J. Cryptol. (2014) 27: 183–209
DOI: 10.1007/s00145-012-9142-5

Journal of
CRYPTOLOGY

Improved Practical Attacks on Round-Reduced Keccak

Itai Dinur
Computer Science Department, The Weizmann Institute, Rehovot, Israel
itai.dinur@weizmann.ac.il

Orr Dunkelman
Computer Science Department, The Weizmann Institute, Rehovot, Israel
and
Computer Science Department, University of Haifa, Haifa, Israel

Adi Shamir
Computer Science Department, The Weizmann Institute, Rehovot, Israel

Communicated by Willi Meier

Received 26 June 2012
Online publication 29 December 2012

Abstract. The Keccak hash function is the winner of NIST's SHA-3 competition, and so far it showed remarkable resistance against practical collision finding attacks: After several years of cryptanalysis and a lot of effort, the largest number of Keccak rounds for which actual collisions were found was only 2. In this paper, we develop improved collision finding techniques which enable us to double this number. More precisely, we can now find within a few minutes on a single PC actual collisions in the standard Keccak-224 and Keccak-256, where the only modification is to reduce their number of rounds to 4. When we apply our techniques to 5-round Keccak, we can get in a few days near collisions, where the Hamming distance is 5 in the case of Keccak-224 and 10 in the case of Keccak-256. Our new attack combines differential and algebraic techniques, and uses the fact that each round of Keccak is only a quadratic mapping in order to efficiently find pairs of messages which follow a high probability differential characteristic. Since full Keccak has 24 rounds, our attack does not threaten the security of the hash function.

Key words. Cryptanalysis, SHA-3, Keccak, Collision, Near-collision, Practical attack.

1. Introduction

The Keccak hash function [5] is the winner of NIST's SHA-3 competition. The hash function uses the sponge construction [4] to map arbitrarily long inputs into fixed length outputs, and its official versions have an internal state size of $b = 1600$ bits, and an

output size n of either 224, 256, 384, or 512 bits. The internal permutation of Keccak consists of 24 application of a non-linear round function, applied to the 1600-bit state. Previous papers on Keccak, such as [17], include analysis of Keccak versions with a reduced internal state size, or with different output sizes. However, in this paper, we concentrate on the official Keccak versions, and the only way in which we modify them is by reducing their number of rounds.

Previous results on Keccak's internal permutation include zero-sum distinguishers presented in [2], and later improved in [6,7,11]. Although zero-sum distinguishers can distinguish the full internal permutation of Keccak from a random permutation, they have very high complexities, and they seem unlikely to threaten the core security properties of Keccak (namely, collision resistance, preimage resistance and second-preimage resistance). Other results on Keccak's internal permutation include a differential analysis given in [12]. Using techniques adapted from the rebound attack [16], the authors construct differential characteristics which give distinguishers on up to 8 rounds of the permutation, with complexity of about 2^{491}. However, in their method it is not clear how to reach the starting state differences of these characteristics from valid initial states of Keccak's internal permutation, since in sponge constructions a large portion of the initial state of the permutation is fixed and cannot be chosen by the cryptanalyst. Thus, although the results of [12] seem to be more closely related to the core security properties of Keccak than zero-sum distinguishers, they still do not lead to any attacks on the Keccak hash function itself.

Currently, there are very few results that analyze reduced-round variants of the full Keccak (rather than its building blocks): in [3], Bernstein described preimage attacks which extend up to 8 rounds of Keccak, but are only marginally faster than exhaustive search, and use a huge amount of memory. More recently, Naya-Plasencia, Röck and Meier presented practical attacks on Keccak-224 and Keccak-256 with a very small number of rounds [18]. These attacks include a preimage attack on 2 rounds, as well as collisions on 2 rounds and near-collisions on 3 rounds. In this paper, we extend these collision attacks on Keccak-224 and Keccak-256 by 2 additional rounds: we find actual collisions in 4 rounds and actual near-collisions in 5 rounds of Keccak-224 and Keccak-256, with Hamming distance 5 and 10, respectively.

The collisions and near-collisions of [18] were obtained using low Hamming weight differential characteristics, starting from the initial state of Keccak's permutation. Such low Hamming weight characteristics are also the starting point of our new attacks, but we do not require the characteristics to start from the initial state of the permutation. Given a low Hamming weight starting state difference of a characteristic, we can easily extend it backwards by one round, and maintain its high probability (as done in [12]). However, due to the very fast diffusion of the inverse linear mapping used by Keccak's permutation, the new starting state difference of the extended characteristic has a very high Hamming weight. We call this starting state difference a *target difference*, since our goal is to find message pairs which have this difference after one round of the Keccak permutation (after the fixed round, this difference will evolve according to the characteristic with high probability).[1] One of the main tools we develop in this paper is

[1] We note that the target difference is not a valid initial difference of the permutation, which fixes many of the state bits to pre-defined values. As a result, the high probability characteristic cannot be used to extend the results of [18] by an additional round.

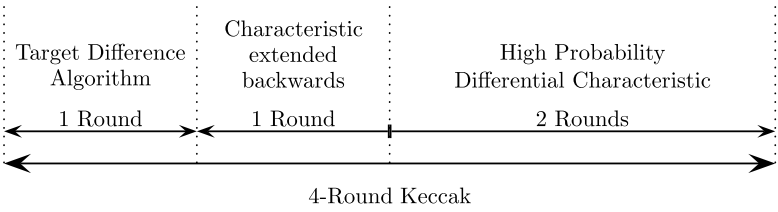


Fig. 1. Extending a 2-round differential characteristic by two additional rounds.

an algorithm that aims to achieve this goal, namely, to find message pairs which satisfy a given target difference after one Keccak permutation round. We call this algorithm a *target difference algorithm*, and it allows us to extend our initial characteristic by two additional rounds (as shown in Fig. 1): we first extend the characteristic backwards by one round to obtain the target difference (while maintaining the characteristic's high probability). Then, we use the target difference algorithm to link the characteristic to the initial state of Keccak's permutation, through an additional round. We note that the final link, which efficiently bypasses Keccak's first Sbox layer, uses algebraic techniques rather than standard probabilistic techniques. Our methods are thus related to previously published work (such as [1]) which combines algebraic and differential techniques in block cipher cryptanalysis.

In the domain of hash function, the target difference algorithm is related to several cryptanalytic techniques that were developed in recent years. In particular, it is related to the work of Khovratovich, Biryukov and Nikolic [15], where, similarly to our algorithm, the authors use linear algebra to quickly satisfy many conditions of a differential characteristic. However, these techniques seem to work best on byte-oriented hash functions, whose internal structure can be described using a few sparse equations, which is not the case for Keccak. Our algorithm is also closely related to the work of Khovratovich [14] that exploits structures (which aggregate internal states of the hash function) in order to reduce the amortized complexity of collision attacks: the attacker first finds a truncated differential characteristic and searches for a few pairs of initial states that satisfy it. Then, using the structures and the initially found pairs, the attacker efficiently obtains many additional pairs that satisfy the truncated characteristic. However, in the case of Keccak, there are very few characteristics that can lead to a collision with high probability, and it seems unlikely that they can be joined in order to form the truncated differential characteristic required in order to organize the state differences into such structures. Moreover, it seems difficult to find even one pair of initial states that satisfy the target difference for Keccak. Another attack related to the target difference algorithm is the rebound attack [16]. In this attack, the cryptanalyst uses the available degrees of freedom to efficiently link and extend two truncated differential characteristics, both forwards and backwards, from an intermediate state of the hash function. However, once again, such high probability truncated characteristics are unlikely to exist for Keccak. Moreover, it is not clear how to use the rebound attack to link the backward characteristic to the initial state of the permutation. Thus, our target difference algorithm can be viewed as an asymmetric rebound attack, where one side of the characteristic is fixed.

Our full attacks have two parts, where in the first part we execute the target difference algorithm in order to obtain a sufficiently large set of message pairs that satisfy the

target difference after the first round. In the second part of the attack, we try different message pairs in this set in order to find a pair whose difference evolves according to a characteristic whose starting state is the target difference. Since the target difference algorithm does not control the differences beyond the first round, the second part of the attack is a standard probabilistic differential attack (which only searches for collisions or near-collisions obtained from message pairs within a specific set). The high probability differential characteristic beyond the first round ensures that the time complexity of the second part of the attack is relatively low.

Although the target difference algorithm is heuristic, and there is no provable bound on its running time, it was successfully applied with its expected complexity to many target differences defined by the high probability differential characteristics. Consequently, we were able to find actual collisions for 4 rounds of Keccak-224 and Keccak-256 within minutes on a standard PC. By using good differential characteristics for an additional round, we found near-collisions for 5 rounds of Keccak-224 and Keccak-256. However, this required more computational effort (namely, a few days on a single PC), since the extended characteristics have lower probabilities.

The paper is organized as follows. In Sect. 2, we briefly describe Keccak, and in Sect. 3 we introduce our notations. In Sect. 4, we give a comprehensive overview of the target difference algorithm and describe the properties of Keccak that it exploits. In Sect. 5, we present our results on round-reduced Keccak. In Appendix A, we describe the full details of the target difference algorithm, and in Appendix B, we propose an alternative algorithm, which has a better understood time complexity. Since the original algorithm gave us very good results in practice, we did not use this alternative version. However, it may be more efficient in some cases, especially if someone finds longer high probability characteristics for Keccak's permutation.

2. Description of Keccak

In this section, we give short descriptions of the sponge construction and the Keccak hash function. More details can be found in the Keccak specification [5].

The sponge construction [4] works on a state of b bits, which is split into two parts: the first part contains the first r bits of the state (called the outer part of the state) and the second part contains the last $c = b - r$ bits of the state (called the inner part of the state).

Given a message, it is first padded and cut into r-bit blocks, and the b state bits are initialized to zero. The sponge construction then processes the message in two phases: In the absorbing phase, the message blocks are processed iteratively by XORing each block into the first r bits of the current state, and then applying a fixed permutation on the value of the b-bit state. After processing all the blocks, the sponge construction switches to the squeezing phase. In this phase, n output bits are produced iteratively, where in each iteration the first r bits of the state are returned as output and the permutation is applied.

The Keccak hash function uses multi-rate padding: given a message, it first appends a single 1 bit. Then, it appends the minimum number of 0 bits followed by a single 1 bit, such that the length of the result is a multiple of r. Thus, multi-rate padding appends at least 2 bits and at most $r + 1$ bits.

The official Keccak versions have $b = 1600$ and $c = 2n$, where $n \in \{224, 256, 384, 512\}$. The 1600-bit state can be viewed as a 3-dimensional array of bits, $a[5][5][64]$, and each state bit is associated with 3 integer coordinates, $a[x][y][z]$, where x and y are taken modulo 5, and z is taken modulo 64.

The Keccak permutation consists of 24 rounds, which operate on the 1600 state bits. Each round of the permutation consists of five mappings $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$. Keccak uses the following naming conventions, which are helpful in describing these mappings:

- A row is a set of 5 bits with constant y and z coordinates, i.e., $a[*][y][z]$.
- A column is a set of 5 bits with constant x and z coordinates, i.e., $a[x][*][z]$.
- A lane is a set of 64 bits with constant x and y coordinates, i.e., $a[x][y][*]$.
- A slice is a set of 25 bits with a constant z coordinate, i.e., $a[*][*][z]$.

The five mappings are given below, for each x, y, and z (where the state addition operations are over GF(2)):

1. θ is a linear map, which adds to each bit in a column, the parity of two other columns:

$$\theta\colon a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1].$$

   In this paper, we also use the inverse mapping, θ^{-1}, which is more complicated and provides much faster diffusion: for θ^{-1}, flipping the value of any input bit, flips the value of more than half of the output bits.
2. ρ rotates the bits within each lane by $T(x, y)$, which is a predefined constant for each lane:

$$\rho\colon a[x][y][z] \leftarrow a[x][y]\big[z + T(x, y)\big].$$

3. π reorders the lanes:

$$\pi\colon a[x][y][z] \leftarrow a[x'][y'][z], \quad \text{where } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \end{pmatrix}.$$

4. χ is the only non-linear mapping of Keccak, working on each of the 320 rows independently:

$$\chi\colon a[x][y][z] \leftarrow a[x][y][z] + \big((\neg a[x+1][y][z]) \wedge a[x+2][y][z]\big).$$

   Since χ works on each row independently, in can be viewed as an Sbox layer which simultaneously applies the same 5 bits to 5 bits Sbox to the 320 rows of the state. We note that the Sbox function is an invertible mapping, and our techniques are heavily based on the observation that the algebraic degree of each output bit of χ as a polynomial in the five input bits is only 2. We also note that the algebraic degree the inverse mapping χ^{-1} is 3 (as noted in [5]).
5. ι adds a round constant to the state:

$$\iota\colon a \leftarrow a + \mathrm{RC}[i_r].$$

   We omit the values of $\mathrm{RC}[i_r]$, as they are not needed for our analysis.

3. Notations

Given a message M, we denote its length in bits by $|M|$. Unless specified otherwise, in this paper we assume that $|M| = r - 8$. Namely, we consider only single-block messages of maximal length such that $|M|(\text{modulo } 8) \equiv 0$, which give us the maximal number of degrees of freedom for single-block messages containing an integral number of bytes.[2] Given M, we denote the initial state of the Keccak permutation as the 1600-bit word $\overline{M} \triangleq M \parallel p \parallel 0^c$, where $\parallel$ denotes concatenation, and p denotes the 8-bit pad 10000001.

The first three operations of Keccak's round function are linear mappings, and we denote their composition by $L \triangleq \rho \circ \pi \circ \theta$. We sometimes refer to L as a "half round" of the Keccak permutation, where $\iota \circ \chi$ represents the other half. We denote the Keccak nonlinear function on 5-bit words defined by varying the first index by $\chi_{|5}$. The difference distribution table (DDT) of this function is a two-dimensional 32×32 integer table, where all the differences are assumed to be over GF(2). The entry $\text{DDT}(\delta^{\text{in}}, \delta^{\text{out}})$ specifies the number of input pairs to this Sbox with difference δ^{in} that give the output difference δ^{out} (i.e., the size of the set $\{x \in \{0,1\}^5 \mid \chi_{|5}(x) + \chi_{|5}(x + \delta^{\text{in}}) = \delta^{\text{out}}\}$).

We denote a 1600-bit difference in the state of Keccak's permutation after i rounds by ΔS_i (e.g., ΔS_0 is the initial difference, $\Delta S_{0.5}$ is the difference after the application of L and ΔS_1 is the difference after the application of the first round function). We denote the 1600-bit target difference ΔS_1, which is the input of the target difference algorithm, by Δ_T. The output of the algorithm is a subset of ordered pairs of single block messages $\{(M_1^1, M_1^2), (M_2^1, M_2^2), \ldots, (M_k^1, M_k^2)\}$ that satisfy this difference after one round R, namely $R(\overline{M}_i^1) + R(\overline{M}_i^2) = \Delta_T \ \forall i \in \{1, 2, \ldots, k\}$.

4. Overview of the Target Difference Algorithm

When designing the target difference algorithm, we face two problems: first, the target difference Δ_T extends backwards, beyond the first Keccak Sbox layer to $\Delta S_{0.5}$, with very low probability (due to its high Hamming weight). The second problem is that the initial state of the permutation fixes many of the state bits to pre-defined values, and the initial states that we use must satisfy these constraints. On the other hand, Keccak has several useful properties that we can exploit in our target difference algorithm. In this section, we describe these properties in detail and give an overview of the algorithm.

4.1. *The Properties of Keccak Exploited by the Target Difference Algorithm*

Property 1. Keccak-224 and Keccak-256 allow the user to control many of the 1600 state bits of the initial state of the permutation. Thus, given a target difference, we expect many solutions to exist (namely, one-block message pairs which have the 1600-bit target difference after one permutation round): since we consider message pairs where each message is of length $r - 8 = 1600 - 8 - c$ bits (1144 for Keccak-224, and 1080 for Keccak-256), given an arbitrary 1600-bit target difference, there is an expected number of $2^{2(1600-8-c)-1600} = 2^{1584-2c}$ message pairs of this length that satisfy this difference

[2] In general, we can choose messages of different lengths which do not necessarily have an integral number of bytes. However, the additional restrictions do not have a significant impact on our attacks.

(regardless of the value of the inner part of the state). Thus, the algorithm has 704 and 560 degrees of freedom for Keccak-224 and Keccak-256, respectively.

Despite the large number of available degrees of freedom, the number of possible solutions varies significantly according to the target difference. To demonstrate this, we use the fact that L^{-1} has very fast diffusion (i.e., even an input with one non-zero bit is mapped by L^{-1} into a roughly balanced output). We consider the case where $t > 0$ out of the 320 Sboxes of the target difference are active (i.e., they have a non-zero output difference). Each one of the $320 - t$ non-active Sbox zero output differences is uniquely mapped backwards to a zero input difference into the first Sbox layer. Using the Keccak Sbox DDT, it is easy to see that each one of the t active Sbox output differences is mapped to more than 8 possible input differences. Thus, the number of possible state differences $\Delta S_{0.5}$ is more than $8^t = 2^{3t}$. Since L is invertible and acts deterministically on the differences, the number of possible input differences to the Keccak compression function ΔS_0 remains the same. If we require that the last $c+8$ bits of ΔS_0 are zero, for t large enough, we still expect more than 2^{3t-c-8} valid solutions. When the target difference is chosen at random, we have $t \approx 310$ (since the probability that an Sbox output difference is zero is $\frac{1}{32}$). This gives more than $2^{930-448-8} = 2^{474}$ expected solutions for Keccak-224, and more than $2^{930-512-8} = 2^{410}$ expected solutions for Keccak-256. On the other hand, consider the extreme case of $t = 1$ (i.e., the target difference has only one active Sbox). Clearly, this Sbox cannot contribute more than 31 possible values of $\Delta S_{0.5}$. Since L^{-1} has very fast diffusion, these possible differences are mapped to at most 31 roughly balanced non-zero possible input differences ΔS_0, and we do not expect the last $c + 8$ bits of any of them to be zero. To conclude, target differences with a small number of active Sboxes are likely to have no solutions at all. On the other hand, a majority of the target differences have a very large number of expected solutions for Keccak-224 and Keccak-256. Note that having a large number of solutions does not imply that it is easy to find any one of them, since their density is still minuscule.

Property 2. The algebraic degree of the Keccak Sboxes is only 2. This implies that given a 5-bit input difference δ^{in} and a 5-bit output difference δ^{out}, the set of values $\{v_1, v_2, \ldots, v_l\}$ such that $\chi_{|5}(v_i) + \chi_{|5}(v_i + \delta^{\text{in}}) = \delta^{\text{out}}$ is an affine subset. Since $(v_i + \delta^{\text{in}}) + \delta^{\text{in}} = v_i$, then $v_i + \delta^{\text{in}} \in \{v_1, v_2, \ldots, v_l\}$, implying $\{v_1, v_2, \ldots, v_l\} = \{v_1 + \delta^{\text{in}}, v_2 + \delta^{\text{in}}, \ldots, v_l + \delta^{\text{in}}\}$. Thus, both coordinates of the ordered pairs give the same subset whose size is $\text{DDT}(\delta^{\text{in}}, \delta^{\text{out}})$.

We note that similar observations were used in [10] to prove that when $\text{DDT}(\delta^{\text{in}}, \delta^{\text{out}}) = 2$ or 4, the same holds. In the specific case of Keccak, we also use 3-dimensional affine subsets of pairs that satisfy the Sbox difference transition $(\delta^{\text{in}}, \delta^{\text{out}})$, for which $\text{DDT}(\delta^{\text{in}}, \delta^{\text{out}}) = 8$. On the other hand, since the algebraic degree of the inverse Sbox is 3, which is reduced to 2 (rather than 1) after differentiation, the output values that satisfy an input and an output difference do not necessarily form an affine subset.

Property 3. For any non-zero 5-bit output difference δ^{out} from a Keccak Sbox, the set of possible input differences, $\{\delta^{\text{in}} | \text{DDT}(\delta^{\text{in}}, \delta^{\text{out}}) > 0\}$, contains at least 5 (and up to 17) 2-dimensional affine subspaces. These affine subspaces can be easily pre-computed using the DDT, for each one of the 31 possible non-zero output differences. However, we note that there is no output difference for which the set of possible input differences contains an affine subspace of dimension 3 or higher.

4.2. *Formulating the Problem*

Given Δ_T, an arbitrary message pair (M^1, M^2) in which $|M^1| = |M^2| = r - 8$ is a solution to our problem if $R(\overline{M}^1) + R(\overline{M}^2) = \Delta_T$. This can be formulated using two constraints on the 1600-bit words $(\overline{M}_1, \overline{M}_2)$:

1. The last $c + 8$ bits of $\overline{M}^1$ and $\overline{M}^2$ are equal to $p \parallel 0^c$, where p denotes the 8-bit pad 10000001.
2. $R(\overline{M}^1) + R(\overline{M}^2) = \Delta_T$ (where R is the permutation round of Keccak).

We can easily formulate the first constraint using linear equations on the bits of $\overline{M}_1$ and $\overline{M}_2$. Since Keccak's Sbox has an algebraic degree of 2 over GF(2), we can formulate the second constraint as a system of quadratic equations on these bits. Standard heuristic techniques for solving such systems include using the available degrees of freedom to fix some message values (or values before the first Sbox layer) in order to linearize the system. However, these techniques require many more than the available number of degrees of freedom when used in a trivial way. For example, in order to get linear equations after one round of Keccak's permutation, we can fix 3 out of the 5 bits entering an Sbox (after the first linear layer), such that there are no two consecutive unknown input bits entering the Sbox. Using this technique reduces the single quadratic term in the symbolic form of each of the Sbox'es output bits to a linear term. However, this requires fixing $320 \cdot 3 = 960$ bits per massage, and $2 \cdot 960 = 1920$ bits in total, which is significantly more than the 704 available degrees of freedom for Keccak-224 (and clearly more than the available number of degrees of freedom for the other Keccak versions). Consequently, we have to repeat the linearization procedure a huge number of times, with different fixed values, in order to find a solution.

A Two-Phase Algorithm Although we expect our quadratic system to have many solutions,[3] solving all the equations at once seems difficult. Thus, we split the problem into easier tasks by exploiting the low algebraic degree of Keccak's Sbox to a greater extent than in the standard techniques: as described in Property 2 of Sect. 4.1 given an input difference and an output difference to an Sbox, all the pairs of input values that satisfy them form an affine subset. This suggests an algorithm with two phases, where in the first phase (called the *difference phase*) we find an input difference to all the Sboxes, and in the second phase (called the *value phase*) we obtain the actual values of the message pairs that lead to the target difference.

Using this two-phase approach, the ordered pairs produced by our algorithm satisfy two additional properties: the 1600-bit *input difference* of the initial states ΔS_0 is fixed to some 1600-bit value Δ_I (i.e., $\overline{M}_i^1 + \overline{M}_i^2 = \Delta_I \ \forall i \in \{1, 2, \ldots, k\}$), and the set composed of all the initial states defined by the first message in each ordered pair (i.e., $\bigcup\{\overline{M}_i^1\} \ \forall i \in \{1, 2, \ldots, k\}$) forms an affine subset. The algorithm outputs the ordered pairs as the fixed 1600-bit input difference Δ_I, and some basis for the affine subset $\bigcup\{\overline{M}_i^1\} \ \forall i \in \{1, 2, \ldots, k\}$. We note that the large number of degrees of freedom allows

[3] As demonstrated in Sect. 4.1, a large number of solutions is expected unless there are many non-active Sboxes. This should be contrasted to differential attacks, where the attacker searches for differential characteristics with many non-active Sboxes, which ensure that the differential transitions occur with high probability.

us to restrict the set of solutions (i.e., the set of message pairs that satisfy the target difference) to a smaller subset (but still large enough for our purposes) that can be found relatively easily. In particular, the algorithm considers only message pairs with a fixed difference Δ_I, for which all the solutions can be found by solving linear equations.

The two constraints above, which define our quadratic equation system, are broken into two sets of constraints, since we have to simultaneously enforce two *difference constraints* (given as constraints on the 1600-bit word Δ_I):

Difference Constraint 1. *The last $c + 8$ bits of Δ_I are equal to zero.*

Difference Constraint 2. *The difference transition $L(\Delta_I) \rightarrow \Delta_T$ is possible, i.e., there exists some* 1600*-bit word W such that $\chi(W) + \chi(W + L(\Delta_I)) = \Delta_T$ (note that since L is a linear function, $L(\Delta_I)$ is well-defined).*

The first difference constraint simply equates bits of the input difference Δ_I to zero (456 bits for Keccak-224 and 520 bits for Keccak-256), while the second difference constraint assigns to every 5 bits of $L(\Delta_I)$ that enter an Sbox, several possible values which are not related by simple affine equations.

In the second phase, we enforce additional *value constraints* (given on the 1600-bit word $\overline{M}^1$):

Value Constraint 1. *The last $c + 8$ bits of $\overline{M}^1$ are equal to $p \parallel 0^c$, where p denotes the* 8*-bit pad* 10000001.

Value Constraint 2. $R(\overline{M}^1) + R(\overline{M}^1 + \Delta_I) = \Delta_T$.

Note that the first difference constraint and the first value constraint on each $\overline{M}_i^1$ also ensure that the same value constraint holds for $\overline{M}_i^2$ (i.e., the last $c + 8$ bits of $\overline{M}_i^2$ are equal to $p \parallel 0^c$).

Given a single 1600-bit Sbox layer input difference $\Delta S_{0.5}$, Property 2 of Sect. 4.1 implied that enforcing the two value constraints simply reduces to solving a union of two sets of linear equations. On the other hand, it is not clear how to simultaneously enforce both of the difference constraints, since given an output difference to an Sbox δ^{out}, all the possible input differences δ^{in} such that $\text{DDT}(\delta^{\text{in}}, \delta^{\text{out}}) > 0$, are not related by simple affine relations.

4.3. *The Difference Phase*

Unsuccessful Attempts to Enforce the Difference Constraints We can try to enforce both difference constraints by assigning the undetermined $1600 - c - 8$ bits of Δ_I, in such a way that the second difference constraint will hold. This usually involves iteratively constructing an assignment for Δ_I, by guessing several undetermined bits at a time, and filtering the guesses by verifying the second difference constraint. However, this is likely to have a very large time complexity, since L diffuses the bits of Δ_I in a way that forces us to guess many bits before we can start filtering the guesses. Moreover, for any Δ_T, the fraction of input differences satisfying the first difference constraint that

also satisfy the second difference constraint is very small. Thus, most of the computational effort turns out to be useless, since the guesses are likely to be discarded at later stages of the algorithm. Another approach is to guess $L(\Delta_I)$ by iteratively guessing the 5-bit Sbox input differences, and filtering the guesses by verifying the first difference constraint. For similar reasons, this approach is likely to have a very large time complexity.

A Better Approach Both of these approaches are very strict, since each guess made by the algorithm commits to a specific value for some of the bits of Δ_I, or $L(\Delta_I)$, and restricts the solution space significantly. Thus, we use Property 3 of Sect. 4.1, which gives us more flexibility, and significantly reduces the time complexity: given any non-zero 5-bit output difference to a Keccak Sbox, the set of possible input differences contains at least five 2-dimensional affine subspaces. Consequently, in order to enforce the second difference constraint, for each Sbox with a non-zero output difference (i.e., an active Sbox), we choose one of the affine subsets (which contains 4 potential values for the 5 Sbox input bits of $L(\Delta_I)$), instead of choosing specific values for these bits. This enables us to maintain an affine subspace of potential values for $L(\Delta_I)$, starting with the full 1600-dimensional space, and iteratively reducing its dimension by adding affine equations in order to enforce the second difference constraint for each Sbox. In addition to these affine equations that we add per active Sbox, we also have to add the linear equations for the non-active Sboxes (which equate their 5 input difference bits to zero), and the additional $c+8$ linear equations that enforce the first difference constraint. All of these equations are added to a linear system of equations that we denote by E_Δ.

Since the $c+8$ equations that enforce the first difference constraint do not depend on the target difference, we add them to E_Δ before we iterate the Sboxes. While iterating over the active Sboxes, we add equations on $L(\Delta_I)$ in order to enforce the second difference constraint and hope that for each Sbox, we can add equations such that E_Δ remains consistent. Note that the equations in E_Δ in each stage of the algorithm depend on the order in which we consider the active Sboxes, and on the order in which we consider the possible affine subsets of input differences for each Sbox. Thus, if we reach an Sbox for which we cannot add equations in order to enforce the second constraint (while maintaining the consistency of E_Δ), we can simply change the order in which we consider the active Sboxes, or the order in which we consider the affine subsets for each Sbox, and try again. Since we cannot predict in advance the orderings that give the best result, we choose them heuristically, as described in Appendix A.

4.4. *The Value Phase*

In case the difference phase procedure described above succeeds, it actually outputs an affine subspace of candidate input differences, rather than a single value for Δ_I. Next, we can commit to a specific value for Δ_I and run the value phase, hoping that the set of all linear equations defined by the value constraints has a solution. Namely, we allocate another system of equations, which we denote by E_M, and add the equations on $\overline{M}^1$ that enforce the first value constraint. We then add the additional linear equations that enforce the second value constraints for all the Sboxes, and output the solution to the system, if it exists. However, once again, this approach is too strict, and may force us

to repeat the value phase a huge number of times with different values for Δ_I, until we find a solution. Thus, we do not choose a single value for Δ_I in advance. Instead, we reduce the linear subset of candidates for Δ_I gradually by fixing the input difference to each one of the active Sboxes, until a single value for Δ_I remains. Thus, we continue to maintain E_Δ throughout the value phase, and iteratively add the additional 2 equations which are required to uniquely specify a 5-bit input difference for each active Sbox, among the 2-dimensional affine subsets chosen in the difference phase. Once we fix the input difference to an Sbox, we immediately obtain linear equations on $\overline{M}^1$, and we can check their consistency with the current equations in E_M. In case the equations in E_M are not consistent for a certain Sbox, we can try to choose another input difference for it. This gives different equations on $\overline{M}^1$, which may be consistent and allow us to continue the process.

Similarly to the difference phase, the equations in E_M in each stage of the algorithm depend on the order in which we consider the active Sboxes, and on the order in which we consider the possible input differences for each Sbox. Thus, once again, if at some stage of the value phase we cannot add any consistent equations to E_M, we can change one of these orderings and try again, hoping to obtain a valid solution.

We stress again that both phases of the algorithm are not guaranteed to succeed. The success of each phase depends on the target difference, and on orderings which are chosen heuristically. As a result, we may have to iterate both phases of the algorithm an undetermined number of times with modified orderings, hoping to obtain better results.

5. Application of the Target Difference Algorithm to Round-Reduced Keccak

Since we would like to use the target difference algorithm in order to find collisions and near-collisions in Keccak, it is crucial to verify the algorithm's success on target differences which lead to these results. Thus, before we run the algorithm, we have to find such high probability differential characteristics, and to obtain the target differences which are likely to be the most successful inputs to the algorithm. As described in the introduction, once we find a high probability differential characteristic with a low Hamming weight starting state difference ΔS_2, we extend it backwards to obtain the target difference $\Delta_T = \Delta S_1$ (while maintaining its high probability). We then use the target difference algorithm to link the extended characteristic backwards to the initial state of Keccak's permutation, with an additional round. Thus, any low Hamming weight characteristic for r rounds of Keccak's permutation can be used to obtain results on a round-reduced version of Keccak with $r + 2$ round. Specifically, in this section we demonstrate how we use 2-round characteristics in order to find collisions for 4 rounds of Keccak-224 and Keccak-256, and how to use 3-round characteristics in order to find near-collisions for 5 rounds of these Keccak versions.

5.1. *Searching for Differential Characteristics*

We reuse the notion of a *column parity kernel* or *CP-kernel* that was defined in the Keccak submission document [5]: a 1600-bit state difference is in the CP-kernel if all of its columns have even parity. It is easy to see that such state differences are fixed points of the function θ, which does not increase their Hamming weight. Since ρ and π just

reorder the bits of the state, the application of L to a CP-kernel does not change its total Hamming weight. In addition, there is a high probability that such low Hamming weight differential states are fixed points of χ. Thus, when we start a differential characteristic from a low Hamming weight CP-kernel, we can extend it beyond the Sbox layer, χ, to one additional round of the Keccak permutation, with relatively high probability and without increasing its Hamming weight. However, extending such a characteristic to more rounds in a similar way is more challenging, since we have to ensure that the state difference before the application of θ remains in the CP-kernel at the beginning of each round.

Using Previous Results In [9], [12] and [18], the authors propose algorithms for constructing low Hamming weight differential characteristics for Keccak. These algorithms successfully find differential characteristics that stay in the CP-kernel for 2 rounds (named *double kernel trails* in [18]), some of which lead to collisions on the n-bit extract taken from the final state after 2 rounds, with high probability. However, when trying to extend each one of these characteristics by another round, the state difference is no longer in the CP-kernel and thus its Hamming weight increases significantly (from less than 10 to a few dozen bits). Nevertheless, the Hamming weight of the characteristics is still relatively low, and they can lead with reasonably high probability to near-collisions on the n output bits extracted. Beyond 3 rounds, the Hamming weight of the characteristics becomes very high (more than 100), and it seems unlikely that they can be extended to give collisions or near-collisions with reasonable probability. The currently known double kernel differential trails only extend forward to at most three rounds with reasonably high probability (higher than 2^{-100}).

Our attacks on round-reduced Keccak make use of the type of differential characteristics that were found in [9], [12] and [18], namely low Hamming weight characteristics that stay in the CP-kernel for 2 rounds. The double kernel trails with the highest probability have Hamming weight of 6 at the input to the initial round, and due to their low hamming weight, we could easily find all these characteristics within a minute on a standard PC. There are 571 such characteristics out of which, 128 can give collisions for Keccak-224 and 64 can give collisions for Keccak-256. However, when trying to extend the characteristics by an additional round, we were not able to find any characteristic that gives collisions for Keccak-224 (or Keccak-256) with reasonable probability. Thus, our best 3-round characteristics lead only to near-collisions, rather than collisions. The characteristics that give the near-collisions with the smallest difference Hamming weight for Keccak-224 and Keccak-256 are, again, double kernel trails with 6 non-zero input bits. The best 3-round characteristics for Keccak-224 lead to near-collisions with a difference Hamming weight of 5, and for Keccak-256, the best 3-round characteristics leads to a near-collision difference Hamming weight of 8. Examples of these characteristics are found in Appendix C.

Extending the Characteristics Backwards Since the characteristics that we use start with a low Hamming weight state difference, we can extend them backwards by one round without reducing their probability significantly (as done in [12]): we take this low Hamming weight initial state difference ΔS_2, and choose a valid state difference input to the previous Sbox layer $\Delta S_{1.5}$ which could produce it. We then apply L^{-1},

and obtain a new initial state difference for the extended characteristic ΔS_1, which serves as a target difference Δ_T for our new algorithm. Note that the target difference is not in the CP-kernel (otherwise, we would have found a low Hamming weight differential characteristic that stays in the CP-kernel for 3 rounds). Thus, when we apply L^{-1} to $\Delta S_{1.5}$, we usually obtain a roughly balanced target difference, with only a few non-active Sboxes. This is significant to the success of the target difference algorithm, which strongly depends on the number of active Sboxes in the target difference (as demonstrated in Sect. 4.1). In case the target difference obtained from a characteristic has too many non-active Sboxes, we can try to select another target difference for the characteristic, by tweaking $\Delta S_{1.5}$.

Assuming that the algorithm succeeds and we obtain a sufficiently large linear subspace of message pairs (such that it contains at least one pair whose difference evolve according to the characteristic), we can find collisions for 4 rounds and near-collisions for 5 rounds of Keccak-224 and Keccak-256. For example, if we have an extended characteristic which can give collisions for 3 round of Keccak-256 with probability 2^{-24}, we need a linear subspace which contains at least 2^{24} message pairs in order to find a collision on 4-round Keccak-256 with high probability.

5.2. *Applying the Target Difference Algorithm to the Selected Differential Characteristics*

We tested our target difference algorithm using a standard PC, on dozens of double-kernel trails with Hamming weight of 6. For each one of them, after tweaking $\Delta S_{1.5}$ at most once, we could easily compute a target difference where all of the 320 Sboxes are active. We then ran the target difference algorithm on each one of these targets. For both Keccak-224 and Keccak-256, the target difference algorithm eventually succeeded: the basic procedure of the difference phase always succeeded within the first two attempts (after changing the order in which we considered the Sboxes), while the value phase was more problematic, and we had to iterate its basic procedure dozens to thousands of times in order to find a good ordering of the Sboxes and obtain results. For Keccak-224, the algorithm typically returned an affine subspace of message pairs with a dimension of about 100 within one minute. For Keccak-256, the dimension of the affine subspaces of message pairs returned was typically between 35 and 50, which is smaller compared to the typical result size for Keccak-224 (as expected since we have fewer degrees of freedom). In addition, unlike Keccak-224, for Keccak-256 we had to rerun the algorithm (starting from the difference phase) a few times, when the value phase did not seem to succeed for the choice of candidate input difference subset. Hence, the running time of the algorithm was typically longer—between 3 and 5 minutes, which is, of course, still practical.

5.3. *Obtaining Actual Collisions and Near-Collisions for Round-Reduced Keccak-224 and Keccak-256*

Obtaining Collisions After successfully running the target difference algorithm, we were able to find collisions for 4-round Keccak for each tested double-kernel trail with Hamming weight of 6 (which leads to a collision). Since the probability of each one of these differential characteristics is greater than 2^{-30}, the probability that a random pair

which satisfies its corresponding target difference leads to a collision, is greater than 2^{-30}. Thus, we expect to find collisions quickly for both Keccak-224 and Keccak-256, once the target difference algorithm returns a set of more than 2^{30} message pairs. However, even though the subsets we used contained more than 2^{30} message pairs, we were not able to find collisions within several of these subsets for Keccak-224, and for many of the subsets for Keccak-256. As a result, we had to rerun the target difference algorithm and obtain additional sets of message pairs, until a collision was found. Thus, the entire process of finding a collision typically takes about 2–3 minutes for Keccak-224, and 15–30 minutes for Keccak-256. The reason that there were no 4-round collisions within many of the subsets of message pairs is the incomplete diffusion of the Keccak permutation within the first two rounds. Since our subsets of message pairs are relatively small (especially for Keccak-256), and the values of all the message pairs within a subset are closely related, some close relations between a small number of bits still hold before the Sbox layer of the second round (e.g., the value of a certain bit is always 0, or the XOR of two bits is always 1). Some of these non-random relations make the desired difference transition into the second Sbox layer impossible, for all the message pairs within a subset. We note that we were still able to find collisions rather quickly, since it is easy to detect the cases where the difference transition within the second Sbox layer is impossible[4] (which allowed us to immediately rerun the target difference algorithm). In addition, when this difference transition is possible, we were always able to find collisions within the subset. Two concrete examples of colliding message pairs for Keccak-224 and Keccak-256 are given in Appendix D.

Obtaining Near-Collisions In order to obtain near-collisions on 5-round Keccak-224 and Keccak-256, we again start by choosing suitable differential characteristics. Out of all the characteristics that we searched, we chose the differential characteristics described in Appendix C, which lead to near-collisions of minimal Hamming weight for the two versions of Keccak. The results of the target difference algorithm when applied to targets chosen according to these characteristics, were similar to the results described in Sect. 5.2. However, compared to the probability of the characteristics leading to a collision, the probability of these longer characteristics is lower: the probability of the characteristics are 2^{-57} and 2^{-59} for Keccak-224 and Keccak-256, respectively. Thus, obtaining message pairs whose differences propagate according to these characteristics, and lead to 5-round near-collisions, is more difficult than obtaining collisions for 4 rounds of Keccak-224 and Keccak-256. However, for each such main characteristic, there are several secondary characteristics which diverge from the main one in final two rounds and give similar results. Thus, the probabilities of finding near collisions with a small Hamming distance for 5 rounds of Keccak-224 and Keccak-256 are higher than the ones stated above. In addition, by using some simple message modification techniques within the subsets returned by the target difference algorithm, we were able to

[4] In order to detect that the difference transition within the second Sbox layer is impossible for all the pairs in our subset, we try several arbitrary pairs in the subset, and observe if at least one has the desired difference after two rounds. Since we only need to check one Sbox layer transition, we expect that if this transition is indeed possible, we will find a corresponding message pair very quickly. Otherwise, we have to find a different set of message pairs by running the difference phase again.

further reduce the workload of finding conforming message pairs. Thus, for Keccak-224, we obtained near-collisions with a Hamming distance of 5, which is the same as the output Hamming distance of the main characteristic that we used. For Keccak-256, the main characteristic that we used has an output Hamming distance of 8, but we were only able to find message pairs which give a near-collision with a slightly higher Hamming distance of 10. All of these near-collisions were found within a few days on a standard PC. Examples of such near-collisions are given in Appendix D.

5.4. *Applying the Target Difference Algorithm to Other Versions of Keccak*

Although there are only four official versions of Keccak, the hash function is defined for any capacity c and rate r such that $c + r = 1600$ (and also for other state sizes, which we do not consider in this paper). Thus, it is interesting to apply the target difference algorithm different versions of Keccak, and estimate the maximal value of c (or minimal value of r) for which the algorithm is able to solve random challenges.

We applied the target difference algorithm for Keccak with different values of c on dozens of randomly chosen target differences. According to our simulations, the algorithm is able to quickly solve (within 10 minutes) a substantial fraction of the challenges (of at least 10 %) for values of c close to 650 (for which the algorithm has 300 degrees of freedom). For values of c which are larger than 650, the success rate of the algorithm seems to decrease rapidly as c grows. Thus, it is very unlikely that the current version of our algorithm can succeed in reasonable time to solve challenges for Keccak-384 (where $c = 768$, and there are only a few dozens of available degrees of freedom). For Keccak-512 (where $c = 1024$), only a negligible fraction of target differences have a solution, and it is not possible to solve an overwhelming majority of them (without exploiting additional degrees of freedom in multi-block messages).

Solving Keccak Collision Challenges In 2011, the Keccak team announced challenges for 48 reduced-round Keccak instances [19]. By applying our techniques to two of these instances, we were able to obtain the best known collision attacks for three and four rounds (with a capacity of $c = 160$ bits).

6. Conclusions and Future Work

In this paper, we presented practical collision and near-collision attacks on reduced-round variants of Keccak-224 and Keccak-256. Our attacks are based on a novel target difference algorithm, which is used to link high probability differential characteristics for the Keccak internal permutation to legal initial states of the hash function. Consequently, we were able to significantly improve the best known previous results on Keccak, by doubling (from 2 to 4) the number of rounds for which collisions can be found in a practical amount of time.

Our target difference algorithm is clearly limited by the number of available degrees of freedom, and it seems difficult to extend it to reach target differences spanning 2 or more rounds of the Keccak permutation. However, it seems very likely that the algorithm will be useful in the future if longer high probability differential characteristics are found for the Keccak permutation. In particular, finding new high probability differential characteristics, starting from a low Hamming weight state difference and extending forwards more than 3 rounds, remains a challenging task.

Acknowledgements

The second author was supported in part by the Israel Science Foundation through grant No. 827/12.

Appendix A. Details of the Target Difference Algorithm

In this section, we give the full details of the target difference algorithm. The general structures of the difference phase and the value phase of the algorithm are similar: both include a main procedure, which iterates a basic procedure, until it succeeds.

A.1. *Calculating a Set of Candidate Input Differences*

We give the details of the difference phase, which finds a set of candidate input differences $(\Delta_{I_1}, \Delta_{I_2}, \ldots, \Delta_{I_i}, \ldots)$, given a target difference Δ_T. To find such a set, we have to simultaneously enforce the two difference constraints. As described in Sect. 4, we maintain a linear system of equations, which we denote by E_Δ, whose solution basis spans some affine subspace of potential values for $L(\Delta_I)$. We can easily obtain a basis for the set of potential values for Δ_I by multiplying each element of the basis for $L(\Delta_I)$ by the matrix which defines L^{-1}.

We assume that Δ_T contains t active Sboxes and $320-t$ non-active Sboxes whose input and output differences are zero. Before executing the algorithm, we have to initialize a data structure that is used repeatedly in both phases of the target difference algorithm. This data structure stores the subsets of input differences for each active Sbox (which are precalculated per possible non-zero Sbox output difference, as described in Property 3 in Sect. 4.1). In addition, it specifies the order in which we consider the specific subsets for each one of the t active Sboxes, and the order in which we consider the active Sboxes themselves. As described in Sect. 4, these orderings have a significant effect on the success and running time of the algorithm.

The input difference subsets, for each of the t active Sboxes, are stored in a sorted *input difference subset list*, or *IDSL*. The heuristic algorithm we use to pick the sorting order of each IDSL is specified in Appendix A.3, but since it is irrelevant in this section, we assume some arbitrary ordering of each one of the IDSLs. In addition, each one of the IDSLs contains a pointer to an input difference subset, initialized to point at the first element in the list. The IDSLs are stored in the main *input difference subset data structure*, or *IDSD*. The IDSD contains t entries (one entry per active Sbox), sorted according to an IDSD order (which may differ from the natural order of the Sboxes). The initial IDSD order is chosen randomly, and is updated during both phases of the algorithm.

The Basic Procedure The steps of the basic procedure of the difference phase are given below:

1. Initialize an empty linear equation system E_Δ with 1600 variables for the unknown bits of $L(\Delta_I)$.
2. By inverting L, compute the coefficients of the $c+8$ linear equations that equate the last $c+8$ bits of Δ_I to zero, and add the equations to E_Δ.

3. For each of the $320-t$ non-active Sboxes, add to E_Δ the 5 equations which equate the corresponding bits to zero.
4. Check if E_Δ is consistent, and if not, output "Fail".
5. Iterate over the t active Sboxes according to the IDSD order, and for each one of them:
   (a) Obtain the current 2-dimensional subset from the Sbox IDSL according to the pointer, and obtain the $5-2=3$ affine equations that define this subset.
   (b) If E_Δ is consistent after adding the 3 equations, add the equations and continue to the next Sbox. Otherwise, continue to the next subset in the Sbox IDSL, by incrementing the pointer and going to step (a). If the end of the IDSL is reached, output "No Solution".
6. Output E_Δ.

Our initial linear equation system contains $c+8$ linear equations. Since each 2-dimensional subset of an active Sbox contributes 3 affine equations, the active Sboxes contribute a total of $3t$ equations. The non-active Sboxes contribute $5(320-t)$ linear equations, since each non-active Sbox contributes 5 equations. Altogether, we have $c+8+3t+5(320-t)=1600+8+c-2t$ linear equations in E_Δ. Thus, for t large enough, if the algorithm succeeds, the solution subset is of dimension at least $1600-(1600+8+c-2t)=2t-c-8$ (for example, for $t=310$ and $n=224$, we get a solution subset of dimension at least $620-448-8=164$). If all the coefficients of the equations were chosen uniformly at random, then for $t>n+4$ we expect no dependencies among the equations, and thus the dimension of the solution subset is exactly $2t-c-8$. However, our equations are clearly not random, and thus we can only verify the procedure's success experimentally.

Note that the equations added in step 2 are independent, since L is invertible (and thus all the 1600 linear equations that define L^{-1} are independent) and equating each bit to zero reduces the remaining dimension by 1. In addition, the equations of all the Sboxes are independent, since each Sbox gives equations on a distinct set of 5 variables. However, there is no guarantee that the combined equation set is independent, or consistent. In addition, note that dependencies among the equations may foil the procedure in case they are inconsistent. On the other hand, in case the dependent equations are consistent, they do not decrease the dimension of the solution subset, which gives us more flexibility in the value phase.

The Main Procedure In case the basic procedure outputs "Fail", then clearly there are no 1600-bit valid state pairs that satisfy the target difference, and the algorithm aborts. If the procedure outputs "No Solution", then our current choice of 2-dimensional subsets gives inconsistent equations, and we change our choice, using a heuristic algorithm, hoping to obtain better results. The main idea behind the heuristic is to change the order in which we consider the Sboxes, such that the "problematic" groups of Sboxes (which tend to produce the inconsistent equations) are pushed to the front of the IDSD order. When considering the first few Sboxes, the number of equations in E_Δ is relatively small, and we have more options to choose consistent equations for these Sboxes. Thus, when the "problematic" Sbox groups are considered first, it seems more likely that the basic procedure will succeed. The main procedure of the difference phase is given below, where $T1$ is some fixed constant threshold:

1. Initialize a counter to 0. Initialize the IDSD by resetting all the IDSL pointers to the beginning of the lists, and randomizing the IDSD Sbox order.
2. Execute the basic procedure. If the procedure succeeds, output the current E_Δ. If the procedure outputs "Fail", abort. Otherwise (i.e., when the procedure outputs "No Solution"), go to step 3.
3. Increment the counter. If the counter's value is equal to $T1$, go to step 1.
4. Reset the pointer of the failed Sbox IDSL to its value before the last basic procedure.
5. Change the IDSD order by bringing the failed Sbox to the front (and pushing the rest of the Sboxes one position backwards).
6. Go to step 2.

Since there is nothing that ensures that the main procedure of the difference phase halts, it may run a very long time without finding a solution (either because there is no solution, or it is extremely difficult to find one). The procedure may even enter a loop by setting the state of the IDSD in steps 4 and 5 to a previously considered state. Thus, after $T1$ executions of the basic procedure which output "No Solution", we start over by re-randomizing the IDSD order and resetting all the IDSL pointers.

A.2. *Choosing Δ_I and the Set of Message Pairs*

After we successfully enforce the difference constraints on the input difference Δ_I in the difference phase and obtain a set of candidate input differences, we are ready to execute the value phase. As described in Sect. 4, in this phase we iteratively narrow down the dimension of the set of candidate input differences, until only one candidate for Δ_I remains. While doing so, we simultaneously narrow down the set of pairs of possible initial states: we start with a set containing all the possible 1600-bit initial state pairs with the last $c+8$ bits equal to $p \parallel 0^c$, and finally obtain the set $\{(\overline{M}_1^1, \overline{M}_1^2), (\overline{M}_2^1, \overline{M}_2^2), \ldots, (\overline{M}_k^1, \overline{M}_k^2)\}$, where all pairs have the difference Δ_I and satisfy the target difference (the actual message can be easily obtained from the corresponding 1600-bit value, by removing these fixed last $c+8$ bits).

Similarly to the procedure that we use for the difference phase, we continue to maintain the set of linear equations which define the set of possible input differences, E_Δ. We iterate over the active Sboxes according to the current IDSD order, and for each Sbox we choose a 5-bit input difference that is consistent with the current equations in E_Δ. We then narrow the set of possible input differences down by adding the additional $5-3=2$ equations that define this 5-bit input difference to E_Δ.

In addition to maintaining E_Δ, we maintain another system of linear equations, whose solution basis spans the set of possible values for $L(\overline{M}^1)$ (the 1600-bit state obtained by invoking L on the first message in each one of the ordered pairs). This set of equations is used to simultaneously enforce the two value constraints given in Sect. 4, and is denoted by E_M. Given E_M, we can easily obtain a basis for the set $\overline{M}_i^1 \ \forall i \in \{1, 2, \ldots, k\}$, by multiplying each element of the basis for $L(\overline{M}_i^1)$ by the matrix that defines L^{-1}.

Initially, we add to E_M all the $c+8$ linear equations that enforce the first value constraint on the first initial state of the ordered pairs. We then enforce the second value constraint on each Sbox independently: for each Sbox with output difference δ^{out}

(computed from Δ_T), once we determine its input difference δ^{in}, we also obtain the linear equations that restrict the values of the 5 variables of $L(\overline{M}^1)$ to the affine subset $A(\delta^{\text{in}}, \delta^{\text{out}})$ (where $A(\delta^{\text{in}}, \delta^{\text{out}})$ denotes the subset of values $\{v_1, v_2, \ldots, v_l\}$ defined in Property 2 of Sect. 4.1). We then enforce the second value constraint on the Sbox by adding these linear equations to E_M. Note that in the difference phase, we ensured that in each stage of the value phase, for each Sbox with a fixed output difference, there exists some input difference whose equations can be added to E_Δ, while maintaining its consistency. In addition, we ensured that the affine subset of values that corresponds to these input and output differences is non-empty. However, we did not ensure that the equations that restrict the values of the variables of $L(\overline{M}^1)$ to this subset are consistent with the equations that are currently in E_M.

The Basic Procedure The steps of the basic procedure of the value phase are given below:

1. Initialize an empty linear equation system E_M with 1600 variables for the unknown bits of $L(\overline{M}^1)$.
2. By inverting L, compute the coefficients of the $c + 8$ linear equations that equate the last $c + 8$ bits of $L(\overline{M}^1)$ to $p \parallel 0^c$ (p is the 8-bit pad 10000001), and add the equations to E_M.
3. Iterate the t active Sboxes according to the IDSD order:
   (a) Using Δ_T, obtain δ^{out} for the Sbox.
   (b) Obtain all the Sbox input differences that are consistent with E_Δ (denoted by $\{\delta_i^{\text{in}}\}$), and sort them in descending order according to the size of the affine subset of values that satisfy the input-output difference (i.e., make sure that $\text{DDT}(\delta_i^{\text{in}}, \delta^{\text{out}}) \geq \text{DDT}(\delta_{i+1}^{\text{in}}, \delta^{\text{out}})$).
   (c) Iterate the sorted input differences obtained in the previous step, starting with δ_1^{in} (which gives the largest affine subset of values):
      (i) Using the current input difference, δ_i^{in} obtain the linear equations that define the affine subset $A(\delta_i^{\text{in}}, \delta^{\text{out}})$.
      (ii) If E_M is consistent with these linear equations, add the equations to E_M. In addition, add the additional equations on the input difference (that were not added in the difference phase), which are defined by δ_i^{in}, to E_Δ, and continue to the next Sbox in step 3.
      (iii) Otherwise, continue to the next input difference in step 3(c). If no more input differences remain, output "No Solution".
4. Output the fully determined 1600-bit value of Δ_I, in addition to the equation system that defines the message values, E_M.

The algorithm makes a heuristic choice in step 3(c), by considering the input differences for each Sbox, starting with the one that gives the largest affine subset of values. The idea is to keep the number of equations in E_M as small as possible, in order to reduce the probability of failure. In addition, when we generate fewer independent equations in E_M at the end of the process, we get a larger set of pairs that satisfy the target difference.

The Main Procedure Similarly to the basic procedure of the difference phase, there is no guarantee that this procedure succeeds. Hence, we may need to repeat it several times with different choices of input differences for each Sbox, which result in different systems of equations, E_M. The steps of the main procedure of the value phase are given below, where $T2$ is some fixed constant threshold:

1. Initialize a counter to 0.
2. Set E_Δ to the equation system returned by the last successful execution of the difference phase.
3. Execute the basic procedure of the value phase. If the procedure succeeds, output Δ_I and E_M. Otherwise (i.e., the procedure outputs "No Solution"), continue to step 4.
4. Increment the counter. If the counter's value is equal to $T2$, output "No Solution".
5. Change the IDSD order by bringing the failed Sbox to the front (and pushing the rest of the Sboxes one position backwards).
6. Go to step 2.

Note the difference between the structures of the two phases of the algorithm: the only input to the difference phase is the value of Δ_T (which we assume to be fixed). Thus, unless it returns "Fail", there is no reason to stop its execution at any point. On the other hand, the value phase depends on the output of the difference phase, E_Δ. Since the particular choice of candidate set for Δ_I may foil the procedure, we terminate the value phase after the number of unsuccessful executions of its basic procedure reaches some threshold, $T2$. In this case, we restart the difference phase, hoping that another choice of candidate set for Δ_I will give better results.

A.3. *Sorting the Input Difference Subset Lists*

In this section, we give the details of how we sort the IDSLs, by specifying how to compare two input difference subsets in the list. Given a non-zero Sbox output difference, all the maximal possible input difference subsets are of dimension 2, and add 3 linear equations to E_Δ. This does not give an obvious reason to prefer one subset over another in the difference phase. However, the input differences within each input difference subset give affine subsets of different sizes: there are 20 specific non-zero Sbox output differences, δ^{out}, whose DDT entries $\text{DDT}(\delta^{\text{in}}, \delta^{\text{out}})$, have values of 2, 4, and 8. For the remaining 11 output differences, the non-zero DDT entries attain only the values 2 and 4. In the value phase, we prefer input differences that give large subsets of values. Thus, in the difference phase, we give precedence to input difference subsets containing such input differences: assume that Δ_T assigns a specific Sbox an output difference δ^{out}, and we want to compare two input difference subsets $\{\delta_1^{\text{in}}, \delta_2^{\text{in}}, \delta_3^{\text{in}}, \delta_4^{\text{in}}\}$ and $\{\delta_5^{\text{in}}, \delta_6^{\text{in}}, \delta_7^{\text{in}}, \delta_8^{\text{in}}\}$ such that $\text{DDT}(\delta_1^{\text{in}}, \delta^{\text{out}}) \geq \text{DDT}(\delta_2^{\text{in}}, \delta^{\text{out}}) \geq \text{DDT}(\delta_3^{\text{in}}, \delta^{\text{out}}) \geq \text{DDT}(\delta_4^{\text{in}}, \delta^{\text{out}}) > 0$ and $\text{DDT}(\delta_5^{\text{in}}, \delta^{\text{out}}) \geq \text{DDT}(\delta_6^{\text{in}}, \delta^{\text{out}}) \geq \text{DDT}(\delta_7^{\text{in}}, \delta^{\text{out}}) \geq \text{DDT}(\delta_8^{\text{in}}, \delta^{\text{out}}) > 0$. We first compare the sizes of the largest subsets of values, $\text{DDT}(\delta_1^{\text{in}}, \delta^{\text{out}})$ and $\text{DDT}(\delta_5^{\text{in}}, \delta^{\text{out}})$, and give precedence to the input difference subset for which the size is bigger. If the sizes are equal, we compare $\text{DDT}(\delta_2^{\text{in}}, \delta^{\text{out}})$ and $\text{DDT}(\delta_6^{\text{in}}, \delta^{\text{out}})$, and so forth.

Appendix B. An Alternative Value Phase

The value phase presented in Sect. A.2 chooses Δ_I and the set of message pairs iteratively, which gives it more options, and reduces its running time compared to more restrictive algorithms which choose Δ_I in advance. However, it has two disadvantages: it restricts the messages to one block (by assuming a zero initial value for the inner part of the state), and more significantly, has no known bound on its expected running time. We can easily overcome the first disadvantage by choosing a prefix of $m - 1$ arbitrary blocks, and calculating the inner part of the state obtained after running the permutation on the prefix. We then use the value of the inner state in order to apply the algorithm on an additional final block (which is the only one in which we use a difference between the two messages). The only change from the single-block version is that we initialize the values of the equations in step 2 of the basic procedure of the value phase according to the new inner state value.

Given that we choose a common prefix of several blocks for all message pairs, and run the target difference algorithm only on the last block, the difference in the inner states obtained after the prefix for each pair, is zero. Thus, the difference phase of the multi-block variant presented above is completely identical to the original single-block version. In fact, since the difference phase is completely independent of the inner state value, we can apply it and obtain suggestions for Δ_I before we even choose the prefix. This simple observation allows us to design a completely different value phase, whose expected complexity can be easily calculated using reasonable assumptions (and given that the difference phase succeeds).

In the alternative value phase, we first pick a fixed Δ_I from the input difference candidate set computed in the difference phase. This fixed input difference gives many linear equations that involve message bits which we can easily control, and thus we can satisfy a large portion of the equations regardless of the value of the inner part of the state. However, after fixing many message bits in order to satisfy equations, we may remain with some linear equations which only involve the inner part of the state, that cannot be directly controlled. Thus, we compute the inner part of the state for arbitrary message prefixes, hoping that one of them satisfies the remaining equations by chance. The complexity of the value phase is measured by the expected number of invocations of Keccak's internal permutation, which depends on the number of linear equations that cannot be satisfied by the message bits (i.e., involve only the variables of the inner part of the state). Thus, in the alternative value phase, we exploit the possibility to vary the inner part of the state, instead of varying the input difference Δ_I (as we do in the original value phase). The steps of the alternative value phase are given below:

1. Choose an input difference Δ_I from the candidate input differences outputted by the difference phase.
2. Initialize an empty linear equation system E_M with 1600 variables for the unknown bits of $L(\overline{M}^1)$.
3. For each Sbox:
   (a) Using Δ_I and Δ_T, compute the Sbox input and input differences $(\delta^{\text{in}}, \delta^{\text{out}})$.
   (b) Add the equations that define $A(\delta^{\text{in}}, \delta^{\text{out}})$ to E_M.

4. By inverting L, compute the coefficients of the 8 linear equations that define the padding. If these equations are consistent with E_M, add them and continue to the next step. Otherwise, go to step 1.
5. Compute the coefficients of the c linear equations that define the inner part of the state. Add them to E_M, by allocating additional c variables for their undetermined values (the vector of values of E_M contain c undetermined values).
6. Perform Gauss Elimination on E_M, and obtain the dependent equations, where the coefficients of the variables of $L(\overline{M}^1)$ are zero. These give linear equations on the c variables of the inner state.
7. Choose a common message prefix of $m-2$ blocks, each containing r bits, $M_1^*, M_2^*, \ldots, M_{m-2}^*$, where $m > 1$.
8. Compute the inner part of the state, C_{m-2}, after running the Keccak permutation on the prefix.
9. Iterate over the 2^r possible values of the message block with index $m-1$, M_{m-1}^*, in some order:
   (a) Execute the Keccak permutation on the message block M_{m-1}^* and the inner state C_{m-2}, and obtain the next inner state C_{m-1}.
   (b) Check if the equations obtained in step 6 hold for C_{m-1}. If they do, output Δ_I and E_M (without the trivial equations), in addition to the message prefix $M_1^*, M_2^*, \ldots, M_{m-1}^*$. Otherwise, go to the beginning of step 9 (choose the next value of M_{m-1}^*).

Note that in step 2, we add no equations to E_M for non-active Sboxes, and for active Sboxes we add at most 4 equations (since all the non-zero DDT entries that correspond to a non-zero output difference have a value of at least 2). Thus, after step 3, E_M contains a subset of dimension at least $1600 - (4 \cdot 320) = 320$. Since L^{-1} has a very fast diffusion, it is very unlikely that none of the initial states defined by this large subset attain the value of the 8-bit pad (and it is even more unlikely if we consider messages of different lengths, which is not necessarily a multiple of 8). Consequently, it is very unlikely that the procedure will fail even once on step 4. Thus, we ignore this possibility and assume that step 9 is the time complexity bottleneck of the procedure. Assuming that we get q dependent equations in step 6 of the attack, we obtain q linear equations on the inner state bits after $m-1$ blocks, C_{m-1}. Thus, assuming that the Keccak permutation behaves randomly, we expect to run it 2^q times in step 9, until all the q equations are satisfied. Note that we can calculate the expected running time of the algorithm only after selecting the exact value of Δ_I from the candidate set of input differences. Hence, it may be useful to run steps 1–6 several times, and choose the value of Δ_I which gives the best expected running time.

Although we cannot bound its expected complexity, in practice, the running time of the original value phase can be much lower than the running time of the alternative value phase. Thus, it is advisable to run the original value phase in case the expected running time of the alternative value phase is too high, and to calculate the expected running time of the alternative value phase in case the original value phase seems to be unsuccessful.

Appendix C. Differential Characteristics for Keccak

In this section, we give examples of 3-round differential characteristics, which lead to collisions on 4-round Keccak-224 and Keccak-256, and 4-round characteristics, which lead to near-collisions on 5-round Keccak-224 and Keccak-256.

The differential characteristics are given as a sequence of the starting state differences in each round. In all the presented characteristics, all the active Sboxes get an input difference with a Hamming weight of 1, and we assume that they produce the same differences as outputs (which occurs with probability 2^{-2}). In order to calculate the probability of the final transition, we only consider active Sboxes which effect the output bits (since we truncate the final state to obtain the hashed output). Each state difference is given as a matrix of 5×5 lanes of 64 bits, ordered from left to right, where each lane is given in hexadecimal using the little-endian format. The symbol '-' is used in order to denote a zero 4-bit difference value. For example, consider the second state difference in Characteristic 1: each of the first two lanes has a zero difference, and only the LSB of the third lane contains a non-zero difference.

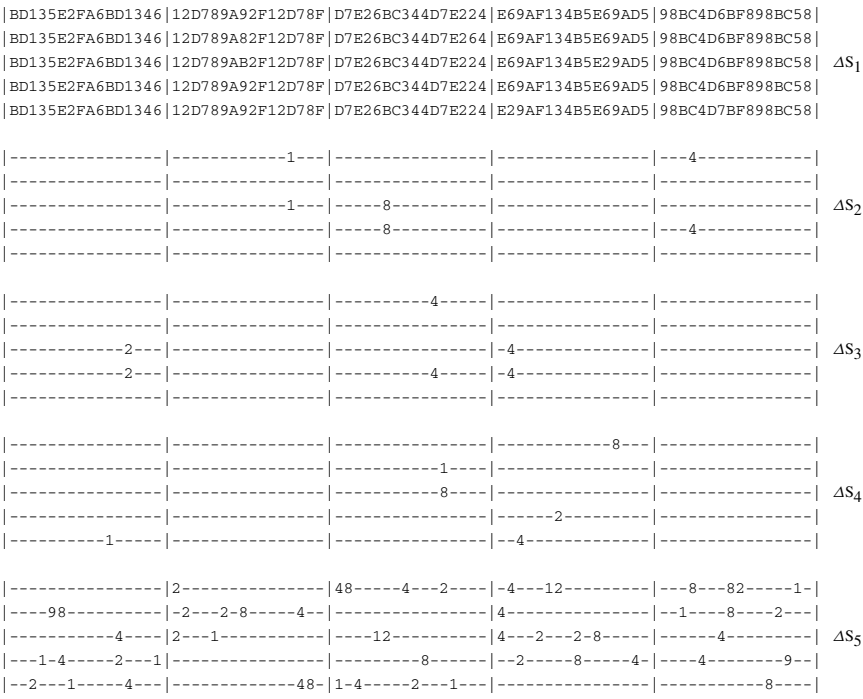
Appendix D. Actual Collisions and Near-Collisions for Round-Reduced Keccak-224 and Keccak-256

We give several examples of collisions and near-collisions for Keccak-224 and Keccak-256. The padded messages and output values are given in blocks of 32-bits ordered from left to right, where each block is given in hexadecimal using the little-endian format.

```
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2656F26B|357C4789AF3-6AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2656F26B|357C4789AF3-6AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2676F26B|357C4789AF3-6AF1|78D3526BC4A74C4D| ΔS1
|26978AF134CB835E|AF224C4D78366789|C4DAE35E265EF26B|357C4789AF3-4AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF226C4D78366789|C4DAE35E2656F26B|35FC4789AF3-6AF1|78D3526BC6A74C4D|

|----------------|----------------|---------------1|-------4--------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------| ΔS2
|----------------|----------------|----------------|-------4--------|----8-----------|
|----------------|----------------|---------------1|----------------|----8-----------|

|----------------|----------------|----------------|--8-------------|2---------------|
|4---------------|----------------|----------------|----------------|2---------------|
|----------------|----------------|----------------|--8-------------|----------------| ΔS3
|----------------|----------------|----------------|----------------|----------------|
|4---------------|----------------|----------------|----------------|----------------|

|----------------|----------------|----------------|----------------|----------------|
|-----------8----|-----------2----|----------------|----------------|----------------|
|----------------|----------------|-----------1----|----------------|-----------1----| ΔS4
|---------1------|-------4--------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

The probability of each one of the first two transitions is 2^{-12}. The probability of the third transition is 1, since there are no active Sboxes which affect the output.

Characteristic 1: A 3-round characteristic leading to collisions on Keccak-224 and Keccak-256 with probability 2^{-24}



The characteristic leads to near-collisions with a Hamming distance of 5 for Keccak-224, and 8 for Keccak-256. The probability of each one of the first three transitions is 2^{-12}. The probability of the final transition is 2^{-21} for Keccak-224 and 2^{-23} for Keccak-256. The total probability is 2^{-57} for Keccak-224 and 2^{-59} for Keccak-256.

Characteristic 2: A 4-round characteristic leading to near-collisions on Keccak-224 and Keccak-256

M1=
C4F31C32 4C59AE6D 5D19F0F4 25C4E44B D8853032 8D5E12F2 BB6E6EE2 27C33B1E 6C091058 EB9002D5
3BA4A06F 4A0CC7F1 CCB55E51 8D0DD983 2B0A0843 9B21D3B0 53679075 526DDED2 48294844 6FF4ED2C
1ACE2C15 471C1DC7 D4098568 F1EBF639 EAF7B257 09FDAE87 688878E6 4875EB30 C9C32D80 3C9E6FCB
3C2ABCFA E6A4807B 2AB281B8 812332B3

M2=
A4D30EF7 80BB8F69 90C048DF EB7213B9 A6650424 3A65F63E 8C268881 B651B81F AADAFA3C EE2CA5C3
DB78EAC2 C8EAE779 442F9C35 3221E287 B3017A5A 90790712 1B1C8BDC E08B10A8 9A9D25CA 1BE7AAAC
4E2F3E9C 73717DAD 5566015A A198CFB9 5A1CA8C2 A0E3348A AE6C0BB1 3980F9E4 A4FA8B91 6E81A989
89A9BCAA E12BF1F1 30EF9595 812E8B45

Output=
61FB1891 F326B6D5 24DD94DF 73274984 05DA9A1D 3FD359B9 78B8393B F2E7990B

The messages were found using the target difference algorithm on the target difference given by Characteristic 1.

Collision 1: A collision for 4-round Keccak-256

```
M1=
FAC7AC69 2710BE04 8462C382 7ABF1BF9 D065CD30 DB64DEB8 1410CD30 C837D79B 22E446B7 31E9BD55
A6B2074C C86E32CC DE50A10A F7BAAA58 D96CBC88 9FBD75F6 5E0D735A D22AA663 16A574AA 7DB08692
558AB029 109B4D30 86CE5DCA 13A295C7 E7C9D94B 648794D2 62EE3CF8 69439337 8CAB9F15 AC7C3267
90F41CBE A20E6893 B4781F24 0BA37647 F29A67A0 81F628D0

M2=
CE5FBC81 47710FCC 462C92E0 48F5D2CF F92F6EC3 053E64E1 570780B9 F838EC54 8F74809F 66B4AC6F
70DD1843 BF34F0C5 5010C89A D8791148 D5CC073E 3239AEBC 7DF48D79 0EC7767B FB081604 AFA975B9
F8EFAE0F ED793473 479E931C F2F80A74 7192D08F 5EB5AB27 F1CAC04E F394232D 48656B2A A3471644
DB74E60A 05FB3B18 41DC27C3 8384BF53 32534C3E 811C00B5

Output=
826B10DC 0670E4E1 5B510CDA AB876AA8 B50557ED 267932FB AA4D38E8
```

The messages were found using the target difference algorithm on the target difference given by Characteristic 1.

Collision 2: A collision for 4-round Keccak-224

```
M1=
23296F07 44536A2B 16E1E363 09B509F9 639CA324 2B834133 61457E6D 9CF07597 6797B3D4 D1715ABA
6D8F4F9F 70D12920 E014BB37 54C32ADE 6117B3FB 30114566 4BA7D70A 00F055F0 71CFFDD4 B53F2563
E223A16D CC8DDAC4 7A59836B A53FBDDE 9FFEC45F 6A3476DC 7349BB92 56AF6E92 83866932 56624032
A936E410 60AC00FA 7E7C61F9 81583CAC

M2=
49D48DE2 9FA843CA 747C88E0 55425134 098CA5B3 C97DC68A B82BC6FD 0F864996 26B13425 D9F73B75
932CD02F FB12E036 47706100 9DEFFFE4 79435F9C DA727EF0 D9CA67C6 520BE2D1 19CF3933 3136D1A9
EEBEA9DD 150CA247 D494BF4A 492EFB26 11CB4C8D F5A10A05 69128FF4 B142742F CA59FE32 4FE68436
068F76AB 041A673E 461575B5 81AA2A54

Output1=
407D4466 FEA8B231 EC968181 DF902165 23C219FF 54571D70 2800F506 E818644B

Output2=
407D4466 FEA8B231 EC928181 FF902165 23C019FF 1C571D74 2800F516 E810656B
```

The messages were found using the target difference algorithm on the target difference given by Characteristic 2.

Near-Collision 1: A near collision with Hamming distance of 10 for 5-round Keccak-256

```
M1=
7DBC1AA9 62A70B2A C2BDAF81 4A4D484B 672F6FAF ED312C83 24BC1974 16946039 6B46EDF6 1AE571A0
EDA59D6E 7561766D 8F0B4C57 3C05C569 715B7DF9 53F81761 F6D43507 6E040495 9B5C08AB 5130BA66
22AF7F5C 755840F2 2E893F59 4C4A730F 8C4F425D 182F8D00 E98515ED E29251AD 853AB863 DC46A7AC
9FB7BB08 14767EFC 5345C7AF AA774E81 8A01A570 81D65453
```

```
M2=
5659C936 AF3BA787 809C1CE6 B287F81B E0A5E769 ECCEB8A0 72506F44 1A1B2A02 EE9AE408 D16A9358
BF03C4D6 90845C46 0C0441CC 8203EA8D 6D122EB1 9193F64F 55C3A6A7 47377ED6 D26E806F DEC2CBF8
A3B8949E A91B248D 420B13BC BEAB4166 EE348CF6 DB6CCD82 122F6BDA 2FBFA7E4 75E8A429 F397BC46
7E9DE824 6A973A22 371FD02D 92035083 267D1C7A 812EDE70

Output1=
85373497 97D871C2 FBD0A823 584C0ED4 C1B3BF4F BC408766 0584B08D

Output2=
85373497 97D871C2 FBD0A823 784C0ED4 E1B1BF5F BC408776 0584B08D
```

The messages were found using the target difference algorithm on the target difference given by Characteristic 2.

Near-Collision 2: A near collision with Hamming distance of 5 for 5-round Keccak-224

References

[1] M.R. Albrecht, C. Cid, Algebraic techniques in differential cryptanalysis, in Dunkelman [13], pp. 193–208

[2] J.-P. Aumasson, W. Meier, Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. NIST mailing list, 2009

[3] D.J. Bernstein, Second preimages for 6 (7? (8??)) rounds of Keccak? NIST mailing list, 2010

[4] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Sponge functions. Presented at the ECRYPT Hash Workshop, 2007

[5] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, The Keccak SHA-3 submission. Submission to NIST (Round 3), 2011

[6] C. Boura, A. Canteaut, Zero-sum distinguishers for iterated permutations and application to Keccak-f and Hamsi-256, in *Selected Areas in Cryptography*, ed. by A. Biryukov, G. Gong, D.R. Stinson. Lecture Notes in Computer Science, vol. 6544 (Springer, Berlin, 2010), pp. 1–17

[7] C. Boura, A. Canteaut, C. De Cannière, Higher-order differential properties of Keccak and *Luffa*, in *FSE*, ed. by A. Joux. Lecture Notes in Computer Science, vol. 6733 (Springer, Berlin, 2011), pp. 252–269

[8] A. Canteaut (ed.), *Fast Software Encryption—19th International Workshop*, FSE 2012, Washington, DC, USA, March 19–21, 2012. Lecture Notes in Computer Science, vol. 7549 (Springer, Berlin, 2012), Revised selected papers

[9] J. Daemen, G. Van Assche, Differential propagation analysis of Keccak, in Canteaut [8], pp. 422–441

[10] J. Daemen, V. Rijmen, Plateau characteristics. *IET Inf. Secur.* **1**(1), 11–17 (2007)

[11] M. Duan, X. Lai, Improved zero-sum distinguisher for full round Keccak-f permutation. Cryptology ePrint Archive, Report 2011/023, 2011

[12] A. Duc, J. Guo, T. Peyrin, L. Wei, Unaligned rebound attack: application to Keccak, in Canteaut [8], pp. 402–421

[13] O. Dunkelman (ed.), *Fast Software Encryption, 16th International Workshop*, FSE 2009, Leuven, Belgium, February 22–25, 2009. Lecture Notes in Computer Science, vol. 5665 (Springer, Berlin, 2009), Revised selected papers

[14] D. Khovratovich, Cryptanalysis of hash functions with structures, in *Selected Areas in Cryptography*, ed. by M.J. Jacobson Jr., V. Rijmen, R. Safavi-Naini. Lecture Notes in Computer Science, vol. 5867 (Springer, Berlin, 2009), pp. 108–125

[15] D. Khovratovich, A. Biryukov, I. Nikolic, Speeding up collision search for byte-oriented hash functions, in *CT-RSA*, ed. by M. Fischlin. Lecture Notes in Computer Science, vol. 5473 (Springer, Berlin, 2009), pp. 164–181

[16] F. Mendel, C. Rechberger, M. Schläffer, S.S. Thomsen, The rebound attack: cryptanalysis of reduced Whirlpool and Grøstl, in Dunkelman [13], pp. 260–276

[17] P. Morawiecki, M. Srebrny, A SAT-based preimage analysis of reduced KECCAK hash functions. Cryptology ePrint Archive, Report 2010/285, 2010
[18] M. Naya-Plasencia, A. Röck, W. Meier, Practical analysis of reduced-round Keccak, in *Progress in Cryptology—INDOCRYPT 2011*, ed. by D.J. Bernstein, S. Chatterjee. Lecture Notes in Computers Science, vol. 7107 (Springer, Heidelberg, 2011)
[19] The Keccak team, Keccak crunchy crypto collision and pre-image contest, 2011