Differential Cryptanalysis of Keccak Variants

Stefan Kölbl, Florian Mendel, Tomislav Nad, Martin Schläffer

IAIK, Graz University of Technology, Austria

Abstract. In October 2012, NIST has announced Keccak as the winner of the SHA-3 cryptographic hash function competition. Recently, at CT-RSA 2013, NIST brought up the idea to standardize Keccak variants with different parameters than those submitted to the SHA-3 competition. In particular, NIST considers to reduce the capacity to the output size of the SHA-3 standard and additionally, standardize a Keccak variant with a permutation size of 800 instead of 1600 bits. However, these variants have not been analyzed very well during the SHA-3 competition. Especially for the variant using an 800-bit permutation no analysis on the hash function has been published so far.
In this work, we analyze these newly proposed Keccak variants and provide practical collisions for up to 4 rounds for all output sizes by constructing internal collisions. Our attacks are based on standard differential cryptanalysis contrary to the recent attacks by Dinur at al. from FSE 2013. We use a non-linear low probability path for the first two rounds and use methods from coding theory to find a high-probability path for the last two rounds. The low probability path as well as the conforming message pair is found using an automatic differential path search tool. Our results indicate that reducing the capacity slightly improves attacks, while reducing the permutation size degrades attacks on Keccak.

Keywords: hash functions, SHA-3, collision attack, differential cryptanalysis

1 Introduction

In October 2012, NIST has announced Keccak [2] as the winner of the SHA-3 cryptographic hash function competition [16], which was held between 2008 and 2012 [15]. Traditionally, cryptographic hash functions take as input a string of arbitrary finite length and produce a fixed sized output of n bits. As a consequence, the following main security requirements are defined for cryptographic hash functions:

- **Preimage Resistance:** For a given output y it should be computationally infeasible to find any input x' such that $y = f(x')$.
- **Second Preimage Resistance:** For given $x, y = f(x)$ it should be computationally infeasible to find any $x' \neq x$ such that $y = f(x')$.
- **Collision Resistance:** It should be computationally infeasible to find two distinct inputs x, x' such that $f(x) = f(x')$.

Table 1. Summary of collision attacks on Keccak.

Variants	Hash Size	Capacity	Permutation	Rounds	Complexity	Reference
Keccak ($c = 2n$)	224	448	1600	4	practical	[8]
	256	512	1600	4	practical	[8]
	256	512	1600	5	2^{115}	[9]
	384	768	1600	3	practical	[9]
	384	768	1600	4	2^{147}	[9]
	512	1024	1600	3	practical	[9]
Keccak ($c = n$)	224	224	1600	4	practical	this work
	256	256	1600	4	practical	this work
	384	384	1600	4	practical	this work
	512	512	1600	4	practical	this work
	224	224	800	4	practical	this work
	256	256	800	4	practical	this work
	384	384	800	4	2^{102}	this work
other variants	all	≤ 640	1600	4	practical	this work
	all	≤ 352	800	4	practical	this work

For any ideal hash function with n-bit output size, we can find preimages or second preimages with a complexity of 2^n, and collisions with a complexity of $2^{n/2}$ using generic attacks. However, in practice the security of a hash function is not necessarily linked to the hash output size. This is addressed by the sponge construction [1] which is the underlying design principle of Keccak. The sponge construction provides an internal capacity of c bits and allows an arbitrary hash value output size of n bits. Therefore, the security is given by $s = \min(c/2, n)$ bits against (second-) preimage attacks and by $s = \min(c/2, n/2)$ bits against collision attacks. As a consequence, the SHA-3 candidates of Keccak submitted to the competition were defined with a capacity of $c = 2n$ bits.

At CT-RSA 2013, NIST proposed the idea to standardize Keccak variants with different parameters than those submitted to the SHA-3 competition. More specifically, NIST proposes to reduce the capacity c to n instead of $2n$ bits. Due to the reduced capacity, NIST may also consider to standardize a smaller 800-bit permutation for small capacities. Unfortunately, these variants have not been analyzed very well during the SHA-3 competition.

Our Contribution. In this work, we analyze these new and the original variants of Keccak using the same attack strategy. We use a standard differential attack, which allows us to better compare the security of all variants. Our results show that reducing the capacity does not lead to much better differential attacks on Keccak. On the other hand, reducing the permutation size b from 1600 to 800 bits even increases the security against differential attacks. To summarize, we are able to provide practical results for up to 4 rounds of all Keccak variants proposed for SHA-3 with a permutation size of 1600 bits. This includes the Keccak variants supporting arbitrary output sizes. A summary of our results and related work can be found in Table 1.

Related Work. The collision resistance of Keccak with permutation size of $b = 1600$ has already been investigated by a number of researchers. The first practical attack on Keccak with $c = 512$ and $n = 256$ has been published by Naya-Plasencia et al. in [17]. They have presented a 2-round collision attack which uses an efficient method to find high probability differential characteristics using the column parity (or kernel) property. By using these characteristics and connecting them with the input using an algebraic method, Dinur et al. have presented a 4-round collision attack for Keccak with $c = 448$ and $n = 224$, and for Keccak with $c = 512$ and $n = 256$ in [8]. Furthermore, in [9] Dinur et al. have presented the first attacks on reduced Keccak with $c = 768$ and $n = 384$, and Keccak with $c = 1024$ and $n = 512$. For both variants practical collision attacks on 3 rounds were shown. Moreover, they have shown theoretical attacks on Keccak with $c = 512$ and $n = 256$ reduced to 5 rounds and Keccak with $c = 768$ and $n = 384$ reduced to 4 rounds with a complexity of 2^{115} and 2^{147}, respectively.

Outline. The paper is structured as follows. In Section 2 we provide a short description of Keccak. In Section 3 we give an overview of the basic attack strategy. Section 4 presents our method to find high probability characteristics. In Section 5, we discuss our approach to connect these characteristics to the input, using an automated search tool.

2 Description of Keccak

Keccak is a family of hash functions based on the sponge construction, with state sizes $b \in \{25, 50, 100, 200, 400, 800, 1600\}$. Keccak uses a b-bit permutation Keccak-$f[b]$ and a multi-rate padding scheme. A specific instance of Keccak is defined by the two parameters r (the rate) and c (the capacity) with $b = r+c$. For the NIST SHA-3 competition, the Keccak designers have defined one instance for each output size $n \in \{224, 256, 384, 512\}$ bits. All four instances use the 1600-bit permutation with capacity $c = 2n$, where n is the hash output size. Additionally, to these four variants the Keccak designers also specify a variant supporting arbitrary output sizes which they named Keccak[]. This variant has capacity $c = 576$ and rate $r = 1024$.

The permutation Keccak-f used in Keccak operates on a three-dimensional state with elements in $\mathbb{F}_2$. The dimensions for this state are $5 \times 5 \times w$ with $w \in \{1, 2, 4, 8, 16, 32, 64\}$. This allows to represent each lane as a w-bit word. A three-dimensional array is used, $S[x][y][z]$, to describe the state. The hash h for a message m is computed in the following way for Keccak with rate r and capacity c:

1. Initialise the state $S[x][y][z] = 0$ for $x = 0 \dots 4$, $y = 0 \dots 4$ and $z = 0 \dots w$.
2. Compute the padded message $M = m||10^*1$ such that M is a multiple of r.
3. Absorb the next r-bit message block by computing $S[x][y] = S[x][y] \oplus M_i$ and update the state by computing $S =$ Keccak-$f(S)$. Repeat this process until all message blocks are absorbed.

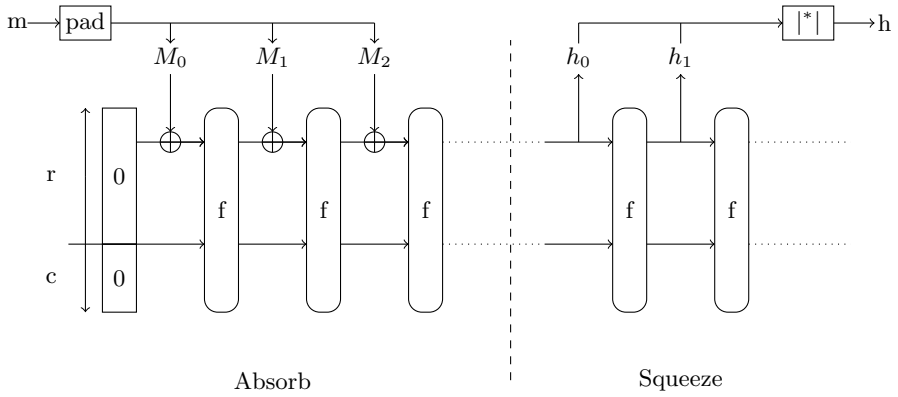

Fig. 1. Overview of the sponge construction, which is used in the Keccak hash function.

4. Concatenate the first r bits to the hash value. Compute $S = \text{Keccak-}f(S)$ and repeat this process to get the desired number of output bits.

This procedure is also outlined in Fig. 1.

2.1 Keccak-f

Keccak uses the iterative permutation Keccak-f operating on $\mathbb{F}_2^w$, with w being the lane size. The permutation consists of multiple rounds in which five functions are used in sequence $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$. The number of rounds n_r depends on the lane size:

$$n_r = 12 + 2\log_2(w) \tag{1}$$

Apart from ι, all functions are equivalent for each round.

Description of θ. This step adds to every bit of the state $S[x][y][z]$ the bitwise sum of the neighbouring columns $S[x-1][*][z]$ and $S[x+1][*][z-1]$. This procedure can be described by the following equation:

$$\theta : S[x][y][z] \leftarrow S[x][y][z] + \sum_{n=0}^{4} S[x-1][n][z] + \sum_{n=0}^{4} S[x+1][n][z-1] \tag{2}$$

Description of ρ. This step rotates the bits in every lane by a constant value.

$$\rho : S[x][y][z] \leftarrow S[x][y][z + C(x,y)] \tag{3}$$

where $C(x,y)$ is a constant value.

Description of π. This function transposes the lanes using the following function:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} \tag{4}$$

Description of χ. This step is the only non-linear step in Keccak and operates on each row of 5 bits.

$$\chi : S[x][y][z] \leftarrow S[x][y][z] \oplus ((\neg S[x+1][y][z]) \wedge S[x+2][y][z]) \tag{5}$$

It can be seen as applying a 5-bit S-box in parallel to all rows.

Description of ι. This steps adds a round dependent constant to the state. For a list of the constants see [2].

3 Differential Cryptanalysis of Keccak

In this section, we give a brief overview of our attack strategy on the Keccak hash function. We use standard differential cryptanalysis which has first been published to cryptanalyze the block cipher DES [3] and was later applied to hash functions as well. The basic idea of our attack is the same as used by Wang et al. in the cryptanalysis of the MD4-family of hash functions [19, 20]. The hash function is split into two parts. We first construct a high-probability differential characteristic for the second part and then, use a low-probability differential characteristic and message modification to connect with the input in the first part. In more detail, we perform the following 4 steps:

1. Find a differential characteristic for the hash function that results in a collision and holds with a high probability for the last few rounds of the hash function.
2. Find a differential characteristic (not necessary with high probability) for the first few rounds of the hash function.
3. Use message modification techniques to find conforming message pairs for the differential characteristic in the first few rounds.
4. Use the message pairs of the previous step to find a solution for the high-probability characteristic in the last few rounds of the hash function.

This strategy has already been used in [8] by Dinur et al. in the attack on 4 rounds of Keccak with $c = 512$ and $n = 256$. In this paper, we revisit the attack, extend it, and apply it to other variants of Keccak. Our attack differs from the attack of Dinur et al. in several ways. First of all, we show how to construct high probability differential characteristics for the last few rounds of Keccak that result in a (internal) collision for more than 256 bits. This allows us to construct collisions for larger output sizes of Keccak reduced to 4 rounds including the variant of Keccak supporting variable output sizes.

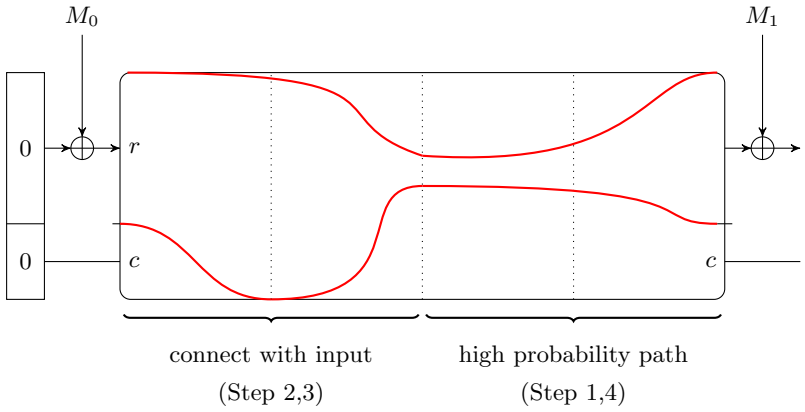

Fig. 2. Outline of our 4-round differential attack strategy.

Second, we present a more general technique to perform Step 2 and 3 of the attack. We use an automatic search tool implementing a guess-and-determine strategy that constructs a differential characteristic and uses message modification techniques to find conforming message pairs. For this purpose, we use a similar tool as published by Mendel et al. in the analysis of SHA-2 [13,14]. Fig. 2 shows an high-level overview of our attack strategy.

In Step 1 of the attack, we search for 2-round high-probability characteristics which lead to (internal) collisions. To find collisions for larger output sizes than in [8], we use a linearized version of Keccak and methods from coding theory (see Sect. 4).

In Step 2 and Step 3 of the attack, we need to connect the input difference of the high-probability characteristic with the fixed input value given by the capacity c. We solve this problem by searching for a differential characteristic and conforming message pair using an automatic search tool (see Sect. 5). The difficulty of finding a solution depends on the size of the capacity c. By improving the search strategy of our tool, we are able to solve the problem for larger values of the capacity c.

4 Finding Colliding High-Probability Characteristics

High-probability differential characteristics can be constructed for Keccak by using the column parity property of θ. Using this property, Naya-Plasencia et al. presented the first practical collision attacks on round-reduced versions of Keccak in [17]. Similar differential characteristics over 2 rounds were also used by Dinur et al. in [8]. A method to construct all column parity paths up to a given Hamming weight is described in [17], and a full characterization of kernel paths was done in [6]. However, no low-weight paths over three consecutive rounds exist [2].

4.1 Differential Characteristics and Coding Theory

To find a good characteristic for 2 rounds of Keccak, we use a linearized model of the Keccak hash function. Therefore, we replace all non-linear operations by a linear approximation resulting in a linear code over $\mathbb{F}_2$. Finding characteristic in the linear code is not difficult, since it depends only on the differences at the input. The probability that the characteristic holds in the original hash function is related to the Hamming weight of the characteristic. In general, a characteristic with low Hamming weight has a higher probability than one with a high Hamming weight. Hence, for finding a characteristic with high probability, i.e. with low Hamming weight, we use probabilistic algorithms from coding theory. It has been shown in the past [4, 11, 12, 18] that these algorithms work quite well. Furthermore, we can impose additional restrictions on the characteristic by forcing certain bits/words to zero. Note that this is needed to find suitable characteristics for Keccak resulting in an (internal) collision for the hash function. In the following we will briefly discuss the linear approximation and algorithms we were using to find the characteristics.

Linear Approximation of Keccak. The only non-linear transformation in Keccak is χ. There are many ways to approximate χ by a linear function. For our analysis we decided to use the identity function, since it comes very close to the original definition and we are aiming for sparse characteristics.

$$\chi : S[x][y][z] \leftarrow S[x][y][z] \tag{6}$$

All other transformations are linear facilitating our approach.

4.2 Finding low-weight Codewords

To find codewords with low Hamming weight we use the publicly available CodingTool Library[1]. It implements the probabilistic algorithm from Canteaut and Chabaud [5] to search for codewords with low Hamming weight. Moreover, it provides some other usable functionalities that turned out to be very useful for our purpose. With this tool we can find good characteristics for different choices of c and n in a few seconds on a standard PC. Table 2 and Table 3 show the best (lowest Hamming weight) characteristics we have found for a different set of parameters. It has to be noted that we can use these characteristics to construct internal collisions for Keccak with capacity up to 416 resp. 832 bits. However, for Keccak with variants with $c = 2n$ this is too small to attack versions with output sizes larger than 208 resp. 416 bits. Therefore, we also give the results for characteristics resulting only in a collision for the hash function. The results are characteristics that can be used for collision attacks for up to 448 resp. 832 bits.

Using this general approach the whole (linear) search space is covered and arbitrary differences in the state words are possible. However, it turned out that

[1] `http://www.iaik.tugraz.at/content/research/krypto/codingtool/`

Table 2. Low-weight differential characteristics for 2 rounds resulting in an internal collision for Keccak with capacity c.

Permutation	Capacity	Weight	Kernel Path
800	320	16	yes
	352	16	yes
	384	16	yes
	416	20	yes
1600	640	20	yes
	704	20	yes
	768	20	yes
	832	28	yes

Table 3. Low-weight differential characteristics for 2 rounds resulting in a collision for Keccak with hash size n.

Permutation	Hash Size	Weight	Kernel Path
800	320	12	yes
	352	12	yes
	384	20	yes
	416	30	no
	448	32	yes
1600	512	16	yes
	640	20	yes
	704	20	yes
	768	20	yes
	832	28	no

the best characteristics we have found are indeed column parity kernel paths. In hindsight the same differential characteristics could have been found using the method described in [17].

Extending the Approach to 3 Rounds. Using the same method one could try to construct differential characteristics for more than 2 rounds. Unfortunately, we did not find any sparse solutions which is conform with the work by Daemen and Van Assche in [6]. Another approach we tried was to non-linearly propagate the linear paths for 2 rounds forward using the automatic tool described in Sect. 5. As we can not linearly combine these paths, we use a brute force algorithm to check if a combination results in a collision after three rounds. However, since the search space is too large to cover, we restricted ourselves to a combination of only a few candidates. Unfortunately, we could not find a solution, which confirms that a sparse 3-round path is unlikely to exist.

5 Non-Linear Characteristics and Message Modification

Once we have determined a high-probability characteristic for the second half of Keccak, we need to connect this path with the constraints at the input of the Keccak permutation. In [8], Dinur et al. have used their target difference algorithm to find a solution for both differences and values of the input message.

In our work, we use the improved differential path search algorithm of Mendel et al. [14]. Using this automated search tool, complex nonlinear differential characteristics can be found. Additionally, the tool can be used for solving nonlinear equations involving conditions on state and message words (i.e. for message modification).

Using the bitsliced propagation method used in [14], we were not able to find a solution for the first two rounds of Keccak. The problem is, that the linear functions used in Keccak are significantly larger than the linear functions used in SHA-2. However, using the linear propagation method proposed in [10] and some other minor improvements, we are able to find solutions for the first two rounds of Keccak. In the following, we give a brief description of the search algorithm.

5.1 Search for Differential Characteristics and Message Pairs

The basic idea of this search algorithm is to pick and guess previously unrestricted bits. After each guess, the information due to these restrictions is propagated to other bits. If an inconsistency occurs, the algorithm backtracks to an earlier state of the search and tries to correct it. Similar to [14], we denote these three parts of the search by decision (guessing), deduction (propagation), and backtracking (correction). Then, the search algorithm proceeds as follows.

Let U be a set of bits. Repeat the following until U is empty:

Decision (Guessing)

1. Pick randomly (or according to some heuristic) a bit in U.
2. Impose new constraints on this bit.

Deduction (Propagation)

3. Propagate the new information to other variables and equations as described in [14].
4. If an inconsistency is detected start backtracking, else continue with step 1.

Backtracking (Correction)

5. Try a different choice for the decision bit.
6. If all choices result in an inconsistency, mark the bit as critical.
7. Jump back until the critical bit can be resolved.
8. Continue with step 1.

In the deduction, we use generalized conditions on bits [7]. To propagate information, we use the techniques of [10] and during the search, we backtrack as shown in [14]. In the message search, we additionally consider linear conditions on two related bits ($X_j \oplus X_k = \{0, 1\}$) as proposed in [13].

Note that in each stage different bits are chosen (guessed). In total we have two stages which can be summarized as follows.

1. **Characteristic Search:** In the first phase we search for a consistent differential characteristic in the state words. Therefore, we only add unconstrained bits '?' to the set U.

2. **Message Search:** In the second stage we search for a conforming message. In this phase, we only add bits with many linear two-bit conditions to the set U. This ensures that bits which influence a lot of other bits are guessed first.

Note that we dynamically switch between the two stages. Additionally, we restart the search from scratch after a certain amount of inconsistencies to terminate branches which appear to be stuck because of exploring a search space far from a solution.

5.2 Improved Linear Propagation in Keccak

Using the bitsliced propagation method used in [14], we were not able to find a solution for the first two rounds of Keccak. The problem is, that the linear layer $\lambda = \pi \circ \rho \circ \theta$ of Keccak is significantly larger than the linear functions used in SHA-2. We have tried to split the linear layer into bitslices and at least 320 bitslices are needed. In this case the linear information propagates very badly and many contradictions in the linear layer are not detected.

To avoid this problem, we use the linear propagation method of [10]. In this case, a linear system of equations is defined, which contains all equations of the linear functions for X_i and X_i^*, as well as the equations for the linear generalized conditions at the input and output of the function. The resulting system of equation is solved using Gauss-Jordan elimination to detect contradictions and propagate information.

We get the best results when applying this linear approach to the complete linear layer $\lambda = \pi \circ \rho \circ \theta$ of Keccak. We have also performed experiments which include the XORs of the S-box layer χ, or by combining the linear parts of two Keccak rounds. However, the best performance/propagation trade-off was achieved for the linear layer λ.

5.3 Finding Solutions for 2 Rounds of Keccak

Using the automated search tool combined with the linear propagation method, we can efficiently find both, differential characteristics and conforming message pairs for up to two rounds of Keccak. However, the difficulty of finding a solution depends on fine-tuning of the search algorithm based on a number of parameters.

The parameter which influences the search most is the capacity c. If too many bits are fixed by the capacity, then we are not able to find a solution. For the 1600-bit permutation, we could find solutions for capacities of up to 640 bits (ten 64-bit lanes), and for the 800-bit permutation, we could find solutions for capacities of up to 352 bits (eleven 32-bit lanes).

Our experiments have shown, that for a zero value at the input it is harder to find a solution, in particular for larger capacities. The running time of the search algorithm can be improved by prepending a random first message block with random differences. This was used for the results given in the appendix.

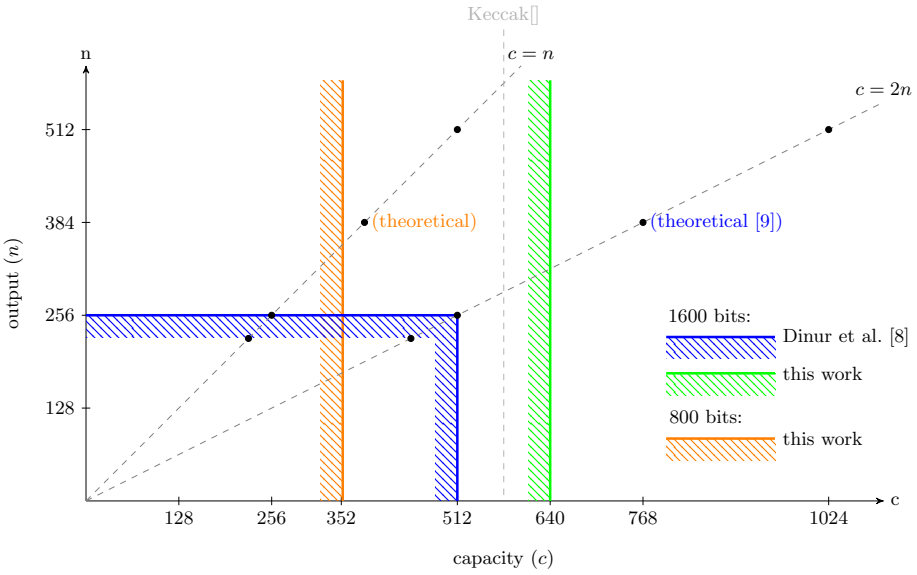

Fig. 3. Overview of all 4-round collision attacks on Keccak with permutation size of 1600 and 800 bits, respectively. Blue: attacks by Dinur et al. and green: our attacks on Keccak with permutation size of 1600 bits. Orange: our attacks on Keccak with permutation size of 800 bits. Additionally, a theoretical 5-round collision attack on Keccak-256 has been published in [9].

6 Results

For the 1600-bit permutation reduced to 4 rounds with a capacity of $c \leq 640$ we can find internal collisions and hence, collisions for the hash function with arbitrary output size. We want to note that that this also includes the Keccak variant with capacity $c = 576$, which was proposed by the Keccak designers for supporting variable output sizes. The confirming message pair and the according differential characteristic is given in Appendix A. Note that for the zero IV of Keccak, our automatic search tool does not work very well. By using a first message block with differences (which could even be meaningful), the tool works much better. In this case a solution was found within minutes on a standard PC.

For the smaller 800-bit permutation we can show internal collisions for a capacity of $c \leq 352$ bits. A confirming message pair and the according differential characteristic is given in Appendix B. Finding this solution took about 140 minutes on a standard PC. This corresponds to about 2^{38} Keccak computations. Note that in an attack on the hash function with a capacity of 352 bits, the values and differences of 32 additional bits can be chosen freely. Based on this, we estimate the complexity to construct an internal collision with capacity 384 to be at most $2^{64+38} = 2^{104}$. However, we expect the complexity to be much smaller in practice. Our results are shown in Fig. 3.

Acknowledgements

This work has been supported in part by the Secure Information Technology Center-Austria (A-SIT), by the Austrian Science Fund (FWF) under grant number TRP 251-N23 (Realizing a Secure Internet of Things - ReSIT), and by the Austrian Research Promotion Agency (FFG) and Styrian Business Promotion Agency (SFG) under grant number 836628 (SeCoS).

References

1. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the Indifferentiability of the Sponge Construction. In: Smart, N.P. (ed.) EUROCRYPT. LNCS, vol. 4965, pp. 181–197. Springer (2008)
2. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak reference. Submission to NIST (Round 3) (January 2011), available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html`
3. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology 4(1), 3–72 (1991)
4. Brier, E., Khazaei, S., Meier, W., Peyrin, T.: Linearization Framework for Collision Attacks: Application to CubeHash and MD6. In: Matsui, M. (ed.) ASIACRYPT. LNCS, vol. 5912, pp. 560–577. Springer (2009)
5. Canteaut, A., Chabaud, F.: A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511. IEEE Transactions on Information Theory 44(1), 367–378 (1998)
6. Daemen, J., Assche, G.V.: Differential Propagation Analysis of Keccak. In: Canteaut, A. (ed.) FSE. LNCS, vol. 7549, pp. 422–441. Springer (2012)
7. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT. LNCS, vol. 4284, pp. 1–20. Springer (2006)
8. Dinur, I., Dunkelman, O., Shamir, A.: New Attacks on Keccak-224 and Keccak-256. In: Canteaut, A. (ed.) FSE. LNCS, vol. 7549, pp. 442–461. Springer (2012)
9. Dinur, I., Dunkelman, O., Shamir, A.: Collision Attacks on Up to 5 Rounds of SHA-3 Using Generalized Internal Differentials. In: Moriai, S. (ed.) FSE. LNCS, Springer (2013), to appear
10. Eichlseder, M., Mendel, F., Nad, T., Rijmen, V., Schläffer, M.: Linear Propagation in Efficient Guess-and-Determine Attacks. In: Budaghyan, L., Helleseth, T., Parker, M.G. (eds.) WCC (2013), `http://www.selmer.uib.no/WCC2013/`
11. Indesteege, S., Preneel, B.: Practical Collisions for EnRUPT. J. Cryptology 24(1), 1–23 (2011)
12. Mendel, F., Nad, T.: A Distinguisher for the Compression Function of SIMD-512. In: Roy, B.K., Sendrier, N. (eds.) INDOCRYPT. LNCS, vol. 5922, pp. 219–232. Springer (2009)
13. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT. LNCS, vol. 7073, pp. 288–307. Springer (2011)
14. Mendel, F., Nad, T., Schläffer, M.: Improving Local Collisions: New Attacks on Reduced SHA-256. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT. LNCS, vol. 7881, pp. 262–278. Springer (2013)

15. National Institute of Standards and Technology: Cryptographic Hash Algorithm Competition (November 2007), available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`
16. National Institute of Standards and Technology: SHA-3 Selection Announcement (October 2012), available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_selection_announcement.pdf`
17. Naya-Plasencia, M., Röck, A., Meier, W.: Practical Analysis of Reduced-Round Keccak. In: Bernstein, D.J., Chatterjee, S. (eds.) INDOCRYPT. LNCS, vol. 7107, pp. 236–254. Springer (2011)
18. Rijmen, V., Oswald, E.: Update on SHA-1. In: Menezes, A. (ed.) CT-RSA. LNCS, vol. 3376, pp. 58–71. Springer (2005)
19. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In: Cramer, R. (ed.) EUROCRYPT. LNCS, vol. 3494, pp. 1–18. Springer (2005)
20. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT. LNCS, vol. 3494, pp. 19–35. Springer (2005)

A Results for Keccak with a 1600-bit Permutation.

Table 4. The 4-round characteristic used for the second block to find an internal collision for Keccak with a 1600-bit permutation and a capacity of 640. Note that a random first block was used in this case.

Round	State				
	737bc39f15b62ce3	4-ae-67d9-f67961	72c17e19ecf12b7b	2ba7b749c7949634	fc-cfc935859fb2e
	3d196398efcd8-85	fce83de1dec57822	585c3e88-e91a216	7abfed54f57e1dd9	d9a96ed7944d8ede
0	147b6be6e6-24fdb	--4a7743-1159181	-1df19ab97369543	77a1e8bca7-c--6f	-5e697e1852d7fd5
	1a9b2c7d9b5a9abf	2913f4ef6ca6b829	4--b84511febc4ff	236c8edaa59db4a3	fa16a175b84e4326
	6c34feb1242754fb	cb2ea33a4c-db176	b2c5aa5a8-df6238	7bafafd7ee121941	8b4cf1f55781e-9f
	96--3182f1fad467	22--9-644fa7e-f-	de--54fb5f2e9a6b	7e--726f824-bd4c	d2--114a6bb11583
	96-171-2f1fad467	26--9-644fa7e-f-	de--54fb5f2e9a6b	7e--726f8244b14c	d2--114a6fb51583
1	96-17112f1fad467	22--b-244fa7e-f-	de--54fb5f2e9a4b	7e--726f8244b14c	d2--114a6bb11583
	96-171-2f1fad467	26--9-644fa7e-f2	de--54fb5f2e9a6b	7e--726f884-b14c	d2--114a6bb11583
	96-171-2f1fad467	22--9-644fa7e-f-	da--5-fb5f2e9a6b	fe--726f8244b14c	d2--114a6ab11583
	----4-8---------	------4---------	---1------------	----------------	----------------
	----4-8---------	------4---------	----------------	----------------	--8-------------
2	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	---1------------	----------------	--8-------------
	----------------	----------------	----------------	----------------	----------------
	----4-8---------	----------4----	----------------	----------------	----------------
	----------------	-------------8--	----------------	----------------	----------------
3	------8---------	----------------	----------------	8---------------	----------------
	----------------	----------4-8--	----------------	8---------------	----------------
	----4-----------	----------------	----------------	----------------	----------------
	----4-8---------	--8-------------	----------1-----	------8---1-----	--8-4-----------
	----------------	-1---4----------	-----4----------	81--------------	----------------
4	---------1-8----	----------------	---------1-----1	---------------1	---------------1
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------

Table 5. A 4-round internal collision for Keccak with a 1600-bit permutation and a capacity of 640.

M_0:

```
0000000082784B27 0000000027B97209 00000000F9E7B4C3 00000000FE890B5C 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

M_1:

```
1AE4DA9BCE0F58C5 3E41B66FEC61367F 60EFB06502B1E522 8F2689B944C6ADA4 3679B40E76AEE052
29023AF14A8D1931 0589A067B9C0882B 9CDCF37544841411 52448031E1488314 295FB9F654DD515D
58783A446CC0DF27 DC575C851C1DA5C0 9F82D47401FC7A76 7D7971B3C8B6D25A EA79DD2396CA4FEE
```

M_2:

```
0000408000000000 0080000000000000 0000000000100000 0000008000100000 0080400000000000
0000000000000000 0100040000000000 0000040000000000 8100000000000000 0000000000000000
0000000001080000 0000000000000000 0000000001000001 0000000000000001 0000000000000001
```

M_0^*:

```
000000006DF2B918 00000000EF86D9FE 0000000040DD1D22 000000000326C57A3 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

M_1^*:

```
93A3B74748B2D4D0 D1224F333B3E30CD E37E9B50203D12F1 9558EABAEB983C68 036275C12894EBCD
E56F4097FFB56F5F 06070F676C145DFD FB11961465177857 C831E04FD29B424E 04AFAA83CF448D0B
59A7AC2AE2163340 E0E482684E961996 778732CDA01B329D BED51FB2554F233A 7830FB4DC95AB3C4
```

M_2^*:

```
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

State after processing third block:

```
8E485EC7CC0271CA BA770EFBBD69EE16 5A3DA8FFD2F4C521 081E39496F095437 756E97B6477B1ED9
833FB0900600EB96 26A93661FE6F9531 86ADA9C976EB9861 9DD0D44634EC35AC F0D14E73C1916C96
3C0FF867406BB4EA 8EDF8F16DABCBAE9 DE3EDE57965FEE6F B34B3B20F466A277 7726B4B7AA8A84D8
272A11E9F2AD4981 046F4AF7DA9F98EC 4788C486729AC3A7 F95AFA8787C36990 06E3748CA8574FDC
929C857723322ED0 6706560C7EE7A3E3 313BB48B67DCCDB2 795A30724698D71C 3BC9CFF2827373AC
```

B Results for Keccak with a 800-bit Permutation.

Table 6. The 4-round characteristic used for the second block to find a internal collision for Keccak with a 800-bit permutation and a capacity of 352. Note that a random first block was used in this case.

Round	State
0	551c2e5a 5b7cc9a2 d7fab224 893-5cd9 f3a536f5 7198b13f c8fb3e45 c82abe5e e85886f1 226465c- -a9f5-d6 d5b9-fb6 47926282 2538236- 996272ee 16f3b671 2fa-3-dd 3aad9-1e 6252-cfd 777383b4 7f928adf c7dcfa85 d8b21bf2 5bf4c55- 27dfd4dd
1	9-25244a 4-721523 612e18b- 8-ae-689 b-e28e-c b-242442 5-731563 61ae18b- 81ae-688 b-e28f-c b-27244a 5-721563 612a18b- 81ae4688 b-e28e-c b-25244a 5-7295e3 6-2e189- 81ae-688 3-c28e-c b-25244a 5-721563 e1ae18b- 91ae8688 b-e2-e-c
2	2------- 1------- -------- -------- -------- -------- 1------- -------- -------- --1----- 2------- -------- -------- ------8- -------- -------- -------- -------- ------8- --1----- -------- -------- -------- -------- --------
3	2------- -----1-- -------- 1------- -------- -------- -----1-- -------1 -------- -------- 2------- -------- -------1 1------- -------- -------- -------- -------- -------- -------- -------- -------- -------- -------- --------
4	2----8-- --1--8-- -----8-- 2------- 2-1----- -1-----1 -------- -------1 -------- -------- --2--24- ------4- --2----- -----2-- -------- -------- -------- -------- -------- -------- -------- -------- -------- -------- --------

Table 7. A 4-round internal collision for Keccak with a 800-bit permutation and a capacity of 352

M_0:

```
8F1075E0 EDFC1488 58EFE9FC 433877BD 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
```

M_1:

```
B051BED6 1DECB0DA D26B2923 01734BC7 2D2002D8
120AB268 10634585 16F789D1 4AD2F036 AF13E319
3DD3C552 0FF14835 8049189A 9786F56E
```

M_2:

```
20000800 00100800 00000800 20000000 20100000
01000001 00000000 00000001 00000000 00000000
00200240 00000040 00200000 00000200
```

M_0^*:

```
3BBBA6F0 97006B85 09124595 C63FA0D6 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
```

M_1^*:

```
1B9837F7 1FFE60AD 13269184 9F567790 6C0191B1
DF45607B 97EE79C4 D963BDDA 00FF2D4A BE6F08C8
1FFBDA2C B787E7C8 34F8358A 8EB37499
```

M_2^*:

```
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
```

State after processing third block:

```
85AF483E 0CED18F7 BC101BB2 2F2CC963 0DCE54BE
30C5B7F8 B5CE439A 465B540D 760424F3 006BB414
045BBB7A 7C14CDA7 F082AF8E 2BA59219 A730ACCD
D5DBAE77 E1598F25 89373578 552A4091 E7D9C411
043CF740 A1D66CA3 F454A015 0E2A1D74 5FA83840
```

C Differential Distribution Table

Table 8. Differential distribution table for χ.

		Δout																													
		00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
Δin	00	32	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	01	-	8	-	8	-	8	-	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	02	-	-	8	-	-	-	8	-	-	-	8	-	-	-	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	03	-	4	-	4	-	4	-	4	-	4	-	4	-	4	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	04	-	-	-	-	8	-	-	-	-	-	-	-	8	-	-	-	-	-	-	-	8	-	-	-	-	-	-	8	-	-	-	-
	05	-	4	-	4	-	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-	-	-	-	-	-	-	-	4	-	4	-
	06	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-
	07	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-
	08	-	-	-	-	-	-	-	-	8	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	8	-	-	-	-	-	-	-
	09	-	-	-	-	-	-	-	-	4	-	-	4	4	-	-	4	-	-	-	-	-	-	-	-	4	-	-	4	4	-	-	4	-
	0A	-	-	4	4	-	-	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	-	-	4	4	-
	0B	-	4	4	-	-	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	-	-	4	4	-	-
	0C	-	-	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-
	0D	-	-	-	-	4	-	-	4	4	-	-	4	-	-	-	-	-	-	-	-	4	-	-	4	4	-	-	4	-	-	-	-	-
	0E	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-
	0F	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-
	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	8	8	8	-	-	-	-	-	-	-	-	-	-	-	-	-
	11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	4	4	4	4	4	4	-	-	-	-	-	-	-	-	-
	12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	-	-	-	4	4	4	4	-	-	-	-	4	4	-	-
	13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	-
	14	-	-	-	-	4	-	4	-	-	-	-	-	4	-	4	-	-	-	-	-	-	4	-	4	-	-	-	-	-	4	-	4	-	-
	15	-	4	-	4	-	-	-	-	-	-	-	-	-	4	-	4	4	-	4	-	-	-	-	-	-	-	-	-	4	-	4	-	-
	16	-	-	4	-	4	-	-	-	-	-	4	-	4	-	-	-	-	-	-	4	-	4	-	-	-	-	-	4	-	4	-	-	-
	17	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	-
	18	-	-	-	-	-	-	-	-	4	4	4	4	-	-	-	-	-	-	-	-	-	-	-	-	4	4	4	4	-	-	-	-	-
	19	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	-
	1A	-	-	-	-	-	-	-	-	4	4	-	-	-	-	4	4	4	4	-	-	-	-	4	4	-	-	-	-	-	-	-	-	-
	1B	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-	-
	1C	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-
	1D	-	-	-	-	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	-	-	-	-	-
	1E	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-
	1F	-	2	2	-	2	-	-	2	2	-	-	2	-	2	2	-	2	-	-	2	-	2	2	-	-	2	2	-	2	-	-	2	-