IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 2, pp. 253–288. DOI:10.46586/tosc.v2022.i2.253-288

Differential Trail Search in Cryptographic Primitives with Big-Circle Chi:

Application to Subterranean

Alireza Mehrdad, Silvia Mella, Lorenzo Grassi and Joan Daemen

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

Abstract. Proving upper bounds for the expected differential probability (DP) of differential trails is a standard requirement when proposing a new symmetric primitive. In the case of cryptographic primitives with a bit-oriented round function, such as Keccak, Xoodoo and Subterranean, computer assistance is required in order to prove strong upper bounds on the probability of differential trails. The techniques described in the literature make use of the fact that the non-linear step of the round function is an S-box layer. In the case of Keccak and Xoodoo, the S-boxes are instances of the chi mapping operating on ℓ-bit circles with ℓ equal to 5 and 3 respectively. In that case the differential propagation properties of the non-linear layer can be evaluated efficiently by the use of pre-computed difference distribution tables.

Subterranean 2.0 is a recently proposed cipher suite that has exceptionally good energy-efficiency when implemented in hardware (ASIC and FPGA). The non-linear step of its round function is also based on the chi mapping, but operating on an $\ell = 257$-bit circle, comprising all the state bits. This making the brute-force approach proposed and used for Keccak and Xoodoo infeasible to apply. Difference propagation through the chi mapping from input to output can be treated using linear algebra thanks to the fact that chi has algebraic degree 2. However, difference propagation from output to input is problematic for big-circle chi. In this paper, we tackle this problem, and present new techniques for the analysis of difference propagation for big-circle chi.

We implemented these techniques in a dedicated program to perform differential trail search in Subterranean. Thanks to this, we confirm the maximum DP of 3-round trails found by the designers, we determine the maximum DP of 4-round trails and we improve the upper bounds for the DP of trails over 5, 6, 7 and 8 rounds.

Keywords: Differential Trail Search · Chi Mapping · Trail Weight Bounds · Subterranean

1 Introduction

One of the main security requirements for any symmetric cryptographic primitive is resistance against differential cryptanalysis (DC) [BS90, BS93]. One way to analyze it is through the study of the differential probability (DP) of differential trails and the proof of upper bounds for it. A differential trail (also called differential characteristic or path) consists of a sequence of state differences through the rounds of the primitive. Its DP is defined as the fraction of all possible input pairs that satisfy the initial difference in the trail and also all intermediate and final differences when going through the rounds. Trails with sufficiently high DP and/or cluster of differential trails forming a high probability differential can be exploited in differential attacks and hence should be absent. A way to

Licensed under Creative Commons License CC-BY 4.0.
Received: 2021-11-23 Revised: 2022-03-01 Accepted: 2022-05-01 Published: 2022-06-10

characterize the DP of a differential trail is by its *weight*. The weight of a round differential is defined as the negative of the binary logarithm of its DP, or equivalently, $\mathrm{DP} = 2^{-w}$. Assuming that the round differentials are independent, the weight of a trail is given by the sum of the weight of its round differentials (i.e. the pairs of differences at the input and output of each round). Clearly, exploiting a trail becomes harder as the weight increases. Therefore, trails with low weight (i.e. with high DP) are a potential weakness and finding lower bounds on the weight of trails (i.e. upper bounds on the DP) provides insights on the security margin of the primitive.

For some primitives, it is possible to derive trail bounds analytically. For instance, for AES there is a simple proof that the minimum weight for 4-round differential trails is 150 [DR02]. For other primitives, bounds are obtained by computer-aided proofs. In this case, a program is used to generate trails over a given number of rounds to find for instance the minimum number of active S-boxes [BPP+17], or the minimum number of AND gates [Hei11, WH19], or the minimum weight [DA12, MDA17, DHAK18]. Such computer-aided proofs often make use of mixed integer linear programming (as in [SHW+14, BPP+17, BJK+16, WH19]), or SAT solvers (as in [MP13]), but can also involve programs dedicated to a specific primitive. As early as 2000, Daemen et al. [DPVAR00] used a dedicated program to prove bounds in Noekeon. Later, dedicated programs have been developed to prove bounds on Keccak-f [DA12, MDA17], Xoodoo [DHAK18], and Subterranean 2.0 [DMMR20].

The approach used in [DA12, MDA17, DHAK18] to lower bound the weight of trails over r rounds, consists in generating all r-round *trail cores* with weight up to a target T. A trail core represents a class of trails with common intermediate differences and the weight of a trail core is by definition the minimum weight over all trails in the class. A key aspect of such dedicated programs is the ability to compute such minimum weight without the need of generating all trails within the class, significantly reducing the computational cost. If trail cores with weight T or below are found, the one with the minimum weight defines a tight lower bound on the weight of all r-round trails. Otherwise, $T + 1$ defines a lower bound that is not necessarily tight. To generate trail cores over multiple rounds, trail cores over two rounds are first generated and then extended. Since a 2-round trail core is fully determined by a single state difference at the input of the linear layer, the generation of 2-round trail cores in these dedicated programs is performed as a tree search, where the nodes of the tree are such state differences. To efficiently prune the tree during the search, methods to compute a lower bound on the weight of a node and all its descendants are introduced. This allows to discard entire branches of the tree as soon as such lower bound exceeds the limit, making the tree search computationally feasible. Extension is performed in the forward direction by iteratively appending a round differential at the end of the trail core and in the backward direction by iteratively prepending a round differential at the beginning of the trail core.

Differential propagation through the circulant map χ_ℓ. During the trail search, to extend trail cores, compute their weight, and to lower bound the weight of a node and all its descendants in the tree, one has to deal with the differential propagation properties of the non-linear layer. In both Keccak-f and Xoodoo permutations, the non-linear layer can be seen as a layer of ℓ-bit S-boxes defined by the mapping χ_ℓ. χ_ℓ is a transformation operating on ℓ-bit strings and defined by

$$[\chi_\ell(x)]_i = x_i + (x_{i+1 \bmod \ell} + 1) \cdot x_{i+2 \bmod \ell}, \quad \forall i \in \{0, \ldots, \ell - 1\}\,.$$

The mapping χ_ℓ is *circulant* and we refer to the ℓ-bit string as a *circle*. Differential propagation through S-boxes, and thus through these χ instances, can be traced efficiently by the use of $2^\ell \times 2^\ell$ difference distribution tables (DDT) that can be pre-computed and easily manipulated as long as $2^{2\ell}$ is sufficiently small. In most ciphers that make use of

χ_ℓ, the circle size is relatively small (e.g. $\ell = 5$ in Keccak-f and $\ell = 3$ in Xoodoo) and this approach works fine. We speak of *small-circle* χ in this case. This approach breaks down if the circle size ℓ is large, as it becomes a challenge to compute and store the corresponding DDT. We speak of big-circle χ in this case. For example, primitives such as Subterranean [CDGP93] and before it Cellhash [DGV91] have all 257 state bits in a single big-circle. In this case, we speak of a *single-circle* χ mapping and $\chi = \chi_\ell$.

In this work, we analyze the differential propagation properties of big-circle χ needed to perform trail search. For the propagation in the forward direction (from the input of χ to the output) we can rely on previous results based on the algebraic degree of χ. For the backward direction (from the output of χ to the input) we introduce new results.

Ciphers using big-circle χ. An immediate target for our results is the Subterranean permutation, for which we introduce a dedicated program to scan the space of all trail cores up to a target weight over a given number of rounds. This tool allows us to improve over known bounds for different number of rounds. In particular, we can prove an upper bound on the DP of 8-round trails of 2^{-116}, which is 2^{18} times smaller than the bound reported in [DMMR20]. The interest in analyzing Subterranean derives from its exceptional performance in hardware, which makes it a good starting point for promising future developments. Among the candidates to the second round of NIST LWC competition, it is in fact the most efficient in terms of throughput/area and stands out for its extremely low energy consumption on different platforms [MHN+20, KPC20, AZ21]. However, its design dates back to 1992 and one can expect to see improved variants in the near future. This is motivated by the fact that using big-circle χ, and thus single-circle χ, presents different advantages over small-circle χ, and one can expect to see new cryptographic primitives with such non-linear layers.

An example of such advantages is the impact on the complexity of algebraic attacks that involve equations of input bits of χ_ℓ in terms of its output bits. The algebraic degree of χ_ℓ is 2, while that of its inverse and the number of monomials in its inverse increase with the circle size ℓ. An algorithm to compute the inverse of χ_ℓ and consequently its degree is introduced in [Dae95, Sect. 6.6.2]. The recent algebraic attacks on Farfalle instantiated with Keccak-f [CFG+18] and with the Xoodoo permutation [CG20] make use of the Meet-in-the-Middle (MitM) strategy to set up a system of algebraic equations solved using the linearization technique. In both attacks, this MitM strategy is efficient due to the fact that the non-linear layers of Keccak-f and of Xoodoo have low degrees both in the forward and in the backward directions. In the case of a big-circle χ, such an MitM strategy would be more cumbersome due to the higher degree of χ^{-1} and number of monomials in the equations.

Another advantage can be seen against higher order differential cryptanalysis [Knu94], such as cube attacks [DS09], where the data complexity increases strongly with the number of rounds attacked. In ciphers where there is a whitening key added to the state before the output, one often *peels off* a round by guessing part of the whitening key. When χ is used as non-linear layer, by guessing the key bits corresponding to a circle, one can compute the input of the circle, effectively peeling off a non-linear layer and potentially reducing the algebraic degree of the whole cryptographic function by a factor 2. Typically, the offline phase of the attack must then be repeated for all such key guesses, i.e., 2^ℓ times. In case of big-circle χ, this may be a prohibitively large factor. By guessing only part of the whitening key bits, say m bits, one can often determine several bits of the χ input, but this depends on the particular output at hand. In the case of cube attacks, one needs to compute the input for all outputs corresponding to the inputs in a cube and the probability of being able to compute some input bits for all these inputs decreases exponentially with $\ell - m$.

Moreover, according to [DMM21], choosing a non-linear layer consisting of big-circle

χ, leads to the same behavior (if not better) compared to short-circle χ in terms of the distribution of differentials with given DP. The number of differentials with given DP gives information about how a non-linear layer may perform in the wide trail strategy. Namely, less differentials with high DP means less opportunity to form trails with high DP over multiple rounds. As an example, using big-circle χ, in place of short-circle χ, results in a smaller number of differentials up to a given weight in Xoodoo and almost the same number in Keccak-f.

Finally, we point out that the computational cost of a χ mapping is *independent of the circle size*, since it consists of one binary addition (XOR), one binary multiplication (AND) and one negation per bit.

For all these reasons, one of the main goal of this work is to provide useful techniques for studying the security of primitives instantiated with big-circle and single-circle χ mapping with respect to classical differential attack. To this end, we present our results for big-circle χ in a generic way, separating them from the techniques introduced to perform trail search specifically for Subterranean, which are presented at the end of the paper in a dedicated section.

Our contribution. In this paper, we analyze the differential propagation properties of big-circle χ that are used for the generation of 2-round trail cores and their extension. The methods we present apply to *all* circle lengths, including even lengths (for which collisions exist, but there are applications where these collisions may be tolerated). In this work, we also present a tool to perform the trail search for Subterranean and use it to improve over known bounds. More in details, we present the following results:

- **A function that lower bounds the weight of a 2-round trail core and all its descendants during the tree search.** According to [MDA17], to efficiently generate 2-round trail cores as nodes of a tree a lower bound function on the weight of the 2-round trail core and all its descendants is needed. We show how to define it when we deal with big-circle χ. This function is composed by a quantity that only depends on the action of the non-linear layer and a quantity that depends on the action of the both linear and non-linear layers of the primitive under analysis. For the former, we present a way to compute it for big-circle χ in Section 4, while for the latter, we present a way to compute it in the case of Subterranean in Section 5.
- **A method to efficiently compute the minimum weight of a trail core.** For a given output difference a, we show how to build an input difference b that minimizes the weight of the differential (b, a). We then use such input difference to compute the minimum weight of the differential. This result is used to compute the exact weight of 2-round trail cores during the tree search and the weight of trail cores obtained during backward extension.
- **A method to efficiently perform backward extension up to a given weight.** For a given difference at the input of χ_ℓ, the set of output differences with weight greater than 0 ($\mathrm{DP}(b, a) > 0$) form an affine space and a recipe to build this affine space is given in [Dae95]. This can be used to perform forward extension efficiently. On the contrary, for a given difference at the output of χ_ℓ, the set of input differences with weight greater than 0 does not have a simple structure. To compute such set, the approach used for small-circle χ is based on *exhaustively* studying all the pairs of outputs with difference a. In this work, we show how to build the input differences for big-circle χ. Moreover, we show how to limit the generation of such input differences to those for which the differential has weight smaller than a certain target.
- **A software tool for differential trail search tailored for Subterranean.** The tool implements the framework of [MDA17] for the tree search adapted to

Table 1: Lower bounds on the weight of differential trails in Subterranean.

# of rounds	1	2	3	4	5	6	7	8
lower bound [DMMR20]	2	8	25	[49, 58]	≥ 54	≥ 65	≥ 70	≥ 98
lower bound (this work)	2	8	25	58	≥ 62	≥ 78	≥ 80	≥ 116

Subterranean and the aforementioned results on big-circle χ. These methods together allow to scan the space of all r-round trail cores up to a given weight that is higher than what was reachable before. An immediate consequence is the ability to improve over known bounds, as summarized in Table 1. As our main results, we prove for the first time that the minimum weight of any 4-round trail in Subterranean is 58 and that any 8-round trail has weight at least 116. Namely, DP $\leq 2^{-116}$, a result that is 2^{18} times smaller than what was reported in [DMMR20]. This allows us to increase the security margin of 8-round Subterranean with respect to differential cryptanalysis.

Organization of the paper. In Section 2, we first recall the concepts of differential trails, trail cores, trail extension and the general strategy to generate all trail cores up to a given target weight. Then we recall the tree-search framework for the generation of 2-round trail cores. In Section 3, after having defined our terminologies, we recall some information about the χ mapping and the way to compute the weight of a differential over χ given its input difference. In Section 4, we introduce our main new results. First, we provide a method to lower bound the weight of a 2-round trail core and all its descendants during the tree search. Then, we show how to compute the minimum weight of a 2-round trail core. Finally, we present an algorithm that generates all possible differences at the input of χ_ℓ for a given difference at the output of χ_ℓ to perform backward extension. Finally, in Section 5 we present techniques and results specific for Subterranean. First, we recall the Subterranean round function, then we present how we instantiate the tree search for Subterranean, and provide the results of our analysis applied to it.

2 Differential trail search

We will recall the concepts of DP, weight, differential trails and trail cores over iterative cryptographic primitives, where we assume that the round function consists of a linear mapping followed by a non-linear mapping of degree 2, as χ. We also recall the general strategy to bound the weight of trails over a given number of rounds, the trail extension and the concept of trail core, and finally how to generate all 2-round trail cores up to a given weight.

Notation. In the following, we interpret input differences and output differences as vectors in $\mathbb{F}_2^\ell$, that is, the vector space of dimension ℓ over $\mathbb{F}_2$. We denote a basis vector with a single 1 at index i as $\mathsf{e}_i \in \mathbb{F}_2^\ell$. We use the symbol $+$ for the addition over different groups and let its meaning depend on the types of the operands.

2.1 Differentials and differential trails

A differential over a transformation α of $\mathbb{F}_2^n$ is an ordered pair (a, b) with $a, b \in \mathbb{F}_2^n$, where a is a difference at the input of α and b is a difference at its output. The *differential*

probability (DP) of a differential (a, b) over α, denoted as $\mathrm{DP}_\alpha(a, b)$, is:

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

If the transformation is clear from the context, we write $\mathrm{DP}(a, b)$. If $\mathrm{DP}_\alpha(a, b) > 0$ we call (a, b) a *possible* differential and we say b is an output difference *compatible* with a and a is an input difference compatible with b.

The weight (also called *restriction weight* [DR02]) of a differential $\mathrm{w_r}(a, b)$ is the logarithm of its DP with base 1/2: $\mathrm{w_r}(a, b) = -\log_2 \mathrm{DP}(a, b)$. In what follows we work with weights instead of DP, simplifying notations and expressions.

2.1.1 Differential trails and their weight

We consider iterative mappings consisting of the iteration of a number or rounds R_i: $f = \mathrm{R}_r \ldots \circ \mathrm{R}_2 \circ \mathrm{R}_1$. Estimating $\mathrm{DP}_f(a^0, a^r)$ for some given differential or finding the differentials (a^0, a^r) over f with high DP requires the study of so-called *differential trails*.

A differential trail Q over an iterative mapping f is a chain of round differentials:

$$Q = a^0 \xrightarrow{\mathrm{R}_1} a^1 \xrightarrow{\mathrm{R}_2} \quad \ldots \quad \xrightarrow{\mathrm{R}_{r-1}} a^{r-1} \xrightarrow{\mathrm{R}_r} a^r \tag{1}$$

An r-round trail is fully specified by the sequence of $r + 1$ differences: $Q = (a^0, a^1, \ldots, a^r)$. The DP of a trail is the probability that a pair of inputs to f with difference a^0 propagates to difference $a^1, a^2, \ldots, a^r$ respectively after $1, 2, \ldots, r$ rounds.

The product of the DP values of the round differentials gives a good indication of the DP of the differential trail and is called its *expected* DP (EDP):

$$\mathrm{EDP}_f(Q) = \prod_{i=1}^{r} \mathrm{DP}_{\mathrm{R}_i}(a^{i-1}, a^i)\,. \tag{2}$$

In the case of trails with $\mathrm{EDP} \gg 2^{-b}$ in b-bit permutations or block ciphers that do not exhibit superboxes (like in, e.g., AES), it is very plausible that $\mathrm{DP}(Q) \approx \mathrm{EDP}(Q)$ – see, e.g., [BDKA21] for more details.

The weight of a differential trail Q is the logarithm of its EDP. It follows from Eq. (2) that $\mathrm{w_r}(Q) = \sum_{i=1}^{r} \mathrm{w_r}(a^{i-1}, a^i)$.

2.2 Finding all trails with weight below some limit

We wish to find lower bounds for the weight of trails over a number of rounds r. To this end, one can generate *all* r-round trails up to a given target weight T_r. If such space is not empty, the trail with the smallest weight defines a tight lower bound on the weight of any r-round trail. Otherwise, $T_r + 1$ defines a non-tight lower bound. In this case, the bound can be improved by scanning the space of all r-round trails up to a bigger target weight.

To generate all r-round trails up to a given weight, we first generate all trails with a lower number of rounds (and weight below some limit weight) and extend them. We do this in a recursive way. Namely, we generate long trails by extending shorter trails.

The conceptually simplest way to generate all r-round trails up to some weight T goes as follows. Since there exists at least one round differential (1-round trail) with weight $\lfloor T/r \rfloor$ in all r-round trails up to weight T, we start by generating all 1-round trails up to $\lfloor T/r \rfloor$. We first extend these 1-round trails forward by $r - 1$ rounds up to weight T. This gives all r-round trails where the profile has 1-round trail up to $\lfloor T/r \rfloor$ in the first position. Then we extend these 1-round trails backward by one round and forward by $r - 2$ rounds up to weight T. This gives all r-round trails where the profile has 1-round trail up to $\lfloor T/r \rfloor$ in the second position. By repeating this for all positions, we can generate all r-round trails with weight up to T.

However, if the round function has a strong mixing layer, it is more efficient to start from all two-round trails up to weight $\lfloor 2T/r \rfloor$ than to start from single-round trails up to $\lfloor T/r \rfloor$. This is the case in, e.g., AES [DR02] and KECCAK [DA12,MDA17]. In such ciphers, there are orders of magnitude less two-round trails with weight $2w$ than single-round trails with weight w for relatively small weight. This results in the fact that the number of trails that have to be extended to r rounds when targeting a certain T is much less if we start from two-round trails. Moreover, using the knowledge of the linear and non-linear layers, generating these two-round trails can be done very efficiently. This was first done in the trail analysis of the lightweight block cipher NOEKEON [DPAR00] and later refined and applied to Keccak-f in [DA12] and in [MDA17]. In [DA12] a technique was presented to generate three-round trails directly.

2.3 Trail extension and trail cores

In our analysis we limit ourselves to round transformations that consist of a linear layer that we denote by λ, a non-linear layer that we denote by χ, and the addition of a round constant or key. This is the shape used in many modern ciphers, like AES, KECCAK, PRESENT, SKINNY, etc.

Since the constant or key addition has no influence in the propagation of differences, it can be omitted in the analysis of differential trails. We extend the trails description by including the differences at the input of each non-linear layer, that we denote by b^i:

$$Q : a^0 \underbrace{\xrightarrow{\lambda} b^0 \xrightarrow{\chi}}_{\mathrm{R}_1} a^1 \underbrace{\xrightarrow{\lambda} b^1 \xrightarrow{\chi}}_{\mathrm{R}_2} a^2 \xrightarrow{\lambda} \dots \xrightarrow{\chi} a^{r-1} \underbrace{\xrightarrow{\lambda} b^{r-1} \xrightarrow{\chi}}_{\mathrm{R}_r} a^r \,. \tag{3}$$

As λ is linear, we have $\mathrm{DP}_\lambda(a^i, b^i) = 1$ if $b^i = \lambda(a^i)$ and 0 otherwise. Clearly, for the round differentials to be possible, we have $b^i = \lambda(a^i)$. The weight of this trail is therefore given by: $\mathrm{w_r}(Q) = \sum_{i=1}^{r} \mathrm{w_r}(b^{i-1}, a^i)$.

In many round functions the non-linear layer has algebraic degree 2 and this is also the case for χ. The DP of a possible differential over such a non-linear layer is fully determined by the input difference, and hence so is the weight. We can thus write $\mathrm{w_r}(b)$ for the weight of a differential (b, a) over a non-linear map of degree 2 and the weight of the trail becomes $\mathrm{w_r}(Q) = \sum_{i=1}^{r} \mathrm{w_r}(b^{i-1})$. It follows that the weight of a valid r-round trail is independent of the last difference a^r.

2.3.1 Trail extension

Given an r-round trail Q, it is possible to extend it to $r+1$ rounds and the extension can be done in the forward or backward direction.

Forward extension consists in appending pairs (b^r, a^{r+1}) to Q, such that $b^r = \lambda(a^r)$ and a^{r+1} is compatible with b^r. When the non-linear layer consists of the parallel application of small S-boxes, as in the case of small-circle χ, the output differences a^{r+1} compatible with a given input difference b^r can be listed efficiently using the DDT. This brute-force approach through the DDT is not applicable with big-circle and single-circle χ. However, for χ_ℓ, such compatible differences form an affine space of size $2^{\mathrm{w_r}(b^r)}$ that can be efficiently described by an offset and basis vectors [Dae95]. An r-round trail Q will give rise to $2^{\mathrm{w_r}(b^r)}$ $(r+1)$-round trails with Q as prefix. The weight of such trails is $\mathrm{w_r}(Q) + \mathrm{w_r}(b^r)$.

Similarly, backward extension consists in prepending pairs (a^{-1}, b^{-1}) to Q, such that b^{-1} is compatible with a^0 and $a^{-1} = \lambda^{-1}(b^{-1})$. This operation will give a set of trails with Q as trailing part and the weight of such trails is given by $\mathrm{w_r}(b^{-1}) + \mathrm{w_r}(Q)$. Again, in the case of small S-boxes and hence of small-circle χ, compatible differences can be computed efficiently through the DDT. In the case of big-circle or single-circle χ this is too expensive and moreover, the set of such compatible differences is not an affine space that can be efficiently scanned. We will tackle this problem in Section 4.4.

Forward and backward extension can be iterated or combined to extend over multiple rounds.

If we are interested in $(r+1)$-round trails with weight up to T, we can discard all extended trails with weight above that limit. We call this *extension up to weight* T.

2.3.2 Differential trail cores

Since the weight of an r-round trail Q is independent of the last difference a^r, the sequence of differences $(a^0, b^0, \ldots, a^{r-1}, b^{r-1})$ – which is Q with the last difference removed – defines a set of r-round trails with the same weight $\mathrm{w_r}(Q)$.

When extending a trail in the backward direction, all differences b^{-1} compatible with a^0 are built. We call the minimum weight over all these compatible differences b^{-1} the *minimum reverse weight* and we denote it by $\mathrm{w_{rev}}(a^0)$. It follows that the weight of any $r+1$-round trail obtained by backward extension of Q can be lower bounded by $\mathrm{w_{rev}}(a^0) + \mathrm{w_r}(Q)$.

Therefore, a sequence $(a^0, b^0, \ldots, a^{r-1}, b^{r-1})$ defines a set of $r+1$-round trails with the sequence as central part and with weight at least $\mathrm{w_{rev}}(a^0) + \mathrm{w_r}(Q)$. We call this set of trails $r+1$-*round trail core* as in [DA12] and denote it by $\tilde{Q}_{r+1}$. We call the quantity $\mathrm{w_{rev}}(a^0) + \mathrm{w_r}(Q)$ the *weight of the trail core*.

Since the aim of our search is to bound the minimum weight of trails, we can restrict our search to trail cores and find bounds for them. Therefore, instead of scanning all r-round trails up to a target weight T_r, we generate r-round trail cores up to T_r. This allows to avoid passing through two non-linear layers, which significantly reduces the computation effort.

According to Section 2.2, and similar to [DA12, MDA17, DHAK18], we can thus start by generating 2-round trail cores (a, b) below a limit weight T_2 and extend them. Since a 2-round trail core is fully determined by the difference a, as $b = \lambda(a)$, the problem of generating all 2-round trail cores up to a certain weight T_2 comes down to the generation of differences a from which we can compute its minimum reverse weight and the weight of $\lambda(a)$, namely $\mathrm{w_{rev}}(a) + \mathrm{w_r}(\lambda(a))$. The feasibility of computing the weight of a 2-round trail core depends on the non-linear layer under analysis. For ciphers like Keccak, they can be easily computed by building the differential distribution table. For other ciphers like Subterranean, their computation can be more involved.

2.4 Generating 2-round trail cores as a tree search

In the approach introduced in [MDA17], differences a are encoded as ordered lists of *units*. Then given a unit, its unique 2-round trail core can be built by computing $b = \lambda(a)$. A unit is a pattern of so called *active* bits, where a difference bit is called active when its value is one, otherwise it is called *passive*.

The unit list representation naturally arranges the difference patterns in a tree, where the root of the tree is the all-zero state corresponding to the empty unit list. The children of a node are obtained by adding one more unit to the list of the node. Equivalently, the parent of a node is obtained by removing the last unit of the node. Siblings have all but the last unit in the list in common.

Generating 2-round trail cores up to some weight consists in traversing portions of this tree. In particular, units shall be defined so that low-weight 2-round trail cores will be situated near the root of the tree and one only has to traverse that part of the tree.

2.4.1 Lower bounding the weight of a node and its descendants

We use the term *weight of a node* to call the weight of the 2-round trail core specified by that node. To traverse the tree more efficiently, it is useful to define a lower bound on the

Algorithm 1 `next()` function.

```
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;
```

weight of a node and of all its descendants.

In this work, we will call such a lower bound the *score* of the node. We denote the score of a node a as $\mathsf{score}(a)$. Therefore $\mathsf{score}(a)$ is such that $\mathsf{score}(a) \leq \mathrm{w}_{\mathrm{rev}}(a) + \mathrm{w}_{\mathrm{r}}(\lambda(a))$ and $\mathsf{score}(a) \leq \mathrm{w}_{\mathrm{rev}}(a') + \mathrm{w}_{\mathrm{r}}(\lambda(a'))$ for all descendants a' of a. It follows that as soon as we encounter a node whose score is above the limit weight, we can safely discard the node and all its descendants.

If units are defined so that their addition to a node is monotonic in the weight, then the score of a node can simply be defined as the weight of the node itself. However, this requirement can lead to complicated definitions of units. It may be more convenient to choose a simple definition of units for which the weight is not monotonic, but that allows to define a score function that is efficiently computable and reasonably tight.

To this end it is convenient to split the score in two parts, one that lower bounds $\mathrm{w}_{\mathrm{rev}}(a)$ and $\mathrm{w}_{\mathrm{rev}}(a')$ and one that lower bounds $\mathrm{w}_{\mathrm{r}}(\lambda(a))$ and $\mathrm{w}_{\mathrm{r}}(\lambda(a'))$. We call the former a *score-at-a* function and the latter a *score-at-b* function. The definition of a *score-at-a* function can be made independent of the linear layer as long as units are defined so that they cannot overlap in a. The definition of a *score-at-b* function instead always depends on the linear layer and typically keeps track of active bits at b that cannot become passive with the addition of new units and considers only their contribution to the weight.

2.4.2 Traversing the tree

The tree traversal is performed in a depth-first fashion, by iteratively calling the function `next()` (Algorithm 1) to look for the next valid node. The traversal starts by calling `next()` on an empty list and ends when `next()` returns false, that is when there are no more nodes worth visiting.

There are three possible moves in the function `next()`. The first is performed by the function `toFirstChild()`, which will look for the first child of the current node by trying to add the smallest possible unit after the last unit. If the node is the root, then it adds the smallest possible unit. If adding a new unit is not possible, it returns false. Otherwise, it returns true and *additional conditions* are checked. Here, the additional conditions are typically whether the score is below the limit weight. If such additional conditions are satisfied, the algorithm has found the next node in the tree. If there is no valid child, e.g. because there are no more units to add or because the score of all children is above the budget, the routine will look for a valid sibling of the current node. This is done by the function `toSibling()` which iterates the value of the last unit in the list. If a sibling is found then the additional conditions are checked. If there are no valid siblings, the algorithm moves back to the parent of the current node, by removing the last unit from the list with the function `toParent()`, and looks for a valid sibling of the parent. This is done recursively until a valid node or the root is reached. In the latter case the function `next()` returns false and the traversal is over.

3 Properties of non-linear mappings χ

In this section, we introduce the χ_ℓ family of transformations and the χ mapping via the notions of translation-invariant mappings. Next, we recall some results presented in [Dae95] about the differential properties of χ_ℓ such as how to compute the weight of a differential over χ_ℓ given its input. Finally, we provide a new method to locally check the compatibility of a given pair of differences at the input and output of χ_ℓ.

Notation for strings. There is a bijection between the vector space $\mathbb{F}_2^\ell$ and the set of strings $\{0,1\}^\ell$, that is, the set of binary strings of length ℓ.

We denote binary strings by lowercase letters, say s, and the i-th bit of s as s_i, where we start indexing from 0. Hence, we have $s = s_0 s_1 s_2 \dots s_{|s|-1}$, where $|s|$ denotes the length of the string s. We denote the number of 1-bits in a string s by $||s||$ and call it *Hamming weight* of s.

We write $s\|s'$ for the concatenation of two strings s and s'. Given a string s, we denote by $(s)^n$ the n-fold concatenation of s. We say that s' is a substring of s if $|s'| \leq |s|$ and there exists an offset i such that $s'_j = s_{j+i}$ for all $j \in [0, |s'| - 1]$.

We use the notation $(0)^\ell$ for the all-zero difference and $(1)^\ell$ for the all-one difference.

Definition 1. A 0-*run* is a string of only zeroes.

So $(0)^n$ is a 0-run of length n.

About circular strings. In some instances we need strings that have an additional feature: they are *circular*. In a *circular string* s, we allow indexing outside the range $[0, |s| - 1]$ by reducing the index modulo $|s|$. As a consequence, in a circular string the leftmost bit s_0 and rightmost bit $s_{|s|-1}$ are neighbors and this has implications for the substring relation.

A substring of a circular string s is a substring of $s\|s$ of length at most $|s|$.

Definition 2 (Substring of a circular string)**.** We say s' is a substring of a circular string s if $|s'| \leq |s|$ and there exists an offset i such that $s'_j = s_{j+i \bmod |s|}$ for all $j \in [0, |s'| - 1]$.

For example $s' = 10000$ is not a substring of string $s = 00110011100$ but it is a substring of the circular string s since $s'_j = s_{j-3 \bmod 11}$ for $0 \leq j < 5$. When speaking about substrings of a string s, we will assume it is known from the context whether s is circular or not.

3.1 Translation-invariant mappings

The transformation χ_ℓ has a symmetry property called *translation-invariance*.

Definition 3 (Translation τ_ℓ)**.** Let $\tau_\ell \colon \mathbb{F}_2^\ell \to \mathbb{F}_2^\ell$ be defined as $\tau_\ell(x) = y$ with

$$\forall i \in \{0, 1, \dots, \ell - 1\} : \qquad y_i = x_{i+1 \bmod \ell}\,.$$

We will often omit the subscript ℓ if it is known from the context. In the following we will assume indexes to be elements of the additive group $\mathbb{Z}/\ell\mathbb{Z}$ and omit the modular reduction.

Clearly, τ is a translation over 1 position *to the left*. It is a linear permutation of $\mathbb{F}_2^\ell$, and with the functional composition it generates a cyclic group $\langle\tau\rangle$ of order ℓ. We denote the elements of this cyclic group as τ^i.

Definition 4 (Translation Invariance)**.** A transformation α_n of $\mathbb{F}_2^\ell$ is translation-invariant if it commutes with τ, that is, $\alpha_n \circ \tau = \tau \circ \alpha_n$.

Clearly, as a cyclic group is commutative, a translation-invariant transformation commutes with all elements of $\langle\tau\rangle$. Moreover, τ is a translation-invariant transformation itself.

A translation-invariant transformation is fully determined by the specification of its first component. The restriction of a transformation α to its first component is its *local map*.

Definition 5 (local map)**.** The *local map* φ_α of a translation-invariant transformation α over $\mathbb{F}_2^\ell$ is given by

$$\varphi_\alpha \colon \mathbb{F}_2^\ell \to \mathbb{F}_2 : \varphi_\alpha(x) = y_0 \text{ with } y = \alpha(x)\,.$$

Thanks to the translation-invariance, we can derive the coordinate functions of the other components from the local map.

Lemma 1. *The coordinate function of component i of a translation-invariant mapping α over $\mathbb{F}_2^\ell$ with local map $\varphi_\alpha(x)$ is given by $\varphi_\alpha(\tau^i(x))$.*

Proof. Let $y = \alpha(x)$. Its i-th component is equal to

$$y_i = [\alpha(x)]_i = [\tau^i \circ \alpha(x)]_0 = [\alpha \circ \tau^i(x)]_0\,,$$

as α commutes with τ^i. Hence, its i-th component is equal to $\varphi_\alpha(\tau^i(x))$. □

In our descriptions it is often useful to indicate the value of a state restricted to a subset of state bits. For that we introduce the expression *to be in a landscape*.

Definition 6 (Landscape)**.** A landscape Λ is a set of states in $\{0,1\}^\ell$ specified by an ordered pair of circular strings $l^{(v)}, l^{(m)} \in \{0,1\}^\ell$. A state a is in landscape Λ if a matches $l^{(v)}$ in the positions where $l^{(m)}$ is 1. Formally:

$$a \in \Lambda \quad \iff \quad \forall j : (a_j + l_j^{(v)}) \cdot l_j^{(m)} = 0\,.$$

We use a shortcut notation for landscape literals by merging $l^{(v)}$ and $l^{(m)}$ in a single string $l^{(t)}$ containing of 3 symbols: 0, 1 and $\star$ with $l_i^{(t)} = l_i^{(v)}$ if $l_i^{(m)} = 1$ and $l_i^{(t)} = \star$ otherwise. We then *compress* $l^{(t)}$ by centering around the position with index 0 that we indicate by underlining it and omitting all unnecessary symbols $\star$.

Example 1. For a circular string of length 11, let the landscape $(l^{(v)}, l^{(m)}) = (01000000000, 11010000001)$, then $l^{(t)} = 01\star0\star\star\star\star\star0$. After compression it gives $0\underline{0}1\star0$.

We say bit i of a state a is in landscape $\Lambda = (l^{(m)}, l^{(v)})$ if the state a translated over i positions is in the landscape. Formally (with some abuse of notation):

$$a_i \in \Lambda \quad \iff \quad \forall j : (a_{i+j} + l_j^{(v)}) \cdot l_j^{(m)} = 0\,.$$

3.2 The family of mappings χ_ℓ and χ

The mapping χ_ℓ is a family with domain $\mathbb{F}_2^\ell$ of any dimension $\ell \geq 3$. We denote members of this family by $\chi_3, \chi_4, \chi_5, \ldots$. The mappings χ_ℓ for all $\ell > 2$ form a family of mappings all specified by the same local map.

Definition 7 (χ_ℓ)**.** χ_ℓ is a translation-invariant transformation of $\mathbb{F}_2^\ell$ with local map

$$\varphi_\chi(x) = x_0 + (x_1 + 1)x_2\,.$$

We say χ_ℓ has circle size ℓ.

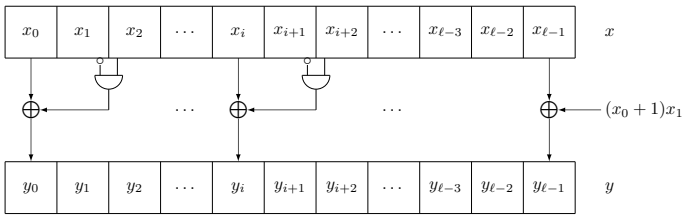

Figure 1: χ_ℓ with input x and output y.

Using landscapes, we can specify χ_ℓ as: $y_i = x_i + 1$ if $x_i \in \measuredangle 01$ and $y_i = x_i$ otherwise. Fig. 1 represents χ_ℓ with input x and output y that are clearly circular strings.

We use the symbol χ to denote non-linear layers that consists of the parallel application on a number of instances of χ_ℓ to an array of n bits, that we typically call the *state*. We call the parts of a state that the instances of χ_ℓ operate on *circles*. Typically, the array consists of m circles of equal length ℓ. For example χ in Keccak-f[1600] has 320 circles of length 5 [BDPA11], χ in Ascon has 64 circles of length 5 [DEMS14] and χ in Xoodoo has 128 circles of length 3 [DHAK18]. The size of the state is $n = m \times \ell$.

If $m = 1$, we speak of single-circle χ when $\chi = \chi_\ell$ and $n = \ell$. This is the case for Subterranean where we have $\ell = 257$.

3.3 Differentials over χ_ℓ given its input difference

Given a difference b at the input of χ_ℓ, we denote by $\mathcal{A}(b)$ the set of compatible output differences

$$\mathcal{A}(b) \coloneqq \{a \in \mathbb{F}_2^\ell \mid \exists x \in \mathbb{F}_2^\ell \text{ s.t. } a = \chi_\ell(x - b) - \chi_\ell(x)\}, \tag{4}$$

where $x - b$ means the addition of the additive inverse of b to x. As observed in [Dae95, Sect. 6.9], since the map χ_ℓ has algebraic degree two, the set $\mathcal{A}(b)$ is an affine space. The cardinality of this affine space is $2^{\mathrm{w_r}(b)}$. It is possible to find such set in an efficient way by exploiting the fact that the map χ_ℓ has degree two. Indeed, the components of $a = \chi_\ell(x - b) - \chi_\ell(x)$ are given by

$$a_i = [\chi_\ell(x - b)]_i - [\chi_\ell(x)]_i = [\chi_\ell(b)]_i + b_{i+2} \cdot x_{i+1} + b_{i+1} \cdot x_{i+2} \tag{5}$$

when $[\chi_\ell(b)]_i$ represents the i-th bit of the output of χ_ℓ for a certain input difference b and input vector x. For a given input difference, the bits b_i are fixed and the bits x_i are variables.

Proposition 1 ([Dae95, Sect. 6.9.1])**.** *Given $b \in \mathbb{F}_2^\ell$, the set $\mathcal{A}(b)$ defined in Eq. (4) is an affine subspace of the form*

$$\mathcal{A}(b) = \chi_\ell(b) + \langle \mathrm{M}^{(b)} \times \mathsf{e}_0, \mathrm{M}^{(b)} \times \mathsf{e}_1, \ldots, \mathrm{M}^{(b)} \times \mathsf{e}_{\ell-1} \rangle \,,$$

where

$$\mathrm{M}^{(b)} \coloneqq \begin{bmatrix} 0 & b_2 & b_1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & b_3 & b_2 & \ldots & 0 & 0 \\ 0 & 0 & 0 & b_4 & \ldots & 0 & 0 \\ 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ b_{\ell-1} & 0 & 0 & 0 & \ldots & 0 & b_0 \\ b_1 & b_0 & 0 & 0 & \ldots & 0 & 0 \end{bmatrix}. \tag{6}$$

A method to build offset and basis vectors for the affine space was presented in [Dae95, Sect. 6.9]. Later, Bertoni et al. in [BDPA11, Sect. 2.3] introduced a method to build basis vectors with the minimum Hamming weight in KECCAK where χ consists of parallel applications of the χ_5 mapping, however proofs are missing there. We recall the rules to build basis vectors with minimum Hamming weight in Table 6, and provide proofs for these rules in Appendix A that result in generalizing these rules for any circle size. We also introduce a new method to build an offset with minimum Hamming weight using local rules as in Tables 5 and 7, and provide a proof for them in Appendix A.

3.3.1 Computing $\mathrm{w_r}(b)$ for a given b

As mentioned in [Dae95, Sect. 6.9], the weight of any differential (b, a) over χ_ℓ depends only on b and can be computed using the following propositions.

Proposition 2. *Given compatible input/output differences $b, a \in \mathbb{F}_2^\ell$, the weight $\mathrm{w_r}(b, a)$ is equal to the rank of the matrix $\mathrm{M}^{(b)}$ given in Eq. (6).*

Proof. By definition of the weight and due to Eq. (6), we have the following

$$\begin{aligned}\mathrm{w_r}(b, a) &= \ell - \log_2\left(|\{x \in \mathbb{F}_2^\ell \mid \chi_\ell(x-b) - \chi_\ell(x) = a\}|\right)\\ &= \ell - \log_2\left(|\{x \in \mathbb{F}_2^\ell \mid \mathrm{M}^{(b)} \times x = a - \chi_\ell(b)\}|\right)\\ &= \ell - (\ell - \mathrm{rank}(\mathrm{M}^{(b)})) = \mathrm{rank}(\mathrm{M}^{(b)}).\end{aligned}$$

□

In particular, the weight of any differential (b, a) can be computed using Proposition 3 and Proposition 4 (whose proofs are given in Appendix B).

Proposition 3. *For a **non**-fully active difference $b \in \mathbb{F}_2^\ell \setminus \{(1)^\ell\}$ at the input of χ_ℓ, the weight of b equals its Hamming weight plus the number of strings 001 it contains, denoted by $\#_{001}(b)$:*

$$\mathrm{w_r}(b) = ||b|| + \#_{001}(b). \tag{7}$$

Proposition 4. *The weight of a fully active difference $b = (1)^\ell$ at the input of χ_ℓ is $\ell - 1$.*

It follows that:

Corollary 1. *For each $b \in \mathbb{F}_2^\ell \setminus \{(0)^\ell\}$ at the input of χ_ℓ: $2 \le \mathrm{w_r}(b) \le \ell - 1$.*

The proofs of Corollary 1 is given in Appendix B.

3.4 A method for locally checking compatibility

Since χ_ℓ can be defined as a local map, we can check compatibility between an input difference and an output difference *locally* by considering substrings. We introduce this new method in the following lemma.

Lemma 2. *Given four adjacent bits of an input difference $b \in \mathbb{F}_2^\ell \setminus \{(1)^\ell\}$, the output difference $a \in \mathbb{F}_2^\ell$ is **not** compatible with b if one of the following conditions is satisfied:*

- *$b_i \in \underline{0}0$ and $b_{i-1} \neq a_{i-1}$; or*
- *$b_i \in \underline{1}01$ and $b_{i-1} \neq a_{i-1} + a_i$.*

Proof. If b is compatible with a, $a \in \mathcal{A}(b)$ and $a = \chi_\ell(x-b) - \chi_\ell(x)$ for some $x \in \mathbb{F}_2^\ell$. Since χ_ℓ is translation-invariant, we focus on $i = 0$. From Eq. (5), if $b_0 \in \underline{0}0$, then

$$a_{-1} = [\chi_\ell(b)]_{-1} + b_1x_0 + b_0x_1 = b_{-1} + b_0b_1 + b_0 + b_1x_0 + b_0x_1 = b_{-1}\,.$$

Similarly, if $b_0 \in \underline{1}01$, then

$$a_{-1} + a_0 = b_{-1} + b_0b_1 + b_0 + b_1x_0 + b_0x_1 + b_0 + b_1b_2 + b_1 + b_2x_1 + b_1x_2 = b_{-1}\,.$$

□

Invertibility. As a direct consequence of Lemma 2 and of other analogous results presented in Appendix A, we have that the χ_ℓ function is **not** invertible if ℓ is even.

Lemma 3. *If ℓ is an even integer, then there are four input differences b compatible with output difference $a = (0)^\ell$, that is:*

$$(0)^\ell, \quad (10)^{\ell/2}, \quad (01)^{\ell/2}, \quad (1)^\ell. \tag{8}$$

Such a result result can be easily verified by e.g. exploiting the formula for offset and basis vectors to build the affine spaces $\mathcal{A}(b)$ for such b's presented in Appendix A. It follows that for even ℓ collisions exist and hence χ_ℓ is non-invertible. Based on Proposition 3, the minimum weight for having such collision is $\ell/2$.

4 Differential properties of χ^{-1}

In this section we present three tools for dealing with the inverse of χ in trail generation. The first one is useful in two-round trail core generation: an efficiently computable and reasonably tight score-at-a function. The other two are useful in backward extension: a method to compute the minimum reverse weight of a difference at the output of χ and an algorithm to generate all input differences for a given output difference of χ up to some weight.

A mapping χ consists of χ_ℓ if it is single-circle and the parallel application of a number of χ_ℓ mappings otherwise. Therefore, the weight and minimum reverse weight of a state is simply that of its circles. Moreover, backward extension can be done at the level of the circles. Therefore, in this section we work at the level of χ_ℓ. We denote differences at the input of χ_ℓ by b and differences at its output as a.

We start this with the score-at-a function for χ_ℓ that we denote it as $\mathsf{score}_{\chi^{-1}}(a)$. Let us call states a' that can be constructed from a by replacing passive bits with active bits *descendants of* a. Then, a function qualifies as a score-at-a function if it lower bounds for the minimum reverse weight of a and all its descendants.

Definition 8 (descendants of an output difference a)**.** We denote the set of descendants of a by $\mathsf{desc}(a)$.

$$\mathsf{desc}(a) := \{a' \in \mathbb{F}_2^n \mid \forall i : (a'_i + 1) \cdot a_i = 0\}.$$

Note that according to this definition $a \in \mathsf{desc}(a)$.

Definition 9 (Score-at-a function)**.** A function $f(a)$ that takes as input a difference pattern a and returns an integer qualifies as a *score-at-a* with respect to χ_ℓ if

$$\forall a' \in \mathsf{desc}(a) : f(a) \leq \mathrm{w}_{\mathrm{rev}}(a'), \ .$$

We first define *dense* substrings and dense circular strings that we need in our descriptions.

Definition 10 (Dense string)**.** We call a string *dense* if its leftmost bit and rightmost bit are both 1 and it does not contain string 00 as substring.

Examples of dense strings are $1, 11, 101, 111, 1101, 1011, 1111, \ldots$.

Definition 11 (Dense circular string)**.** We call a circular string *dense* if it does not contain string 00 as substring, and non-dense otherwise.

Examples of circular dense strings are 01101 and 101010, while circular strings 1101001 and 010110 are non-dense.

We will first present our score-at-a for dense output differences of χ_ℓ and then for the more complicated case of non-dense output differences.

4.1 Score-at-a for a dense output difference of χ_ℓ

Lemma 4. *For a dense output difference a of χ_ℓ, $f(a) = \left\lceil \frac{\ell}{2} \right\rceil$ qualifies as a score-at-a function.*

Proof. Per definition, the minimum reverse weight of a is the minimum weight over all input differences b compatible with a. A compatible input difference b can have no substrings 0000 as it would give rise to a string 00 in a (see Lemma 2) that contradicts the fact that a is dense.

So, any compatible b can be written as a concatenation of strings $\in \{1, 01, 001, 0001\}$. Each one of the substrings from $\{1, 01, 001, 0001\}$ in b contributes to its weight: due to Proposition 3 strings 1 and 01 contribute 1, while 001 and 0001 contribute 2.

The sum of the lengths of the substrings must match ℓ. Let q_0, q_1, q_2, q_3 denote the number of $1, 01, 001, 0001$ substrings in b and $\mathbf{q} = (q_0, q_1, q_2, q_3)$. The minimum reverse weight is upper bound by the following quantity:

$$\min_{\mathbf{q} \text{ with } q_0+2q_1+3q_2+4q_3=\ell} q_0 + q_1 + 2q_2 + 2q_3 =$$
$$\min_{\mathbf{q} \text{ with } q_0+2q_1+3q_2+4q_3=\ell} \frac{(q_0 + 2q_1 + 3q_2 + 4q_3) + q_0 + q_2}{2} = \min_{q_0,q_2} \frac{\ell + q_0 + q_2}{2} .$$

Clearly, the minimum is achieved by minimizing $q_0 + q_2$. This corresponds with forming b by as many possible 01 or 0001 strings that fit in ℓ bits:

- for even ℓ the weight is $\ell/2$, achieved by input differences solely consisting of strings 01 and 0001.
- for odd ℓ, the weight is $(\ell + 1)/2$, achieved by input differences that have a single string 1 or 001 and all remaining strings 01 or 0001.

This proves that the minimum reverse weight of a is lower bound by $\left\lceil \frac{\ell}{2} \right\rceil$. Moreover, when adding active bits to a dense output difference it remains dense, and therefore $\left\lceil \frac{\ell}{2} \right\rceil$ qualifies as a score-at-a. □

4.2 Score-at-a for a non-dense output difference of χ_ℓ

For non-dense differences we define a score-at-a function that makes use of a decomposition of a in dense strings interleaved with 0-runs of length at least 2. We call this the dense-string decomposition.

Definition 12 (Dense-string decomposition)**.** The *dense-string decomposition* of a non-dense circular string s is

$$s = \tau^k(d_0 \,\|\, (0)^{p_0} \,\|\, d_1 \,\|\, (0)^{p_1} \,\|\, \ldots \,\|\, d_{\mu-1} \,\|\, (0)^{p_{\mu-1}}) .$$

for the smallest offset $k \in [0, \ell - 1]$ and with d_i dense strings and $p_i \geq 2$ for all $i \in [0, \mu - 1]$.

We reserve the term *dense substring* for the dense strings d_i in the dense-string decomposition of an output difference a of χ_ℓ.

In this section we approach the construction and correctness proof of $\mathsf{score}_{\chi^{-1}}(a)$ for non-dense strings in an incremental way. We first show a lower bound for score-at-a functions in the form of the minimum weight of input differences in a certain class defined by a: the *score-bounding input differences*. A score-bound input difference for a has for each dense substring in a a so-called *semi-dense* substring. We observe that the weight of semi-dense substrings can be split in a context-free *inherent* weight and a context-dependent *bonus* weight, and this leads to an intermediate non-tight score-at-a function based on the inherent weights only. We then analyze the relative positioning of semi-dense substrings

in score-bounding input differences given the dense-string decomposition of a, and lower bound the contribution of the bonus weights, arriving at a tighter score-at-a function, that we call $\mathsf{score}_{\chi^{-1}}(a)$.

Any input difference compatible with an output difference a or any of its descendants must satisfy the conditions stated in the following lemma.

Lemma 5. *Let a be a non-dense output difference, then an input difference b compatible with a or with any $a' \in \mathsf{desc}(a)$ shall satisfy the following conditions. For each dense substring in a with l and r denoting the positions of its leftmost and rightmost bit respectively:*

1. *b shall have at least one active bit in the interval $[l, l+2]$,*
2. *b shall have at least one active bit in the interval $[r, r+2]$,*
3. *b shall have no substrings 0000 in the interval $[l, r+2]$.*

We call a difference b that satisfies these three conditions a score-bounding input difference *for output difference a.*

Proof. We prove each of the conditions separately:

1. If b has 000 in position $[l, l+2]$, then $a'_l = 0$. Now, a_l is the leftmost bit of a dense string and is 1 and hence it follows $a'_l = 1$, so this is not possible.
2. If b has 000 in position $[r, r+2]$, then $a'_r = 0$. Now, a_r is the rightmost bit of a dense string and is 1 and hence it follows $a'_r = 1$, so this is not possible.
3. If b has 0000 in position $[i, i+3]$ then a' has 00 in position $[i, i+1]$ and so must a. A dense string has no 00 substrings so this cannot occur for $i \in [l, r]$ and therefore the interval $[l, r+2]$ of b cannot have a substring 0000. □

Score-bounding input differences have for each dense substring in a a substring that we call *semi-dense*.

Definition 13 (Semi-dense string)**.** We call a string *semi-dense* if its leftmost bit and rightmost bit are both 1 and it does not contain string 0000 as substring.

Examples of semi-dense strings are $1, 11, 101, 111, 1101, 110001$, but 100001 not.

Corollary 2. *Let b be a score-bounding input difference for a non-dense output difference a. Then for each dense substring in a in position $[l, r]$, there shall be a semi-dense string in b with leftmost bit in interval $[l, l+2]$ and rightmost bit in interval $[r, r+2]$.*

Proof. This follows immediately from Lemma 5 with $a' = a$. □

The contribution of a semi-dense substring of a score-bounding input difference b to its weight is simply its Hamming weight plus the number of 001-strings it contains, plus 1 if there is a 00-string at its left. This possible 00-string at its left is not part of the semi-dense substring and hence depends on the context. However, the other part of the contribution can be computed from the semi-dense substring alone. We call that part its *inherent* weight and the contribution due to the 00-string at its left its *bonus* weight. We will now provide exact definitions.

Definition 14 (Inherent and bonus weight of substrings)**.** The inherent weight of a substring of b is its Hamming weight plus the number of 001 strings it contains.

The bonus weight of a substring of b in position $[l, r]$ is 1 if there is a string 001 in interval $[l-2, l+1]$.

Lemma 6. *The inherent weight of a semi-dense string of length n is lower bound by $\lfloor (n/2) + 1 \rfloor$.*

Proof. A semi-dense string consists of the concatenation of strings $\in \{1, 01, 001, 0001\}$, where the leftmost substring is 1. The sum of the lengths of these strings must match n. Let q_0, q_1, q_2, q_3 denote the number of $1, 01, 001, 0001$ substrings in b and $\mathbf{q} = (q_0, q_1, q_2, q_3)$. The weight of a semi-dense string of length n is lower bound by:

$$\min_{\mathbf{q} \text{ with } q_0+2q_1+3q_2+4q_3=n} q_0 + q_1 + 2q_2 + 2q_3 =$$
$$\min_{\mathbf{q} \text{ with } q_0+2q_1+3q_2+4q_3=n} \frac{(q_0 + 2q_1 + 3q_2 + 4q_3) + q_0 + q_2}{2} = \min_{q_0,q_2} \frac{n + q_0 + q_2}{2} .$$

Clearly, the minimum is achieved by minimizing $q_0 + q_2$. As the semi-dense string starts with 1, we have $q_0 \geq 1$. If n is odd, we take $q_0 = 1$ and $q_2 = 0$ resulting in weight $\frac{n+1}{2}$. If n is even, either $q_0 = 2$ and $q_2 = 0$ or $q_0 = 1$ and $q_2 = 1$ and the weight is $\frac{n}{2} + 1$. □

With the elements we now have we can already build a score-at-a function.

Corollary 3 (A loose score-at-a)**.** *The function for a non-dense output difference a that returns $\sum_i \max\left(1, \left\lfloor \frac{|d_i|}{2} \right\rfloor\right)$ with the sum taken over the dense substrings d_i of a qualifies as a score-at-a function.*

Proof. For a dense substring of length n in a, the corresponding semi-dense substring in a score-bounding input difference b has length between $\max(1, n-2)$ and $n+2$. For $n \leq 2$, the semi-dense substrings with length 1 minimize the inherent weight: it is 1. For $n > 2$, the semi-dense substrings of length $n-2$ have minimum inherent weight that is equal to $\left\lfloor \frac{n}{2} \right\rfloor$. Adding the inherent weights of the semi-dense substrings corresponding to all dense substrings of a gives the lower bound in the corollary. □

From the function in Corollary 3 we can build a tighter score function by taking into account the bonus weight of the semi-dense substrings in score-bounding input differences b corresponding to dense substrings in a.

Lemma 7. *Consider a single-bit dense substring in a at position $l(= r)$. Then the corresponding semi-dense string in b has inherent weight 1 if it is one of the following three:*

- *a single 1 in position l,*
- *a single 1 in position $l + 1$,*
- *a single 1 in position $l + 2$.*

All other semi-dense string choices have strictly higher inherent weight.

Proof. The three conditions of Lemma 5 reduce to a single condition, namely to have an active bit in the interval $[l, l+2]$. The lowest inherent weight is 1 and that is only obtained for a single-bit semi-dense string. □

Lemma 8. *Consider a dense substring in a with even length n in position $[l, r(= l+n-1)]$. Then the corresponding semi-dense string in b has inherent weight $n/2$ if it is one of the following types:*

- *length $n-1$ in position $[l+1, r]$,*
- *length $n-1$ in position $[l+2, r+1]$.*

- *length* $n-2$ *in position* $[l+2, r]$.

All other semi-dense string choices have strictly higher inherent weight.

Proof. Clearly all three types of strings satisfy the conditions of Lemma 5, so they are valid corresponding semi-dense strings. They are the only possible semi-dense strings with leftmost bit in interval $[l, l+2]$ and rightmost bit in interval $[r, r+2]$ that have length either $n-1$ or $n-2$ and hence having inherent weight $n/2$. The other two possible lengths are n and $n+1$ and they have inherent weight $n/2+1$. □

Lemma 9. *Consider a dense substring in a with odd length* $n > 1$ *in position* $[l, r]$. *Then the corresponding semi-dense string in b with inherent weight* $(n-1)/2$ *is in position* $[l+2, r]$ *and so has length* $n-2$. *All other semi-dense string choices have strictly higher inherent weight.*

Proof. Clearly a semi-dense string in position $[l+2, r]$ satisfies the conditions of Lemma 5 so it is a valid corresponding semi-dense string. It is the only possible semi-dense string with leftmost bit in interval $[l, l+2]$ and rightmost bit in interval $[r, r+2]$ that has length $n-2$ and hence has inherent weight $(n-1)/2$. The three other two possible lengths are $n-1, n$ and $n+1$ and they have inherent weight $n/2$ and higher. □

Using the previous three lemmas we can now tighten the score function by taking into account the bonus weight of each semi-dense string in score-bounding input difference b of an output difference a.

Lemma 10 (Bonus weights due to dense substrings longer than a single bit)**.** *A dense substring* d_i *longer than 1 in a adds a bonus weight 1 on top of the minimum inherent weight to the score function if at least one of the following conditions are satisfied:*

1. d_i *has odd length,*
2. d_i *has* 000 *at its left,*
3. d_{i-1} *is longer than 1.*

Proof. We prove the points of the lemma one by one:

1. From Lemma 9 it follows that the only semi-dense string with inherent weight $(n-1)/2$ has its leftmost bit in position $l+2$, so it has 00 in position $[l, l+1]$, implying a bonus bit.
2. From Lemma 9 and Lemma 8 it follows that for semi-dense strings with minimum inherent weight corresponding to dense substrings in a with length $n > 1$ we have $b_l = 0$. The highest possible position of the rightmost bit of the semi-dense string corresponding to d_{i-1} is $l-3+1$, namely if d_{i-1} has length 1 and the 0-run at the left of d_i has length 3. Therefore b has 00 in position $[l-1, l]$, implying a bonus bit.
3. Let the 0-run at the left of d_i have length 2, so the dense substring d_{i-1} has its rightmost bit in $l-3$. The highest possible position of the rightmost bit of the semi-dense string corresponding to d_{i-1} is $l-2$, namely if d_{i-1} has even length. Therefore b has 00 in position $[l-1, l]$, implying a bonus bit. □

Lemma 11 (Bonus weights)**.** *A dense substring* $d_i = 1$ *adds a bonus weight 1 on top of the minimum inherent weight to the score function if at least one of the following conditions are satisfied:*

1. d_i *has* 0000 *at its left.*
2. d_i *has* 000 *at its left and* d_{i-1} *has even length*

3. d_{i-1} has odd length and is longer than 1.

Proof. In all 3 cases the highest possible position of the rightmost bit of the semi-dense string corresponding to d_{i-1} is $l-3$. Therefore b has 00 in position $[l-2, l-1]$, implying a bonus bit. □

Taking into account these bonus weights gives a tighter score-at-a function, that we denote as $\mathsf{score}_{\chi^{-1}}(a)$r

Definition 15 ($\mathsf{score}_{\chi^{-1}}(a)$)**.** Let a be an output difference of χ_ℓ. If a is dense, $\mathsf{score}_{\chi^{-1}}(a) = \lceil \frac{\ell}{2} \rceil$. If a is non-dense, let d_i be the dense substrings of a. We have

$$\mathsf{score}_{\chi^{-1}}(a) = \sum_i \left\lceil \frac{|d_i|}{2} \right\rceil + B_i \,,$$

with $B_i = 1$ if

- d_i has even length and it has 000 at its left or d_{i-1} is longer than 1.
- $d_i = 1$ and it has at its left 0000, or 000 preceded by d_{i-1} of even length, or 00 preceded by d_{i-1} of odd length longer than 1.

and 0 otherwise.

Proposition 5. *The function* $\mathsf{score}_{\chi^{-1}}(a)$ *defined in Definition 15 qualifies as a score-at-a function.*

Proof. For dense a, the proposition follows directly from Lemma 4. For non-dense a, we look at the contribution of the three types of dense substrings to the score:

- A dense substring d_i in a of odd length different from 1 has inherent weight $(|d_i|-1)/2$ due to Lemma 9 and bonus weight 1 due to Lemma 10, resulting in $\left\lceil \frac{|d_i|}{2} \right\rceil$.
- A dense substring d_i in a of even length has inherent weight $|d_i|/2$ due to Lemma 8. This corresponds to the contribution $\left\lceil \frac{|d_i|}{2} \right\rceil$. The value of B_i corresponds to the bonus weight that is 1 due to Lemma 10 if it has 000 at its left or if d_{i-1} is longer than 1.
- A dense substring $d_i = 1$ in a of length 1 has inherent weight 1 due to Lemma 7. This corresponds to the contribution $\left\lceil \frac{|d_i|}{2} \right\rceil$. The value of B_i corresponds to the bonus weight that is 1 due to Lemma 11 if it has at its left 0000, or 000 preceded by d_{i-1} of even length, or 00 preceded by d_{i-1} of odd length longer than 1. □

4.3 Computing $\mathrm{w}_{\mathrm{rev}}(a)$ given non-dense a

In this section, we present a method for computing the exact value of $\mathrm{w}_{\mathrm{rev}}(a)$ for a *non-dense* given difference at the output of χ_ℓ.

Regarding the case of *dense* string and since our main focus is big circles (namely, $\ell \gg 1$ is a large value), the result provided in Lemma 4 is sufficient for our goal. Indeed, we have $\mathrm{w}_{\mathrm{rev}}(a) \geq \lceil \frac{\ell}{2} \rceil$. Since we want to generate 2-round trail cores up to a target weight T_2, if this weight is smaller than $\lceil \frac{\ell}{2} \rceil$ we can safely assume that this case cannot happen and the output difference contains at least a substring of two consecutive zeroes.

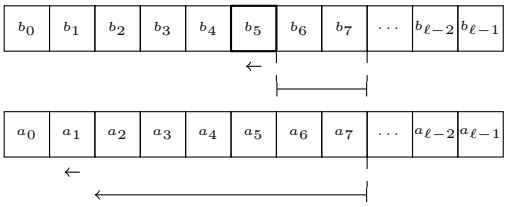

Figure 2: Required bits at the input and output of χ_ℓ to set one bit of representative minimum input difference at b_5.

The "representative minimum input difference". Although it is possible that several input differences realize the minimum reverse weight for a given output difference, we limit ourselves to generate only one of them, that we call *representative minimum input difference*. Thus, given a *non-dense* a, the weight of its representative minimum input difference is the minimum reverse weight of a. We provide an algorithm that sets the value of one bit of the representative minimum input difference b_{i-1}, given the output difference and two bits of the representative minimum input difference, namely b_i and b_{i+1}. Working iteratively, we can build the entire representative minimum input difference. As an example, Fig. 2 represents the required bits of the representative minimum input difference b and output difference a to set b_5.

Algorithm for generating the "representative minimum input difference". Our algorithm first determines two adjacent bits of b and then the remaining bits in an iterative fashion. A pseudo-code of this algorithm is given in Algorithm 2.

Since we assume a is not dense, then there is a 0-run of length at least 2 in a. W.l.o.g., we assume that the 0-run of length more than 1 start at position i, namely $a_i \in \underline{0}0$. We consider the four possible cases for $b_i \| b_{i+1}$, namely $b_i \in \{\underline{0}0, \underline{0}1, \underline{1}0, \underline{1}1\}$. For each one of these cases, the algorithm generates all the other bits of the input differences iteratively, using the rules that are described in Proposition 6. Given these four input differences corresponding to the cases $b_i \in \{\underline{0}0, \underline{0}1, \underline{1}0, \underline{1}1\}$, the compatible one with the lowest weight is the representative minimum input difference.

Proposition 6. *Let $a \in \mathbb{F}_2^\ell \setminus \{\mathbf{0}\}$ be a non-dense output difference of χ_ℓ. Let's denote by b the "representative minimum input difference". For each $i \in \{0, 1, \ldots, n-1\}$, given a and $b_i \| b_{i+1} \in \mathbb{F}_2^2$ two adjacent bits of the representative minimum input difference:*

- $b_{i-1} \leftarrow 1$ *if one of the following conditions is satisfied:*
  1. $b_i \in \underline{0}0$ *and* $a_i \in 1\underline{\star}$,
  2. $b_i \in \underline{0}1$ *and* $a_i \in 111\underline{\star}1$,
  3. $b_i \in \underline{1}0$ *and* $a_i \in 00(10)^m \star \underline{1}0$, *where* $m = 2m' + 1$,
  4. $b_i \in \underline{0}1$ *and* $a_i \in \{0(01)^m 1\underline{\star}1, 00(10)^m 0\underline{\star}1\}$, *where* $m = 2m' + 1$;
- *in all other cases:* $b_{i-1} \leftarrow 0$.

Proof. We assume $i = 0$ and prove that $b_{-1} = 1$ minimizes the weight when one of the listed conditions is satisfied. But, the proof can be generalized for any i since χ_ℓ is translation-invariant. To minimize the weight of the input difference, the sum of the number of (001)-strings and the Hamming weight should be minimized. We analyze the conditions separately and use Lemma 2 to check compatibility.

1. Here, $b_{-1} = a_{-1} = 1$ because otherwise, the input and output are incompatible.

2. If $b_0 \in \underline{0}1$ and $a_0 \in 111\underline{\star}1$, then to affect active bit at a_{-3}, at least one bit in the interval $[-3,-1]$ of the input must be active. Hence, we have three sub-cases:
   (a) if $b_0 \in 100\underline{0}1$, then the input is not compatible with the output because $b_{-2} \neq a_{-2}$;
   (b) if $b_0 \in 10\underline{0}1$, then b can be compatible with the output, and this 4-bit part of b weighs 3;
   (c) if $b_0 \in \star 1\underline{0}1$, then the input can be compatible with the output, and this 4-bit part of b weighs at most 3 but it can weigh 2.

   Therefore, setting b_{-1} to 1 is the best choice.

3. We start by proving the result for $m' = 0$, and then we prove it for a generic m'.
   - Case $m' = 0$: then $b_0 \in \underline{1}0$ and $a_0 = 0010 \star \underline{1}0$. To affect the active bit at a_{-3} we need to have at least one active bit in the interval $[-3,-1]$. In order to prove this case, we need to consider 7 bits of the input then, we have three sub-cases:
     (a) if $b_0 \in \star\star 100\underline{1}0$, then the input can be compatible with the output, and this 7-bit part of b weighs at least 4;
     (b) if $b_0 \in \star\star\star10\underline{1}0$, then in case of $a_0 = 0010\underline{1}10$, b is not compatible with the output, and in case of $a_0 = 0010\underline{0}10$ this 7-bit part of b weighs at least 3;
     (c) if $b_0 \in \star\star\star\star 1\underline{1}0$, then the input can be compatible with the output, and this 7-bit part of b weighs at least 3;

     In order to cover both $a_0 = 0010\underline{1}10$ and $a_0 = 0010\underline{0}10$ and have the minimum possible weight, $b_{-1} = 1$ is the best choice.
   - More generally, assume $b'_{-1} = 1$ minimizes the weight when $a'_0 \in 00(10)^{2m'+1}\star\underline{1}0$, then we prove $b_{-1} = 1$ minimizes the weight when $a_0 \in 00(10)^{2(m'+1)+1} \star \underline{1}0$.

     In the case of $a_0 \in 00(10)^{2m'+3} \star \underline{1}0$, the two passive bits at positions $-4m' - 9$ and $-4m' - 8$ do not influence our analysis. Therefore, we represent this difference as $a_0 \in \mathrm{xx}(10)^{2m'+3} \star \underline{1}0$ when bits that do not play any role in our analyse are denoted by x's. To affect the leftmost active bit at $a_{-4m'-7}$, there should be at least one active bit in the interval $[-4m' - 7, -4m' - 5]$. Just like previous case, we have 3 sub-cases and among them $b_{-4m'-5} = 1$ minimizes the weight.

     An active bit at $b_{-4m'-5}$ affects both active bits at $a_{-4m'-7}$ and $a_{-4m'-5}$ then, we have $a_0 \in \mathrm{xxxxx}0(10)^{2m'+1} \star \underline{1}0$. Since x's do not play any role in our analyse we can consider them as passive bits.

     Based on our assumption, $b'_{-1} = 1$ minimizes the weight when $a'_0 \in 00(10)^{2m'+1}\star \underline{1}0$. Therefore, $b_{-1} = 1$ minimizes the weight in this case.

4. The conditions can be analyzed in the same way as case 3.

As adding active bits cannot decrease the weight, this proves the proposition. □

Special Case: 0-run with length at least 4. In the case in which a has a 0-run of length 4 or more, then we can speed up the construction of the representative minimum input difference. Indeed, in such a case, we can fix the bits of b that correspond to the 0-run of length more than 3 in a to zero, as we are going to prove in the Lemma 12. Therefore, we can immediately fix the value of some parts of the representative minimum input difference and furthermore, instead of considering 4 different cases ($b_i \in \{\underline{0}0, \underline{0}1, \underline{1}0, \underline{1}1\}$), we only have one case.

Algorithm 2 Generating the representative minimum input difference for a given output.

Input: A non-dense output difference $a \in \mathbb{F}_2^\ell$
Output: The representative minimum input difference $b \in \mathbb{F}_2^\ell$

if $(\exists j$ s. t. $a_j \in \underline{0}000)$ **then**
  $b_j \| b_{j+1} \| b_{j+2} \| b_{j+3} = 0000$
  for all $i = (j) \to (0)$ & $(n-1) \to (j+5)$ **do**
    if (a_i & b_i satisfies one of the conditions in Proposition 6) **then**
      $b_{i-1} \leftarrow 1$
    else
      $b_{i-1} \leftarrow 0$
  return b
else if $(\exists j$ s. t. $a_j \in \underline{0}0)$ **then**
  $b_j^{(0)} \| b_{j+1}^{(0)} = 00$, $b_j^{(1)} \| b_{j+1}^{(1)} = 01$, $b_j^{(2)} \| b_{j+1}^{(2)} = 10$, $b_j^{(3)} \| b_{j+1}^{(3)} = 11$
  for all $k = 0 \to 3$ **do**
    for all $i = (j) \to (0)$ & $(n-1) \to (j+3)$ **do**
      if (a_i & b_i satisfies one of the conditions in Proposition 6) **then**
        $b_{i-1}^{(k)} \leftarrow 1$
      else
        $b_{i-1}^{(k)} \leftarrow 0$
    if ($b^{(k)}$ is compatible with a and $\mathrm{w_r}(b^{(k)})$ is the minimum) **then**
      return $b^{(k)}$

Lemma 12. *Let a be an output difference of χ_ℓ that has a 0-run with length 4 or more. Then, a compatible input difference b with minimum reverse weight has a 0-run of the same length in the same position as the one in a.*

Proof. Since χ_ℓ is translation-invariant, let's assume the 0-run is in $[0, m]$ with $3 \leq m < \ell$. In order to minimize the weight, we have to find the compatible difference that minimizes the sum of Hamming weight and $\#_{001}$. To this end, we aim to minimize $\#_{001}$ in compatible differences with minimum Hamming weight.

According to the definition of χ_ℓ, each bit in the input difference b_i affects three bits of the output, namely in the interval $[i-2, i]$. Therefore, having active bits at the input in the interval $[2, m]$ only affects bits of the output in the interval $[0, m]$. Since there is no active bit in the interval $[0, m]$ of the output, we put zero-string in the interval $[2, m]$ of the input difference to minimize its Hamming weight. In this case and when $3 \leq m$, having active bit(s) at b_0 or b_1 makes the input incompatible with the output (see Lemma 2). So, we set $b_0 \| b_1 = 00$ that implies the interval $[0, m]$ of the input is fully passive. □

Verification. We experimentally verified the rules provided in Proposition 6 for all non-dense output differences of χ_ℓ when $3 \leq \ell \leq 22$.

4.4 Generating all input differences of χ_ℓ given its output difference

Using a similar method as in Section 2.4.2, we show how to generate all input difference of χ up to a target weight for a given output difference a. Since χ_ℓ acts on each circle independently, the total weight over χ is given by the sum of the weight of each circle. Therefore, we work at the level of χ_ℓ. The algorithm generates a tree where each node of the tree represents an input difference of χ_ℓ.

The algorithm starts from the root node that is a fully passive input difference ($b = (0)^\ell$). The algorithm iteratively applies the function `next()` (Algorithm 1) to visit the next node in the tree. Let us denote by b_t the active bit of b with the smallest index, so t represents

the position of its leftmost active bit. Then, the three functions used during the search are defined as follows:

toFirstChild() adds an active bit in b_{t-1} if $t > 0$, and returns true. It returns false if $t = 0$. When the current node is empty, it adds one active bit at position $\ell - 1$.

toSibling() moves leftmost active bit of input difference by one position to the left ($b_t \to b_{t-1}$), if $t > 0$. If $t = 0$ it returns false and true otherwise.

toParent() removes the leftmost bit. It returns false if the result is an empty input difference and true otherwise.

The search ends by returning false.

Here, additional conditions consist of verifying that the weight of the input difference does not exceed the target weight, and checking compatibility of the input and output. Since the interval $[t, \ell - 1]$ of b is known in each node, using local compatibility rules given in Lemma 2, it is possible to check compatibility once the value of the input difference bit at the interval $[\ell - 3, \ell - 1]$ is fixed. That is due to the fact that the provided rules check the compatibility of bit b_{i-1} with the output given bits at the interval $[i, i + 2]$ of b. Hence, after setting 3 bits of the input of χ_ℓ, once we add a new bit, we can check compatibility since the interval $[t, \ell - 1]$ is known.

5 Differential trail search in Subterranean

We applied the techniques presented in Sections 2 to 4 to perform differential trail search in Subterranean. We report on the bounds on the weight of differential trails in Table 1 for different number of rounds. In this section, we first provide details about Subterranean and then, we explain how we achieved the results of Table 1.

5.1 The round function R in Subterranean

The round function R in Subterranean operates on a 257-bit state and consists of four steps: $\mathrm{R} = \pi \circ \theta \circ \iota \circ \chi$, where χ is the only non-linear mapping. For each $0 \leq i < 257$ we have:

$$\begin{aligned} \chi: &\quad s_i \leftarrow s_i + (s_{i+1} + 1) \cdot s_{i+2} \ , \\ \iota: &\quad s_0 \leftarrow s_0 + 1 \ , \\ \theta: &\quad s_i \leftarrow s_i + s_{i+3} + s_{i+8} \ , \\ \pi: &\quad s_i \leftarrow s_{12 \cdot i} \ , \end{aligned}$$

with indices taken modulo 257. Fig. 3 illustrates the round function of Subterranean and the steps to compute a single bit s_{92} at the output of the round function. The Subterranean 2.0 cipher suite consists of 3 primitives: an extendable output function (XOF) Subterranean-XOF, a deck function Subterranean-Deck and a session authenticated encryption scheme Subterranean-SAE. The former two have 8 blank rounds between the absorbing and the squeezing phases and the latter has 8 blank rounds after the loading of the session diversifier and before the generation of each tag. The rationale is that it is assumed to be infeasible to exploit difference propagation over 8 or more rounds.

5.2 Tree-search in Subterranean

We define a unit as a single active bit at a and we denote it by its index position in the state. Units are ordered by the order relation over natural numbers. Let $i_k \in \{0, 1, \ldots, n - 1\}$

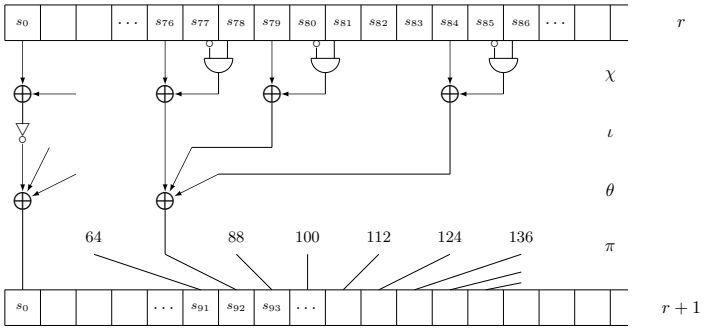

Figure 3: SUBTERRANEAN round function.

denote the position of an active bit at a, i.e., $a_{i_k} = 1$. Given a difference pattern a, its *unit list* is the ordered list $[i_1, i_2, \ldots, i_j]$, where $a_{i_k} = 1$ if and only if $k \in \{1, \ldots, j\}$ and $i_k < i_{k+1}$ $\forall k \in \{1, \ldots, j-1\}$.

5.2.1 Translation-invariance and canonicity

The round function of many cryptographic primitives exhibits translation symmetry if we make abstraction of round key or round constant addition. For example, in AES the state is a 4-by-4 array of bytes and the unkeyed round function is invariant under any (cyclic) translation of the array in the plane [DR02]. Similarly, in Keccak-f[1600] the state is an 5-by-5-by-64 array of bits and the round function with round constant addition removed is invariant under any of the 64 (cyclic) translations in the direction of the z-axis [BDPA11]. The state of ASCON is a 5-by-64 array of bits and the round function with round constant addition removed is invariant under any of the 64 (cyclic) translation in the direction of the x-axis [DEMS14]. In all cases the set of translations that the round function is invariant to forms a group and we denote it as $\mathcal{T}$.

This translation-invariance is inherited by trails and trail cores and for instance in each trail core $\tilde{Q}$ in Keccak-f[1600] there are 63 other trails that just consist of translated versions of the difference patterns in $\tilde{Q}$. Translation invariance naturally partitions the state space in classes where two states a and a' are in the same class if and only if $\exists \tau \in \mathcal{T} : a' = \tau(a)$.

When generating 2-round trail cores, one wishes to avoid double work to only consider one pattern a per equivalence class. With this purpose we appoint a unique representative per class that we call *canonical* and we will only consider 2-round trail cores where a is canonical. When using χ as a non-linear layer, it makes sense to have a round function with some form of translation-invariance and therefore it makes sense to make use of canonicity.

In [MDA17] canonicity has been defined at the level of the unit lists. A unit is a bit pattern in the state and therefore can be translated. Similarly, a unit list is a set of units and its translation is just the set of all units translated. Equivalence of unit lists with respect to translation is therefore defined as follows:

Definition 16 (Translation-equivalence of unit lists)**.** Two unit lists $U = [u_1, u_2, \ldots u_j]$ and $U' = [u'_1, u'_2, \ldots u'_j]$ are equivalent with respect to $\mathcal{T}$ if and only if $\exists \tau \in \mathcal{T} : U' = \tau(U)$.

Clearly, two unit lists can only be translation-equivalent if they have the same number of units. Moreover, if two unit lists corresponding to a given state are translation-equivalent, the differences a they encode are translation-equivalent. On the other hand, it is not immediate that equivalence of differences a and a' implies equivalence of corresponding unit lists. This requires the definition of unit lists that is symmetric in τ: if u is a valid

unit, then $\tau(u)$ is a valid unit for all $\tau \in \mathcal{T}$. This requirement was not made explicit in [MDA17] but is satisfied by the definition of units in that paper and follow-up work.

To define the canonicity of a unit list, Mella et al. define an order relation among unit lists and define the canonical unit list to be the smallest within its equivalence class. This order relation is simply *lexicographical order*: comparing the unit lists starting from the smallest units and the first units they differ in determine the order.

Definition 17 (Lexicographical order of unit lists)**.** [MDA17, Sect. 3.2] Given two unit lists $U = [u_1, u_2, \dots u_j]$ and $U' = [u'_1, u'_2, \dots u'_j]$ in the same equivalence class, we say that $U \leq U'$ if $\exists k \leq j$ s. t. $u_k \leq u'_k$ and $u_m = u'_m$ $\forall m < k$.

They proved that with this order relation, the parent of a canonical unit list is a canonical unit list in [MDA17, Lemma 1].

We define canonical unit lists as follows.

Definition 18 (Canonicity)**.** A unit list is canonical if it is the smallest in its equivalence class with respect to some order relation defined on unit lists.

Hence, when traversing the tree of 2-round trail cores, once we encounter a node that is non-canonical, we can prune it and all its descendants.

5.2.2 Canonicity in Subterranean

In Subterranean, all steps are translation-invariant except π. However, the analysis given in Section 5.2.1 is still valid.

Let $\tilde{Q}_2 = \xrightarrow{\chi} a \xrightarrow{\lambda} b \xrightarrow{\chi}$ be a 2-round trail core. Then, by translating the position of active bits at a by a certain offset x, the position of active bits at b shifts by $150x$. As a simple example, consider a 2-round trail core with only one active bit at a_0. After λ there will be three active bits, namely b_0, b_{64} and b_{85}. If we translate the position of the active bit by one, i.e. we move it to a_1, then the three active bits after λ will be at b_{150}, b_{214} and b_{235}. The last three active bits at b are just translated version of the first three active bits but with offset 1×150. Since the χ_ℓ map is translation-invariant, these two trail cores are equivalent. As a result, in our tree search we can consider them in an equivalence class.

Notice that in this case, the number of unit lists in each equivalence class is 257, that is the size of the state in Subterranean. Thus, considering only one unit list among all the 257 in each class significantly reduces the search space.

5.2.3 Traversing the tree in Subterranean

We call the first unit of a unit list $[i_1, i_2, \dots, i_j]$ the *tail* bit and denote it by $t = i_1$, and the last unit of it the *head* bit and denote it by $h = i_j$. Then, the three functions used in Algorithm 1 during the search are defined as follows:

`toFirstChild()` adds an active bit in a_{h+1} if $h < \ell - 1$, and returns true. If $h = \ell - 1$, then it returns false.

`toSibling()` moves head bit of the unit list by one position to the right ($a_h \to a_{h+1}$), if $h < \ell - 1$. If $h = \ell - 1$ it returns false, true otherwise.

`toParent()` removes the head bit. If this results in an empty unit list it returns false and true otherwise.

Here, additional conditions consist of canonicity check and whether the score is in budget.

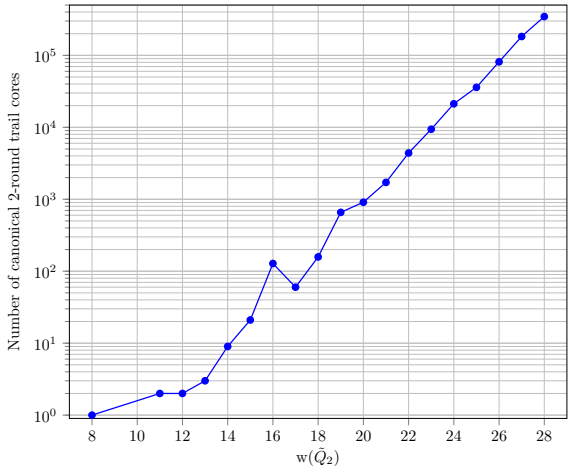

Figure 4: The number of canonical 2-round trail cores in Subterranean up to weight 28.

5.2.4 The score-at-b in Subterranean

We will denote the score-at-b that we use in Subterranean by $\mathsf{score}_{\mathrm{sub}}(a)$. Based on the properties of the linear layer, it is possible to identify active bits at b that cannot turn to passive by the addition of any new unit. We refer to such bits as *stable*. Then we compute the score caused by b by computing $\mathrm{w_r}(b')$ where b' is a difference state composed by only the stable bits of b.

Due to the property of θ, an active bit at a_i at the input of θ affects three bits at positions i, $i-3$ and $i-8$ after θ. If the distance between two active bits at a is greater than 8 then the active bits caused by them at the output of θ cannot overlap. Since the function `toFirstChild()` adds a new head bit and `toSibling()` only moves the current head bit to the right, the bits of the state after θ that are at distance bigger than 8 from the head bit will not change. Therefore, we define the function $\mathsf{score}_{\mathrm{sub}}$ as follows.

Definition 19 ($\mathsf{score}_{\mathrm{sub}}$)**.** Given a difference a with unit list $[i_1, i_2, \ldots, h]$, $\mathsf{score}_{\mathrm{sub}}(a) = \mathrm{w_r}(\pi(\bar{a}))$ where

$$\bar{a}_j = \begin{cases} [\theta(a)]_j \text{ for } i \text{ s.t. } |h-j| > 8, \\ 0 \text{ otherwise.} \end{cases}$$

5.3 Differential trail bounds

Our primary goal was to scan the space of all 8-round trail cores of Subterranean up to weight $T_8 = 115$, but we applied our tool to find bounds also for smaller number of rounds. To scan all 8-round trail cores up to 115, we first generate all 4-round trail cores up to weight $\lfloor 115/2 \rfloor = 57$. This took almost 307 hours on an Intel Xeon Silver 4110 processor running at 3GHz, using a single core. According to our analysis, we can also make a prediction on the time required to generate 8-round trail cores with bigger weight. E.g., in order to be able to scan all 8-round trail cores up to weight 128 we need to generate all 4-round trail cores up to weight $\lfloor 128/2 \rfloor = 64$ and then extend them. We expect it to take almost one year to scan the space of all 4-round trail cores up to 64 without parallelization. However, the extension of 2-round trails to 3, 4 and more rounds would be easy to parallelize and this would indeed reduce the time, generating all 2-round trail cores would take a substantial effort to parallelize. Hence, using n cores instead of one does not mean n times faster processing, but it will clearly increase the speed.

Table 2: Weight distribution table of canonical 2-round trail cores in SUBTERRANEAN up to weight 28.

$w_r(b)$ / $w_{rev}(a)$	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
2	-	1	-	-	-	-	-	2	-	-	-	-	-	1	-	-	-	-	-	-
3	-	-	-	1	-	-	1	-	1	2	-	-	-	-	1	1	1	-	-	-
4	-	-	-	2	2	2	12	111	9	8	28	25	56	421	19	26	51	34	46	592
5	-	1	-	1	1	3	3	23	32	237	32	58	281	136	308	860	97	145	537	
6	-	-	-	1	3	5	6	46	63	543	589	846	2795	7657	2953	3924	7952	8321		
7	-	-	2	2	5	9	49	257	108	437	640	1649	2519	7248	11550	28207	14548			
8	-	-	1	1	7	10	33	81	210	800	1497	3188	5832	16107	28196	85270				
9	-	-	1	3	10	24	72	197	855	1766	3259	6985	20312	44438	54708					
10	-	1	-	2	13	31	103	359	844	1875	5192	10444	24625	55736						
11	-	-	1	2	14	49	117	374	1217	3474	9595	21646	51896							
12	1	-	1	2	11	50	208	781	1945	4958	14616	34435								
13	-	-	1	5	16	86	264	708	2556	6799	21167									
14	-	-	2	7	27	94	502	1090	3726	11041										
15	-	-	2	8	26	149	438	1688	4846											
16	-	-	3	7	35	151	500	1940												
17	-	-	2	9	41	202	869													
18	-	1	1	14	53	213														
19	-	1	-	9	62															
20	-	-	3	20																
21	-	-	2																	
22	-	1																		

Table 3: 3-round trail core weight histogram. Each unit corresponds to 257 trail cores.

weight	25	28	29	30	31	32	33	34	35	36	37	38	39	40
# trail cores	1	1	2	6	7	5	3	12	34	34	51	42	81	114

Differential trail bound for 2 rounds. To cover the space of all 8-round trail cores up to $T_8 = 115$, we start by generating all 2-round trail cores $\xrightarrow{\chi} a \xrightarrow{\lambda} b \xrightarrow{\chi}$ up to $T_2 = \lfloor 115/4 \rfloor = 28$. Table 2 lists the number of 2-round trail cores, that are different modulo translation, regarding the value of $w_{rev}(a)$ and $w_r(b)$. Note that we report only on the number of canonical trail cores so, each unit in this table corresponds to 257 trail cores. Fig. 4 also represents the total number of canonical 2-round trail cores in SUBTERRANEAN regarding their weight $w(\tilde{Q}_2)$.

Our search confirms that a 2-round trail core has at least weight 8 (as reported in [DMMR20]), which corresponds to a single active bit in a and consequently 3 active bits in b. Then, there is no trail up to the weight 10, since it is not possible for active bits at a to cancel each others after θ and locate in certain places of b that results in trail with weight smaller than 11.

If we have two active bits at a that do not cancel each others after θ, there will be 6 active bits at b. In most of the cases the two active bits at a weigh 4, and the corresponding 6 active bits at b weigh 12, that results in many 2-round trail cores with weight 16.

For a 2-round trail core of weight 17, there should be overlap after θ and the corresponding active bits at b must be located in a limited certain places. This corresponds to drop in Fig. 4 between weights 16 and 18.

Differential trail bound for 3 rounds. A 3-round trail core $\tilde{Q}_3 = \xrightarrow{\chi} a^1 \xrightarrow{\lambda} b^1 \xrightarrow{\chi} a^2 \xrightarrow{\lambda} b^2 \xrightarrow{\chi}$ has weight $w(\tilde{Q}_3) = w_{rev}(a_1) + w_r(b_1) + w_r(b_2)$. This is satisfied if either $w_{rev}(a_1) \leq w_r(b_2)$ or $w_{rev}(a_1) > w_r(b_2)$:

- The former case implies that $2w_{rev}(a_1) + w_r(b_1) \leq T_3$. Therefore we can obtain all 3-round trail cores $\tilde{Q}_3$ with $w_{rev}(a_1) \leq w_r(b_2)$ by generating all 2-round trail cores

Table 4: A canonical 4-round differential trail core with the minimum weight.

state	weight	active bit positions
a_1	12	$\{0, 5, 8, 10, 12, 15, 16, 18, 21\}$
b_1	7	$\{65, 66, 85, 86, 87\}$
b_2	11	$\{7, 28, 134, 198, 200, 219\}$
b_3	28	$\{16, 18, 22, 39, 54, 86, 88, 107, 118, 139, 152, 173, 188, 211, 252\}$

$\tilde{Q}_2 = \xrightarrow{\chi} a^1 \xrightarrow{\lambda} b^1 \xrightarrow{\chi}$ satisfying $2\mathrm{w}_{\mathrm{rev}}(a_1) + \mathrm{w}_{\mathrm{r}}(b_1) \leq T_3$ and then extending each of them by one round in the forward direction.

- The latter case implies that $\mathrm{w}_{\mathrm{rev}}(a_2) + 2\mathrm{w}_{\mathrm{r}}(b_2) < T_3$. Therefore we can obtain all 3-round trail cores $\tilde{Q}_3$ with $\mathrm{w}_{\mathrm{rev}}(a_1) > \mathrm{w}_{\mathrm{r}}(b_2)$ by generating all 2-round trail cores $\tilde{Q}_2 = \xrightarrow{\chi} a^2 \xrightarrow{\lambda} b^2 \xrightarrow{\chi}$ satisfying $\mathrm{w}_{\mathrm{rev}}(a_2) + 2\mathrm{w}_{\mathrm{r}}(b_2) < T_3$ and then extending each of them by one round in the backward direction.

We adapted our tool to support the check of the required inequalities during the tree-search. For our experiments, we set the target weight T_3 to 40 and we report the number of trail cores found in Table 3. As before, we report only on the number of canonical trail cores, that is each unit in this table corresponds to 257 trail cores. Our search confirms that a 3-round trail core has at least weight 25, as reported in [DMMR20].

Differential trail bound for 4 rounds. In each 4-round trail core with weight up to 57 either the first or the last 2-round sub-trail core should weigh smaller than or equal to $\lfloor 57/2 \rfloor = 28$. Therefore, scanning the space of all 2-round trail cores up to 28 and extending them by two rounds in both directions assures us that we scanned all 4-round trail cores up to 57. Based on our results, there is no 4-round trail core up to weight 57 that means the lower bound on the weight of 4-round trail cores is 58. We actually found one canonical trail with weight 58, which is equal to the one already listed in [DMMR20] and given in Table 4. Here we limit ourselves to point out that other canonical 4-round trail cores with weight 58 may exist, but their research is not our primary goal. It is worth noticing, that a tight bound for the minimum weight of any trail over 4 rounds in Subterranean was missing before this work and we proved it.

Observation. During our search, we found that by extending all 2-round trail cores $\xrightarrow{\chi} a^3 \xrightarrow{\lambda} b^3 \xrightarrow{\chi}$ with $\mathrm{w}_{\mathrm{r}}(b^3) < 5$ in the backward direction, there exist no 4-round trail core up to the weight 60. Therefore, given a 4-round trail core $\tilde{Q}_4$ with weight $T_4 \leq 60$, the following condition holds:

$$\mathrm{w}_{\mathrm{r}}(a^1) + \mathrm{w}_{\mathrm{r}}(b^1) + \mathrm{w}_{\mathrm{r}}(b^2) \leq T_4 - 5\,.$$

Based on this observation, if we target $T_4 \leq 60$ then we can first generate all 3-round trail cores up to $T_4 - 5$ that results in minimizing the search space.

Similarly, we found that by extending all 2-round trail cores $\xrightarrow{\chi} a^1 \xrightarrow{\lambda} b^1 \xrightarrow{\chi}$ with $\mathrm{w}_{\mathrm{rev}}(a^1) < 4$ in the forward direction, there exist no 4-round trail core up to the weight 60. Therefore, given a 4-round trail core $\tilde{Q}_4$ with weight $T_4 \leq 60$, the following condition holds:

$$\mathrm{w}_{\mathrm{r}}(a^2) + \mathrm{w}_{\mathrm{r}}(b^2) + \mathrm{w}_{\mathrm{r}}(b^3) \leq T_4 - 4\,.$$

Based on this observation, if we target $T_4 \leq 60$ then, we can start from 3-round trail cores up to $T_4 - 4$ that again makes the search space smaller.

Differential trail bound for 5 rounds. Any 5-round trail core with weight smaller than T_5 has at least a 4-round sub-trail core of weight smaller than $T_5 - 2$ due to the fact that the weight over one round is at least 2. Since there is no 4-round trail core with weight up to 57, we conclude there exists no 5-round trail core up to the weight $57 + 2 = 59$.

However, from our search on 4 rounds we know there is no 4-round trail core up to weight 60 such that $\mathrm{w}_{\mathrm{rev}}(a^1) < 4$. It means if we extend these 4-round trails by one round in the forward direction, there is no 5-round trail core up to weight $60 + 2$ such that $\mathrm{w}_{\mathrm{rev}}(a^1) < 4$. The same holds for the backward extension when we conclude there is no 5-round trail core up to weight 62 such that and $\mathrm{w}_{\mathrm{r}}(b^4) < 5$ since there exists no 4-round trail core up to weight 60 when $\mathrm{w}_{\mathrm{r}}(b^4) < 5$. It means, for all 5-round trail cores up to weight 62, $\mathrm{w}_{\mathrm{r}}(b^4) > \mathrm{w}_{\mathrm{rev}}(a^1) \geq 4$.

Since we scanned the space of all 4-round trail cores up to weight 57 and there is no trail up to this weight then $\mathrm{w}(\tilde{Q}_5) \geq 58 + 4 = 62$. This result improves the known lower bound over 5 rounds.

Differential trail bound for 6 rounds. In each 6-round trail core with weight smaller than T_6, either the first or the last 3-round sub-trail core should weigh $\lfloor T_6/2 \rfloor$ or less. We extended all 3-round trail cores up to weight 38 in both forward and backward directions by three rounds to scan the space of all 6-round trail cores up to weight 77. Since there exist no 6-round trail with weight 77, the lower bound on the weight of any 6-round trail is 78, which improves the known lower bound.

Differential trail bound for 7 rounds. Any 7-round trail core with weight smaller than T_7 has at least a 6-round sub-trail core of weight smaller than $T_7 - 2$. This follows from the fact that the weight over one round is at least 2. Hence, the extension of all 6-round trail cores $\tilde{Q}_6$ with $\mathrm{w}(\tilde{Q}_6) \leq T_6$ by one round allows to scan the whole space of 7-round trail cores $\tilde{Q}_7$ with $\mathrm{w}(\tilde{Q}_7) \leq T_6 + 2$. Since there exists no 6-round trail core with weight 77, then there exists no 7-round trail with weight 79, which implies that the lower bound on the weight of 7-round trails is at least 80. This result improves the previously known lower bound.

Differential trail bound for 8 rounds. In each 8-round trail core with weight smaller than 115, either the first or the last 4-round sub-trail core should weigh smaller than or equal to $\lfloor 115/2 \rfloor = 57$. Since there is no 4-round trail core with weight smaller than 58, then there is no 8-round trail core with weight smaller than 115, which implies that the lower bound on the weight of 8-round trails is at least 116. Equivalently, the upper bound on the DP of 8-round trails is at most 2^{-116} that improves the given bound in [DMMR20] (2^{-98}) by a factor 2^{18}.

Related work. We conclude by briefly discussing the relation of our work with the one recently proposed by Song et al. in [STSH21]. First of all, in there, authors propose several attacks on different primitives of Subterranean based on one-round differentials, on the contrary, we investigated 8 rounds of Subterranean.

Besides this, in [STSH21], authors pointed out that the χ function in Subterranean can be re-written as the non-linear layer of SIMON [BSS+13], a family of lightweight block ciphers proposed by Beaulieu et al. Its round function contains $(x \lll \alpha) \cdot (x \lll \beta) + (x \lll \gamma)$, where $x \lll i$ indicates the cyclic left shift over i bits, while the χ function in Subterranean can be re-written as $x + (x \ggg 1) \cdot (x \ggg 2) + (x \ggg 2)$ where $x \ggg i$ indicates the cyclic right shift over i bits. This suggests that the techniques for searching differential trails of SIMON can be potentially applied to Subterranean as well. Even if we do not exclude it, we point out that SIMON is a Feistel scheme, which means its round function is an involution and the same round function is used both in the forward and in the

backward direction. As a result in SIMON, extending trails in both directions is possible without further investigation on the backward direction or lower bounding the weight of the inverse of the non-linear layer as in [LLW17]. But in the case of SUBTERRANEAN, to efficiently extend trails in the backward direction, it is crucial to lower bound the weight of the inverse of χ.

Therefore, the techniques for differential trail search in SIMON cannot be easily applied to 8 blank rounds of SUBTERRANEAN. In addition, providing a method to compute a lower bound on the weight of the inverse of χ is crucial for scanning larger space of 8-round differential trails.

Acknowledgments

Joan Daemen, Lorenzo Grassi and Alireza Mehrdad are supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA. Silvia Mella is 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*.

References

[AZ21] Mark D. Aagaard and Nusa Zidaric. ASIC Benchmarking of Round 2 Candidates in the NIST Lightweight Cryptography Standardization Process. Cryptology ePrint Archive, Report 2021/049, 2021. https://ia.cr/2021/049.

[BDKA21] Nicolas Bordes, Joan Daemen, Daniël Kuijsters, and Gilles Van Assche. Thinking Outside the Superbox. In *Advances in Cryptology - CRYPTO 2021*, volume 12827 of *LNCS*, pages 337–367. Springer, 2021.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The keccak reference, 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.

[BS90] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *Advances in Cryptology - CRYPTO 1990*, volume 537 of *LNCS*, pages 2–21. Springer, 1990.

[BS93] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. *IACR Cryptol. ePrint Arch.*, page 404, 2013.

[CDGP93] Luc J. M. Claesen, Joan Daemen, Mark Genoe, and G. Peeters. Subterranean: A 600 Mbit/Sec Cryptographic VLSI Chip. In *Proceedings 1993 International Conference on Computer Design – ICCD 1993*, pages 610–613. IEEE Computer Society, 1993.

[CFG+18] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jian Guo, Jérémy Jean, Jean-René Reinhard, and Ling Song. Key-Recovery Attacks on Full Kravatte. *IACR Trans. Symmetric Cryptol.*, 2018(1):5–28, 2018.

[CG20] Tingting Cui and Lorenzo Grassi. Algebraic Key-Recovery Attacks on Reduced-Round Xoofff. In *Selected Areas in Cryptography - SAC 2020*, volume 12804 of *LNCS*, pages 171–197. Springer, 2020.

[DA12] Joan Daemen and Gilles Van Assche. Differential Propagation Analysis of Keccak. In *Fast Software Encryption - FSE 2012*, volume 7549 of *LNCS*, pages 422–441. Springer, 2012.

[Dae95] Joan Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis.* K.U.Leuven, 1995. http://jda.noekeon.org/.

[DEMS14] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2, 2014. https://ascon.iaik.tugraz.at/.

[DGV91] Joan Daemen, René Govaerts, and Joos Vandewalle. A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgård's One-Way Function Based on a Cellular Automaton. In *Advances in Cryptology - ASIACRYPT 1991*, volume 739 of *LNCS*, pages 82–96. Springer, 1991.

[DHAK18] 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.

[DMM21] Joan Daemen, Alireza Mehrdad, and Silvia Mella. Computing the Distribution of Differentials over the Non-linear Mapping χ. In *Security, Privacy, and Applied Cryptography Engineering - SPACE 2021*, volume 13162 of *LNCS*, pages 3–21. Springer, 2021.

[DMMR20] Joan Daemen, Pedro Maat Costa Massolino, Alireza Mehrdad, and Yann Rotella. The Subterranean 2.0 Cipher Suite. *IACR Trans. Symmetric Cryptol.*, 2020(S1):262–294, 2020.

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

[DPVAR00] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie proposal: NOEKEON. In *First Open NESSIE Workshop*, pages 213–230, 2000.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Springer, 2002.

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299. Springer, 2009.

[Hei11] Ethan Heilman. Restoring the Differential Resistance of MD6 . Cryptology ePrint Archive, Report 2011/374, 2011. https://ia.cr/2011/374.

[Knu94] Lars R. Knudsen. Truncated and Higher Order Differentials. In *Fast Software Encryption - FSE 1994*, volume 1008 of *LNCS*, pages 196–211. Springer, 1994.

[KPC20] Mustafa Khairallah, Thomas Peyrin, and Anupam Chattopadhyay. Preliminary Hardware Benchmarking of a Group of Round 2 NIST Lightweight AEAD Candidates. Cryptology ePrint Archive, Report 2020/1459, 2020. https://ia.cr/2020/1459.

[LLW17] Zhengbin Liu, Yongqiang Li, and Mingsheng Wang. Optimal differential trails in simon-like ciphers. *IACR Trans. Symmetric Cryptol.*, 2017(1):358–379, 2017.

[MDA17] 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.

[MHN+20] Kamyar Mohajerani, Richard Haeussler, Rishub Nagpal, Farnoud Farahmand, Abubakr Abdulgadir, Jens-Peter Kaps, and Kris Gaj. FPGA Benchmarking of Round 2 Candidates in the NIST Lightweight Cryptography Standardization Process: Methodology, Metrics, Tools, and Results. Cryptology ePrint Archive, Report 2020/1207, 2020. https://ia.cr/2020/1207.

[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.

[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.

[STSH21] Ling Song, Yi Tu, Danping Shi, and Lei Hu. Security analysis of subterranean 2.0. *Des. Codes Cryptogr.*, 89(8):1875–1905, 2021.

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

A C++ program has been written in order to scan the space of all 8-round trail cores of Subterranean up to weight 115 by generating all 4-round trail cores up to weight 57. This program, along with a list of all 3-round trail cores up to weight 40, is available at

https://github.com/Subterranean2/DifferentialTrailSearch

A Generate the affine subspace $\mathcal{A}(b)$

In the following, we show how to construct offset and basis vectors of the affine subspace $\mathcal{A}(b)$ with *minimum* Hamming weight using local rules based on the landscape of bits b_i's.

To be able to specify a specific offset, we introduce the concept of "canonical offset".

Definition 20 (Canonical offset)**.** Given the subspace $\mathcal{A}(b) \subseteq \mathbb{F}_2^\ell$, we define the canonical offset $c \in \mathbb{F}_2^\ell$ such that:

Algorithm 3 Generating canonical offset and basis vectors.

Input: size of the state ℓ and $b \in \mathbb{F}_2^\ell$
Output: Canonical offset and basis vectors with minimum Hamming weight.

if $(b = (1)^\ell)$ **then**
  if (ℓ even) **then**
    $c = (0)^\ell$
  else
    $c = \mathsf{e}_0$
  for all $i \in \{0, 1 \dots n-1\}$ **do**
    $v^{(i)} = \mathsf{e}_i + \mathsf{e}_{i+1}$
else
  for all $i \in \{0, 1 \dots n-1\}$ **do**
    if $(b_i \in \{\underline{1}00, \underline{1}101\})$ **then**
      $c_i = 1$
    else
      $c_i = 0$
    if $(b_i \in \{\underline{1}00, \underline{1}1, 0\underline{0}1\})$ **then**
      $v^{(j)} = \mathsf{e}_{i-1}$
      $j \leftarrow j + 1$
    else if $(b_i \in \underline{1}01)$ **then**
      $v^{(j)} = \mathsf{e}_{i-1} + \mathsf{e}_i$
      $j \leftarrow j + 1$

1. it has the minimum Hamming weight among all possible offsets, that is $||c|| \leq ||o||$ for each offset $o \in \mathcal{A}(b)$;
2. let $\{i_1^{(o)}, \dots, i_{||o||}^{(o)}\}$ (with $i_j^{(o)} < i_{j+1}^{(o)}$ for each $j \in \{1, \dots, ||o|| - 1\}$) be the position(s) of the active bit(s) in $o \in \mathbb{F}_2^\ell$. If $||c|| \geq 1$, then

$$\forall o \in \mathcal{A}(b) \text{ such that } ||c|| = ||o|| \text{ and } \forall j \in \{1, \dots, ||c||\} : \qquad i_j^{(c)} \leq i_j^{(o)}.$$

The way to generate a valid offset as $o = \chi_\ell(b)$, and also basis vectors with the minimum Hamming weight for a given input difference of χ_5 is already discussed in [BDPA11, Sect. 2.3]. Here, we generalize the results and provide a new method to compute the canonical offset. To this end, we present rules that allow to build the canonical offset and basis vectors with minimum Hamming weight first for a not fully active input difference and then for a fully active input difference. Such rules are summarized in Tables 5 to 7. Algorithm 3 also presents a pseudo-code that takes in input a difference b and returns the canonical offset and minimum Hamming weight basis vectors of the affine subspace $\mathcal{A}(b)$.

A.1 Generating offset and basis for a not fully active input difference

For input differences $b \in \mathbb{F}_2^\ell \setminus \{(1)^\ell\}$, we locally generate the canonical offset and basis vectors.

Proposition 7. *Given a not fully active difference $b \in \mathbb{F}_2^\ell \setminus \{(1)^\ell\}$ at the input of χ_ℓ, $c_i = 1$ only if $b_i \in \{\underline{1}00, \underline{1}101\}$.*

Proof. Since χ_ℓ is translation-invariant, we only prove it for $i = 0$. Here, we should find bits of the output that are always active independently of the initial state x_i's since they have no fixed value. From Eq. (5):

$$a_0 = b_0 + b_1 \cdot b_2 + b_2 + b_2 \cdot x_1 + b_1 \cdot x_2. \tag{9}$$

Table 5: The positions of active bits in c for $b \in \mathbb{F}_2^\ell \setminus \{(1)^\ell\}$.

$b_i \in$	$\underline{1}00$	$\underline{1}101$
c_i	1	1

Table 6: The values of basis vectors $v^{(j)}$ for b_i.

	$b \in \mathbb{F}_2^\ell \setminus \{(1)^\ell\}$				$b = (1)^\ell$
$b_i \in$	$\underline{1}01$	$\underline{1}00$	$\underline{1}1$	$0\underline{0}1$	$\underline{1}$
$v^{(j)}$	$\mathsf{e}_{i-1} + \mathsf{e}_i$	e_{i-1}	e_{i-1}	e_{i-1}	$\mathsf{e}_i + \mathsf{e}_{i+1}$

- If $b_0 \in \underline{1}00$, then $a_0 = b_0 = 1$ independently of x.
- If $b_0 \in \underline{1}101$, then:

$$a_0 + a_1 = b_0 + b_2 \cdot (1 + x_1 + x_3) + (b_1 + b_3) \cdot (x_2 + b_2 + 1) \tag{10}$$

In Eq. (10), if $b_1 + b_3 = 0$ and $b_2 = 0$, then $a_0 + a_1$ is independent of x. It implies that $a_0 + a_1 = 1$ only if $b_0 = 1$. Since we already analyzed the case when $b_1 = b_2 = b_3 = 0$ ($b_0 \in \underline{1}00$), here we focus on the case $b_1 = b_3 = 1$ ($b_0 \in \underline{1}101$). In this last case, $a_0 + a_1 = 1$ or equivalently, either a_0 or a_1 is active in this case. Since the position of the active bits of the canonical offset is the minimum, then $a_0 = 1$. □

Now, we present the rules to generate basis vectors with the minimum Hamming weight for $b \in \mathbb{F}_2^\ell \setminus \{(1)^\ell\}$, that are also summarized in Table 6. We highlight that the method to generate basis vectors for $\ell = 5$ is presented in [BDPA11, Sect. 2.3]. We generalized their method for any arbitrary ℓ (even or odd) and we provided a proof for it in Proposition 8.

Proposition 8. *Given a not fully active difference $b \in \mathbb{F}_2^\ell \setminus \{(1)^\ell\}$ at the input of χ_ℓ:*

1. $v^{(j)} = \mathsf{e}_{i-1} + \mathsf{e}_i$ *if* $b_i \in \underline{1}01$,
2. $v^{(j)} = \mathsf{e}_{i-1}$ *if* $b_i \in \{\underline{1}00, \underline{1}1, 0\underline{0}1\}$.

Proof. Working as in the previous proof and since the value of b is given, we look for a_i's that depend on x. We analyze the conditions separately:

1. If $b_0 \in \underline{1}01$ then from Eq. (5) we have $a_{-1} = b_{-1} + x_1$ and $a_0 = x_1$. Since both a_0 and a_1 depend on the same bit of the input namely x_1, the basis vector has two active bits at a_{-1} and a_0.
2. According to Eq. (5), if at least one of the b_0 or b_1 are 1, then a_{-1} depends on x. Since we have already considered the case when there is a (101)-string in the landscape of b, we do not consider it again. Therefore, to have at least one active bit at $b_0 \| b_1$ and no (101)-string, we have $b_0 \in \{\underline{1}00, \underline{1}1, 0\underline{0}1\}$. In this case, base vector has a single active bit at a_{-1}. □

A.2 Generating offset and basis for a fully active input difference

Next, we treat the case of fully active input difference.

Proposition 9. *Given a fully active difference $b = (1)^\ell$ at the input of χ_ℓ:*

- $c = (0)^\ell$ *if ℓ is even,*

Table 7: The offset value for $b = (1)^\ell$.

ℓ	even	odd
c	$(0)^\ell$	e_0

- $c = \mathsf{e}_0$ *if* ℓ *is odd.*

Proof. From Eq. (5):

$$\sum_{i=0}^{\ell-1} a_i = (1 + x_1 + x_2) + (1 + x_2 + x_3) + \cdots + (1 + x_0 + x_1) = \begin{cases} 0, & \text{if } \ell \text{ is even} \\ 1, & \text{otherwise.} \end{cases} \quad \square$$

By Proposition 9, it follows that collisions can occur for $b = (1)^\ell$ and ℓ even, that means χ_ℓ is not invertible in this case (as we already know).

Proposition 10. *Given a fully active difference* $b = (1)^\ell$ *at the input of* χ_ℓ, $\forall i \in \{0, 1, \ldots, n-1\} : v^{(i)} = \mathsf{e}_i + \mathsf{e}_{i+1}$.

Proof. From Eq. (5) $a_0 = 1 + x_1 + x_2$ and $a_1 = 1 + x_2 + x_3$ so both a_0 and a_1 depend on x_2. Since χ_ℓ is translation-invariant, $\forall i \in \{0, 1, \ldots, n-1\}$, a_i and a_{i+1} depend on x_{i+2}. $\square$

B Proofs

B.1 Proof of Proposition 3

The rank of a matrix corresponds to the maximum number of linearly independent rows/columns. Let us denote by r_i $(0 \leq i < n)$ the i-th row of the matrix $\mathrm{M}^{(b)}$ given in Eq. (6). We say a row is active if it has at least one active bit. By definition the rank of matrix $\mathrm{M}^{(b)}$ for $b \neq \mathbf{1}$ is the number of unequal active rows.

Since χ_ℓ is translation-invariant, we consider a difference with a single active bit b_0 at the input of χ_ℓ with $\ell \gg 1$. In such a case, both $r_{\ell-2}$ and $r_{\ell-1}$ are active and $r_{\ell-2} \neq r_{\ell-1}$. Since we have two unequal active rows, the weight is 2. Adding another active bit to the b leads to one of the following cases:

- $b_0 = b_1 = 1$: the rows $r_{\ell-2}$, $r_{\ell-1}$ and r_0 are active and unequal. In this case, the weight is 3. Therefore, adding an active bit at b_1 implies adding 1 to the weight;
- $b_0 = b_2 = 1$: the rows $r_{\ell-2}$, $r_{\ell-1}$, r_0 and r_1 are active. Since $r_{\ell-1} = r_0$, the weight is $4 - 1 = 3$. So, adding an active bit at b_2 implies adding 1 to the weight;
- $b_0 = b_i = 1$ for $3 \leq i < n - 2$: the rows $r_{\ell-2}$, $r_{\ell-1}$, r_{i-2} and r_{i-1} are active and unequal. In this case, the weight is 4. Hence, adding an active bit at b_i when $3 \leq i < n - 2$ implies adding 2 to the weight;
- $b_0 = b_{\ell-2} = 1$: the rows $r_{\ell-4}$, $r_{\ell-3}$, $r_{\ell-2}$ and $r_{\ell-1}$ are active, and $r_{\ell-3} = r_{\ell-2}$ thus, the weight is $4 - 1 = 3$. So, adding an active bit at $b_{\ell-2}$ increases the weight by 1;
- $b_0 = b_{\ell-1} = 1$: then rows $r_{\ell-3}$, $r_{\ell-2}$ and $r_{\ell-1}$ are active and unequal. In this case, the weight is 3. So, adding an active bit at $b_{\ell-1}$ implies adding 1 to the weight.

As a result, if an active bit has a distance of at least two bits with other active bits, it adds 2 to the rank of $\mathrm{M}^{(b)}$. Hence, whenever an active bit forms a (001)-string (or (100)-string), it adds two to the weight. Otherwise, each active bit increases the weight by one.

B.2 Proof of Proposition 4

The weight of a given input difference b is the rank of matrix $\mathrm{M}^{(b)}$. After setting b to $(1)^\ell$ and row reduction, the rank of the matrix $\mathrm{M}^{(b)}$ is $\ell - 1$.

B.3 Proof of Corollary 1

Regarding the minimum value and based on Proposition 3, if b has one active bit, then $||b|| + \#_{001}(b) = 1 + 1 = 2$. If b has two active bits, then $1 \leq \#_{001}(b) \leq 2$, which implies that $||b|| + \#_{001}(b) \geq 2 + 1 = 3$. If b has at least three active bits, then $||b|| + \#_{001}(b) \geq 3$.

Regarding the maximum value, it can be achieved in the following cases:

- $b = \mathbf{1}$;
- $||b|| = n - 1$ (and so $\#_{001}(b) = 0$);
- $||b|| = n - 2$ and $\#_{001}(b) = 1$ (that is, the two passive bits in b are consecutive).

In all other cases, $\mathrm{w_r}(b) \leq n-2$. Note that $\#_{001}(b) = x \geq 2$ if at least $2x$ bits in b are equal to zero, which implies that $||b|| \leq n-2x$, that is $||b||+\#_{001}(b) \leq n-2x+x = n-x \leq n-2$.