(x_2)$ be their corresponding outputs. We say $b = x_1 \oplus x_2$ is an input difference of α and $a = y_1 \oplus y_2$ is an output difference and we call the pair (b, a) a *differential* over α. The difference probability (DP) of a differential (b, a) is defined as

$$\mathrm{DP}(b, a) = \frac{|\{x \in \mathbb{F}_2^n \mid \alpha(x - b) - \alpha(x) = a\}|}{2^n} .$$

When $\mathrm{DP}(b, a) > 0$, we say that a is *compatible* with b through α. The *restriction weight* of a differential, denoted by $\mathrm{w_r}$, is defined as

$$\mathrm{w_r}(b, a) = -\log_2 \mathrm{DP}(b, a) .$$

Let α be an iterative mapping, that consists of the repetition of a number of rounds p_i: $\alpha = p_r \circ \cdots \circ p_2 \circ p_1$. A differential over p_i is called a *round differential.* An *r-round differential trail* over α is a sequence of r round differentials.

Let the round function be defined as the composition of a linear layer p_L and a non-linear layer p_S. We use a redundant representation of trails where we specify the difference after each layer:

$$Q = a^0 \xrightarrow{p_L} b^0 \xrightarrow{p_S} a^1 \xrightarrow{p_L} b^1 \xrightarrow{p_S} a^2 \xrightarrow{p_L} \cdots \xrightarrow{p_S} a^r .$$

The restriction weight of a trail is the sum of the weight of its round differentials: $\mathrm{w_r}(Q) = \sum_{i=1}^{r} \mathrm{w_r}(a^{i-1} \xrightarrow{p_i} a^i)$. Since p_L is linear, the weight of a trail only depends on the weight over the non-linear layers: $\mathrm{w_r}(Q) = \sum_{i=1}^{r} \mathrm{w_r}(b^{i-1} \xrightarrow{p_S} a^i)$. If the non-linear layer p_S has algebraic degree 2 (as in Ascon), the weight of a differential over p_S only depends on its input difference b [Dae95]. Hence, the weight of the trail is given by $\mathrm{w_r}(Q) = \sum_{i=1}^{r} \mathrm{w_r}(b^{i-1})$.

Since the weight of an r-round trail Q is independent of the first and last differences of the trail, the sequence of differences $(b^0, a^1, \dots, a^{r-1}, b^{r-1})$ – which is Q with the first and last differences removed – defines a set of r-round trails with the same weight $\mathrm{w_r}(Q)$. On the other hand, for a given a^1 there exist several differences b^0 that are compatible with a^1 through p_S^{-1}. The minimum weight over all these compatible states b^0 is called the *minimum reverse weight* of a^1 and it is denoted by $\mathrm{w_{rev}}(a^1)$ [DV12]. It follows that the sequence $\tilde{Q} = (a^1, \dots, a^{r-1}, b^{r-1})$ defines a set of r-round trails with weight at least $\mathrm{w_{rev}}(a^1) + \sum_{i=2}^{r} \mathrm{w_r}(b^{i-1})$. $\tilde{Q}$ is called *r-round differential trail core* [DV12].

2.1.2 Correlation and linear trails

Let α be a transformation over $\mathbb{F}_2^n$. A linear approximation over α consists of a pair (a, b) of selection vectors over $\mathbb{F}_2^n$, called *input mask* and *output mask*, respectively. The *correlation* C of a linear approximation (a, b) is the correlation between the Boolean functions $a^T \cdot x$ and $b^T \cdot \alpha(x)$:

$$\mathrm{C}(a, b) = \frac{|\{x \in \mathbb{F}_2^n \mid a^\mathsf{T} x + b^\mathsf{T} \alpha(x) = 0|}{2^{n-1}} - 1\,.$$

The *correlation weight* is denoted by $\mathrm{w_c}(a, b)$ and is defined as

$$\mathrm{w_c}(a, b) = -\log_2 \mathrm{C}^2(a, b)\,.$$

Similar to a differential trail, an *r-round linear trail* is defined as a sequence of linear masks. As in [BDPV11b, DHVV18b] we study linear propagation from the output to the input. To this end, we rephase the round function so that the trail first encounters p_L and then p_S of each round (as in the differential case). Notice that such rephasing does not affect the trail analysis.

A linear trail is represented as

$$Q = a^0 \xrightarrow{{p_L}^\mathsf{T}} b^0 \xrightarrow{p_S^{-1}} a^1 \xrightarrow{{p_L}^\mathsf{T}} b^1 \xrightarrow{p_S^{-1}} a^2 \xrightarrow{{p_L}^\mathsf{T}} \cdots \xrightarrow{p_S^{-1}} a^r\,.$$

where a_0 is the output mask (after the last round) and a_r is the input mask (before the first round). A mask a_i at the output of p_L maps to a mask $b_i = {p_L}^\mathsf{T}(a_i)$ before p_L. If the linear mapping p_L is seen as the multiplication by a matrix M, then ${p_L}^\mathsf{T}$ denotes the linear mapping obtained by the multiplication by M^T. To denote the propagation from the output of p_S to its input, we use p_S^{-1}.

The correlation weight of a linear trail is the sum of the correlation weights of the round linear approximations composing the trail. Given that ${p_L}^\mathsf{T}$ is linear and that, when p_S has algebraic degree 2, the correlation weight depends only on the value of the output mask [Dae95], the weight of a linear trail is given by $\mathrm{w_c}(Q) = \sum_{i=1}^{r} \mathrm{w_c}(b^{i-1})$

Similar to the differential case, an *r-round linear trail core* [DHVV18b] is a sequence $\tilde{Q} = (a^1, \dots, a^{r-1}, b^{r-1})$ that defines a set of r-round linear trails with weight at least $\mathrm{w_{rev}}(a^1) + \sum_{i=2}^{r} \mathrm{w_r}(b^{i-1})$.

2.1.3 Unified representation of trail cores

As done in [BDPV11b] with Keccak-p and in [DHVV18b] with Xoodoo, we use a unified representation of trails and trail cores. In fact, also in the case of Ascon, there are strong similarities in the study of propagation of differential and linear trails. For differential trails we consider the propagation of differences from input to output and for linear trails we consider the propagation of masks from output to input. A trail core is specified by:

$$\tilde{Q} = a^1 \xrightarrow{p_L^*} b^1 \xrightarrow{p_S^*} a^2 \xrightarrow{p_L^*} b^2 \xrightarrow{p_S^*} a^3 \xrightarrow{p_L^*} \cdots \xrightarrow{p_S^*} b^{r-1} .$$

where

- $p_L^* = p_L$, and $p_S^* = p_S$ for differential trails, and
- $p_L^* = {p_L}^{\mathsf{T}}$, and $p_S^* = p_S^{-1}$ for linear trails.

We refer to differences and masks as *state patterns*, or only *states* or *patterns*, when we generally talk about trails. A pattern a_i represents a difference at the output of p_S in a differential trail and a mask at the input of p_S in a linear trail. A pattern b_i represents a difference at the input of p_S in a differential trail and a mask at the output of p_S in a linear trail. We use the term weight, denoted by w, when we generically refer to $\mathrm{w_r}$ and $\mathrm{w_c}$.

2.2 Strategy of the trail search

In our trail search, we aim to scan the space of all r-round trails with weight below a certain threshold T_r, where r is usually a small number like 3,4, or 6. A naive way to generate them would be to generate all 1-round trails (i.e. round differentials and linear approximations) with weight below $\lfloor T_r/r \rfloor$ and then extend them to r rounds. The value of T_r that can be achieved is limited by the quantity of such 1-round trails, which grows exponentially with the weight, and the cost of extending them. The number of 1-round trails can be reduced when symmetry properties are taken into account. For instance, in Xoodoo it can be reduced roughly by a factor 128 thanks to the fact that both the linear and non-linear layers are invariant with respect to translations parallel to the planes [DHVV18b]. While in Keccak-f[1600] it can be reduced by a factor 64 thanks to the translation invariance along the lanes [MDV17]. Even with such reductions, it is shown that this number still grows exponentially with the weight [MDV17, DHVV18b].

However, as demonstrated in [MDV17, DHVV18b], the number of trails with a given weight per round decreases with the number of rounds. That is, the number of 2-round trails with weight below $\lfloor 2T_r/r \rfloor$ is smaller than the number of 1-round trails with weight below $\lfloor T_r/r \rfloor$. Therefore, a more convenient approach for Keccak-p like primitives consists in starting from 2-round trails and extend them. This allows to achieve much higher values of T_r for the same number of rounds r.

Actually, to prove bounds, it is not necessary to generate all r-round trails. We can limit ourselves to r-round trail cores, since the weight of a trail core lower bounds the weight of all trails in it. Therefore, we can start from 2-round trail cores and extend them.

This strategy was used for Keccak [DV12, MDV17], Xoodoo [DHVV18b], and Subterranean [MMGD22] and we will use it also in this work. In fact, also in the case of Ascon, starting from 2-round trail cores instead of 1-round trails significantly reduces the number of patterns to extend. The symmetry properties of Ascon allows us to reduce the number of 1-round trails and the number of 2-round trail cores with weight per round ($\mathrm{w}/\#r$). In particular, the linear and non-linear layers of Ascon are invariant with respect to translation along the horizontal axis and it allows to reduce them by a factor 64. In Fig. 1, we depict these reduced numbers with weight per round ($\mathrm{w}/\#r$).

2.2.1 Generating 2-round trail cores as a tree search

A method to generate all 2-round trail cores with weight below a given threshold T_2 was introduced in [DPAR00], applied to Keccak-p in [DV12], and improved and refined

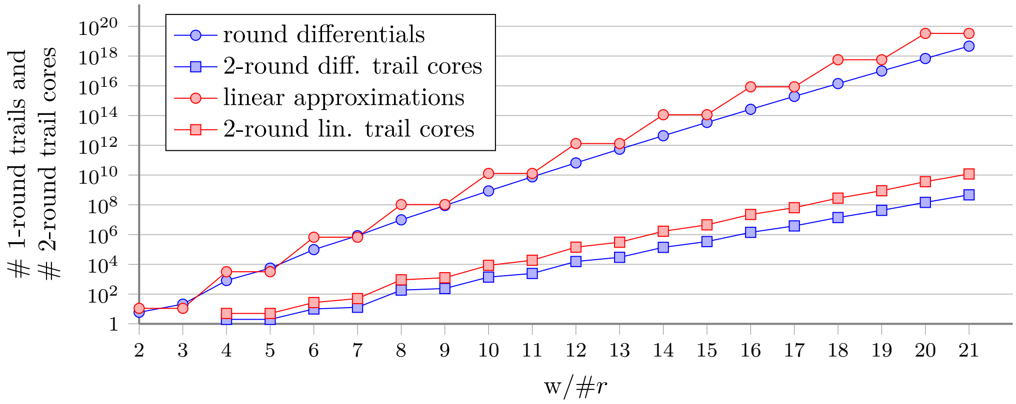


Figure 1: Number of 1-round trails and 2-round trail cores with weight per round (w/#r), divided by 64.

in [MDV17]. Later, similar method was applied to Xoodoo in [DHVV18b] and also Subterranean in [MMGD22].

We now recall the main idea at the basis of the refined method of [MDV17], which consists in seeing all 2-round trail cores as nodes of a tree that is properly traversed to get only those nodes with weight below T_2. In Section 4, we will explain how to instantiate it for Ascon to perform an efficient search.

A 2-round trail core is a pair (a, b) with weight $\mathrm{w}_{\mathrm{rev}}(a) + \mathrm{w}(b)$. To build them we have two choices: either we build a and then compute $b = p^*_L(a)$ or we build b and we compute $a = {p^*_L}^{-1}$. Each node of the tree is encoded as an ordered list of *units*, called *unit-list*. A unit is a set of *active* bits at a (if we are building a) or at b (if we are building b), where a bit is called active if it equals one, otherwise it is called *passive*. For instance, in Keccak-p and Xoodoo a type of unit is the *orbital*, which is a pair of active bits in the same column at a [MDV17, DHVV18b], while in Subterranean a unit is a single active bit at a [MMGD22].

The choice of building first a or b, the definition of units and their order relation influence the efficiency of the 2-round trail core generation. Therefore, it requires a good understanding of the linear and non-linear layers of the round function and their propagation properties.

Traversing the tree. The tree traversal is performed in a depth-first fashion, where a program iteratively calls the function `next()` (Algorithm 1) to generate the next valid node, as in [MMGD22]. The traversal starts by calling `next()` on an empty unit-list, and ends when it results again in the empty unit list.

The function `next()` traverses the tree with three possible moves: `toFirstChild()`, `toSibling()` and `toParent()`. If the node is an empty unit-list, then it adds the smallest possible unit. The function `toFirstChild()` returns false if adding a new unit is not possible. Otherwise it returns true. Then *additional conditions* are checked to see if we can prune the tree. If the `toFirstChild()` function returns false or the additional conditions are not satisfied, the routine will look for the next valid node in the tree by generating a sibling for the current node using the function `toSibling()`. The function `toSibling()` iterates the value of the last unit of the unit-list. If a sibling is found then the additional conditions are checked. If there are no valid siblings, the algorithm calls the function `toParent()` to remove the last unit from the unit-list and look for a valid sibling of the parent node in a recursive way.

Algorithm 1 `next()` function [MMGD22]

```
if (toFirstChild() == true) then
    if (additional conditions are satisfied) then
        return true;
do
    while (toSibling() == true) do
        if (additional conditions are satisfied) then
            return true;
while (toParent() == true)
return false;
```

Pruning the tree. To efficiently traverse the tree, at each move we check whether the node satisfies some additional conditions or not. To this end, we make use of two tools: *canonicity* and *score*, whose definition fully depends on the specification of the linear and non-linear layers.

- **Canonicity:** Without considering round constant and key addition, the round function of many cryptographic primitives exhibits translation symmetry. This symmetry allows to divide the state space into equivalence classes where all patterns in a class have the same properties and weight. Therefore, we aim to generate only one pattern per equivalence class, that is called *canonical*.
- **Score:** The score of a node is defined as a lower bound on the weight of a node and all its descendants. This tool allows us to prune entire sub-trees as soon as we reach a node whose score is higher than T_2. It should be tight enough to allow efficient pruning, but also efficiently computable.

2.2.2 Trail core extension

After generating all 2-round trail cores with weight below T_2, we need to extend them to generate trail cores over more rounds. Extension is done incrementally one round at the time. Namely, we first extend the 2-round trail cores by one round to generate 3-round trail cores with weight below a given T_3. Then we extend the obtained 3-round trail cores by one round to generate 4-round trail cores with weight below a given T_4 and so on.

Given an r-round trail core $\tilde{Q} = (a^1, b^1, \ldots, b^{r-1})$, one can extend it to $(r+1)$ rounds in both forward and backward directions. In the term forward extension, forward means through p_S^*, so through p_S for differential trails and through p_S^{-1} for linear trails. Backward means through $p_S^{*^{-1}}$, so through p_S^{-1} for differential trails and through p_S for linear trails.

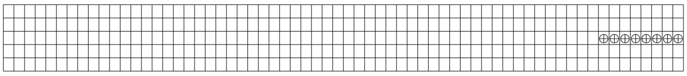
In forward extension, we generate all patterns a^r that are compatible with b^{r-1} through p_S^*, compute $b^r = p_L^*(a^r)$ and finally append (a^r, b^r) to the end of $\tilde{Q}$. The weight of the obtained cores is $\mathrm{w}(\tilde{Q}) + \mathrm{w}(b^r)$.

In backward extension, we generate all patterns b^0 compatible with a^1 through $p_S^{*^{-1}}$, then compute the corresponding $a^0 = p_L^{*^{-1}}(b^0)$, and prepend them to $\tilde{Q}$. The weight of these 3-round trail cores is obtained by subtracting $\mathrm{w}_{\mathrm{rev}}(a^1)$ and then adding $\mathrm{w}_{\mathrm{rev}}(a^0) + \mathrm{w}(b^0)$.

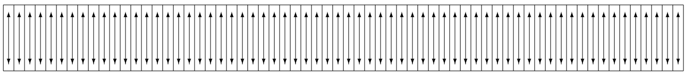
By repeating the aforementioned process, one can extend a trail core over multiple rounds in any direction.

3 The Ascon permutation

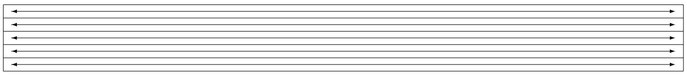
Ascon family includes the authenticated encryption schemes Ascon-128 and Ascon-128a [DEMS21b], the hash functions Ascon-Hash and Ascon-Hasha and the extendable



(a) Adding round constant p_C.



(b) Non-linear layer p_S.



(c) Linear layer p_L.

Figure 2: ASCON's round function p.

output functions (XOF) ASCON-XOF and ASCON-XOFA. The AE schemes are based on the duplex construction [BDPV11a], while the hashing and XOF functions are based on the sponge construction [BDPV07,BDPV08]. All family members are based on the ASCON permutation, which is also used in ISAP [DEM+20], another finalist of the NIST LWC competition.

3.1 Ascon round specification

The ASCON permutation operates on a state of 320 bits arranged in five 64-bit rows $x_0, \ldots, x_4$. The number of rounds is a tunable parameter. It is 12 in the initialization and finalization phase of all ASCON schemes, while it changes for the data processing phase. It is 6 for ASCON-128, 8 for ASCON-128A, ASCON-HASHA, and ASCON-XOFA, and 12 for ASCON-HASH and ASCON-XOF.

The round function of ASCON is denoted by p and consists of three steps: $p = p_L \circ p_S \circ p_C$. The function p_C, that can be seen in Fig. 2a, adds a round constant to row x_2 of the state. The non-linear layer p_S applies 64 parallel 5-bit S-boxes, denoted $\mathcal{S}$, to the columns of the state, as in Fig. 2b. The non-linear part of the S-box $\mathcal{S}$ is based on the χ shift-invariant mapping [Dae95]. We denote χ applied to an n-bit circle of bits as χ_n, so the S-box in KECCAK-p is χ_5 [BDPV11b]. We hence can describe $\mathcal{S}$ as χ_5 preceded and followed by two linear mappings, each consisting of 3 bitwise additions. We depict it in Fig. 3.

Finally, p_L applies a linear function to each row independently as in Fig. 2c and is defined as follows:

$$
\begin{aligned}
x_0 &\leftarrow x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \\
x_1 &\leftarrow x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \\
x_2 &\leftarrow x_2 \oplus (x_2 \ggg \ \ 1) \oplus (x_2 \ggg \ \ 6) \\
x_3 &\leftarrow x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17) \\
x_4 &\leftarrow x_4 \oplus (x_4 \ggg \ \ 7) \oplus (x_4 \ggg 41)
\end{aligned}
\tag{1}
$$

Clearly, in p_L there is no inter-row mixing and this is compensated by the linear mappings in p_S.

3.2 Propagation properties through the round

Since the S-box $\mathcal{S}$ is based on the χ_5 mapping also used in KECCAK-p, it inherits some interesting properties from it that were discussed in [Dae95] and that we summarize here.

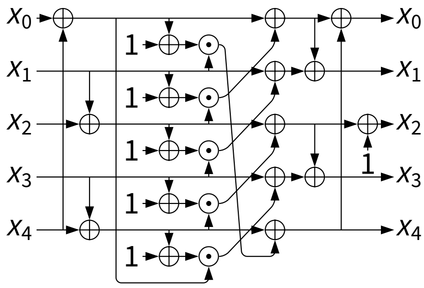


Figure 3: ASCON's S-box $\mathcal{S}$.

Difference propagation properties. Since p_S has algebraic degree 2, given a difference b at the input of p_S, the space of compatible differences a at the output of p_S form a linear affine space $\mathcal{A}(b)$ with $2^{\mathrm{w_r}(b)}$ elements [Dae95]. We can compute offset and basis for such space starting from offset and basis over χ_5, that are reported in [BDPV11b]. In particular, for a given difference b at the input of $\mathcal{S}$, we map it at the input of χ_5 through the first linear layer of bitwise additions, we take the offset and basis that determine the affine space at the output of χ_5, and finally we map them through the second linear layer. We provide offset and basis vectors for all possible 31 non-zero differences at the input of $\mathcal{S}$ in Table 7. Among the 31 non-zero differences, 5 have weight 2, 15 have weight 3, and 11 have weight 4. Therefore, the weight of b is at least twice the number of active columns in b.

Difference propagating through the inverse of p_S is different. For a given difference a at the output of p_S, the set of compatible differences b at the input of p_S is not an affine space, but we can exhaustively list them. The list of the differences b compatible with a is needed to compute $\mathrm{w_{rev}}(a)$ which is required for our trail search. Among the 31 non-zero differences, 10 have 9 compatible differences, 10 have 10 compatible differences, 6 have 11 compatible differences, and 5 have 12 compatible differences. Moreover, 20 have minimum reverse weight 2, and 11 have minimum reverse weight 3.

Mask propagation properties. For a given output mask b, the space of input mask a with a non-zero correlation with b is a linear affine space with $2^{\mathrm{w_c}(b)}$ elements [Dae95]. Again, to build a representation of such space, we rely on the specification of offset and basis over χ_5 [BDPV11b]. We provide offset and basis vectors for all possible 31 non-zero masks at the output of $\mathcal{S}$ in Table 8. Among the 31 non-zero masks, 10 have weight 2, and 21 have weight 4.

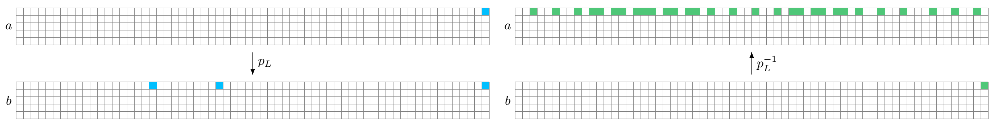
Given a mask a at the input of p_S, we can list the compatible masks b at the output of p_S, which do not form an affine space. Among the 31 non-zero masks, 10 have 10 compatible masks, and 20 have 13 compatible masks, and 1 has 16 compatible masks. Moreover, 30 have minimum reverse weight 2, and 1 has minimum reverse weight 4.

Notice that a linear trail has always even weight.

As explained in Section 2.1.2, the propagation of masks through the linear layer p_L is deterministic: an output mask b fully determines the corresponding input mask a by $b = {p_L}^{\mathsf{T}}(b)$. The transpose ${p_L}^{\mathsf{T}}$ has the same shape as p_L itself, the only difference is that the right shifts become left shifts.

4 Generating 2-round trail cores in Ascon as tree-search

In this section, we explain how we generate all 2-round trail cores in ASCON, with weight below a given target T_2, using the tree-based approach of Section 2.2.1. To this end, we first define units and their order relation. Then we give a description of the techniques



(a) Propagation from a to b (b) Propagation from b to a

Figure 4: Propagation through p_L

used to traverse the tree and, to do it in an efficient way, we define the score function and discuss canonicity. After identifying the techniques used in the tree-search in Section 4.1, we give a more detailed description on the two-level tree search in Section 4.2, and in Section 4.3 we give a description of an alternative representation of p_L^*.

4.1 Concepts and techniques

Active bits as units. For the tree-based approach we have to define units and their ordering and the most important criteria for this choice are the ability to define an efficient score function and deal with canonicity efficiently. The linear mapping p_L^* does not have a particular structure like the column parity mixers in Xoodoo or Keccak-p, and the obvious choice for units would be (coordinates of) active bits. We can choose to have the units be active bits in a or in b. In other words, we either build the state at a and we compute $b = p_L^*(a)$, or we build the state at b and we compute $a = {p_L^*}^{-1}(b)$.

Active bits in a as units. If the units are defined as active bits in a, adding a unit affects 3 bits in b. If some of these bits are active in the parent, this addition cancels them. We call the effect of active bits in a parent that are not present in the child *cancellation*. The inverse of the row mapping p_L^* is *dense*: it maps a row with a single active bit to a row with many active bits. If the units are defined as active bits in b, adding a unit affects many active bits in a, risking the cancellation of many more active bits. We illustrate this asymmetry for the mapping p_L on row 0 in Fig. 4. It works similarly for ${p_L}^{\mathsf{T}}$. So with units defined at a an efficient score is more likely to be easy as there is less opportunity for cancellation. So we define our units as active bits in a. Note that cancellation only takes place in b and an active bit in a will be present in all its children.

Score function based on number of active columns. The non-linear layer p_S operates in parallel on 5-bit columns. This is similar to Xoodoo where the non-linear layer is the parallel operation of χ_3 on 3-bit columns and Keccak-p, where it is the parallel operation of χ_5 on 5-bit rows. χ_3 and χ_5 are instantiations of χ that has the property that adding an active bit to an input difference does not decrease the weight, and that adding an active bit at the output does not decrease the minimum reverse weight. This also holds for linear masks. In p_S this is not the case due to the presence of additional linear mappings in the S-box. So, adding an active bit to a column in a may decrease its minimum reverse weight and adding an active bit to a column in b may decrease its weight. Still, each active column in a contributes at least 2 to its minimum reverse weight and each active column in b contributes at least 2 to its weight. Moreover, adding active bits to a column in a or b cannot make it passive. So we can base the score function on the number of active columns.

Row-index-first lexicographic ordering. In a all the active columns can be accounted for in the score, in b only those that cannot become passive due to cancellation when adding units. This is where the ordering comes in. Units are defined by their coordinates

(i, j) and there are two natural orderings, both lexicographic: i-first or j-first. In i-first the active bits in row $i = 0$ come before those in row 1 etc., in j-first those in column $j = 0$ come before those in column 1, etc. The i-first ordering works well with p_L^*. This is because this mapping is the parallel application of 5 linear mappings that operate on the rows separately. In the i-first ordering units are added row by row, where units are always added in the row of the last unit or after it. Let us call the i-coordinate of the last unit i'. Then rows in a with $i < i'$ will be the same for all descendants of a state. As p_L^* operates on rows separately, this will also be the case for the rows in b with $i < i'$. That means that we can take as score function two times the number of active columns in a plus the number of columns in b that are active in the rows with $i < i'$.

Two-level tree: active rows and active bits. When we look at the children of a node we see two kinds. Children where a unit is added to a row that already contains active bits on the one hand and children where a unit is added to a row that does not on the other. In the former case the last active row of b cannot be taken into account for the score and in the latter case it can. We address this distinction by defining the units in a two-level structure. At the top level the units are active rows, where an active row groups all active bits in the same row. We will call the top level the *row tree* and its unit-lists *row-lists*. This means that the children of a node in the row tree have the first active rows in common with their parent, but have one more active row. The consequence is that when navigating in the row tree, for the score function we can count all active rows at a and at b. We call this score function the `Score-state()` function. An active row is a unit list too, where the units are active bits (within a specific row) listed in a so called *bit-list*. The consequence is that the children of a node in the row tree are also arranged according to a tree, that we will call a *bit tree*. More exactly, the children of a node in the row tree with last active row at i' are $4 - i'$ bit trees. For example if $i' = 2$, the children are grouped in two bit trees: one that groups the states with last row at row index $i = 3$ and one that groups the ones with last row index $i = 4$. The two-level tree search is detailed more in Section 4.2.

Score in the bit tree: the case of index 2. Each bit tree contains $2^{64} - 1$ nodes so it would be good to also prune these trees using a score function. Clearly, all active columns of a and the active columns at b due to all active rows but the last can be counted in this score. However, this does not help in states with a single active row and also not when these rows have sparse bit-lists. We will now explain that we can also include active bits from the last active row in b. Let us take a look at row $i = 2$. Adding a unit at position j affects three bits in b, in positions j, $j - 1$ and $j - 6$, so it affects bits in b only in the interval $[j - 6 \bmod 64, j]$. Here we adopt the following convention for intervals where we take into account the circular structure of the rows of the state: $[x, y]$ with $y \geq x$ is the set of indexes $\{x, x + 1, x + 2, \dots y\}$ and $[x, y]$ with $y < x$ is the set of indexes $\{x, x+1, \dots, 63, 0, 1 \dots y\}$. Assume we have an active row (bit-list) where the j-coordinate of the last active bit is j'. The range of j for the last active bit in its children is $[j' + 1, 63]$, so if $j' > 5$ the range of corresponding affected bits in b is $[j' - 5 \bmod 64, 63]$. In other words, any bit in b in the interval $[0, j' - 5]$ will be there for all children in the bit tree and therefore the corresponding active columns can be counted in the score. This becomes interesting as soon as $j' > 5$.

Score in the bit tree: general case. The efficiency of this technique depends on the (circular) distance between the affected bits in b: the smaller the better. In $j = 2$ this distance is only 6 but for the other rows, these distances are much larger. For example for $j = 0$, the bit positions are $0, 19, 28$ and the shortest interval that encloses all three is $[0, 28]$. We will call the length of this interval the *span*. For $j = 1$, the bit positions

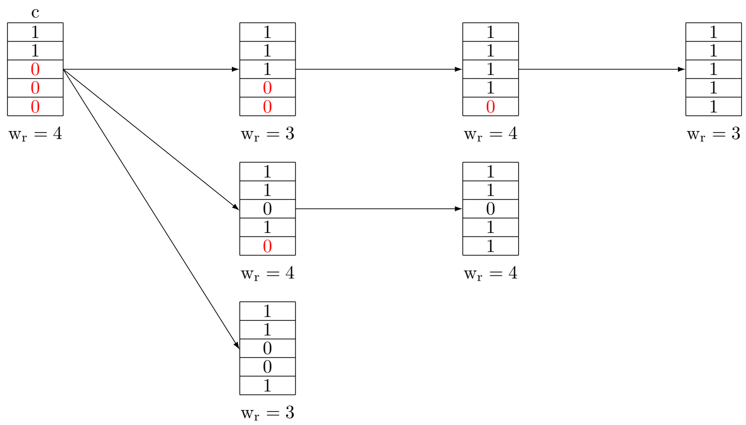


Figure 5: The score of a column difference with the first two stable bits set to $(1,1)$ is 3.

$0, 39, 61$ can be enclosed in an interval of length 25: $[39, 0]$. We can address this problem by adopting an alternative representation of the row that is used to compute the score in the so called `Score-row()` function. A more detailed explanation on the new representation is given in Section 4.3.

Refining the score of b. Computing the score based on twice the number of active columns in b is sub-optimal. In fact, while we are working on row i, all active bits at rows i' with $i' < i$ are stable and thus we can consider their contribution to the weight. In particular, for a given active column, only bits in rows i' with $i' \geq i$ can be added and this may potentially decrease the weight (though not below 2), but it may not. We define a lower bound on the weight of each active column, that we call *score* of the column, as the minimum among the weight of the column and the weight of all possible columns that can be obtained by adding bits in $i' \geq i$. Then, the score of a state is the sum over the score of all columns.

We illustrate an example in Fig. 5, with column differences and restriction weight. Let the first two bits of column c in Fig. 5 be set to $(1, 1)$. These bits are stable and we denote them in black, while we denote in red the three bits that can become 1 later in the search. On the right of c we list the six possible column values that we can obtain by adding bits to c in row 2, 3 or 4. The restriction weight of each column is reported below the column and we can see that the minimum weight among them is 3. So, we can define the score of c to be 3. If there are several active columns whose score is higher than 2, then the score of b will grow more quickly and pruning comes earlier.

Pruning the tree using canonicity. Clearly, both p_L and p_S are shift-invariant with respect to horizontal shifts (along the j-axis). A state that is the *smallest* in its class of states that are equivalent modulo horizontal shift is called *canonical*. The natural order to determine which state is smallest is a lexicographical ordering on the row-list: state X is smaller than state Y if the first row in its row-list is smaller than the first row in the row-list of Y. If they have equal first rows, we compare the 2nd row and so on. The order of rows is similarly defined using lexicographic ordering of their bit-lists, where we compare j-coordinates of active bits starting from the first one.

It was proven in [MDV17] that with such an order relation, the children of a non-canonical node are not canonical. This implies that whenever a non-canonical node is encountered, the full subtree can be pruned. For an active row we can define its *period*:

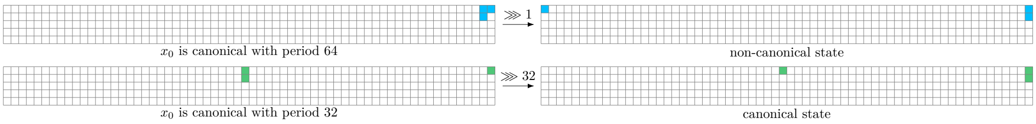


Figure 6: The row x_0 in the left states is canonical with different period. In the top figure, since the period of x_0 equals the row length, translation results in a non-canonical state. In the bottom figure since the period of x_0 is smaller than the row length, translation can result in a canonical state.

it is the smallest offset such that a shift of the row over that offset leaves it invariant. The period must be a divisor of 64 (the row length) and the vast majority of row values has period 64. If the first active row of a canonical state has period 64, all its children are canonical. This means that in that case we do not have to check for canonicity in subsequent active rows. Otherwise, we have to check canonicity by shifting any newly added active row over all multiples of the period and comparing. Examples are given in Fig. 6.

In general, only if the partial state consisting of the stable rows has period smaller than the row length, these checks must be done when adding an active bit.

4.2 Two-level tree

We represent a 2-round trail core (a, b) by the positions of its active bits in a. An active bit is determined by its coordinates (i, j) in the state with i the row coordinate and j the column coordinate and $0 \leq i < 5$ and $0 \leq j < 64$.

The bit-list of an active row is of the following form

$$a_i = [(i, j_1), (i, j_2), \ldots, (i, j_\ell)], \tag{2}$$

with $j_k < j_{k+1} \forall k \in \{1, \ldots, \ell - 1\}$. We have that $a_{i,j_k} = 1$ if and only if $k \in \{1, \ldots, \ell\}$.

At state-level, the row-list of a state a is a list of the form

$$a = [a_{i_1}, a_{i_2}, \ldots, a_{i_{r-1}}, a_{i_r}] \tag{3}$$

with $i_s < i_{s+1}$ $\forall s \in \{1, \ldots, r - 1\}$. We have that $a_{i,j} = 1$ if and only if $(i, j) \in \bigcup_s a_{i_s}$. The *smallest* value that an active row a_i can assume is $[(i, 0)]$.

We use two sets of functions to walk through the tree. One is the set of functions that operate on the bit-list of the last active row. The other is the set of functions that operate on the row-list.

We start by describing the former, where we assume the bit-list of the last active row is as in Eq. (2).

toFirstChildRow() If $1 + j_\ell < 64$, it adds $(i, 1 + j_\ell)$ to the bit-list and returns true. It returns false otherwise.

toSiblingRow() If $1 + j_\ell < 64$, it iterates the last bit in the list, i.e. (i, j_ℓ) becomes $(i, 1 + j_\ell)$ and returns true. It returns false otherwise.

toParentRow() It removes the last bit of the list, resulting in $a_i = [(i, j_1), (i, j_2), \ldots, (i, j_{\ell-1})]$. If it leaves the bit-list empty, it returns false and true otherwise.

The following functions operate on the row-list, where the row-list of the current node is as in Eq. (3).

toFirstChildState() If $1+i_r < 5$, it adds $a_{1+i_r} = [(1+i_r, 0)]$ to the row-list and returns true. It returns false otherwise.

ToSiblingState() It calls `nextRow()` on the last active row and if that returns true, it returns true. Otherwise, it checks whether the last active row is the bottom row, i.e., $i_r = 4$. If so, it returns false. If not, it moves the last active row one row index down, i.e. $i_r = 1 + i_r$, and there takes the smallest active row value $a_{i_r} = [(i_r, 0)]$ and returns true.

toParentState() It first removes the last active row from the list, resulting in $a = [a_{i_1}, a_{i_2}, \ldots, a_{i_{r-1}}]$. If this leaves the row-list empty it returns false and the search is over. Otherwise, it returns true.

The complete search works as follows. The tree traversal starts by calling `nextState()` on a state with a single active row set with a single active bit at position 0 and ends when `nextState()` returns false, that is when the row-list is empty. Its behavior is similar to that of the function `next()`. To prune the row tree the procedure calls `Score-state()` on the current canonical state.

The function `nextRow()` in Algorithm 3 is called by `ToSiblingState()` to iterate the last active row through a bit tree. It starts by checking `Score-row()` and if it is below the budget then it calls `toFirstChildRow()`. Here, a canonicity check is done on the whole state to only return canonical states. If there is no valid child either because `Score-row()` is above the budget or a canonical child has not been found, the procedure will look for a sibling by calling the function `toSiblingRow()`. Here again, a canonicity check is performed and if a canonical sibling has been found then the procedure returns true, otherwise the function `toParentRow()` is called.

4.3 The alternative row representation

The active bits in a row are indexed by j, and we index them by an alternative coordinate k that has a relation with j as $k = j \times q \bmod 64$, with q odd. Then, the row component function of p_L can be reformulated in terms of the new representation and this gives a mapping that only differs in the shift offsets. For a good choice of q we obtain a mapping with minimum span that we call *alternative representation.* Minimizing the span requires a specific factor q per row so, we have alternative representation for each row of Ascon. For $a_j = a'_{jq}$ and $b_j = b'_{jq}$, the alternative representation is defined as follows:

$$\begin{aligned}
p_L : b_j &\leftarrow a_j \oplus a_{j+s} \oplus a_{j+t} \\
b'_{jq} &\leftarrow a'_{jq} \oplus a'_{(j+s)q} \oplus a'_{(j+t)q} \\
b'_{jq} &\leftarrow a'_{jq} \oplus a'_{jq+sq} \oplus a'_{jq+tq} \\
p'_L : b'_k &\leftarrow a'_k \oplus a'_{k+sq} \oplus a'_{k+tq}
\end{aligned}$$

Since the alternative representation has the minimum span, more active bits in b are guaranteed to stay active after adding a unit. The active bits in b that remain active after adding new units to a are called *stable* bits. In the alternative representation, the bits in b become stable sooner than in the original representation and more active columns can be accounted in `Score-row()`. For instance, p_L acts on the first row as $b_j \leftarrow a_j \oplus a_{j+19} \oplus a_{j+28}$. After multiplying the shift offsets by all odd numbers, we found that $q = 7$ results in the minimum span. So, the alternative representation of the linear diffusion layer for the first row is defined as $b'_k \leftarrow a'_k \oplus (a'_k \ggg 5) \oplus (a'_k \ggg 4)$. Fig. 7 provides a comparison between the original and alternative representation of p_L over row 0 where the number of stable bits, that are depicted by blue cells, is higher in the case of alternative representation.

Algorithm 2 Functions to navigate through a row tree

```
function NEXTSTATE()
    if (toFirstChildState() == true) then
        if (Score-state() < T_2) then
            return true;
    do
        while (ToSiblingState() == true) do
            if (Score-state() < T_2) then
                return true;
    while (toParentState() == true)
    return false;
end function

function TOFIRSTCHILDSTATE()
    if (i_l = 4) then              ▷ Last active row index has reached the bottom row
        return false;
    a ← a ∪ [(1 + i_l, 0)];        ▷ Set the last active row to the smallest active row value
    return true
end function

function TOSIBLINGSTATE()
    if (nextRow() == true) then
        return true;
    if (i_l = 4) then
        return false;
    i_l ← 1 + i_l;                 ▷ The last active row is moved one row index down
    a_{i_l} = [(i_l, 0)]           ▷ Set the last active row to the smallest active row value
    return true
end function
```

Algorithm 3 Function to navigate through a bit tree

```
function NEXTROW()
    if (Score-row() < T_2) then
        if ((toFirstChildRow()) && (is canonical) ) then
            return true;
    do
        while ((toSiblingRow()) && (is canonical) ) do
            return true;
    while (toParentRow() == true)
    return false;
end function
```

We denote by p'_L the alternative linear mapping of p_L for each row such that

$$p_L = \pi_{q^{-1}} \circ p'_L \circ \pi_q$$

where $\pi_q(j) = q \times j \bmod 64$. Fig. 8 illustrates how p'_L in the new representation works for row 0. The list of parameters for the different rows of the alternative representation of p_L with the minimum span are listed in Table 2. The alternative representation of $p_L{}^\mathsf{T}$ corresponds to the mapping obtained with $-q$.

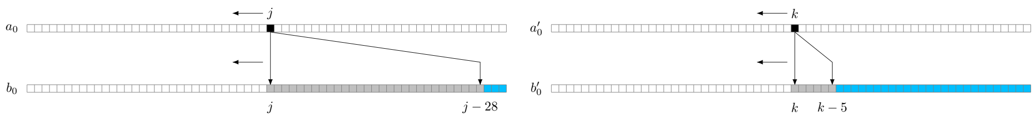


Figure 7: The grey cells at b and b' represent the span in the original and alternative representation of row x_0, respectively. The original representation (left figure) results in a lower number of stable bits at b (blue cells) compared to its alternative representation on the right.

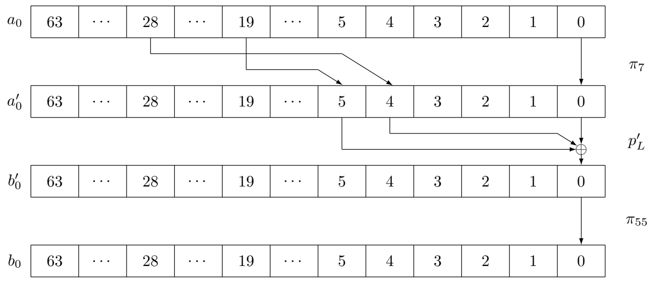


Figure 8: The linear mapping p_L for row 0 can be seen as its alternative representation p'_L surrounded by two multiplication layers, illustrated for bit b_0.

5 Extension in Ascon

In this section, we explain how we perform trail core extension in Ascon. We partially rely on previous works on Keccak-p [DV12, MDV17] and Xoodoo [DHVV18b, DHP+20], given that extension deals with the non-linear layer of Ascon p_S which is based on χ_5.

Given a trail core $\tilde{Q} = (a^1, \ldots, b^{r-1})$, we recall that forward extension by one round consists in building all patterns a^r that are compatible with b^{r-1} over p_S^* and compute $b^r = p_L^*(a^r)$. While backward extension consists in building all patterns b^0 that are compatible with a^1 over p_S^* and compute $a^0 = {p_L^*}^{-1}(b^0)$.

The non-linear layer of Ascon can be seen as the parallel application of 64 5-bit S-boxes, acting on each column independently. Therefore, we can treat extension at column level. If b^0 and a^1 are compatible over p_S^*, then the j-th column of b^0 is compatible with the j-th column of a^1 over $\mathcal{S}$, for any column index $0 \leq j < 64$. To build all states b^0 that

Table 2: List of parameters for the original and alternative representation of the linear diffusion layer of Ascon.

	original representation			alternative representation			
row	offset$_1$	offset$_2$	span	q	offset$_1$	offset$_2$	span
0	19	28	28	7	4	5	5
1	61	39	25	41	5	63	6
2	1	6	6	1	1	6	6
3	10	17	17	19	3	62	5
4	7	41	30	47	7	9	9

are compatible with a^1, we first need to identify the active columns in a^1, namely, the non-zero columns. Then, for each active column, we build all compatible column values at b^0 through $\mathcal{S}$. By combining them, we can finally build all compatible states b^0.

Similarly, we can build all compatible state patterns a^r given b^{r-1}.

5.1 Extension as a tree search

Extension can be performed as a tree search [MDV17, DHVV18b], where we incrementally build b^0 or a^r. To this end we need to define units, their order relation, and a score function. In this case we don't have to deal with canonicity since canonical 2-round trail cores yields canonical r-round trail cores. A trail core is a sequence of state patterns. Translating each pattern of the sequence by a fixed offset results in an equivalent trail core with the same weight. We can define a canonical trail core as the smallest among its translated versions. We can say that a core $(a^1, b^1, \ldots, b^{r-1})$ is smaller than a core $(\bar{a}^1, \bar{b}^1, \ldots, \bar{b}^{r-1})$ if a^1 is smaller than $\bar{a}^1$, or if $a^1 = \bar{a}^1$ and b^1 is smaller than $\bar{b}^1$, etc. However, we can choose any intermediate pattern in the sequence instead of a^1 to start the comparison. It is then natural to start from the $(r-1)$-round trail core from which the r-round core is generated. We say that an r-round trail core is canonical if the $(r-1)$-round trail core from which it is generated is canonical. It follows that the generation of only canonical 2-round trail cores, yields to canonical r-round trail cores naturally.

Differently from the tree search for the generation of 2-round trail cores where a unit was an active bit, here units are determined by the compatible column values. At each move in the tree, we fix the value of an active column of the state. To efficiently traverse the tree we need a score function that lower bounds the weight of the $(r+1)$-round trail cores obtained.

In forward extension this translates into lower bounding $\mathrm{w}(b^r)$ while we are building a^r. The addition of a unit at a^r can cancel some bits at b^r because of the action of p_L^*. To define a good score function, we consider the stable bits at b^r, that are active bits that cannot be cancelled with the addition of any new unit. We represent stable bits by a stability mask $\mathcal{M}$, that is a state where a bit is 1 to indicate that the bit in that position is stable and 0 otherwise. Then $b^r \wedge \mathcal{M}$ gives the stable bits of b^r, and also the column of b^r that will be active in all its descendants. We can define the score as twice the number of active columns in $b^r \wedge \mathcal{M}$.

In backward extension we have to lower bound $\mathrm{w}_{\mathrm{rev}}(a^0) + \mathrm{w}(b^0)$ while we are building b^0. While the addition of a unit at b^0 cannot turn active bits into passive, adding a unit at b^0 can potentially cancel many bits at a^0, since the inverse of p_L^* is dense. In Keccak-p [DV12, MDV17], this problem was overcome by not considering the contribution of a^0 and by bounding $\mathrm{w}_{\mathrm{rev}}(a^0) + \mathrm{w}(b^0)$ with a bound on $\mathrm{w}(b^0)$ only. However, this is sub-optimal. In this work, we use stability masks to determine the stable bits of a^0 and thus consider also its contribution.

In general, the goal is to make the number of stable bits in the stability masks grow as quickly as possible while traversing the tree, so that more columns are counted in the score and pruning happens as early as possible. To this end, the order relation among the units must be carefully defined.

5.2 Forward Extension

For forward extension, we follow the approach used in [DHVV18b] for Xoodoo, that is the following. All patterns a^r that are compatible with b^{r-1} over p_S^* form an affine space $\mathcal{A}(b^{r-1})$ with $2^{\mathrm{w}(b^{r-1})}$ elements. We represent such space through an offset and a basis. Each column at b^{r-1} defines an offset and basis for the space of compatible columns over $\mathcal{S}$, according to Table 7 and Table 8. The state offset, that we denote by $\mathtt{o}$, is built by gathering together all the column offsets. It will be zero in all column positions that are passive in b^{r-1}. For each column vector $\mathtt{u}$ specified by each active column j, we build a state vector $\mathtt{v}$ that is all zero except column j that has value $\mathtt{u}$. The basis has $\mathrm{w} = \mathrm{w}(b^{r-1})$ elements that we denote by $\{\mathtt{v}_1, \mathtt{v}_2, \ldots, \mathtt{v}_\mathrm{w}\}$. Therefore, $\mathcal{A}(b^{r-1}) = \mathtt{o} + \langle \mathtt{v}_1, \mathtt{v}_2, \ldots, \mathtt{v}_\mathrm{w}\rangle$.

Of course, brute-force scanning the whole affine space becomes unaffordable when $\mathrm{w}(b^{r-1})$ is large. However, we only need to construct those states a^r such that the weight of b^r is below a given threshold. For this reason, it is practical to directly consider the affine space mapped through p_L^*, namely before the next p_S^*. We denote such space by $\mathcal{B}(b^{r-1}) = p_L^*(\mathcal{A}(b^{r-1})) = \mathtt{o}^* + \langle \mathtt{v}_1^*, \mathtt{v}_2^*, \ldots, \mathtt{v}_\mathrm{w}^*\rangle$, with $\mathtt{o}^* = p_L^*(\mathtt{o})$ and $\mathtt{v}_k^* = p_L^*(\mathtt{v}_k)$.

We scan the space $\mathcal{B}(b^{r-1})$ through a tree-based search as follows. The root of the tree is the offset $\mathtt{o}^*$. The units are the indexes of the basis vectors, ordered by the natural number ordering. A unit-list $\mathcal{K} = \{k_1, \ldots, k_m\}$ encodes the element of the affine space given by $\mathtt{o} + \mathtt{v}_{k_1}^* + \cdots + \mathtt{v}_{k_m}^*$.The children of $\mathcal{K}$ are all nodes of the form $\mathcal{K} \cup k_{m+1}$ with $k_{m+1} \in \{k_m + 1, \ldots, k_\mathrm{w}\}$.

We need to define stability masks so that the number of stable bits increases quickly with k. A technique to do it consists in triangularizing the basis $\mathcal{V}^* = \{\mathtt{v}_1^*, \mathtt{v}_2^*, \ldots, \mathtt{v}_\mathrm{w}^*\}$. We perform triangularization in Ascon as follows. We start with an empty basis $\mathcal{T}$. We loop on all possible bit positions considering the lexicographic order relation on coordinates (i, j). If a basis vector is found with an active bit in position (i, j), then such basis vector is added to $\mathcal{T}$ and removed from $\mathcal{V}^*$. The same vector is also added to all remaining vectors in $\mathcal{V}^*$ that have bit (i, j) active, to make it passive. After triangularization, we obtain a new representation of $\mathcal{B}(b^{r-1})$ as $\mathtt{o}^* + \langle t_1, t_2, \ldots, t_\mathrm{w}\rangle$. If the first active bit in t_k is in position (i_k, j_k), then, by construction, all bits in position $(i, j) \le (i_k, j_k)$ are passive in all vectors $t_{k+1}, \ldots, t_\mathrm{w}$. We call (i_k, j_k) the pivot position of vector t_k. For each k, we define the stability mask $\mathcal{M}_k$ as a state that is 1 in the pivot position and in all positions smaller than the pivot (i.e. in all $(i, j) \le (i_k, j_k)$) and 0 otherwise. In addition we consider the position of the stable bits in the offset as $\mathcal{O} = \bigwedge_{i=1}^{\mathrm{w}} \overline{t_i}$. We add them to each stability mask: $\mathcal{M}_k = \mathcal{M}_k \vee \mathcal{O}$.

If the last unit in the list of a node b^r is k, then all bits in $b^r \wedge \mathcal{M}_k$ will be active in all descendants of b^r. Therefore, all active columns of $b^r \wedge \mathcal{M}_k$ will be active in all descendants of b^r and each will contribute at least 2 to the weight. We define the score as twice the number of active columns of $b^r \wedge \mathcal{M}_k$.

5.3 Backward Extension

Given a^1, the patterns b^0 that are compatible with a^1 over p_S^* do not form an affine space, so we shall use a different approach than the one for forward extension.

We present two methods to perform backward extension. In the first one, presented in Section 5.3.1, we follow the method used in [DV12] for Keccak-p, that builds on the compatible column values, and we introduce some optimizations. Notice that [MDV17] presents some optimizations for backward extension in Keccak-p, that exploit the structure of the linear step θ, which is a column parity mixer. Such techniques do not apply to Ascon since its linear layer has a different structure. In the second method, presented in Section 5.3.2, we build an envelope space that contains the set of compatible patterns, with the aim of growing the number of active columns in a^0 more quickly. The former method is more effective when the number of active columns in a^1 is small enough, say less than 12. The second method is more effective when there are many active columns in a^1. In our code we use both of them, considering the number of active columns at hand.

5.3.1 Extension using compatible patterns

For each active column position j in a^1, let $\mathcal{B}_j = \{\mathtt{v}_{j,1}, \ldots, \mathtt{v}_{j,n(j)}\}$ denote the set of compatible column patterns at the input of p_S^*. The number of compatible patterns b^0 is given by $\prod_j |\mathcal{B}_j|$. Since $n(j)$ ranges between 9 and 12 for compatible differences and is 10, 13 or 16 for compatible masks, the number of patterns b^0 grows very quickly with the number of active columns in a^1 and it can be unaffordable to generate all of them. However, we need to generate only those such that $\mathrm{w}_{\mathrm{rev}}(a^0) + \mathrm{w}(b^0)$ is smaller than a given threshold T. We can do it using a tree-based approach where the nodes of the tree are the patterns b^0 and units and score function are defined as follows.

The root of the tree is the fully passive state. The units are the indexes of the elements of the sets $\mathcal{B}_j$ ordered by the lexicographic order over (j, k). A unit-list can contain at most one element per set of column patterns for a given index j. At height h in the tree, all the first h active columns are set. Only the leaves of the tree give compatible patterns.

The score function shall bound the quantity $\mathrm{w}_{\mathrm{rev}}(a^0) + \mathrm{w}(b^0)$ for a node and all its descendants. It is defined as $\mathsf{score}_a + \mathsf{score}_b$ with score_a that bounds $\mathrm{w}_{\mathrm{rev}}(a^0)$ and score_b that bounds $\mathrm{w}(b^0)$.

We start with the explanation of score_b that we compute as in [DV12, MDV17]. We order the elements of each $\mathcal{B}_j$ by increasing weight so that $\mathrm{w}(\mathtt{v}_{j,k}) \leq \mathrm{w}(\mathtt{v}_{j,k+1})$ for all k. We denote by w_j the minimum of such weights, that is $\mathrm{w}_j = \mathrm{w}(\mathtt{v}_{j,1})$. For a node at height h, the first h active columns are set and their value cannot change by the addition of a new unit. Each of the remaining active column will contribute to the weight by at least w_j. Therefore, for a node b^0 we define $\mathsf{score}_b(b^0) = \mathrm{w}(b^0) + \sum_{h<j} \mathrm{w}_j$.

For Keccak-p [DV12, MDV17], $\mathsf{score}_a = 2$ since a non-passive state has weight at least 2. This is sub-optimal because it does not take into account the contribution of the active bits at a^0. In this work, we define score_a based on the stable bits of a^0 in the following way. We map each set $\mathcal{B}_j$ before p^*_L obtaining $\mathcal{A}_j = \{\mathtt{v}^*_{j,1}, \ldots, \mathtt{v}^*_{j,n(j)}\}$, where $\mathtt{v}^*_{j,k} = {p^*_L}^{-1}(\mathtt{v}^*_{j,k})$. At height h, one element of each $\mathcal{A}_j$ with $j \leq h$ has been added to a^0 and any element of $\mathcal{A}_j$ can potentially be added for all $j > h$. The OR of the elements that can still be added gives the set of bits that can be potentially cancelled at a^0. Its negation gives the stable bits. Therefore, for each h, we define the stability mask

$$\mathcal{M}_h = \overline{\bigvee_{h<j} \left(\bigvee_{k} \mathtt{v}^*_{j,k} \right)} = \bigwedge_{h<j} \left(\bigwedge_{k} \overline{\mathtt{v}^*_{j,k}} \right) .$$

For a node a^0 at height h, all bits of $a^0 \wedge \mathcal{M}_h$ will be active in all descendants of a^0. Therefore, all active columns of $a^0 \wedge \mathcal{M}_h$ will be active in all descendants of a^0 and each will contribute at least 2 to the weight. We define score_a as twice the number of active columns of the state $a^0 \wedge \mathcal{M}_h$.

The ordering of the elements in each $\mathcal{B}_j$ by increasing weight implies that the right-siblings of a node have weight (resp. score) greater than or equal to the weight (resp. score) of that node. It follows that when a node is encountered whose score is greater than the given threshold all its descendants and also all its siblings can be pruned.

As an additional optimization, we observe that during the backward extension of a trail core $\tilde{Q}_r = (a^1, \ldots, b^{r-1})$, $\mathrm{w}_{\mathrm{rev}}(a^1)$ is replaced by $\mathrm{w}(b^0)$ which can be larger than $\mathrm{w}_{\mathrm{rev}}(a^1)$. If $\mathrm{w}(\tilde{Q}_r) < T_r$ for a given T_r, most of the times we want $\mathrm{w}(\tilde{Q}_r) - \mathrm{w}_{\mathrm{rev}}(a^1) + \mathrm{w}(b^0)$ to be still smaller than T_r. So, during the search we perform the additional check $\mathsf{score}_b < T_r - (\mathrm{w}(\tilde{Q}_r) - \mathrm{w}_{\mathrm{rev}}(a^1))$.

5.3.2 Extension using the envelope space

This method aims at prioritizing the growth of the number of active columns in a^0, so that $\mathrm{w}_{\mathrm{rev}}(a^0)$ grows as quickly as possible.

First, we build a space that contains the set of compatible states b^0's, that we call *envelope space* and denote by $\mathcal{E}$. To do this, for each active column at a^1 we define the envelope space of its compatible column patterns as $0 + \langle \mathsf{e}_0, \mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3, \mathsf{e}_4 \rangle$, where $\mathsf{e}_i \in \mathbb{F}_2^5$ has a single active bit in position i. The envelope space $\mathcal{E}$ is the union of all these envelope spaces and its dimension is five times the number of active columns in a^1.

We scan $\mathcal{E}$ in a tree-based fashion as done in Section 5.2, where the root of the tree is the offset (in this case the all zero state) and we iteratively add basis vectors. Since the envelope space is much larger than the actual space of compatible states, we must define a score function that is very efficient and allows to prune the tree as soon as possible. To this end, we try to make the number of stable bits in a^0 to grow as quickly as possible. A way to do it is to consider the envelope space before p_L^* and triangularize its basis.

Let $\mathcal{E} = \langle \mathtt{v}_1, \ldots, \mathtt{v}_{5n} \rangle$, where n denotes the number of active columns in a^1. Since p_L^* is linear, we can transpose the envelope space $\mathcal{E}$ before p_L^* and get $\mathcal{E}^* = \langle \mathtt{v}_1^*, \ldots, \mathtt{v}_{5n}^* \rangle$ with $\mathtt{v}_k^* = {p_L^*}^{-1}(\mathtt{v}_k)$. We triangularize the basis of $\mathcal{E}^*$ based on the lexicographic order relation on coordinates (i, j) and we modify the representation of $\mathcal{E}$ accordingly. That is, when we add a vector $\mathtt{v}_k^*$ to a vector $\mathtt{v}_\ell^*$ in $\mathcal{E}^*$, we add $\mathtt{v}_k$ to $\mathtt{v}_\ell$ in $\mathcal{E}$. We obtain a new representation of $\mathcal{E}^*$ as $\langle t_1^*, \ldots, t_{5n}^* \rangle$. By construction, the triangularized basis contains first all basis vectors with active bits in row 0, then those with active bits in row 1, etc.

For each k, we define the stability mask $\mathcal{M}_k$ as a state that is 1 in all positions smaller or equal than the pivot position of t_k^*, and 0 otherwise. We define score_a as twice the number of active columns of $a^0 \wedge \mathcal{M}_k$. Finally, we define score_b as twice the number of active columns in a^1. In fact, the number of active columns in b^0 is the same of a^1 and each contributes at least 2 to the weight. On the other hand, since we are scanning the envelope space and not only the space of compatible states, we cannot use the weight of b^0, because in this case the addition of a new unit can potentially decrease it.

6 Practical results and improved bounds for Ascon

In this section, we report on our practical results. The improved bounds are reported in Table 1. To scan the different spaces of trail cores, we follow the different strategies presented in [DV12, MDV17, DHP+20, DMA22]. We used parts of KeccakTools [DHVV13] and XooTools [DHVV18a] for some routines for trail extension. All our tests are run on a server equipped with an AMD EPYC 7552 48-Core Processor @2.20GHz. We exploited the multicore architecture to run some of our tests in parallel, but execution times are reported as single core costs in the following. We round up the execution time to the closest integer.

In some cases, we compare our execution time to that reported in [EME22], which uses machines equipped with Intel Xeon E5-2669 and E5-4669 v4 @2.20GHz. Even if the machines are different, and thus execution times are not perfectly comparable, we can observe that our methods allow us to scan larger spaces than what was possible with the solvers-based method of [EME22].

In the following, we denote by $\mathcal{D}_r^T$ the space of all r-round differential trail cores with weight $< T$, i.e. at most $T - 1$. Similarly, we denote by $\mathcal{L}_r^T$ the space of all r-round differential trail cores with weight $< T$.

6.1 Results on 3 rounds: tight bound and all low-weight trails

Since the best known 3-round differential and linear trails have weight 40 [DEMS15] and 28 [DEM15] respectively, we scanned the spaces $\mathcal{D}_3^{41}$ and $\mathcal{L}_3^{30}$ to check whether they are the lightest trails[1]. Our experimental results confirmed the results for differential trails in [EME22, MR22] and proved that 28 is the tight bound for linear trails. In fact, we found 2 differential trail cores of weight 40, 1 linear trail core of weight 28, and no trail cores with lower weight. The search took less than 3 minutes for differential trails and less than 4 seconds for linear trails.

[1] Notice that to prove that they are the lightest trails, it is sufficient to scan the spaces $\mathcal{D}_3^{40}$ and $\mathcal{L}_3^{28}$ and prove that they are empty. To check how many differential trail cores of weight 40 and linear trail cores of weight 28 there exist, we chose to scan larger spaces.

Table 3: Details on the generation of canonical 3-round differential and linear trail cores below target weight 41 and 30, respectively.

search space	# cores	time	search details: step	# cores	time
$\mathcal{D}_3^{41}$	2	3m	$2\mathrm{w}_{\mathrm{rev}}(a_1) + \mathrm{w}(b_1) < 40$	284,561	2m
			forward extension	2	4s
			$\mathrm{w}(b_1) + 2\mathrm{w}(b_2) \leq 40$	15,252	28s
			backward extension	0	2s
$\mathcal{L}_3^{30}$	1	4s	$2\mathrm{w}_{\mathrm{rev}}(a_1) + \mathrm{w}(b_1) < 28$	1,935	1s
			forward extension	1	1s
			$\mathrm{w}(b_1) + 2\mathrm{w}(b_2) \leq 28$	972	1s
			backward extension	0	1s

To scan the above spaces, we followed the approach used in [DHVV18b], which is the following. A 3-round trail core has weight $\mathrm{w}_{\mathrm{rev}}(a^1) + \mathrm{w}(b^1) + \mathrm{w}(b^2)$. We split all trail cores in $\mathcal{D}_3^{41}$ (resp. $\mathcal{L}_3^{30}$) into two sets based on whether $\mathrm{w}_{\mathrm{rev}}(a^1) < \mathrm{w}(b^2)$ or $\mathrm{w}_{\mathrm{rev}}(a^1) \geq \mathrm{w}(b^2)$.

- The former case implies that $2\mathrm{w}_{\mathrm{rev}}(a^1) + \mathrm{w}(b^1) < 40$ (resp. < 28). Such trail cores can be obtained by generating all 2-round trail cores (a^1, b^1) satisfying this inequality and extending them in the forward direction by one round up to 40 (resp. 28).
- The latter case implies that $\mathrm{w}(b^1) + 2\mathrm{w}(b^2) \leq 40$ (resp. ≤ 28). Such trails can be obtained by generating all 2-round trail cores (a^2, b^2) satisfying this inequality and then extending them in the backward direction by one round up to 40 (resp. 28).

Detailed execution times are given in Table 3 together with the number of trail cores found in each step of the search.

Beyond proving bounds for 3-round trails, we are also interested in the distribution of low-weight 3-round trails in Ascon. To this end, we also scanned the space $\mathcal{D}_3^{51}$ (resp. $\mathcal{L}_3^{52}$), and counted all 3-round trails contained in such cores with weight below 51 (resp. 52). To count trails, we used the code for backward extension to build all patterns b^0 compatible with a^1 that satisfy $\mathrm{w}(b^0) + \mathrm{w}(b^1) + \mathrm{w}(b^2) < 51$ (resp. < 52) and we count each of them $\mathrm{w}(b_2)$ times. Results are depicted in Fig. 9. We can notice that, per given (even) weight ≥ 40, the ratio between the number of linear trails and differential trails ranges between 9.7 (for weight 46) and 66.5 (for weight 44). This is due to the fact that the LAT of the Ascon S-box is more dense than its DDT.

6.2 Results on 4 rounds: improved (non-tight) bounds

The best known 4-round differential and linear trails in Ascon have weight 107 and 98 respectively [DEMS15, DEM15], while the previously proved lower bound is 72 for both [EME22].

With our techniques, we scanned the spaces $\mathcal{D}_4^{86}$ and $\mathcal{L}_4^{88}$. We found that both spaces are empty, which implies that any 4-round differential trail has weight at least 86 and any 4-round linear trail has weight at least 88. This improves over known results, even if the new bounds are still not tight.

Our search took around 13 days for differential trails and around 110 days for linear trails. While in [EME22] the authors report a cost of 600 days each for differential and linear trails to prove a bound of 72. Moreover, the authors in [EME22] estimate a cost of

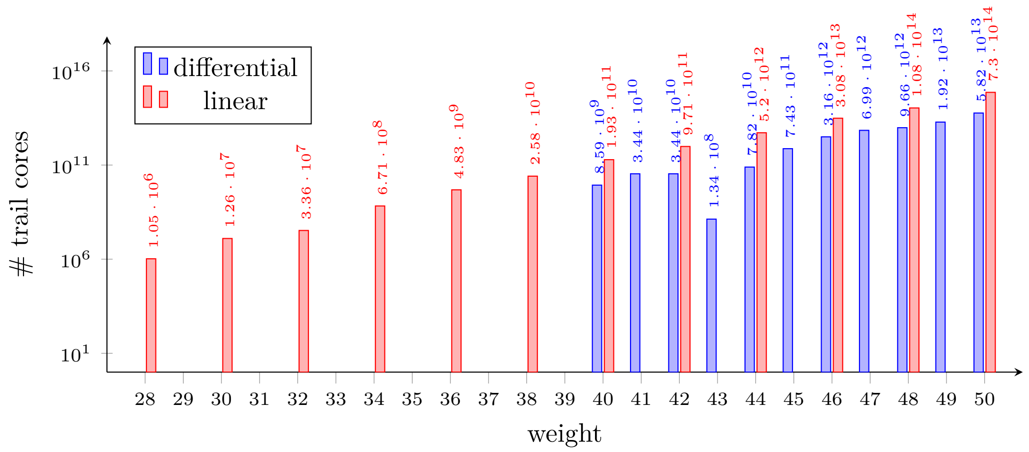


Figure 9: Number of all canonical 3-round trails per weight.

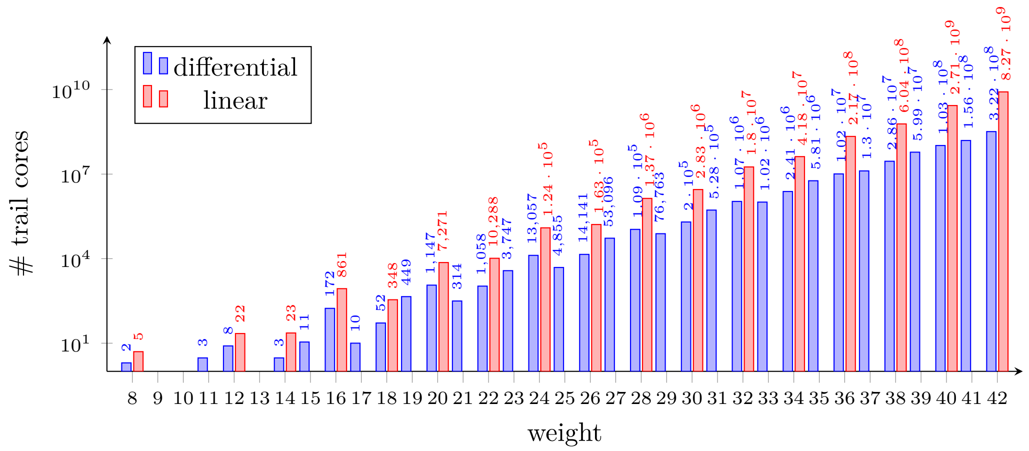


Figure 10: Number of all canonical 2-round trail cores per weight.

6688 days to prove a bound of 80 whereas in [MR22], they estimate 3898 days to prove this bound. Therefore, with our method we could reach higher bounds with significantly less computational cost.

To scan the above spaces, we followed [DHVV18b]. Any 4-round differential (resp. linear) trail core with weight $w_{\text{rev}}(a^1) + w(b^1) + w(b^2) + w(b^3) < 86$ (resp. < 88) has $w_{\text{rev}}(a^1) + w(b^1) < 43$ (resp. < 44) or $w(b^2) + w(b^3) < 43$ (resp. < 44). Otherwise, their sum would be at least 86 (resp. 88). We could thus generate all trail cores in $\mathcal{D}_4^{86}$ (resp. $\mathcal{L}_4^{88}$) by generating all 2-round trail cores in $\mathcal{D}_2^{43}$ (resp. $\mathcal{L}_2^{44}$) and extending them to 4 rounds below 86 (resp. 88). To perform extension to 4 rounds, we first extended to 3 rounds below 84 (resp. 86), since we know that the remaining round has weight at least 2.

Details on the number of trail cores found in each step of the search and the execution times are reported in Table 4. In Fig. 10, we report the number of all 2-round trail cores per given weight. Again, we can observe that (for even weights) the number of 2-round linear trail cores found is significantly higher than the number of 2-round differential trail cores. This difference of course reflects on the costs for extension.

Table 4: Details on the generation of canonical 4-round differential and linear trail cores with weight lower than 86 and 88, respectively. Timings are rounded to the closest integer.

search space	# cores	time	search details step	# cores	time
$\mathcal{D}_4^{86}$	0	310h	generation of $\mathcal{D}_2^{43}$	704,744,005	100h
			forw.ext. to 3 rounds with $\mathrm{w} < 84$	2,421,335	140h
			forw.ext. to 4 rounds with $\mathrm{w} < 86$	0	1m
			back.ext. to 3 rounds with $\mathrm{w} < 84$	2,424	66h
			back.ext. to 4 rounds with $\mathrm{w} < 86$	0	3h
$\mathcal{L}_4^{88}$	0	2641h	generation of $\mathcal{L}_2^{44}$	11,866,934,404	397h
			forw.ext. to 3 rounds with $\mathrm{w} < 86$	44,850,380	1411h
			forw.ext. to 4 rounds with $\mathrm{w} < 88$	0	25m
			back.ext. to 3 rounds with $\mathrm{w} < 86$	40,013	671h
			back.ext. to 4 rounds with $\mathrm{w} < 88$	0	161h

6.3 Results on 5 rounds: new (non-tight) bounds

The best known differential trail over 5 rounds has weight 190 [DEMS15, GPT21], while the best known linear trail has weight 184 [MR22]. As far as we know, there are no proved lower bounds for 5-round trails, before this work. We can prove non-tight bounds of 100 for differential trails and 96 for linear trails. To this end, we scanned the spaces $\mathcal{D}_5^{100}$ and $\mathcal{L}_5^{96}$, which resulted to be both empty. Our search took around 158 days for differential trails and around 127 days for linear trails.

To perform our search, we followed the approach of [DHP+20], to re-use the 2-round trail cores already built. We split the space $\mathcal{D}_5^{100}$ (resp. $\mathcal{L}_5^{96}$) into two sets. The first contains all 5-round trail cores with $\mathrm{w}_{\mathrm{rev}}(a^1) + \mathrm{w}(b^1) < 43$ (resp. < 44). To cover it, we extend all 2-round trail cores in $\mathcal{D}_2^{43}$ (resp. $\mathcal{L}_2^{44}$), that we already have, by 3 rounds in the forward direction below weight 100 (resp. 96). The second set contains all 5-round trail cores with $\mathrm{w}_{\mathrm{rev}}(a^1) + \mathrm{w}(b^1) \geq 43$ (resp. ≥ 44). This implies that $\mathrm{w}(b^2) + \mathrm{w}(b^3) + \mathrm{w}(b^4) < 57$ (resp. < 52). Therefore, we generated all 3-round trail cores in $\mathcal{D}_3^{57}$ (resp. $\mathcal{L}_3^{52}$) and extended them backwards below weight 100 (resp. 96).

Details on the different steps of our search are reported in Table 5. As we didn't need to regenerate the 2-round trail cores in $\mathcal{D}_2^{43}$ and $\mathcal{L}_2^{44}$ (because we already generated them for the search over 4 rounds), we report the corresponding time between parentheses and we don't consider it in the total cost of this search.

6.4 Results on 6 rounds: improved bounds beyond 2^{-128}

The previously proved lower bound on the weight of 6-round trails is 108, for both linear and differential trails [EME22]. With our techniques we can prove that the spaces $\mathcal{D}_6^{129}$ and $\mathcal{D}_6^{132}$ are both empty. It follows that any 6-round differential trail has weight at least 129 and any 6-round linear trail has weight at least 132. Even if our new bounds are still not tight, we are able to prove for the first time that 6-round trails in Ascon have differential probability or squared correlation lower than 2^{-128}.

Our search took around 6 days for differential trails and around 21 days for linear trails. While in [EME22], the authors report a cost of 2 months each for differential and linear trails. Both in this work and in [EME22], results for 6 rounds are built on top of results on 3 and 4 rounds, whose cost is not included in the figures for 6 rounds. Even if we include

Table 5: Results on the generation of canonical 5-round differential and linear trail cores with weight lower than 100 and 96, respectively. Timings between parentheses mean that we can reuse previous results and they are not counted in the total amount of time.

search space	# cores	time	search details step	# cores	time
$\mathcal{D}_5^{100}$	0	3795h	generation of $\mathcal{D}_2^{43}$	704,744,005	(100h)
			forw.ext. by 3 rounds with w $<$ 100	0	3683h
			generation of $\mathcal{D}_3^{57}$	437	112h
			back.ext. by 2 rounds with w $<$ 100	0	3s
$\mathcal{L}_5^{96}$	0	3045h	generation of $\mathcal{L}_2^{44}$	11,866,934,404	(397h)
			forw.ext. by 3 rounds with w $<$ 96		3037h
			generation of $\mathcal{L}_3^{52}$	309	8h
			back.ext. by 2 rounds with w $<$ 96	0	1s

such costs in the total computational cost for 6 rounds, our technique still requires less time compared to [EME22] to reach better bounds.

To scan the space $\mathcal{D}_6^{129}$ (resp. $\mathcal{L}_6^{132}$), we followed the approach of [DMA22]. First, we split the space in two subspaces that we denote $\mathcal{S}_1$ and $\mathcal{S}_2$. The set $\mathcal{S}_1$ contains all 6-round trail cores with $\mathrm{w}_{\mathrm{rev}}(a^1) + \mathrm{w}(b^1) + \mathrm{w}(b^2) < 57$ (resp. < 52) or $\mathrm{w}(b^3) + \mathrm{w}(b^4) + \mathrm{w}(b^5) < 57$ (resp. < 52). The space $\mathcal{S}_2$ is the complement of $\mathcal{S}_1$, that is the space of all 6-round trail cores with $\mathrm{w}_{\mathrm{rev}}(a^1) + \mathrm{w}(b^1) + \mathrm{w}(b^2) \geq 57$ (resp. ≥ 52) and $\mathrm{w}(b^3) + \mathrm{w}(b^4) + \mathrm{w}(b^5) \geq 57$ (resp. ≥ 52).

The details of our search are summarized here.

Scanning $\mathcal{S}_1$ starting from $\mathcal{D}_3^{57}$ (resp. $\mathcal{L}_3^{52}$). The space $\mathcal{S}_1$ can be scanned by extending all 3-round trail cores in $\mathcal{D}_3^{57}$ (resp. $\mathcal{L}_3^{52}$) by 3 rounds below weight 129 (resp. 132). We first extended all 3-round trails in the space by 3 rounds in the forward direction and then by 3 rounds in the backward direction. To extend to 6 rounds, we first extended to 4 rounds below 121 (resp. 122) because we know that the two remaining rounds will weight at least 8. Then we extended to 5 rounds below 127 (resp. 130) because we know that the remaining round will weigh at least 2. For both differential and linear case, extension to 5 rounds resulted in an empty set. Therefore, we didn't need to perform extension to 6 rounds.

Scanning $\mathcal{S}_2$ starting from $\mathcal{D}_2^{43}$ (resp. $\mathcal{L}_2^{44}$). The space $\mathcal{S}_2$ is further split into three subsets. In fact, any 6-round trail core with weight below 129 (resp. 132) can be generated by starting from a 2-round trail core of weight below 43 (resp. 44) placed at the beginning, or in the middle, or at the end of the trail. In the first case, the 2-round trail core is extended by four rounds in the forward direction. In the second case, it is extended by two rounds in the forward direction and two rounds in the backward direction. In the last case, it is extended by four rounds in the backward direction.

- **Starting from the beginning.** To extend 2-round trail cores to 6 rounds, we performed extension by one round at the time each time limiting the weight up to which we perform extension, considering the minimum contribution of the remaining rounds.

  First, we extended 2-round trail cores to 3 rounds below $129 - 57 = 72$ (resp. $132 - 52 = 80$) because we are in the case where $\mathrm{w}(b^3) + \mathrm{w}(b^4) + \mathrm{w}(b^5) \geq 57$ (resp.

Table 6: Results on the generation of canonical 6-round differential and linear trail cores with weight lower than 129 and 132, respectively. Timings between parentheses mean that we can reuse previous results and they are not counted in the total amount of time. - means that the step was not performed, because we know it leads to an empty space.

search space	# cores	time	search details: step	# cores	time
$\mathcal{D}_6^{129}$	0	135h	generation of $\mathcal{D}_3^{57}$	437	112h
			forw.ext. by 3 rounds with $w < 129$	0	9h
			backw.ext. by 3 rounds with $w < 129$	0	11h
			generation of $\mathcal{D}_2^{43}$	704,744,005	(100h)
			$\mathcal{D}_2^{43}$ at the beginning		
			- f.e. to 3 rounds with $57 \leq w < 72$	43,465	(140h)
			- f.e. to 4 rounds with $w < 121$	0	3h
			$\mathcal{D}_2^{44}$ in the middle	0	-
			$\mathcal{D}_2^{44}$ at the end	0	-
$\mathcal{L}_6^{132}$	0	493h	generation of $\mathcal{L}_3^{52}$	309	(8h)
			forw.ext. by 3 rounds with $w < 132$	0	7h
			backw.ext. by 3 rounds with $w < 132$	0	450h
			generation of $\mathcal{L}_2^{44}$	11,866,934,404	(397h)
			$\mathcal{L}_2^{44}$ at the beginning		
			- f.e. to 3 rounds with $52 \leq w < 80$	5,171,116	(1411h)
			- f.e. to 4 rounds with $w < 124$	14,082	36h
			- f.e. to 5 rounds with $w < 130$	0	1s
			$\mathcal{L}_2^{44}$ in the middle	0	-
			$\mathcal{L}_2^{44}$ at the end	0	-

≥ 52). Among the obtained 3-round trail cores, we kept only those satisfying $w_{rev}(a^1) + w(b^1) + w(b^2) \geq 57$ (resp. ≥ 52) because otherwise they belong to $\mathcal{S}_1$. Notice that the set of such trail cores is a subset of the set obtained during the search over 4 rounds. In that case in fact, we extended all trail cores in $\mathcal{D}_2^{43}$ (resp. $\mathcal{L}_2^{44}$) to 3 rounds below weight 84 (resp. 86). Therefore, we did not need to perform this step but we just extracted the needed trail cores from such set.

Then, we extended the obtained 3-round trail cores to 4 rounds below $129 - 8 = 121$ (resp. $132 - 8 = 124$) because we know that $w(b^4) + w(b^5) \geq 8$, since any 2-round trail has weight at least 8.

The obtained 4-round trail cores were then extended to 5 rounds below $129 - 2 = 127$ (resp. $132 - 2 = 130$) because we know that $w(b^5) \geq 2$.

Finally, we extended the obtained 5-round trail cores to 6 rounds below 129 (resp. 132).

Notice that, for differential trails, extension to 4 rounds already resulted in an empty set. Therefore, extension to 5 and 6 rounds was not performed. For linear trails, it is extension to 5 rounds that gave an empty set. Therefore, we could skip extension to 6 rounds.

- **Starting from the middle.** We can assume that $w_{rev}(a^1) + w(b^1) \geq 43$ (resp.

≥ 44) because the other case is covered in the previous step. First, we need to perform forward extension to 4 rounds below $129 - 43 = 86$ (resp. $132 - 44 = 88$) because $\mathrm{w}_{\mathrm{rev}}(a^1) + \mathrm{w}(b^1) \geq 43$ (resp. ≥ 44). Notice that we already performed this search in Section 6.2. In fact, this was part of the search to build $\mathcal{D}_4^{86}$ (resp. $\mathcal{L}_4^{88}$), which is empty. Therefore, we did not need to perform this step of the search.

- **Starting from the end.** We can assume that $\mathrm{w}_{\mathrm{rev}}(a^1) + \mathrm{w}(b^1) \geq 43$ (resp. ≥ 44) and $\mathrm{w}(b^2) + \mathrm{w}(b^3) \geq 43$ (resp. ≥ 44), because the opposite is already covered in the two previous steps. First, we need to perform backward extension to 4 rounds below $129 - 43 = 86$ (resp. $132 - 44 = 88$) because $\mathrm{w}_{\mathrm{rev}}(a^1) + \mathrm{w}(b^1) \geq 43$ (resp. ≥ 44). Again, we already performed this search in Section 6.2 to build $\mathcal{D}_4^{86}$ (resp. $\mathcal{L}_4^{88}$). As we already know that this leads to an empty set, we can jump this step of the search.

Figures on the number of trail cores found in each step of the search and details on the execution time of each step are given in Table 6. When we can reuse trail cores generated in previous searches, we put the corresponding computational time between parentheses and we don't include it in the total cost. When a step is not performed because we know that it leads to an empty space, we put a dash.

6.5 Results on 8 rounds: improved (non-tight) bounds

Since $\mathcal{D}_4^{86}$ and $\mathcal{L}_4^{88}$ are empty, we can claim that also $\mathcal{D}_8^{172}$ and $\mathcal{L}_8^{176}$ are empty. In fact, if we split any 8-round differential (resp. linear) trail with weight < 172 (resp. < 176) in two 4-round trails, at least one of the two must have weight < 86 (resp. < 88). Otherwise, their sum would be ≥ 172 (resp. ≥ 176). Therefore, all 8-round differential (resp. linear) trails with weight below 172 (resp. 176) can be obtained by the extension of all 4-round trails with weight below 86 (resp. 88). But, we know that such 4-round trails do not exist. Therefore, also such 8-round trails do not exist. It follows that 172 is a lower bound on the weight of any 8-round differential trail and 176 is a lower bound on the weight of any 8-round linear trail. Such bounds improve over previous known bound, which was 144 for both differential and linear trails. However, they are still non-tight.

6.6 Results on 12 rounds: improved bounds beyond 2^{-256}

With a reasoning similar to the one used for 8 rounds, we can prove that the spaces $\mathcal{D}_{12}^{258}$ and $\mathcal{L}_{12}^{264}$ are empty, given that the spaces $\mathcal{D}_6^{129}$ and $\mathcal{L}_6^{132}$ are empty. It follows that any 12-round differential trail has weight at least 258 and any 12-round linear trail has weight at least 264. Such bounds improve over previous known bound, which was 216 for both differential and linear trails. Even if our new bounds are still non-tight, they allow us to prove for the first time that 12-round trails in Ascon have differential probability or squared correlation lower than 2^{-256}.

6.7 Results on 7, 9, 10, and 11 rounds: improved (non-tight) bounds

Based on the results obtained for 4, 5, and 6 rounds, we can derive bounds on 7, 9, 10, and 11 rounds. We explain how to do it for 10 rounds by combining the results for 4 and 6 rounds. Then we show how to obtain bounds for the other numbers of rounds similarly.

We can cover the space $\mathcal{D}_{10}^{215}$ (resp. $\mathcal{L}_{10}^{220}$) in the following way. We split the set in two subsets. The first contains all 10-round trail cores with $\mathrm{w}_{\mathrm{rev}}(a^1)+\mathrm{w}(b^1)+\mathrm{w}(b^2)+\mathrm{w}(b^3) < 86$ (resp. 88), while the second set is its complement. We can cover the first set by extending all 4-round trail cores in $\mathcal{D}_4^{86}$ (resp. $\mathcal{L}_4^{88}$) by 6 rounds in the forward direction below 215 (resp. 220). The second set contains all 10-round trails with $\mathrm{w}_{\mathrm{rev}}(a^1)+\mathrm{w}(b^1)+\mathrm{w}(b^2)+\mathrm{w}(b^3) \geq 86$ (resp. 88), which implies that the other 6 rounds have weight below 129 (resp. 132). Therefore, we can cover it by extending all 6-round trail cores in $\mathcal{D}_6^{129}$ (resp. $\mathcal{L}_6^{132}$) by 4 rounds in the backward direction below 215 (resp. 220). Since both $\mathcal{D}_4^{86}$ and $\mathcal{D}_6^{129}$ (resp. $\mathcal{L}_4^{88}$ and $\mathcal{L}_6^{132}$) are empty, then also $\mathcal{D}_{10}^{215}$ (resp. $\mathcal{L}_{10}^{220}$) is empty. Therefore, 215 and 220 are lower bounds on the weight of 10-round differential and linear trails, respectively.

For the other numbers of rounds, we consider the combination that yields the best bounds. For 7 rounds, we can prove a bound of 131 for differential trails and 134 for linear trails, considering the results on 6 rounds and that 1 round weights at least 2. For 9 rounds, we combine the results for 4 and 5 rounds and obtain a bound of 186 for differential trails and 184 for linear trails. Finally, for 11 rounds we obtain a bound of 229 for differential trails and 228 for linear trails, by combining the results for 5 and 6 rounds.

For the sake of comparison, we can apply the same reasoning to the results presented in [EME22]. We can derive bounds for r rounds from the bounds on $r-1$ rounds, considering that one round has minimum weight 2. For differential and linear trails, this gives a bound of 74 for 5 rounds, of 110 for 7 rounds, of 146 for 9 rounds, and 182 for 11 rounds.

7 Conclusions

In this work, we presented a dedicated tool for trail search in Ascon, based on the 2-round trail core generation methods given in [MDV17] and improved methods for extension based on the works done in [DV12, DHVV18b]. Using these techniques, we proved tight bound for 3-rounds linear trails and improved the existing bounds for other number of rounds. In particular, we prove for the first time bounds beyond 2^{-128} for 6 rounds, and for 12 rounds bounds beyond 2^{-256}. Our approach improves and proves bounds in a reasonable amount of time and it confirms that dedicated tools can still outperform methods based on general-purpose solvers.

As a takeaway from this and previous works on Keccak-p [MDV17], Xoodoo [DHVV18b], and Subterranean [MMGD22] we highlight that:

- For the 2-round trail search stage, the linear layers of Ascon and Subterranean allow a simpler definition of units compared to Keccak-p and Xoodoo where a more complex linear layer is used.

- A non-linear layer based on the parallel application of small S-boxes (as in Keccak-p, Xoodoo and Ascon) implies a simpler analysis of the propagation properties compared to the non-linear layer of Subterranean. In the latter case, the backward extension is more complex, and the definition of the minimum reverse weight requires a thorough proof which makes it more complicated.

Acknowledgements

Solane El Hirch and Silvia Mella are supported by the Cryptography Research Center of the Technology Innovation Institute (TII), Abu Dhabi (UAE), under the TII-Radboud project with title *Evaluation and Implementation of Lightweight Cryptographic Primitives and Protocols*.
Alireza Mehrdad and Joan Daemen are supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA.

References

[BDPV07] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge functions. https://keccak.team/files/SpongeFunctions.pdf, 2007.

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

[BDPV11a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[BDPV11b] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. The keccak reference, January 2011.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *Advances in Cryptology - CRYPTO 2016*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption. In *Cryptographic Hardware and Embedded Systems - CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, 2017.

[com14] CAESAR committee. CAESAR: Competition for authenticated encryption: Security, applicability, and robustness, 2014.

[Dae95] Joan Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis.* PhD thesis, K.U.Leuven, 1995.

[DEM15] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Heuristic tool for linear cryptanalysis with applications to CAESAR candidates. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 490–509. Springer, 2015.

[DEM+20] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. Isap v2.0. *IACR Trans. Symmetric Cryptol.*, 2020(S1):390–416, 2020.

[DEMS15] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Cryptanalysis of ascon. In Kaisa Nyberg, editor, *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, volume 9048 of *Lecture Notes in Computer Science*, pages 371–387. Springer, 2015.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. submission to caesar competition. Technical report, 2016.

[DEMS21a] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.

[DEMS21b] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. submission to nist. Technical report, 2021.

[DHP+20] Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Xoodyak, a lightweight cryptographic scheme. *IACR Trans. Symmetric Cryptol.*, 2020(S1):60–87, 2020.

[DHVV13] J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer. KeccakTools software. https://github.com/KeccakTeam/KeccakTools, 2013.

[DHVV18a] J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer. XooTools software. https://github.com/XoodooTeam/Xoodoo, 2018.

[DHVV18b] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. The design of xoodoo and xoofff. *IACR Trans. Symmetric Cryptol.*, 2018(4):1–38, 2018.

[DMA22] Joan Daemen, Silvia Mella, and Gilles Van Assche. Tighter trail bounds for xoodoo. Cryptology ePrint Archive, Paper 2022/1088, 2022. https://eprint.iacr.org/2022/1088.

[DPAR00] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie proposal: the block cipher Noekeon. Nessie submission, 2000. http://gro.noekeon.org/.

[DR20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition*. Information Security and Cryptography. Springer, 2020.

[DV12] Joan Daemen and Gilles Van Assche. Differential propagation analysis of keccak. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 422–441. Springer, 2012.

[EME22] Johannes Erlacher, Florian Mendel, and Maria Eichlseder. Bounds for the security of ascon against differential and linear cryptanalysis. *IACR Trans. Symmetric Cryptol.*, 2022(1):64–87, 2022.

[GPT21] David Gérault, Thomas Peyrin, and Quan Quan Tan. Exploring differential-based distinguishers and forgeries for ASCON. *IACR Trans. Symmetric Cryptol.*, 2021(3):102–136, 2021.

[MDV17] Silvia Mella, Joan Daemen, and Gilles Van Assche. New techniques for trail bounds and application to differential trails in Keccak. *IACR Trans. Symmetric Cryptol.*, 2017(1):329–357, 2017.

[MMGD22] Alireza Mehrdad, Silvia Mella, Lorenzo Grassi, and Joan Daemen. Differential trail search in cryptographic primitives with big-circle chi: Application to subterranean. *IACR Trans. Symmetric Cryptol.*, 2022(2):253–288, 2022.

[MP13] Nicky Mouha and Bart Preneel. Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Report 2013/328, 2013. https://ia.cr/2013/328.

[MR22] Rusydi H. Makarim and Raghvendra Rohit. Towards tight differential bounds of ascon: A hybrid usage of smt and milp. *IACR Transactions on Symmetric Cryptology*, 2022(3):303–340, 2022.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers. In *Advances in Cryptology - ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 158–178. Springer, 2014.

[TMC+21] Meltem Sonmez Turan, Kerry McKay, Donghoon Chang, Cagdas Calik, Lawrence Bassham, Jinkeon Kang, and John Kelsey. Status report on the second round of the nist lightweight cryptography standardization process, 2021.

[WH19] Hongjun Wu and Tao Huang. TinyJAMBU: A Family of LightweightAuthenticated Encryption Algorithms, 2019.

A Representation of the affine spaces over $\mathcal{S}$.

In Table 7, for each possible column difference, we provide a representation of the affine space of compatible differences at the output of $\mathcal{S}$, the restriction weight, and the minimum reverse weight. In Table 8, for each possible column mask, we provide a representation of the affine space of compatible masks at the input of $\mathcal{S}$, the correlation weight, and the minimum reverse weight.

Table 7: Space of compatible differences at the output of p_S, restriction weight, and minimum reverse weight for all possible column differences.

difference	Affine space after $\mathcal{S}$	$\mathrm{w_r}(\cdot)$	$\mathrm{w_{rev}}(\cdot)$
00000	00000	0	0
00001	$01001 + \langle 00010, 00100, 10001 \rangle$	3	2
00010	$10001 + \langle 00010, 00100, 01000 \rangle$	3	2
00011	$00001 + \langle 00100, 01000, 10001 \rangle$	3	3
00100	$00110 + \langle 01000, 10000 \rangle$	2	2
00101	$10001 + \langle 00010, 01001, 01100 \rangle$	3	3
00110	$00001 + \langle 00010, 00100, 01000, 10000 \rangle$	4	2
00111	$00010 + \langle 00001, 00100, 01000 \rangle$	3	3
01000	$00110 + \langle 00001, 01000, 10000 \rangle$	3	2
01001	$00001 + \langle 00010, 10001, 10100, 11000 \rangle$	4	2
01010	$00001 + \langle 00101, 00110, 01000, 10000 \rangle$	4	2
01011	$00010 + \langle 00001, 00100, 01000, 10000 \rangle$	4	2
01100	$00001 + \langle 10001, 11000 \rangle$	2	2
01101	$00001 + \langle 00010, 00100, 10001, 11000 \rangle$	4	3
01110	$00001 + \langle 00101, 00110, 10000 \rangle$	3	2
01111	$01000 + \langle 00001, 00100, 10000 \rangle$	3	3
10000	$01001 + \langle 00010, 10001 \rangle$	2	2
10001	$10001 + \langle 00010, 00100 \rangle$	2	2
10010	$00001 + \langle 00010, 00100, 01000, 10001 \rangle$	4	3
10011	$00010 + \langle 00110, 01000 \rangle$	2	2
10100	$00100 + \langle 00001, 00010, 01000 \rangle$	3	3
10101	$00101 + \langle 00010, 10100, 11000 \rangle$	3	2
10110	$10000 + \langle 00001, 00010, 00100, 01000 \rangle$	4	2
10111	$00010 + \langle 00110, 01000, 10000 \rangle$	3	2
11000	$00100 + \langle 00001, 00010, 01000, 10000 \rangle$	4	2
11001	$01000 + \langle 00101, 10110, 11000 \rangle$	3	2
11010	$00001 + \langle 00100, 01001, 01010, 10000 \rangle$	4	2
11011	$00010 + \langle 00001, 00110, 01000, 10000 \rangle$	4	3
11100	$00001 + \langle 00010, 10001, 11000 \rangle$	3	3
11101	$01000 + \langle 00110, 10101, 11000 \rangle$	3	3
11110	$01000 + \langle 00001, 00010, 00100, 10000 \rangle$	4	2
11111	$00010 + \langle 00001, 00110, 10000 \rangle$	3	3

Table 8: Space of compatible masks at the input of p_S, correlation weight, and minimum reverse weight for all possible column masks.

mask	Affine space before $\mathcal{S}$	$\mathrm{w_c}(\cdot)$	$\mathrm{w_{rev}}(\cdot)$
00000	00000	0	0
00001	$00011 + \langle 01000, 10001\rangle$	2	2
00010	$01100 + \langle 00011, 10000\rangle$	2	2
00011	$00100 + \langle 00001, 00010, 01000, 10000\rangle$	4	2
00100	$01100 + \langle 00001, 00010\rangle$	2	2
00101	$00100 + \langle 00001, 00010, 01000, 10000\rangle$	4	2
00110	$00001 + \langle 10001, 10010\rangle$	2	2
00111	$00010 + \langle 10001, 11010\rangle$	2	2
01000	$10001 + \langle 01010, 01100\rangle$	2	2
01001	$00001 + \langle 00010, 00100, 01000, 10001\rangle$	4	2
01010	$00001 + \langle 01001, 01010, 01100, 10000\rangle$	4	2
01011	$00010 + \langle 00001, 00100, 10010, 11000\rangle$	4	2
01100	$10000 + \langle 00001, 00010, 00100, 01000\rangle$	4	2
01101	$00001 + \langle 00010, 00101, 01000, 10000\rangle$	4	2
01110	$00001 + \langle 10001, 10010, 10100, 11000\rangle$	4	2
01111	$00010 + \langle 00001, 01010, 01100, 10000\rangle$	4	2
10000	$00011 + \langle 01000, 10101\rangle$	2	2
10001	$10001 + \langle 00100, 01000\rangle$	2	2
10010	$00001 + \langle 00101, 00110, 01000, 10000\rangle$	4	2
10011	$00001 + \langle 00011, 00100, 01000, 10000\rangle$	4	2
10100	$00100 + \langle 00001, 00010, 01000, 10100\rangle$	4	4
10101	$10000 + \langle 00001, 00010, 00100, 01000\rangle$	4	2
10110	$00001 + \langle 00100, 01000, 10001, 10010\rangle$	4	2
10111	$00001 + \langle 00100, 01000, 10001, 10010\rangle$	4	2
11000	$00001 + \langle 00010, 00100, 01000, 10001\rangle$	4	2
11001	$00100 + \langle 00010, 01100\rangle$	2	2
11010	$00001 + \langle 00010, 00100, 10001, 11000\rangle$	4	2
11011	$00100 + \langle 00001, 00010, 01100, 10000\rangle$	4	2
11100	$00001 + \langle 00010, 01000, 10001, 10100\rangle$	4	2
11101	$01000 + \langle 00010, 01101\rangle$	2	2
11110	$00001 + \langle 00010, 10001, 10100, 11000\rangle$	4	2
11111	$00100 + \langle 00001, 00010, 01100, 10000\rangle$	4	2