Differential propagation analysis of Keccak

Joan Daemen and Gilles Van Assche

STMicroelectronics

Abstract. In this paper we introduce new concepts that help read and understand low-weight differential trails in KECCAK. We then propose efficient techniques to exhaustively generate all 3-round trails in its largest permutation below a given weight. This allows us to prove that any 6-round differential trail in KECCAK-f[1600] has weight at least 74. In the worst-case diffusion scenario where the mixing layer acts as the identity, we refine the lower bound to 82 by systematically constructing trails using a specific representation of states.

Keywords: cryptographic hash function, KECCAK, differential cryptanalysis, computer-aided proof

1 Introduction

The goal of cryptanalysis is to assess the security of cryptographic primitives. Finding attacks or properties not present in ideal instances typically contributes to the cryptanalysis of a given primitive. Building upon previous results, attacks can be improved over time, possibly up to a point where the security of the primitive is severely questioned.

In contrast, cryptanalysis can also benefit from positive results that exclude classes of attacks, thereby allowing research to focus on potentially weaker aspects of the primitive. Interestingly, weaknesses are sometimes revealed by challenging the assumptions underlying positive results. Nevertheless, both attacks and positive results can be improved over time and together contribute to the understanding and estimation of the security of a primitive by narrowing the gap between what is possible to attack and what is not.

Differential cryptanalysis (DC) is a discipline that attempts to find and exploit predictable difference propagation patterns to break iterative cryptographic primitives [6]. For ciphers, this typically means key retrieval, while for hash functions, this is the generation of collisions or of second preimages. The basic version makes use of differential trails (also called characteristics or differential paths) that consist of a sequence of differences through the rounds of the primitive. Given such a trail, one can estimate its differential probability (DP), namely, the fraction of all possible input pairs with the initial trail difference that also exhibit all intermediate and final difference when going through the rounds.

A more natural way to characterize the power of trails in unkeyed primitives is by their weight w. In general the weight of a trail is is the sum of the weight

of its round differentials, where the latter is the negative of its binary logarithm. For many round functions, including that of KECCAK-f and Rijndael, the weight equals the number of binary equations that a pair must satisfy to follow the specified differences. Assuming that these conditions are independent, the weight of the trail relates to its DP as $\mathrm{DP} = 2^{-w}$ and exploiting such a trail becomes harder as the weight increases. For a primitive with, say, b input and output bits, the number of pairs that satisfy these conditions is then 2^{b-w}. The assumption of independence does not always apply. For instance, a trail with $w > b$ implies redundant or contradictory conditions on pairs, for which satisfying pairs may or may not exist. Another example where this independence assumption breaks down are the plateau trails that occur in Rijndael [10]. These trails, with weight starting from $w = 30$ for 2 rounds, have a DP equal to 2^{z-w} with $z > 0$ for a fraction 2^{-z} of the keys and zero for the remaining part. In general, they occur in primitives with strong alignment [3] and a mixing layer based on maximum-distance separable (MDS) codes.

In the scope of DC, positive results can be established by finding a lower bound on the weight of any trail over a specified number of rounds. For instance, the structure of Rijndael and the properties of its diffusion operations allow to analytically derive such lower bounds [9]. Such results can be transposed to the permutations underlying the hash function Grøstl [13]. Other examples include a lower bound on the number of active S-boxes in JH [18] or computer-aided proofs on the weight of trails in NOEKEON [8] and on the minimum number of active AND gates in MD6 [17,14].

KECCAK is a sponge function submitted to the SHA-3 contest [16,4,2]. Recently, new results were published on the differential resistance of this function and among those heuristic techniques were proposed to build low-weight differential trails [12,15]. These gave the currently best trails for 3, 4 and 5 rounds of the underlying permutation KECCAK-f[1600]. In particular, Duc et al. found a trail of weight 32 for 3 rounds, and this motivated us to systematically investigate whether trails of lower weight exist. Also, there are some similarities between KECCAK and MD6, but unlike MD6, the permutation used in the proposed SHA-3 candidate KECCAK has no significant lower bounds on the weight of trails. So the philosophy behind [17,14] was another source of inspiration and motivation for our research.

Lower bounds on symmetric trails were already proven in [4]. They provide lower bounds with weight above the permutation width on KECCAK-f[25] to KECCAK-f[200] but only partial bounds in the case of KECCAK-f[1600]. Thanks to the Matryoshka structure [4], a lower bound W on trails in KECCAK-f[$25w$] implies a lower bound $W' = W\frac{w'}{w}$ on w-symmetric trail in KECCAK-f[$25w'$] for $w' > w$. These are summarized in Table 1.

In this paper, we report on techniques to efficiently generate all the trails in KECCAK-f[1600] up to a given weight. We implemented these techniques in a computer program, which allowed us at this point to completely scan the space of 3-round differential trails up to weight 36. This confirmed that the trail found by Duc et al. has minimum weight and allowed us to demonstrate that there are

w	**Lower bound for** KECCAK-$f[25w]$	**Lower bound for** KECCAK-$f[1600]$	
1	30 per 5 rounds	1920 per 5 rounds	tight
2	54 per 6 rounds	1728 per 6 rounds	tight
4	146 per 16 rounds	2336 per 16 rounds	non-tight
8	206 per 18 rounds	1648 per 18 rounds	non-tight

Table 1. Lower bounds above the permutation width on 1- to 8-symmetric trails [4].

no 6-round trails with weight below 74. These results are summarized in Table 2. The source code of the classes and methods used in this paper is available in the KECCAKTOOLS package [5].

As a by-product of this trail search, this paper proposes new techniques to relate the properties of the θ mapping in KECCAK to the weight of differential trails. In the worst-case diffusion scenario where θ acts as the identity, we build upon the results of [4] and [15] to systematically construct so-called in-kernel trails using an efficient representation of states.

Rounds	**Lower bound**	**Best known**
3	32 (this work)	32 [12]
4	-	134 [7]
5	-	510 [15]
6	74 (this work)	1360 [4]
24	296 (this work)	-

Table 2. Weight of differential trails in KECCAK-$f[1600]$.

Further discussions on how to exploit differential trails in KECCAK can be found in [3]. Also, the attacks in [11] combine algebraic techniques with a differential trail.

The paper is organized as follows. In Section 2, we recall the structure of KECCAK and mappings inside its round function. Section 3 focuses on how to represent and extend the differential trails of KECCAK. Section 4 sets up the overall strategy and Section 5 introduces a basic trail generation technique. The advanced techniques are covered in Sections 6 and 7, which address two complementary cases. Finally, Section 8 extends the results from 3 to 6 rounds.

2 KECCAK

KECCAK combines the sponge construction with a set of seven permutations denoted KECCAK-$f[b]$, with b ranging from 25 to 1600 bits [1,4]. In this pa-

per, we concentrate on the permutation used in the SHA-3 submission, namely, KECCAK-$f[1600]$.

The state of KECCAK-$f[1600]$ is organized as a set of $5 \times 5 \times 64$ bits with (x, y, z) coordinates. The coordinates are always considered modulo 5 for x and y and modulo 64 for z. A *row* is a set of 5 bits with given (y, z) coordinates, a *column* is a set of 5 bits with given (x, z) coordinates and a *slice* is a set of 25 bits with given z coordinate.

The round function of KECCAK-$f[1600]$ consists of the following steps, which are only briefly summarized here. For more details, we refer to the specifications [4].

- θ is a linear mixing layer, which adds a pattern that depends solely on the parity of the columns of the state. Its properties with respect to differential propagation will be detailed and exploited in Section 6.
- ρ and π displace bits without altering their value. Jointly, their effect is denoted by $(x, y, z) \xrightarrow{\pi \circ \rho} (x', y', z')$, with (x, y, z) a bit position before ρ and π and (x', y', z') its coordinates afterward.
- χ is a degree-2 non-linear mapping that processes each row independently. It can be seen as the application of a translation-invariant 5-bit S-box. The differential propagation properties will be detailed below.
- ι adds a round constant. As it has no effect on difference propagation, we will ignore it in the sequel.

3 Representing and extending trails

In general, for a function f with domain $\mathbb{Z}_2^b$, we define the *weight* of a differential (u', v') as

$$\mathrm{w}(u' \xrightarrow{f} v') = b - \log_2 \left| \{u \ : \ f(u) \oplus f(u \oplus u') = v'\} \right| .$$

If the argument of the logarithm is non-zero (i.e., the DP is non-zero), we say that u' and v' are *compatible*. Otherwise, the weight is undefined.

The weight of a trail is the sum of the weight of the differentials that compose this trail. In KECCAK-f, we specify differential trails with the differences before each round function. For clarity, we adopt a redundant description by also specifying the differences before and after the linear steps $\lambda = \pi \circ \rho \circ \theta$. An n-round trail is of the following form, where each b_i must be equal to $\lambda(a_i)$,

$$Q = a_0 \xrightarrow{\pi \circ \rho \circ \theta} b_0 \xrightarrow{\chi} a_1 \xrightarrow{\pi \circ \rho \circ \theta} b_1 \xrightarrow{\chi} a_2 \xrightarrow{\pi \circ \rho \circ \theta} \quad \ldots \quad \xrightarrow{\chi} a_n, \tag{1}$$

and has weight $\mathrm{w}(Q) = \sum_i \mathrm{w}(a_i \xrightarrow{\chi \circ \pi \circ \rho \circ \theta} a_{i+1})$. Since $b_i = \lambda(a_i)$, this expression simplifies to $\mathrm{w}(Q) = \sum_i \mathrm{w}(b_i \xrightarrow{\chi} a_{i+1})$.

3.1 Extending forward and trail prefixes

Given a trail as in (1), it is possible to characterize all states that are compatible with $b_n = \lambda(a_n)$ through χ and thus to find all $n+1$-round trails Q' that have Q as its leading part. This process is called *forward extension*.

The χ mapping has algebraic degree 2 and, for a given input difference b_n, the space of compatible output differences forms a linear affine variety $\mathcal{A}(b_n)$ with $|\mathcal{A}(b_n)|$ elements [4]. For a compatible a_{n+1}, the weight $\mathrm{w}(b_n \xrightarrow{\chi} a_{n+1})$ depends only on b_n and is equal to $\mathrm{w}(b_n) \triangleq \log_2 |\mathcal{A}(b_n)|$, with the symbol $\triangleq$ denoting a definition. As χ operates on each row independently, the weight $\mathrm{w}(b)$ can also be computed on each row independently and summed. To construct $\mathcal{A}(b)$, the bases resulting from each active row are gathered. Table 3 displays offsets and bases for the affine spaces of all single-row differences.

	forward propagation							
Difference	offset	base elements				$\mathrm{w}(\cdot)$	$\mathrm{w}^{\mathrm{rev}}(\cdot)$	$\|\| \cdot \|\|$
00000	00000					0	0	0
00001	00001	00010	00100			2	2	1
00011	00001	00010	00100	01000		3	2	2
00101	00001	00010	01100	10000		3	2	2
10101	00001	00010	01100	10001		3	3	3
00111	00001	00010	00100	01000	10000	4	2	3
01111	00001	00011	00100	01000	10000	4	3	4
11111	00001	00011	00110	01100	11000	4	3	5

Table 3. Space of possible output differences, weight, minimum reverse weight and Hamming weight of all row differences, up to cyclic shifts.

As a consequence, the weight of a n-round trail Q is $\mathrm{w}(Q) = \sum_{i=0}^{n-1} \mathrm{w}(b_i)$ and depends only on the n-tuple $(b_0, \ldots, b_{n-1})$. We call the latter a *trail prefix*. All n-round trails sharing this trail prefix and with a_n compatible with b_{n-1} through χ have the same weight.

3.2 Extending backward and trail cores

Similarly, given a trail as in (1), it is possible to construct all states that are compatible with a_0 through χ^{-1} and thus to find all $n+1$-round trails Q' that have Q as its trailing part. This process is called *backward extension*. In contrast to χ, its inverse has algebraic degree 3 and the space of compatible differences is not an affine variety in general. Yet, compatible values can be identified per active row and combined.

For a difference a after χ, we define the *minimum reverse weight* $\mathrm{w}^{\mathrm{rev}}(a)$ as the minimum weight over all compatible b before χ. Namely,

$$\mathrm{w}^{\mathrm{rev}}(a) \triangleq \min_{b \,:\, a \in \mathcal{A}(b)} \mathrm{w}(b).$$

Like for the restriction weight, the minimum reverse weight $\mathrm{w}^{\mathrm{rev}}(a)$ can be computed on each row independently and summed. Values are also shown in Table 3.

Given a $n-1$-round trail prefix $Q = (b_1, \ldots, b_{n-1})$, it is easy to construct a difference b_0 such that the trail prefix $Q' = b_0||Q$ has weight given by $\mathrm{w}(Q') = \mathrm{w}(Q) + \mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1))$. This is the smallest possible weight a n-round trail can have with Q as its trailing part. It follows that a sequence of $n-1$ state values $\tilde{Q} = (b_1, \ldots, b_{n-1})$ defines a set of n-round trails with a weight at least

$$\tilde{\mathrm{w}}(\tilde{Q}) \triangleq \mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) + \sum_{i=1}^{n-1} \mathrm{w}(b_i).$$

We denote the former by the term *trail core* and the latter by its weight. Note that a n-round trail core is determined by only $n-1$ states, although its weight takes n individual weights into account.

KeccakTools implements the representation of trails, trail prefixes and trail cores (see the `Trail` class), as well as the forward and backward extension (see the `KeccakFTrailExtension` class) [5].

4 Towards a bound for trails in Keccak-f[1600]

To find a lower bound on differential trail weights in Keccak-f[1600], our strategy is the following.

- First, we exhaustively generate all 3-round trails up to a given weight T_3. There exists a trail of weight 32 as found by Duc et al. [12]. So by scanning the space of trails up to weight $T_3 \geq 32$, we are sure to hit at least one trail and the trail with minimum weight yields a tight lower bound on 3-round trails.
- Second, we derive a lower bound, not necessarily tight, on the weight of 6-round trails by using the 3-round trails found. Any 6-round trail of weight $2T_3 + 1$ or less satisfies either $\mathrm{w}(b_0) + \mathrm{w}(b_1) + \mathrm{w}(b_2) \leq T_3$ or $\mathrm{w}(b_3) + \mathrm{w}(b_4) + \mathrm{w}(b_5) \leq T_3$. We thus use forward and backward extension from 3-round trails up to weight $2T_3+1$. If such trails are found, the one with the smallest weight defines the lower bound, which is naturally tight. Otherwise, this establishes a lower bound for the weight of 6-round trails to $2T_3 + 2$. In the latter case no trail with weight $2T_3 + 2$ is known so the bound is not necessarily tight.

The reason for targeting 3-round trails in the first phase is the following. The minimum weight of a 1-round trail is 2, with a single active bit in b_0. For the 24 rounds of Keccak-f[1600], this amounts to a lower bound of $24 \times 2 = 48$. Constructing a state a with only two active bits in the same column leads to 2-round trail core with weight 8. Hence, if we base ourselves only on 2-round trail, we reach a lower bound of $12 \times 8 = 96$. If the 3-round trail of weight 32 found by Duc et al. [12] has minimum weight, this would mean that a 24-round trail has weight at least $8 \times 32 = 256$. Also, 3-round trail cores can be constructed by taking into account conditions across one layer of χ. Generating

exhaustively trails of 4 rounds or more up to some weight would probably yield better bounds, but at the same time it is more difficult as several layers of χ must be dealt with. Instead, the two-step approach described above can take advantage of the exhaustive set of trails covered (i.e., all up to weight T_3) to derive a bound based on T_3 instead of on the minimum weight over 3 rounds.

4.1 Generating all 3-round trails up to a given weight

In our approach we generate all 3-round differential trails of the form

$$Q = a_0 \xrightarrow{\pi\circ\rho\circ\theta} b_0 \xrightarrow{\chi} a_1 \xrightarrow{\pi\circ\rho\circ\theta} b_1 \xrightarrow{\chi} a_2 \xrightarrow{\pi\circ\rho\circ\theta} b_2 \xrightarrow{\chi} a_3, \tag{2}$$

up to some weight limit $\mathrm{w}(Q) \leq T_3$. We call this the target space. We do this by searching for all trail cores (b_1, b_2) with weight below T_3. Each such trail core (b_1, b_2) thus represents a set 3-round trails of the form of Eq. (2) with weight not below that of its core. In the scope of this paper, we limited ourselves to $T_3 = 36$.

We covered the set of all 3-round trails up to weight T_3 in three sub-phases:

1. In Section 5, we start with all cores such that $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) \leq 7$, $\mathrm{w}(b_1) \leq 7$ or $\mathrm{w}(b_2) \leq 7$.
2. In Section 6, we generate all remaining cores, except where both a_1 and a_2 are in the kernel.
3. In Section 7, we finish by generating all cores where both a_1 and a_2 are in the kernel.

4.2 Too many states to generate and extend, even when exploiting symmetry

A way to generate all trails in the target space is to first generate all states up to a given weight and then do backward and forward extensions to obtain trail cores. If we define $T_1 \triangleq \lfloor \frac{T_3}{3} \rfloor$, then for $\tilde{\mathrm{w}}(b_1, b_2) \leq T_3$ either $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) \leq T_1$, $\mathrm{w}(b_1) \leq T_1$ or $\mathrm{w}(b_2) \leq T_1$. To cover the target space, we need to consider these cases:

- $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) \leq T_1$, so we have to generate all states a_1 with $\mathrm{w}^{\mathrm{rev}}(a_1) \leq T_1$, compute $b_1 = \lambda(a_1)$ and extend forward the 2-round trail cores (b_1) to get 3-round trail cores.
- $\mathrm{w}(b_1) \leq T_1$, so we have to generate all states b_1 with $\mathrm{w}(b_1) \leq T_1$ and extend forward the 2-round trail cores (b_1).
- $\mathrm{w}(b_2) \leq T_1$, so we have generate all states b_2 with $\mathrm{w}(b_2) \leq T_1$ and extend backward the 2-round trail cores (b_2).

Unfortunately, this brute-force strategy requires a high number of states to cover the whole space for an interesting target weight. E.g., if $T_3 = 36$, then $T_1 = 12$ and there are about $1.42 \times 10^{15} \approx 2^{50}$ states with weight up to 12 in Keccak-f[1600].

We can reduce this number by taking the z symmetry into account. Except for ι, which does not influence difference propagation, all the step mappings of KECCAK-f are invariant when translated along z. Hence, for each trail $Q = (b_0, b_1, \ldots, b_n)$ there exists a trail $Q' = (\mathrm{z}(b_0), \mathrm{z}(b_1), \ldots, \mathrm{z}(b_n))$ of same weight, with z the translation operator along the z axis. In the sequel, we will always consider trails up to translations in z. This reduces the search space by approximately a factor $w = 64$—not exactly a factor w because of states that are periodic in z. Yet, the number of states to extend forward and backward is still about 2^{44}.

5 Generating trails with a low number of active rows

In this section, we generate and extend states with weight up to $T_1' = 7$. This does not cover the whole target space with $T_3 = 36$ but the remaining portion of the target space is limited to trails with a more flat weight profile, i.e., they satisfy $\mathrm{w}(b_i) \geq T_1' + 1 = 8$ for all $i \in \{0, 1, 2\}$ and $\mathrm{w}(b_i) + \mathrm{w}(b_{i+1}) \leq T_2' = T_3 - (T_1' + 1) = 28$ for all $i \in \{0, 1\}$.

More specifically, in this phase we look at the number of active rows in order to generate all trail cores such that $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) \leq T_1'$, $\mathrm{w}(b_1) \leq T_1'$ or $\mathrm{w}(b_2) \leq T_1'$, for $T_1' = 7$. According to Table 3, each active row contributes for at least 2 to the weight. Hence,

$$\mathrm{w}(b) \geq 2\|b\|_{\mathrm{row}} \quad \text{and} \quad \mathrm{w}^{\mathrm{rev}}(b) \geq 2\|b\|_{\mathrm{row}},$$

and we can cover all the states up to weight 7 by generating all states with up to $\lfloor \frac{T_1'}{2} \rfloor = 3$ active rows.

This approach can be refined by looking at the number of active rows not only for one state but for two consecutive states. With χ, the minimum weight a round differential can have is 2. So, $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) \geq 2$ implies that $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_2)) + \mathrm{w}(b_2) \leq \mathrm{w}(b_1) + \mathrm{w}(b_2) \leq T_3 - 2 = 34$ and similarly $\mathrm{w}(b_2) \geq 2$ implies that $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1)) + \mathrm{w}(b_1) \leq T_3 - 2 = 34$. Hence,

$$\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_i)) + \mathrm{w}(b_i) \leq T_3 - 2 = 34 \;\Rightarrow\; \|\lambda^{-1}(b_i)\|_{\mathrm{row}} + \|b_i\|_{\mathrm{row}} \leq \left\lfloor \frac{T_3 - 2}{2} \right\rfloor = 17.$$

In practice, what we did was the following.

- Generate $\mathcal{B} = \{b \;:\; (\|b\|_{\mathrm{row}} \leq 3 \text{ or } \|\lambda^{-1}(b)\|_{\mathrm{row}} \leq 3) \text{ and } \|\lambda^{-1}(b)\|_{\mathrm{row}} + \|b\|_{\mathrm{row}} \leq 17\}$. This is done by first generating all states b with up to 3 active rows and filter on $\|\lambda^{-1}(b)\|_{\mathrm{row}}$, and then generate all states a with up to 3 active rows, compute $b = \lambda(a)$ and filter on $\|b\|_{\mathrm{row}}$.
- Do forward extension of all $b_1 \in \mathcal{B}$ and keep the cores $\tilde{Q} = (b_1, b_2)$ with $\tilde{\mathrm{w}}(\tilde{Q}) \leq T_3$.
- Do backward extension of all $b_2 \in \mathcal{B}$ and keep the cores $\tilde{Q} = (b_1, b_2)$ with $\tilde{\mathrm{w}}(\tilde{Q}) \leq T_3$.

We found a trail core (b_1, b_2) with $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b_1))+\mathrm{w}(b_1)+\mathrm{w}(b_2) = 4+4+24 = 32$ (see also Table 4). It contains the 3-round trail found by Duc et al. [12].

There are $\binom{320}{n}(31)^n$ states with n active rows. As this function grows very quickly, it was not reasonable to extend this search beyond 3 active rows.

The generation of trail cores based on a small number of active rows is implemented in the `KeccakFTrailCoreRows` class [5].

6 Generating trails using the properties of θ

To investigate the remaining part of the target space, we look at the properties of states a with respect to θ, and specifically the parity of its columns, to limit the weight of two-round trails. An important parameter to classify the states a is their column parity, so as to study states in sets of parities. From the column parity, we derive the θ-gap, defined below. With θ-gap g, the effect of θ is to flip $10g$ bits. There are thus at least $10g$ active bits, each either in a or in $\theta(a)$. So, the higher the θ-gap the higher $\mathrm{w}^{\mathrm{rev}}(a) + \mathrm{w}(\lambda(a))$ is likely to be. We can efficiently compute a lower bound for $\mathrm{w}^{\mathrm{rev}}(a) + \mathrm{w}(\lambda(a))$ over all a with a given parity. For the target weights considered in this paper, this allows us to limit the states to consider to those with a parity belonging to a mere handful of values.

We then use the generated states a are to build trail cores by forward and backward extension. As the θ-gap increases, the number of states a to consider decreases since more states a can immediately be excluded. An important case is when all the columns of a have even parity, i.e., a is in the kernel. In this case, the θ-gap is zero and a high number of states must be generated and extended. For this reason, this section focuses only the case where either a_1 or a_2 is not in the kernel. The complementary case is covered in Section 7.

6.1 Properties of θ

As θ is a linear function, its properties are the same whether applied on a state absolute value or on a difference, so we just write "value". The following definitions are from [4].

The *column parity* (or *parity* for short) $P(a)$ of a value a is defined as the parity of the columns of a, namely $P(a)[x][z] = \sum_y a[x][y][z]$. A column is *even* (resp. *odd*) if its parity is 0 (resp. 1). The parity can also be defined on a slice, namely $P(a_z)[x] = \sum_y a[x][y][z]$. When the parity of a state or of a slice is zero (i.e., all its columns are even), we say it is in the *column-parity kernel* (or *kernel* for short).

The mapping θ consists in adding a pattern to the state, which we call the θ-*effect*. The θ-effect of a value a is $E(a)[x][z] = P(a)[x-1][z]+P(a)[x+1][z-1]$. For a fixed θ-effect $e[x][z]$, θ comes down to adding the y-symmetric pattern $e[x][y][z] \triangleq e[x][z](\forall y)$. So θ depends only on column parities and always affects columns symmetrically in y.

A column of coordinates (x, z) is *affected* iff $E(a)[x][z] = 1$; otherwise, it is *unaffected*. Note that the θ-effect always has an even Hamming weight so the number of affected columns is even.

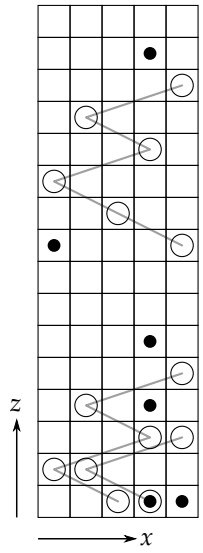

Fig. 1. Example of parity pattern. Each square represents a column. An odd column contains a circle, while an affected column is denoted by a dot. A column can be both odd and affected. The odd columns of a run are connected with a line. The affected columns due to a run are located at the right (resp. top left) of the start (resp. end) column of the run.

The *θ-gap* is defined as the Hamming weight of the θ-effect divided by two. Hence, if the θ-gap of a value at the input of θ is g, the number of affected columns is $2g$ and applying θ to it results in $10g$ bits being flipped.

We have introduced the θ-gap via the θ-effect, but it can be defined directly using the parity itself. For this we introduce an alternative, single-dimensional, representation of a parity $p[x][z]$. We map the (x, z) coordinates to a single coordinate t as $t \rightarrow (x, z) = (-2t, t)$ and denote the result by $p[t]$. In this representation a *run* is a sequence of ones delimited by zeroes. As illustrated on Figure 1, each run induces two affected columns. First, if it starts in coordinates (x, z), it implies an affected column in its right neighbor $(x+1, z)$. And if it ends in (x', z') it implies an affected column in its top-left neighbor $(x' - 1, z' + 1)$. The following lemma links the number of runs to the θ-gap.

Lemma 1. *The parity p has θ-gap g iff $p[t]$ has g runs.*

6.2 The propagation branch number

The *propagation branch number* of a parity p is the minimum weight of the 2-round trail core (b) among states with this parity. More formally,

$$B(p) \triangleq \min\{\tilde{\mathrm{w}}(b) \ : \ P(\lambda^{-1}(b)) = p\}.$$

Owing to the portion of the target space already covered in Section 5, we can limit the propagation branch number to $T_2' = 28$. The strategy is as follows:

- First, we identify and exclude parity patterns p such that the propagation branch number can be proven to exceed $T_2' = 28$.
- Then, for the remaining parity patterns p we look for all states $b = \lambda(a)$ with $P(a) = p$ and $\tilde{\mathrm{w}}(b) \leq T_2' = 28$.
- Finally, we forward and backward extend the states seen as 2-round trail cores up to weight $T_3 = 36$.

Clearly, the kernel states, i.e., states such that $P(a) = 0$ must be considered. For instance, a state a with just two active bits in the same column will have $\mathrm{w}^{\mathrm{rev}}(a) = 4$. Then, $b = \lambda(a) = \pi(\rho(a))$ since θ has no effect in this case, and b also has two active bits. For KECCAK-f[1600], all the rotation constants in ρ are different and these two bits will not be in the same slice, so not in the same row and $\mathrm{w}^{\mathrm{rev}}(a) + \mathrm{w}(b) = 8$. Hence, the propagation branch number of the all-zero parity is at least 8 and thus the all-zero parity pattern must be included.

States that are out of the kernel are likely to have a higher propagation branch number. We now concentrate on how to find a lower bound on the propagation branch number of a given parity pattern.

6.3 Bounding the row branch number

The *row branch number* of a parity p is the minimum number of active rows before and after λ among states with this parity. More formally,

$$B_{\mathrm{rows}}(p) \triangleq \min\{\|\lambda^{-1}(b)\|_{\mathrm{row}} + \|b\|_{\mathrm{row}} \ : \ P(\lambda^{-1}(b)) = p\}.$$

Since an active row has at least propagation weight 2, this means that $B(p) \geq 2B_{\mathrm{rows}}(p)$. We can thus use the row branch number as a way to limit the search to parity patterns for which $\tilde{\mathrm{w}}(b) \leq T_2'$.

For a given parity pattern, we classify the columns as either affected, unaffected odd or unaffected even. We make use of the following properties to find a lower bound on the row branch number.

Lemma 2. *In terms of active rows, θ satisfies the following properties:*

- *An active bit in an affected column before θ will be passive after θ, and vice-versa. So, for each bit $(x, y, z) \xrightarrow{\pi\circ\rho} (x', y', z')$ of an affected column, at least one of row (y, z) in $\lambda^{-1}(b)$ and row (y', z') in b will be active.*
- *An odd unaffected column always contains at least one active bit and this bit stays active after θ. So, for at least one bit $(x, y, z) \xrightarrow{\pi\circ\rho} (x', y', z')$ of an odd unaffected column, both rows (y, z) in $\lambda^{-1}(b)$ and (y', z') in b will be active.*

These properties are translated into Algorithm 1, which returns a lower bound of $B_{\mathrm{rows}}(p)$. The algorithm avoids counting twice an active row by marking (in the sets a and b) the row positions already encountered.

Algorithm 1 Computing a lower bound of $B_{\text{rows}}(p)$

Let a and b be sets of row positions, which are initially empty
$B \leftarrow 0$
for each affected column (x, z) **do**
  for $y \in \mathbb{Z}_5$ **do**
    Let $(x, y, z) \xrightarrow{\pi \circ \rho} (x', y', z')$
    if $(y, z) \notin a$ and $(y', z') \notin b$ **then**
      $B \leftarrow B + 1$
      $a \leftarrow a \cup \{(y, z)\}$ and $b \leftarrow b \cup \{(y', z')\}$
    end if
  end for
end for
for each unaffected odd column (x, z) **do**
  Let $(x, i, z) \xrightarrow{\pi \circ \rho} (x'_i, y'_i, z'_i)$ for $i \in \mathbb{Z}_5$
  if $\{(i, z), i \in \mathbb{Z}_5\} \cap a = \emptyset$ **then**
    $B \leftarrow B + 1$
    $a \leftarrow a \cup \{(i, z), i \in \mathbb{Z}_5\}$
  end if
  if $\{(y'_i, z'_i), i \in \mathbb{Z}_5\} \cap b = \emptyset$ **then**
    $B \leftarrow B + 1$
    $b \leftarrow b \cup \{(y'_i, z'_i), i \in \mathbb{Z}_5\}$
  end if
end for
return B

6.4 Looking for candidate parity patterns

To find trails such that any two consecutive rounds have weight up to $T'_2 = 28$, we have to consider the parity patterns listed in Lemma 3.

Lemma 3. *A 2-round differential trail $Q = (b_0, b_1, b_2)$ in* Keccak-f[1600] *with* $\mathrm{w}(Q) \leq 28$ *necessarily satisfies one of the following properties on the parity of* $a_1 = \lambda^{-1}(b_1)$*:*

- *a_1 is in the kernel, i.e., $P(a_1) = 0$;*
- *the θ-gap of a_1 is 1 with a single run of length 1 or 2; or*
- *the θ-gap of a_1 is 2 or 3 with runs of length 1 each, all starting in the same slice.*

If parities are considered up to translation along z, we can restrict ourselves to parity patterns with runs starting in slice $z = 0$.

To prove this result, we conducted a recursive search as follows. Each parity is represented as a set of runs. First, all parity patterns p with a single run (so θ-gap 1) are investigated. All p with $B_{\text{rows}}(p) \leq \frac{T'_2}{2} = 14$ are stored into a set S. Then, we recursively add runs not overlapping the already added ones (so as to cover θ-gaps higher than 1), and all found p with $B_{\text{rows}}(p) \leq \frac{T'_2}{2} = 14$ are stored into a set S.

To limit the search, we use the following monotonicity property on the number of active rows. Using Lemma 2, changing an unaffected even column into either an unaffected odd or an affected column cannot decrease the number of active rows.

In the recursive search described above, adding a run to a parity pattern p can turn an unaffected odd column into an affected column. Hence, we cannot use the monotonicity property directly on the runs. However, adding a run never turns an affected column back into an unaffected one. So, before recursively adding a run to p, we apply a modified version of Algorithm 1 that does not take unaffected odd columns into account; this modified algorithm is monotonic in the runs. If the value returned by this modified algorithm is already above $\frac{T_2'}{2} = 14$, then there is no need to further add runs. This efficiently cuts the search.

Before being added to the candidate set S, the parity pattern p is tested with the unmodified Algorithm 1. For the remaining parity patterns, we explicitly generated all states a with these parities up to $\tilde{\mathrm{w}}(\lambda(a)) \leq T_2' = 28$. This allowed us to prove Lemma 3.

Algorithm 1 is implemented in the `getLowerBoundTotalActiveRows` function and the recursive search in `lookForRunsBelowTargetWeight` [5].

6.5 Starting from out-of-kernel states

For a given parity pattern p, we can construct all states $b = \lambda(a)$ with $P(a) = p$ and $\tilde{\mathrm{w}}(b) \leq T_2' = 28$. We proceed in two phases.

- In a first phase, we generate all states a such that $P(a) = p$ by assigning all possible 16 values to affected (odd or even) columns and by assigning a single active bit in each unaffected odd column. These states are such that $||a|| + ||\lambda(a)||$ is exactly $10g + 2c$, with g the θ-gap and c the number of unaffected odd columns.
- In a second phase, we take the states generated in the first phase and add pairs of bits to all unaffected columns. By adding a pair of bits, we do not alter $P(a)$.

In both phases, we keep only the states $b = \lambda(a)$ for which $\tilde{\mathrm{w}}(b) \leq T_2' = 28$. As can be seen in Table 3, both the weight and the reverse minimum weight are *monotonic*, i.e., adding an active bit to the state cannot decrease them. We can therefore limit the search by stopping adding pairs of bits when $\tilde{\mathrm{w}}(b)$ is above $T_2' = 28$.

In practice, what we did was the following.

- Let $\mathcal{P}$ be the set of parity patterns satisfying one of the conditions of Lemma 3 except $p = 0$.
- By the method described above, we construct all states in the set $\mathcal{B} = \{b \ : P(\lambda^{-1}(b)) \in \mathcal{P}$ and $\tilde{\mathrm{w}}(b) \leq T_2' = 28\}$.
- Finally, we forward and backward extend the states in $\mathcal{B}$ to 3-round trail cores up to weight $T_3 = 36$.

We again found the same trail core as in Section 5. The trail prefix of weight 32 has $P(a_1) = 0$ (so a_1 is in the kernel) and $P(a_2)$ has one run of length 2 (so a_2 has θ-gap 1). No other trail cores were found.

When extending the states in $\mathcal{B}$, we exhaustively scan all compatible states, thereby including cases where $P(a_1) = 0$ or $P(a_2) = 0$. Hence, we covered the whole target space, except for trails such that both $P(a_1) = 0$ and $P(a_2) = 0$.

7 Generating in-kernel trails

To close the target space, we must look at in-kernel trails of the form in Eq. (2) with both $P(a_1) = 0$ and $P(a_2) = 0$. In the case of in-kernel trails, we were able to be completely cover the space up to weight $T_3 = 40$, and we expect the techniques presented here can cover trails of higher weight. As $P(a_1) = P(a_2) = 0$, the θ operation has no effect and therefore $b_i = \pi(\rho(a_i))$. So this comes down to looking for states $a = a_1$, $b = b_1$, $c = a_2$ and $d = b_2$ connected as:

$$a \xrightarrow{\pi\circ\rho} b \xrightarrow{\chi} c \xrightarrow{\pi\circ\rho} d, \text{ with } P(a) = P(c) = 0. \tag{3}$$

We now summarize how we can efficiently generate all in-kernel three-round trail cores up to some weight and provide more details in following subsections. The key element in our method is the observation that any state b with $P(a) = 0$ and for which there exists a state c with $P(c) = 0$ can be represented in a specific way. The states a and b are iteratively constructed by adding active bits in the form of bit sequences called chains and vortices, defined in Section 7.2 below. Chains and vortices have an even number of active bits per column in a by construction and hence ensure $P(a) = 0$.

In b, there can be zero, one or more slices called knots, which contain three or more active bits. Each of these active bits is the end point of a chain that leads to another knot or that connects back to the same knot. The intermediate active bits of a chain appear pairwise in slices holding exactly two active bits in one column (called orbital slices, see Section 7.1). On top of chains connecting knots, a state b can exhibit a vortex, i.e., a cyclic sequence of active bits that appear pairwise both in the columns of a and in the columns of b.

By starting with an empty state and progressively adding chains, knots and vortices, one can quickly build states a and b that satisfy $P(a) = 0$ and for which there exist c with $P(c) = 0$, leading to 3-round in-kernel trail cores. Any state leading to a in-kernel trail can be represented in this way, and care is taken so that all possible states are generated, up to a given target weight. At each step, a lower bound on the weight of 3-round trail cores containing a and b is computed so as to efficiently limit the search.

As a final step, the generated states a and b are forward-extended to states c and d, limiting to c values in the kernel. Thanks to the properties of χ (see Section 3.1), the compatible states c can be expressed as a linear affine space. It is thereby easy to take the intersection of this affine space with the set of states such that $P(c) = 0$.

7.1 Characterizing the slices in b

Definition 1. *A state b is* tame *if $P(\lambda^{-1}(b)) = 0$ and such that there exists at least one state c compatible with b through χ such that $P(c) = 0$.*

To characterize states b such that $P(c) = 0$, we can reason on the slices b_z of b since χ and P can be jointly described in terms of slices. In particular, each slice c_z of c must be in the kernel, namely, $P(c_z) = 0$, and we have to characterize the slices b_z under that constraint. First, if $b_z = 0$ then $c_z = 0$ and $P(c_z) = 0$. Then, a slice b_z with a single active bit cannot be in the kernel after χ, as at least one column of c_z will have a single active bit. Finally, a slice b_z with two active bits must have its two active bits in the same column for c_z to be in the kernel. By inspection of Table 3, a row with a single active bit at coordinate x, e.g., 00100 transforms into an active row of the form $uv100$ with $u, v \in \{0, 1\}$, so the active bit stays active at x and zero, one or two active bits can appear at $x-2$ and $x-1$ of the same row. So, if the two bits are not in the same column, one of the active bits that stays after χ will not find another active bit in the same column. We summarize this in the next lemma.

Lemma 4. *If b is tame, then each of its slices has either*

- *no active bit,*
- *two active bits in the same column, or*
- *three or more active bits.*

We call an *empty slice* a slice with no active bit, and an *orbital slice* is a slice with two active bits in the same column. A slice that is neither empty not an orbital slice is called a *knot*. We say that a knot is *tame* if it can transform after χ into a slice in the kernel. According to Lemma 4, a tame knot has at least three active bits.

7.2 Characterizing the set of active bits

Since in the kernel θ acts as the identity, the active bits of a are just moved to other positions in b and their number remains the same, i.e., $||a|| = ||b||$. We can therefore represent a and b by a list of active bit positions $(p_i)_{i=1...||a||}$ in either the coordinates (x_i, y_i, z_i) in a or the coordinates (x'_i, y'_i, z'_i) in b, with $(x_i, y_i, z_i) \xrightarrow{\pi \circ \rho} (x'_i, y'_i, z'_i)$.

First, we start with the active bits in a. We say that active bits p_i and p_j are *peer* if they are in the same column in a, i.e., $x_i = x_j$ and $z_i = z_j$. Since each column has an even number of active bits when $P(a) = 0$, an active bit thus always has a peer.[1]

[1] While for columns with two active bits, the peer relationship is unambiguous, in the case of columns with four active bits, we choose which pairs of active bits are peer. Thus we can see the representation of the states as being augmented with additional attributes specifying the peer relationship and there may be several ways to represent the same state. By generating states via this representation, the only risk is to generate more states than necessary.

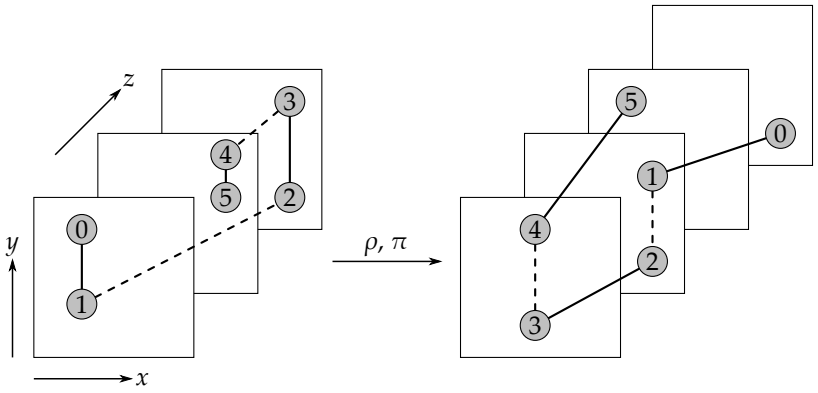

Fig. 2. Schematic example of a chain. An active bit position is represented by a circle with its index. Two active bits connected by a plain line (resp. dashed line) are peer (resp. chained).

Then, we move to the active bits in b. We say that the two active bits p_i and p_j are *chained* if they both lie in the same orbital slice in b. So $x'_i = x'_j$ and $z'_i = z'_j$ and no other active bit is in slice z'_i.

A *chain* is a sequence of bit positions of even length $(p_0, p_1, p_2, \ldots, p_{2n-1})$ such that p_{2k} and p_{2k+1} are peer ($\forall k \in \{0, \ldots, n-1\}$) and that p_{2k+1} and p_{2k+2} are chained ($\forall k \in \{0, \ldots, n-2\}$). In addition, the first and last active bits p_0 and p_{2n-1} must be in knots (either the same one or different ones). The simplest possible chain has length 2 and consists only in two peer active bits. Figure 2 depicts the concept of chain.

The definition of a *vortex* is the same as that of a chain $(p_0, p_1, p_2, \ldots, p_{2n-1})$, except that the first and last active bits p_0 and p_{2n-1} must be chained. In other words, a vortex forms a cycle of bit positions linked alternatively by peer and chained relationships, all in orbital slices.

In a tame state, each active bit position has exactly one peer position. The active bit positions in knots are the end points of chains, while the active bits in orbital slices are chained and belong to chains or vortices. Therefore, any tame state can be represented as a set of vortices and chains connecting knots.

7.3 Generating all tame states

To generate all tame states up to a target weight T_3, we generate states a and b by representing them using the concepts of Sections 7.1 and 7.2. The generation builds (initially empty) states a and b by iterating the following nested loops:

– In the outer loop, we add chains to the existing state. When adding a chain $(p_0, p_1, p_2, \ldots, p_{2n-1})$, the slices that receive the end points p_0 and p_{2n-1} must become knots if they are not already. If $n > 1$, the pairs of (chained)

active bits (i_{2k+1}, i_{2k+2}) are added to empty slices, which become orbital slices. Active bits cannot be added to already constructed orbital slices, as it would contradict the definition of an orbital slice. Enough chains must be added such that each knot contains at least 3 active bits (see Lemma 4).
- For a fixed set of chains produced in the previous step, the inner loop iterates on the number and position of vortices. In a vortex, all active bits are chained, so they must be added to empty slices, which become orbital slices.

With the monotonic lower bound function defined in the next section, we add chains and vortices until this lower bound exceeds T_3.

7.4 Lower-bounding the weight of in-kernel trails

We wish to determine a lower bound on the weight of 3-round in-kernel trail cores (b, d), namely, on $\mathrm{w}^{\mathrm{rev}}(a) + \mathrm{w}(b) + \mathrm{w}(d)$ with $a = \lambda^{-1}(b)$, from a and b only, for use in our trail generation. Since only d is unknown, this implies finding a lower bound on $\mathrm{w}(d)$. This can be done by first determining a lower bound on the Hamming weight $||d||$ and then bounding the weight of any state with given Hamming weight.

To determine a lower-bound on $||d||$, we work on each slice of b. If slice b_z has $u = \|b_z\|_{\mathrm{row}}$ active rows, then the slice c_z has at least u active bits. In addition, $P(c_z) = 0$ implies that the number of active bits must be even, so $||c_z|| \geq 2\lceil \frac{u}{2} \rceil$. Finally, we have $||d|| = ||c||$ so

$$||d|| \geq 2 \sum_z \left\lceil \frac{\|b_z\|_{\mathrm{row}}}{2} \right\rceil .$$

From Table 3, it is easy to verify the following lower bound:

$$\mathrm{w}(d) \geq \hat{\mathrm{w}}(||d||) \triangleq \left\lceil \frac{4||d||}{5} \right\rceil + [1 \text{ if } ||d|| = 1 \text{ or } 2 \pmod 5].$$

Hence, we define the *lower weight* of b as

$$\mathrm{L}(b) \triangleq \mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(b)) + \mathrm{w}(b) + \hat{\mathrm{w}} \left(2 \sum_z \left\lceil \frac{\|b_z\|_{\mathrm{row}}}{2} \right\rceil \right).$$

The lower weight yields a lower bound on the weight of 3-round in-kernel trail cores (b, d) regardless of d.

7.5 Limiting the search by lower-bounding the weight

At each level of the loop described in Section 7.3, the corresponding iteration is aborted, and elements are not further added, if we can be sure that the lower weight $\mathrm{L}(b)$ will become larger than the target weight T_3. Adding a chain to the state can potentially bring new knots and/or new orbital slices. Adding a vortex necessarily brings new orbital slices. Therefore, there is a limit in the

Number	$\tilde{\mathrm{w}}(\cdot)$	$\mathrm{w}^{\mathrm{rev}}(a_1)$	$\mathrm{w}(b_1)$	$\mathrm{w}(b_2)$	$P(a_1)$	$P(a_2)$	Structure of a_1, b_1
1	32	4	4	24	kernel	θ-gap 1	
1	35	12	12	11	kernel	kernel	vortex of length 6
7	36	12	12	12	kernel	kernel	vortex of length 6
7	39	12	12	15	kernel	kernel	vortex of length 6
2	39	12	11	16	kernel	kernel	2 knots connected by 3 chains
41	40	12	12	16	kernel	kernel	vortex of length 6
4	40	12	12	16	kernel	kernel	2 knots connected by 3 chains

Table 4. Summary of all 3-round differential trail cores found in KECCAK-f[1600] up to weight 36, and up to weight 40 for in-kernel trails. The number indicates the number of cores with the same properties indicated in the other columns.

number of knots and orbital slices that must be considered for the generation to be complete up to the target weight.

As a preliminary step, the minimum reverse weight satisfies the following inequality (see Table 3):

$$\mathrm{w}^{\mathrm{rev}}(a) \geq \hat{\mathrm{w}}^{\mathrm{rev}}(||a||) \triangleq \left\lceil \frac{3||a||}{5} \right\rceil .$$

We see from Lemma 4 that each tame knot contributes to at least 3 active bits in a and in b. Furthermore, the number of bits in each slice of a must be even ($P(a) = 0$), so $||a|| \geq 2\left\lceil \frac{3k}{2} \right\rceil$ and $\mathrm{w}^{\mathrm{rev}}(a) \geq \hat{\mathrm{w}}^{\mathrm{rev}}(||a||)$, with k the number of knots. In b, each tame knot has at least 3 active bits on at least 2 different active rows, hence contributing at least 5 to the weight, and so $\mathrm{w}(b) \geq 5k$. Each active row in b contributes to at least one active bit in d so $||d|| \geq 2k$ and $\mathrm{w}(d) \geq \hat{\mathrm{w}}(||d||)$.

For instance, $k = 5$ knots implies that $||a|| \geq 16$ and $\mathrm{w}^{\mathrm{rev}}(a) \geq \hat{\mathrm{w}}^{\mathrm{rev}}(16) = 10$, that $\mathrm{w}(b) \geq 25$ and that $||d|| \geq 10$ and $\mathrm{w}(d) \geq \hat{\mathrm{w}}(10) = 8$, so a lower weight of at least 43. If $T_3 \leq 42$, looking for configurations with from 0 to 4 knots is therefore sufficient, not even counting the orbital slices that also compose chains.

We found cores of weight 35, 36, 39 and 40, as detailed in Table 4. For illustration purposes, examples of trail prefixes are shown in [7]. The search described in this section is implemented in the `TrailCore3Rounds` and `TrailCoreInKernel-AtC` classes [5].

8 Extension to six-round trails

Table 4 summarizes all the 3-round cores found. These trail cores completely represent all the 3-round trails up to weight 36 (or 40 for in-kernel trails). They can be found in [5].

The second phase introduced in Section 4 consists in exhaustively extending forward and backward all the 3-round trail cores into 6-round trails cores. As

no 6-round trail of weight up to 73 were found, we conclude that a 6-round differential trail in KECCAK-f[1600] has at least weight 74. In the specific case of in-kernel trails, no 6-round trail of weight up to 81 were found and we conclude that a 6-round in-kernel differential trail in KECCAK-f[1600] has at least weight 82.

For the 24 rounds of KECCAK-f[1600], a differential trail has at least weight 296, and an in-kernel trail has at least weight 328.

9 Conclusions

We studied and implemented the exhaustive generation of 3-round differential trails in the KECCAK-f[1600] permutation, which allowed us to prove a lower bound on the weight of differential trails. The techniques developed in this paper exploit the properties of the mixing layer in its round function to provide better bounds than what a brute-force method could provide. Table 2 shows that there remains a gap between the best known trails and the lower bound beyond three rounds that calls for future work. Finally, the concepts introduced in this paper, such as chains, vortices, knots and parity runs, help read trails and understand them.

References

1. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *On the indifferentiability of the sponge construction*, Advances in Cryptology – Eurocrypt 2008 (N. P. Smart, ed.), Lecture Notes in Computer Science, vol. 4965, Springer, 2008, `http://sponge.noekeon.org/`, pp. 181–197.
2. ———, *Cryptographic sponge functions*, January 2011, `http://sponge.noekeon.org/`.
3. ———, *On alignment in* KECCAK, ECRYPT II Hash Workshop 2011, 2011.
4. ———, *The* KECCAK *reference*, January 2011, `http://keccak.noekeon.org/`.
5. ———, KECCAKTOOLS *software*, April 2012, `http://keccak.noekeon.org/`.
6. E. Biham and A. Shamir, *Differential cryptanalysis of DES-like cryptosystems*, CRYPTO (A. Menezes and S. Vanstone, eds.), Lecture Notes in Computer Science, vol. 537, Springer, 1990, pp. 2–21.
7. J. Daemen and G. Van Assche, *Differential propagation analysis of* KECCAK, Cryptology ePrint Archive, Report 2012/163, 2012, `http://eprint.iacr.org/`.
8. J. Daemen, M. Peeters, G. Van Assche, and V. Rijmen, *Nessie proposal: the block cipher* NOEKEON, Nessie submission, 2000, `http://gro.noekeon.org/`.
9. J. Daemen and V. Rijmen, *The design of Rijndael — AES, the advanced encryption standard*, Springer-Verlag, 2002.
10. ———, *Plateau characteristics and AES*, IET Information Security **1** (2007), no. 1, 11–17.
11. I. Dinur, O. Dunkelman, and A. Shamir, *New attacks on Keccak-224 and Keccak-256*, Fast Software Encryption 2012, 2012, to appear, draft available from Cryptology ePrint Archive, Report 2011/624.
12. A. Duc, J. Guo, T. Peyrin, and L. Wei, *Unaligned rebound attack: Application to Keccak*, Fast Software Encryption 2012, 2012, to appear, draft available from Cryptology ePrint Archive, Report 2011/420.

13. P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen, *Grøstl – a SHA-3 candidate*, Submission to NIST (round 3), 2011.
14. E. Heilman, *Restoring the differential security of MD6*, ECRYPT II Hash Workshop 2011, 2011.
15. M. Naya-Plasencia, A. Röck, and W. Meier, *Practical analysis of reduced-round Keccak*, Indocrypt 2011, 2011.
16. NIST, *Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family*, Federal Register Notices **72** (2007), no. 212, 62212–62220, `http://csrc.nist.gov/groups/ST/hash/index.html`.
17. R. Rivest, B. Agre, D. V. Bailey, S. Cheng, C. Crutchfield, Y. Dodis, K. E. Fleming, A. Khan, J. Krishnamurthy, Y. Lin, L. Reyzin, E. Shen, J. Sukha, D. Sutherland, E. Tromer, and Y. L. Yin, *The MD6 hash function – a proposal to NIST for SHA-3*, Submission to NIST, 2008, `http://groups.csail.mit.edu/cis/md6/`.
18. H. Wu, *The hash function JH*, Submission to NIST (round 3), 2011.