Practical Collision Attacks against Round-Reduced SHA-3*

Jian Guo[1], Guohong Liao[2], Guozhen Liu[3,1], Meicheng Liu[4], Kexin Qiao[5], and Ling Song[1,4,6(✉)]

[1] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore
guojian@ntu.edu.sg

[2] South China Normal University, Guangzhou, China
liaogh.cs@gmail.com

[3] School of Cyber Security, Shanghai Jiao Tong University, Shanghai, China
liuguozhen@sjtu.edu.cn

[4] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
meicheng.liu@gmail.com

[5] Beijing Unionpay Card Technology Co., Ltd., Beijing, China
qiaokexin@bctest.com

[6] Strategic Centre for Research in Privacy-Preserving Technologies and Systems, Nanyang Technological University, Singapore, Singapore
songling.qs@gmail.com

Abstract. The KECCAK hash function is the winner of the SHA-3 competition (2008 – 2012) and became the SHA-3 standard of NIST in 2015. In this paper, we focus on practical collision attacks against round-reduced SHA-3 and some KECCAK variants. Following the framework developed by Dinur *et al.* at FSE 2012 where 4-round collisions were found by combining 3-round differential trails and 1-round connectors, we extend the connectors to up to three rounds and hence achieve collision attacks for up to 6 rounds. The extension is possible thanks to the large degree of freedom of the wide internal state. By linearizing S-boxes of the first round, the problem of finding solutions of 2-round connectors is converted to that of solving a system of linear equations. When linearization is applied to the first two rounds, 3-round connectors become possible. However, due to the quick reduction in the degree of freedom caused by linearization, the connector succeeds only when the 3-round differential trails satisfy some additional conditions. We develop dedicated strategies for searching differential trails and find that such special differential trails indeed exist. To summarize, we obtain the first real collisions on six instances, including three round-reduced instances of SHA-3, namely 5-round SHAKE128, SHA3-224, and SHA3-256, and three instances of KECCAK contest, namely KECCAK[1440, 160, 5, 160], KECCAK[640, 160, 5, 160]

* This paper is prepared based mainly on [24] and [26]. The work was done when all the authors were working with Nanyang Technological University in Singapore.

and KECCAK[1440, 160, 6, 160], improving the number of practically attacked rounds by two. It is remarked that the work here is still far from threatening the security of the full 24-round SHA-3 family.

Keywords: Cryptanalysis, hash function, SHA-3, KECCAK, collision, linearization, differential, GPU.

1 Introduction

The KECCAK hash function [3] was a submission to the SHA-3 competition [22] in 2008. After four years of intensive evaluation, it was selected as the winner of the competition in 2012. It was then formally standardized in 2015 by the National Institute of Standards and Technology (NIST) of the U.S. as Secure Hash Algorithm-3 (SHA-3) [27]. The SHA-3 family has four instances with fixed digest lengths, namely, SHA3-224, SHA3-256, SHA3-384 and SHA3-512, which correspond to KECCAK$[c] \triangleq$ KECCAK$[r = 1600 - c, c]$ where $c \in \{448, 512, 768, 1024\}$. The SHA-3 family also has two eXtendable-Output Functions (XOFs) SHAKE128 and SHAKE256. To promote cryptanalysis of the KECCAK hash function, the KECCAK designers proposed variants with lower security levels in the KECCAK Crunchy Crypto Collision and Pre-image Contest (the KECCAK contest for short) [2], for which the digest lengths are 80 and 160 bits for pre-image and collision resistance, respectively. For clarity, these variants are denoted by KECCAK$[r, c, n_r, d]$ with parameters $\{r, c, n_r, d\}$ to be specified later.

Since the KECCAK hash function was made public in 2008, there have been intensive cryptanalysis from the public research community [1, 9, 10, 11, 12, 13, 14, 16, 17, 21]. In this paper, we mainly investigate collision attacks on the KECCAK hash function, in particular, those with practical attack complexities. In collision attacks, the aim is to find two distinct messages which lead to the same hash digest. Up to date, the best practical collision attacks against KECCAK[448]/KECCAK[512] is of 4 (out of 24) rounds found by Dinur *et al.* [10] in 2012 and later furnished in the journal version [12]. These 4-round collisions were generated by combining a 1-round connector and a 3-round differential trail. The same authors presented practical collision attacks on 3-round KECCAK[768]/KECCAK[1024], and theoretical collision attacks (with complexities beyond the reach of practical resources) on 5/4-round KECCAK[512]/KECCAK[768] in [11] using internal differentials. For the KECCAK contest, the best solutions reached up to 4 rounds in [12, 18]. To the best of our knowledge, there exist neither practical collision attacks against 5-round KECCAK[448]/KECCAK[512]/KECCAK[768]/KECCAK[1024], nor for any 5-round instances of the KECCAK contest.

Our contributions. Following the framework of Dinur *et al.* [10], we develop a new algebraic and differential hybrid method to launch collision attacks on KECCAK family and present seven real collisions on six KECCAK variants, including 5-round SHAKE128, 5-round SHA3-224, 5-round SHA3-256, and 5-round as well as 6-round instances of the KECCAK collision contest.

These results are obtained by combining a differential trail and a connector which links the initial state of KECCAK to the input difference of the trail. Previously, connectors of KECCAK covered one round [10], while in this paper we propose new connectors of up to three rounds. The improvements on connectors are possible mainly due to S-box linearizations of KECCAK, *i.e.*, the linearization of the nonlinear layer of KECCAK. Specifically, we linearize the first round for constructing 2-round connectors for KECCAK. When the degree of freedom of a 2-round connector is sufficiently large, it can be extended to three rounds by further linearizing (part of) the second round.

In this paper, two types of S-box linearizations are proposed, *i.e.*, *full linearization* and *non-full linearization*, which are achieved based on several crucial observations. First, we observe that the KECCAK S-box can be re-expressed with linear transformations when the input is restricted to certain affine subspaces. Daemen *et al.* [4, 8] and Dinur *et al.* [10] have already noted that when the input and output differences of the KECCAK S-box are fixed, the solution set forms an affine subspace of dimension up to 3. In this work, we show that the maximum subspaces allowing linearization of the S-box is of dimension 2 and any 2-dimensional affine subspace allows S-box linearization. For affine subspaces of dimension 3, six 2-dimensional affine subspaces out of it could allow the linearization. However, we do not have to fully linearize all S-boxes of the first round if only partial output bits of them need to be linear for constructing 2-round connectors. We further observe that non-full S-box linearizations help to save some degrees of freedom. More specifically, when linearizing part (not all) of the output bits of a non-active S-box (defined by whether there is any difference in the input and output), at most 2 degrees of freedom are consumed by specifying 2 binary linear equations over the input bits. For an active S-box whose entry in the differential distribution table (`DDT`) is 8, 4 out of 5 output bits are already linear when the input is chosen from the solution set. Note that to restrict the input to the solution set for such an S-box, 2 binary linear equations should be specified over the input bits. Therefore, for both non-active and active S-boxes, two or fewer degrees of freedom may be enough to linearize part of the output bits, while fully linearizing an S-box consumes at least 3 degrees of freedom.

With these properties in mind, we linearize S-boxes of the first round such that their output bits which affect the input difference of the trail that our connector tries to link to become linear. Therefore the first round function of the KECCAK permutation is transformed into a (partial) linear one. Combining with an inversion method of the S-box layer of the second round, we convert the problem of finding 2-round connectors into that of solving a system of linear equations. Solving the system produces sufficiently many solutions so that at least one pair of them will follow the differential trails over the last rounds. To extend the connector to 3 rounds, we first construct a 2-round connector where the first round is fully linearized by enforcing full S-box linearizations. Then upon the 2-round connector, 3-round connectors are obtained through constructing 2-round connectors over the second and third round, where non-full S-box linearizations are applied to the second round.

A side effect of S-box linearization is a quick reduction in the degree of freedom which in turn determines the existence of 2-round or 3-round connectors. To address this problem, we aim to find differential trails that impose the least possible conditions on the connectors. We design dedicated strategies to find suitable differential trails of 3 and 4 rounds. Our GPU implementations confirmed the correctness of this idea, and successfully found desirable differential trails for our collisions attacks.

Results obtained in this paper are listed in Table 1, compared with the best previous practical collision attacks and related theoretical attacks. In brief, we obtain actual collisions on three 5-round instances of `SHA-3`, *i.e.*, `SHAKE128`, and `SHA3-224`, `SHA3-256`, and three instances of Keccak contest. The number of practically attacked rounds of Keccak instances now is increased to 6. For the instance of contest Keccak$[1440, 160, 6, 160]$, an inner collision [3] on the last 160-bit of the output state is mounted to construct collisions from messages of any block length.

Table 1: Summary of our attacks and comparison with related works

Target$[r, c, d]$	n_r	Complexity	Reference
Keccak$[1024]$	3	Practical	[11]
Keccak$[768]$	3	Practical	
Keccak$[768]$	4	2^{147}	
Keccak$[512]$	5	2^{115}	
Keccak$[512]$	4	Practical	[10, 12]
Keccak$[448]$	4	Practical	
Keccak$[1440, 160, 160]$	4	Practical	
`SHAKE128`	5	Practical†	Sect. 6
`SHA3-224`	5	Practical	
`SHA3-256`	5	Practical	
Keccak$[1440, 160, 160]$	5	Practical	
Keccak$[\ 640, 160, 160]$	5	Practical	
Keccak$[1440, 160, 160]$	6	Practical	
Keccak$[1440,160,160^{\dashv}]^{\ddagger}$	6	Practical	

† For exact complexities, refer to Table 6.
‡ $160^{\dashv}$ corresponds to the last 160-bit collision rather than the general digest collision.

Organization. The rest of the paper is organized as follows. In Section 2, a brief introduction to the `SHA-3` hash function is given. Section 3 first presents the notations used in this paper and subsequently provides a synopsis of the collision attacks against `SHA-3` and Keccak instances. The properties of S-box linearization and non-full linearization are illustrated in Section 4, followed by detailed

explanations over how the 2-round and 3-round connectors are constructed. Section 5 outlines the search for differential trails as well as the highly efficient GPU implementations of KECCAK. In Section 6, the experimental results of collision attacks are given. We conclude the paper in Section 7. Details of the differential trails and actual collisions are postponed to the Appendix.

2 Description of SHA-3

2.1 The Sponge Function

The sponge construction is a framework for constructing hash functions from permutations, as depicted in Fig. 1. The construction consists of three components: an underlying b-bit permutation f, a parameter r called rate and a padding rule. The value $c = b - r$ is called capacity. A hash function following this construction takes in a message M as input and outputs a digest of d bits. Given a message M, it is first padded and split into r-bit blocks. The b-bit state is initialized to be all zeros. The sponge construction then proceeds in two phases. In the absorbing phase, each message block is XORed into the first r bits of the state, followed by application of the permutation f. This process is repeated until all message blocks are processed. Then, the sponge construction switches to the squeezing phase. In this phase, each iteration returns the first r bits of the state as (part of) the output and then applies the permutation f to the current state. This repeats until all d bits digest are obtained.

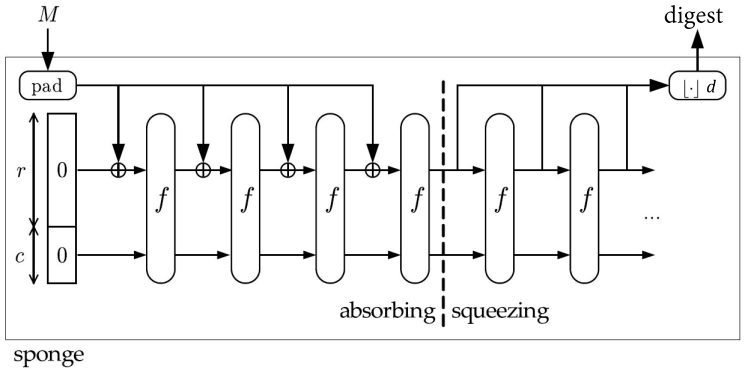

Figure 1: Sponge Construction [3]

2.2 The Keccak Hash Function

The KECCAK hash function follows the sponge construction. The underlying permutation of KECCAK is chosen from a set of seven KECCAK-f permutations, denoted by KECCAK-$f[b]$, where $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ is the width

of the permutation in bits. The default KECCAK employs KECCAK-$f[1600]$. The 1600-bit state can be viewed as a 3-dimensional $5 \times 5 \times 64$ array of bits, denoted as $A[5][5][64]$. Let $0 \leq i, j < 5$, and $0 \leq k < 64$, $A[i][j][k]$ represents one bit of the state at position (i, j, k). Defined by the designers of KECCAK, $A[*][j][k]$ is called a row, $A[i][*][k]$ a column, and $A[i][j][*]$ a lane. Note the lane size is $\frac{b}{25}$ which varies according to b.

The KECCAK-f permutation has $12 + 2l$ rounds ($l = \log_2 \frac{b}{25}$), each of which consists of five mappings $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$.

θ: $A[i][j][k] \leftarrow A[i][j][k] + \sum_{j'=0}^{4} A[i-1][j'][k] + \sum_{j'=0}^{4} A[i+1][j'][k-1]$.

ρ: $A[i][j] \leftarrow A[i][j] \lll T(i, j)$, where $T(i, j)$s are constants.

π: $A[j][2i + 3j] \leftarrow A[i][j]$.

χ: $A[i][j][k] \leftarrow A[i][j][k] + (A[i + 1][j][k] + 1) \cdot A[i + 2][j][k]$.

ι: $A[0][0] \leftarrow A[0][0] + RC_{i_r}$, where RC_{i_r} is the i_r-th round constant.

Here, the additions and multiplications between the terms are in $\mathsf{GF}(2)$. As ι plays no essential role in our attacks, we will ignore it in the rest of the paper unless otherwise stated.

2.3 Instances of Keccak and SHA-3

There are four instances of KECCAK, denoted by KECCAK$[c] \triangleq$ KECCAK$[r = 1600 - c, c]$, where the capacity $c \in \{448, 512, 768, 1024\}$ and the digest length d is half of c. To promote cryptanalysis against KECCAK, the KECCAK design team also proposed variants with lower security levels in the KECCAK contest, where $b \in \{1600, 800, 400, 200\}$, $(d = 80, c = 160)$ for pre-image contest and $(d = 160, c = 160)$ for collision contest. In this paper, we follow the designers' notation KECCAK$[r, c, n_r, d]$ for the instances in the contest, where r is the rate, $c = b - r$ is the capacity, d is the digest size, and n_r is the number of rounds the underlying permutation KECCAK-f is reduced to.

The KECCAK hash function uses the multi-rate padding rule which appends to the original message M a single bit 1 followed by the minimum number of bits 0 and a single bit 1 such that the length of the resulted message is a multiple of the block length r. Namely, the padded message $\overline{M}$ is $M \| 10^*1$.

The SHA-3 standard adopts the four KECCAK instances and names them SHA3-224, SHA3-256, SHA3-384 and SHA3-512, respectively. In these four instances of SHA-3, the message is appended '01' first. After that, the multi-rate padding is applied. The SHA-3 standard also contains two extendable-output functions named SHAKE128 and SHAKE256, which are defined from two instances of KECCAK with the capacity c being 256 and 512 respectively and the digest of any length. For SHAKE, a four-bit suffix '1111' is added to the original message M before applying the multi-rate padding.

3 Notations and Overview

3.1 Notations

We summarize the major notations to be used in this paper here.

c	Capacity of a sponge function
r	Rate of a sponge function
b	Width of a KECCAK permutation in bits, $b = r + c$
d	Length of the digest in bits
$d^{\dashv}$	Length of collision on the last bits
p	Number of fixed bits in the initial state due to padding
n_r	Number of rounds
$\theta, \rho, \pi, \chi, \iota$	The five mappings that comprise a round. A subscript i denotes the mapping at the i-th round, *e.g.*, χ_i denotes the χ layer at the i-th round for $i = 0, 1, 2, \cdots$.
L	Composition of θ, ρ, π and its inverse denoted by L^{-1}
RC_i	Round constant for the i-th round, $i = 0, 1, 2, \cdots$
$\mathtt{R}^i(\cdot)$	KECCAK permutation reduced to the first i rounds
$\mathtt{S}(\cdot)$	5-bit S-box operating on each row of KECCAK state
$\delta_{in}, \delta_{out}$	5-bit input and output differences of an S-box,
$\mathtt{DDT}$	Differential distribution table, and $\mathtt{DDT}(\delta_{in}, \delta_{out}) = \ \lvert\{x : \mathtt{S}(x) + \mathtt{S}(x + \delta_{in}) = \delta_{out}\}\rvert$, where $\lvert \cdot \rvert$ denotes the size of a set.
α_i	Input difference of the i-th round function, $i = 0, 1, 2, \cdots$
β_i	Input difference of χ in the i-th round, $i = 0, 1, 2, \cdots$
w_i	Weight of the i-th round, w_i, $i = 0, 1, 2, \cdots$
$\mathtt{DF}$	Degree of freedom of the solution space of connectors
$\overline{M}$	Padded message of M. Note that $\overline{M}$ is of one block in our attacks.
$M_1 \lvert\lvert M_2$	Concatenation of strings M_1 and M_2

3.2 Overview of the Attack

In this subsection, we give an overview of our collision attacks. Following the framework by Dinur *et al.* [10], as well as many other collision attacks utilizing differential trails, our collision attacks consist of two parts, *i.e.*, a high probability differential trail and a connector linking the differential trail with the initial state, as depicted in Fig. 2. Let ΔS_I and ΔS_O denote the input and output differences of the differential trail, respectively. A connector covering n_{r_1} rounds produces message pairs (M, M') such that

$$\mathtt{R}^{n_{r_1}}(\overline{M}||0^c) + \mathtt{R}^{n_{r_1}}(\overline{M'}||0^c) = \Delta S_I$$

always holds, *i.e.*, the difference after n_{r_1} rounds is always ΔS_I. The differential trail is then fulfilled probabilistically with many such message pairs, and collisions can be found if the first d bits of ΔS_O are zero.

Given an n_{r_2}-round differential of probability 2^{-w}, our $(n_{r_1} + n_{r_2})$-round collision attack proceeds in two stages:

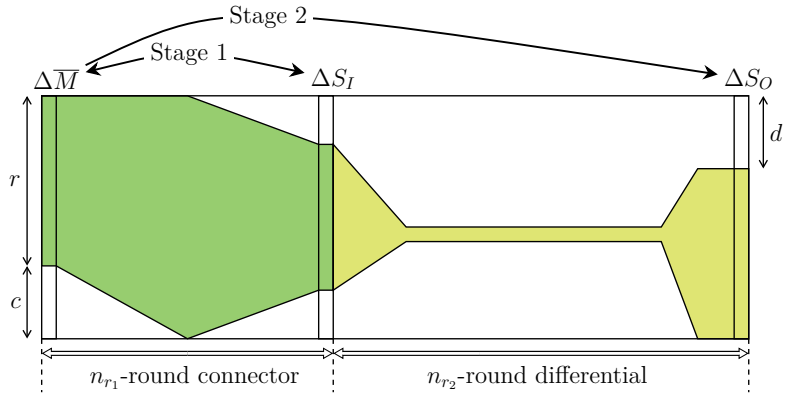

Figure 2: Overview of $(n_{r_1} + n_{r_2})$-round collision attacks

- *Stage 1 — Connecting stage:* Construct an n_{r_1}-round connector and get a subspace of messages bypassing the first n_{r_1} rounds.
- *Stage 2 — Brute-force searching stage:* By brute force, find a colliding pair following the n_{r_2}-round differential trail from the subspace found in stage 1. The time complexity of this stage is 2^w.

The brute-force searching stage is simple, though it may be time-consuming. Therefore, the core steps of the attack are finding good differentials and constructing connectors. In [10], Dinur *et al.* explored a method, which they call *target difference algorithm*, to construct 1-round connectors, namely, to find message pairs (M, M') such that the output difference after one round permutation is ΔS_I. In the next section, we show an algebraic method to extend this connector to two and three rounds, followed by details of the differential trail search in Section 5.

Without further specification, we assume in this paper the messages used are of one block after padding. To fulfil the Keccak padding rule, one needs to fix the last bit of the padded message to be "1", hence the first $r-1$ bits of the state are under the full control of the attacker through the message bits, and the last c bits of the state are fixed to zero as in the initial state specified by Keccak. When applied to SHA3-d, $d \in \{224, 256, 384, 512\}$ (resp. SHAKE), there are $r-4$ (resp. $r-6$) free bits under control, by setting the last 4 (resp. 6) bits of the padded message to be '1110' (resp. '111111') so to be compatible with the specific padding rule. For convenience, the number of extra fixed bits is denoted by p, and only $r-p$ bits are under control. Overall, the constraints of n_{r_1}-connectors are that the last $c+p$ bits of the initial state are fixed, and that the output difference after n_{r_1} rounds is given and fixed (this is determined by the differential trail to be used). We are to utilize the degree of freedom from the first $r-p$ bits of the initial state to find solutions efficiently. As we are aiming at low complexity attacks, finding solutions of connectors should be practical. For

the same reason, the probability of the differential trail should be sufficiently high.

4 S-box Linearization and Connectors

We observe that the number of degrees of freedom for KECCAK instances is $r-p$ which is large for instances like KECCAK$[1440, 160, 5, 160]$. One can then choose some message subsets with special properties such that the starting difference of a given trail is fulfilled deterministically. Since we intend to extend the connector to more than one round, it is natural to consider linearizing the nonlinear mapping χ of the first round by exploiting degrees of freedom, *i.e.*, the expression of each S-box in the first round can be re-written as a linear transformation when the input is restricted to certain subsets. Once χ is linearized, the entire first round becomes linear and then 2-round connectors are possible by applying the core ideas of Dinur *et al.*'s target difference algorithm [10] to the second round.

In this section, we elaborate on techniques for linearizing the KECCAK S-box, both fully and partially, based on which 2-round and 3-round connectors are achieved.

4.1 S-box Linearization

It is obvious that the KECCAK S-box is non-linear when the entire 2^5-sized input space is considered. However, affine subspaces of size up to 4, as to be shown below, could be found so that the S-box can be linearized. Note that the S-box is the only nonlinear mapping of the KECCAK round function. Hence, the entire round function becomes linear when all S-boxes are restricted to such subspaces. Formally, we define:

Definition 1 (Linearizable affine subspace). *Linearizable affine subspaces are affine input subspaces on which S-box substitution is equivalent to a linear transformation. If V is a linearizable affine subspace of an S-box operation $\mathtt{S}(\cdot)$, then $\forall x \in V, \mathtt{S}(x) = A \cdot x + b$, where A is a 5×5 matrix and b is a 5-bit constant vector.*

For example, when input is restricted to the affine subspace {00000, 00001, 00100, 00101} ({00, 01, 04, 05} in hex), where the bits are written as $x_4x_3x_2x_1x_0$, the corresponding output set of the KECCAK S-box is {00000, 01001, 00101, 01100} ({00, 09, 05, 0C} in hex), and the expression of the S-box can be re-written as a linear transformation:

$$y = \begin{pmatrix} 1\,0\,1\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 1\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \end{pmatrix} \cdot x$$

where x and y are bit vector representations of input and output values of the KECCAK S-box with x_0 at the top. By rotation symmetry, four more linearizable affine subspaces can be deduced from the one above.

Exhaustive search for the linearizable affine subspaces of the KECCAK S-box shows:

Observation 1 *Out of the entire 5-dimensional input space,*

a. *there are totally 80 2-dimensional linearizable affine subspaces, as listed in Table 7 in Appendix A.*
b. *there does not exist any linearizable affine subspace with dimension 3 or more.*

For completeness, any 1-dimensional subspace is always linearizable.

Since the affine subspaces are to be used together with differential trails, we are especially interested in linearizable affine subspaces with fixed input and output differences, which is more relevant with the differential distribution table (DDT) of the S-box. We observe:

Observation 2 *Given a 5-bit input difference δ_{in} and a 5-bit output difference δ_{out} such that $\mathtt{DDT}(\delta_{in}, \delta_{out}) \neq 0$, i.e., the solution set $V = \{x : \mathtt{S}(x)+\mathtt{S}(x+\delta_{in}) = \delta_{out}\}$ is not empty, we have*

a. *if $\mathtt{DDT}(\delta_{in}, \delta_{out}) = 2$ or 4, then V is a linearizable affine subspace.*
b. *if $\mathtt{DDT}(\delta_{in}, \delta_{out}) = 8$, then there are* six *2-dimensional subsets $V_i \subset V, i = 0, 1, \cdots, 5$ such that $V_i (i = 0, 1, \cdots, 5)$ are linearizable affine subspaces.*

It is interesting to note the 2-dimensional linearizable affine subspaces obtained from the analysis of DDT cover all the 80 cases in Observation 1. It is already noted in [15] there is one-to-one correspondence between linearizable affine subspaces and entries with value 2 or 4 in DDT. As for the DDT entries of value 8, there are 6 choices of 2-dimensional linearizable affine subspaces. As an example, the 3-dimensional affine subspace corresponding to $\mathtt{DDT}(\mathtt{01}, \mathtt{01})$, *i.e.*, with both input and output differences being 01, is $\{\mathtt{10}, \mathtt{11}, \mathtt{14}, \mathtt{15}, \mathtt{18}, \mathtt{19}, \mathtt{1C}, \mathtt{1D}\}$ and the six 2-dimensional linearizable affine subspaces from it are

$$\{\mathtt{10}, \mathtt{11}, \mathtt{14}, \mathtt{15}\}, \{\mathtt{10}, \mathtt{11}, \mathtt{1C}, \mathtt{1D}\}, \{\mathtt{14}, \mathtt{15}, \mathtt{18}, \mathtt{19}\},$$
$$\{\mathtt{10}, \mathtt{11}, \mathtt{18}, \mathtt{19}\}, \{\mathtt{18}, \mathtt{19}, \mathtt{1C}, \mathtt{1D}\}, \{\mathtt{14}, \mathtt{15}, \mathtt{1C}, \mathtt{1D}\}.$$

When projected to the whole KECCAK state, the direct product of affine subspaces of each individual S-box form affine subspaces of the entire state with larger dimensions. In other words, when all the S-boxes in the round function are linearized, the entire round function becomes linear.

4.2 Non-full S-box Linearization

Suppose we are to construct a connector of two rounds. Given the input difference ΔS_I of the differential trail, *i.e.*, the output difference of the connector α_2, it may be the case that some output bits of the S-boxes in the first round of the connector do not influence ΔS_I. The S-boxes with such output bits do not need full linearizations. That is, it is enough to restrict the input to certain subsets

such that only the output bits of the first round affecting ΔS_I are to be expressed with a linear relation with input bits. Such linearization is called *non-full S-box linearization* and may contribute to a lower consumption of degree of freedom, as to be shown in Observation 3 and 4 below.

Before presenting the two observations, let us introduce some notations. Let the input and output value of χ_0 (the χ mapping of the first round) be x and y respectively. Let $u = (u_0, u_1, \cdots, u_{b-1})$ be a flag vector where $u_i = 1$ $(0 \leq i < b)$ if y_i influences ΔS_I, otherwise $u_i = 0$. Let $U = (U_0, U_1, \cdots, U_{b/5-1})$ where $U_i = u_{5i}u_{5i+1}u_{5i+2}u_{5i+3}u_{5i+4}$, $0 \leq i < b/5$. For the i-th S-box of χ_0, if U_i is not 00000, *aka.* some bits of the corresponding S-box will influence ΔS_I, this S-box should be fully or partially linearized. The value of u is determined by β_1 and α_2 (ΔS_I), which will be further illustrated in the next subsection. In this subsection, we suppose u is already known and focus on non-full S-box linearizations.

Observation 3 *For a non-active* Keccak *S-box, when* U_i *is not* `11111`,

a. `if` $U_i = 00000$`, it does not require any linearization;`
b. `if` $U_i \in \{$`00001, 00010, 00100, 01000, 10000, 00011, 00110, 01100, 11000, 10001`$\}$`, at least 1 degree of freedom is consumed to linearize the output bit(s) of the S-box marked by` U_i`;`
c. `otherwise, at least 2 degrees of freedom are consumed to linearize the output bits of the S-box marked by` U_i`.`

This observation comes from the algebraic relation between the input and output of χ. Suppose the 5-bit input of the S-box is $x_4x_3x_2x_1x_0$ and the 5-bit output $y_4y_3y_2y_1y_0$. Then the algebraic normal forms of the S-box are as follows.

$$\begin{aligned} y_0 &= x_0 + (x_1 + 1) \cdot x_2, \\ y_1 &= x_1 + (x_2 + 1) \cdot x_3, \\ y_2 &= x_2 + (x_3 + 1) \cdot x_4, \\ y_3 &= x_3 + (x_4 + 1) \cdot x_0, \\ y_4 &= x_4 + (x_0 + 1) \cdot x_1. \end{aligned}$$

Take $U_i =$ `00001` as an example. It indicates that y_0 should be linearized. As can be seen, the only nonlinear term in the expression of y_0 is $x_1 \cdot x_2$. Fixing the value of either x_1 or x_2 makes y_0 linear. Without loss of generality, the value of x_1 is assumed to be fixed to 0 or 1. When $x_1 = 0$, we have $y_0 = x_0 + x_2$; otherwise $y_0 = x_0$. When $U_i =$ `01111`, it maps to 4 output bits y_0, y_1, y_2, y_3 which should be linearized. We can fix the value of two bits x_2 and x_4 only. Once x_2 and x_4 are fixed, the nonlinear terms in the algebraic form of all y_0, y_1, y_2, y_3 will disappear. Other cases work similarly. If $U_i =$ `11111`, a full linearization is required by fixing the value of any three input bits which are not cyclically continuous, e.g., (x_0, x_2, x_4).

As noted in Observation 1 and 2, it costs at least three degrees of freedom to fully linearize an S-box, even if it has `DDT` entry of 8. However, Observation 4 shows that two degrees of freedom may be enough to partially linearize an S-box of `DDT` entry 8, and thus 1 bit degree of freedom could be saved.

Observation 4 *For a 5-bit input difference δ_{in} and a 5-bit output difference δ_{out} such that $\mathtt{DDT}(\delta_{in}, \delta_{out}) = 8$, 4 out of 5 output bits are already linear if the input is chosen from the solution set $V = \{x \ : \ \mathtt{S}(x) + \mathtt{S}(x + \delta_{in}) = \delta_{out}\}$.*

Take $\mathtt{DDT}(\mathtt{01}, \mathtt{01}) = \mathtt{8}$ as an example (see Table 6 of [24]). The solution set is $V = \{\mathtt{10}, \mathtt{11}, \mathtt{14}, \mathtt{15}, \mathtt{18}, \mathtt{19}, \ \mathtt{1C}, \mathtt{1D}\}$. We re-write these solutions in the following 5-bit strings where the bits are written as $x_4x_3x_2x_1x_0$.

$$\mathtt{10} : \mathbf{1}\mathtt{00}\mathbf{0}\mathtt{0}, \mathtt{11} : \mathbf{1}\mathtt{00}\mathbf{0}\mathtt{1}, \mathtt{14} : \mathbf{1}\mathtt{01}\mathbf{0}\mathtt{0}, \mathtt{15} : \mathbf{1}\mathtt{01}\mathbf{0}\mathtt{1},$$
$$\mathtt{18} : \mathbf{1}\mathtt{10}\mathbf{0}\mathtt{0}, \mathtt{19} : \mathbf{1}\mathtt{10}\mathbf{0}\mathtt{1}, \mathtt{1C} : \mathbf{1}\mathtt{11}\mathbf{0}\mathtt{0}, \mathtt{1D} : \mathbf{1}\mathtt{11}\mathbf{0}\mathtt{1}.$$

It is easy to see for the values from this set, $x_1 = 0$ and $x_4 = 1$ always hold, making y_0, y_2, y_3, y_4 linear since their algebraic forms could be rewritten as

$$\begin{aligned} y_0 &= x_0 + x_2, \\ y_1 &= (x_2 + 1) \cdot x_3, \\ y_2 &= x_2 + x_3 + 1, \\ y_3 &= x_3, \\ y_4 &= 1. \end{aligned}$$

If the only nonlinear bit y_1 does not influence ΔS_I, setting $x_1 = 0$ and $x_4 = 1$ (consuming two degrees of freedom) is enough for linearizing the remaining four bits.

Therefore, for an S-box, both active and non-active, it may consume less than three degrees of freedom for non-full linearizations, which allows to producing relatively large message spaces for the brute-force collision searching stage.

4.3 2-Round Connector

The core idea of our 2-round connector is to convert the problem to that of solving a system of linear equations. Note the first two rounds of Keccak permutation can be expressed as $\chi_1 \circ L \circ \chi_0 \circ L$ (omitting the ι), as depicted in Fig. 3. Using the techniques discussed above, χ_0 which has fixed input and output differences can be (partially) linearized, *i.e.*, the first three operations $L \circ \chi_0 \circ L$ become (partially) linear. For convenience, the differential transition of the i-th round is denoted by $\alpha_{i-1} \xrightarrow{L} \beta_{i-1} \xrightarrow{\chi} \alpha_i$, where $i = 1, 2, \cdots$. We will give details of the method how input and output differences of χ_0, *i.e.*, β_0 and α_1, are selected later. Now, we show how the χ_1 can be inverted by adding more linear equations of constraints.

In the setting of a 2-round connector, the output difference of χ_1, *i.e.*, α_2 is given as ΔS_I — the input difference of the differential trail. It is not necessary that all S-boxes of χ_1 are active, *i.e.*, with a non-zero difference. Here only active S-boxes are concerned, and each of them is inverted by randomly choosing an input difference with a non-zero number of solutions which is called *compatible* input difference. Formally, given the output difference δ_{out} for the Keccak S-box, its compatible input differences are from the set $\{\delta_{in} : \mathtt{DDT}(\delta_{in}, \delta_{out}) \neq 0\}$.

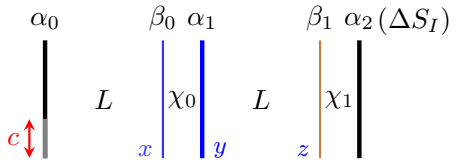

Figure 3: The first two rounds of KECCAK permutation

As noted previously in [4, 8, 10], for any pair of $(\delta_{in}, \delta_{out})$, the solution set $V = \{x : \mathtt{S}(x) + \mathtt{S}(x + \delta_{in}) = \delta_{out}\}$ forms an affine subspace. In other words, V can be deduced from the set $\{0,1\}^5$ by setting up i constraints that turn out to be binary linear equations, when the size of the solution set V is 2^{5-i}. For example, corresponding to $\mathtt{DDT}(\mathtt{03},\mathtt{02})$ is the 2-dimensional affine subspace $\{\mathtt{14},\mathtt{17},\mathtt{1C},\mathtt{1F}\}$ which can be formulated by the following three linear equations as the constraints:

$$\begin{pmatrix} 0\,0\,1\,0\,0 \\ 1\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1 \end{pmatrix} \cdot x = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

It is important to note, under the i linear equations or set V, δ_{in} propagates to δ_{out} deterministically. Hence, each active S-box in χ_1 is inverted by a choice of compatible input difference together with the corresponding i linear equations on the input values. Once the input difference and linear equations for all active S-boxes of χ_1 are enforced and fulfilled together with the linearization of the first round, solutions of the 2-round connector are found. Note that a compatible input difference of χ_1 is a choice of β_1, and α_1 can be uniquely determined by the relation $\alpha_1 = L^{-1}(\beta_1)$. In the remaining part of this subsection, more details on the implementation of this idea are given.

Generation of linear equations. As depicted in Fig. 3, the variables of the equation system are the bit values before χ_0 denoted by vector x. Additionally, y and z are bit vectors of intermediate values for further interpretation where y represents the output after χ_0 and z the bits before χ_1. The main task is to derive all constraints on differences and affine subspaces into that on the variables x. Suppose β_1 and β_0 (details will be given in Section 4.6) are fixed, and ΔS_I (*aka.* α_2) is given, we show how the system of equations could be set up.

With the input difference β_1 and output difference α_2 of χ_1, all the linear equations on the input affine subspaces of the active S-boxes in the second round can be derived and expressed as

$$B \cdot z = t_B, \tag{1}$$

where B is a block-diagonal matrix in which each diagonal block together with corresponding constants in t_B formulates the equations of one active S-box.

A similar procedure can be done for input affine subspaces of the first round given β_0 and α_1, and the corresponding linear equations on x is denoted by

$$A_1 \cdot x = t_{A_1}. \tag{2}$$

Note an additional constraint x needs to fulfill is that the last $(c+p)$-bit of initial state are pre-fixed, which can be expressed as

$$A_2 \cdot x = t_{A_2}. \tag{3}$$

where A_2 is a submatrix of L^{-1} and t_{A_2} is the preset value. Apart from Eq. (2) and (3), some extra equations are imposed to linearize χ_0. To derive these equations, we need to specify the flag vector u identifying the output bits of χ_0 which affect ΔS_I and should be linearized. In fact, the value of u is determined by Eq. (1) which can be re-expressed as

$$B \cdot L \cdot (y + RC_0) = t_B, \tag{4}$$

since $z = L \cdot (y + RC_0)$. Then $u_i = 1$ if y_i appears in Eq. (4) with a non-zero coefficient, otherwise $u_i = 0$. Recall that $U = (U_0, U_1, \cdots, U_{b/5-1})$ where $U_i = u_{5i}u_{5i+1}u_{5i+2}u_{5i+3}u_{5i+4}$, $0 \leq i < b/5$. According to Observation 2, S-boxes with `DDT` entry being 2 or 4 are already linearized due to the fact that the input is confined to the solution set $V = \{x : \mathtt{S}(x)+\mathtt{S}(x+\delta_{in}) = \delta_{out}\}$ by linear equations in (2). So we only need to handle non-active S-boxes and active S-boxes with `DDT` entry 8.

Full S-box linearization. $U_i = \mathtt{11111}$ indicates that fully linearizing the i-th S-box is required.

- For non-active S-boxes, randomly choose any 2-dimensional linearizable affine subspace from 80 such subspaces by imposing 3 linear equations on x.
- For active S-boxes with `DDT` entry 8, randomly choose any 2-dimensional linearizable affine subspace from 6 such subspaces by imposing an extra linear equation on x (2 linear equations are already included in Eq. (2)).

Non-full S-box linearization. $U_i \neq \mathtt{11111}$ indicates that only part of the output bits of the i-th S-box needs to be linearized.

- For non-active S-boxes, linearize the outputs marked by u by imposing a least possible number of equations according to Observation 3. At most 2 linear equations are required.
- For each active S-box with `DDT` 8, if its only non-linear output bit is not marked by U, then we need not handle it. Otherwise, impose an extra linear equation on x in a similar way as what we do for full S-box linearizations.

We denote these extra linear equations for linearizing χ_0 by

$$A_3 \cdot x = t_{A_3}. \tag{5}$$

Suppose bits in y marked by u now can be computed from x linearly through

$$y = L_{\chi_0} \cdot x + t_{L_{\chi_0}}, \tag{6}$$

and further an equivalent system of Eq. (4) can be obtained as

$$B \cdot L \cdot L_{\chi_0} \cdot x + B \cdot L \cdot (t_{L_{\chi_0}} + RC_0) = t_B. \tag{7}$$

We merge the four systems of linear equations on x together, *i.e.*, Eq. (2), (3), (5) and (7) and denote the resulted system of linear equations by E_M:

$$A \cdot x = t_A. \tag{8}$$

Then, solutions fulfilling E_M will be solutions of the 2-round connector.

Algorithm of the 2-round connector. Since Eq. (3) remains static, and given differences $(\beta_0, \alpha_1, \beta_1, \alpha_2)$, Eq. (2) and (1) are determined, the core part is the generation of Eq. (5), *i.e.*, linearizing χ_0, which is sketched in the procedure *basicLinearization* (see Alg. 2). The procedure *mainLinearization* (see Alg. 1) invokes *basicLinearization* and generates the final system of linear equations E_M whose solutions are exactly the solutions of the 2-round connector.

The inputs to *mainLinearizztion* are $\beta_0, \alpha_1, \beta_1, \alpha_2(\Delta S_I)$ and E_M, where E_M is initialized with Eq. (2) and (3). Similarly, *basicLinearization* takes as input the same β_0, α_1 and E_M, as well as a flag vector U generated in *mainLinearization*. If *basicLinearization* succeeds, it will return to *mainLinearization* the (partial) linear mapping of χ_0, *i.e.*, $(L_{\chi_0}, t_{L_{\chi_0}})$, and the updated E_M. Further, the success of *mainLinearization* will give rise to the final system of equations E_M.

In *basicLinearization*, a procedure named *preProcess* is called on Line 1 which is described more at length in Alg. 3. *preProcess* identifies all possible S-boxes of χ_0 whose output bits marked by U are already linear under the initial E_M, so the number of equations in Eq. (5) will be minimized. In fact, the conditions of Line 11 and 16 would never hold if E_M does not contain Eq. (3), and they hold more likely for large Eq. (3), *i.e.*, large $c + p$. Therefore, *preProcess* is crucial to the success of solving Keccak instances with a relatively large capacity, such as `SHA3-256`.

Note that Alg. 1 and 2 do not succeed all the time. To overcome this problem, we repeat random picks of compatible input differences β_1 (β_0 changes accordingly) for a given ΔS_I, until the main procedure succeeds.

4.4 3-Round Connector

We construct 3-round connectors upon a 2-round connector. Since almost all output bits of the first round affect α_3 (the difference after the third round), full linearizations are enforced on the first nonlinear layer χ_0 while constructing a 2-round connector for the first two rounds. Once a 2-round connector succeeds, the resulted system of equations E_M on x can be re-expressed as an equivalent

Algorithm 1: mainLinearization.

Input: E_M, $\beta_0, \alpha_1, \beta_1$, and α_2
Output: updated E_M

Derive Eq. (1) from (β_1, α_2);
Derive Eq. (4) and U;
Initialize a counter *cnt* to a large integer;
while $cnt > 0$ **do**
    // Linearize χ_0 and obtain (L_{χ_0}, t_{χ_0}) and update E_M.
    if *basicLinearization*$(E_M, \beta_0, \alpha_1, U)$ *succeeds* **then**
        Derive Eq. (7) using $(L_{\chi_0}, t_{L_{\chi_0}})$ output by *basicLinearization*;
        if *Eq. (7) is consistent with* E_M **then**
            Add Eq. (7) to E_M;
            Output E_M;
            return success;
    $cnt = cnt - 1$;
return failure;

Algorithm 2: basicLinearization.

Input: E_M, β_0, α_1, and U
Output: updated E_M, L_{χ_0}, t_{χ_0}

l_{sb},$L_{\chi_0}, t_{L_{\chi_0}}$ = preProcess(β_0, α_1, U); // Generate the primary $L_{\chi_0}, t_{L_{\chi_0}}$ and a list of S-boxes that need further processing.
for *each S-box sb in* l_{sb} **do**
    Initialize an empty list l_{lin} of set of equations;
    if *sb is active* **then**
        Load to l_{lin} the 6 sets; // Each set has 1 equation.
    else
        Load to l_{lin} all sets of least possible equations that allow linearization of output bits marked by U;
    while *There is any untested set in* l_{lin} **do**
        Randomly choose an untested set of equations;
        if *The set of equations are consistent with* E_M **then**
            Add it to E_M and update $L_{\chi_0}, t_{L_{\chi_0}}$;
            Break;
        else
            if *All sets have been tested* **then**
                Output "No solution in *basicLinearization*";
                return failure;
Output E_M, L_{χ_0}, t_{χ_0}; // E_M is updated with Eq. (5).
return success;

Algorithm 3: preProcess.

Input: β_0, α_1, U

Output: A list l_{sb} of S-boxes that need further processing, $L_{\chi_0}, t_{L_{\chi_0}}$

```
Initialize an empty list l_sb, a matrix L_χ0 and a vector t_{L_χ0};
for i = 0 → b/5 − 1 do
    if U_i > 0 then
        Extract δ_in, δ_out of the i-th S-box from β_0, α_1;
        if DDT(δ_in, δ_out) = 2, 4 then
            Update L_χ0, t_{L_χ0};
        if DDT(δ_in, δ_out)) = 8 then
            if The only nonlinear output bit is not marked by U_i then
                Update L_χ0, t_{L_χ0};
            else
                if Among the 6 2-dimensional linearizable subspaces, some one
                  is already specified in E_M then
                    Update L_χ0, t_{L_χ0};
                else
                    Add this S-box to l_sb;
        if the S-box is not active then
            if Any set of equations for linearizing the output bits marked by U_i
              is already contained in E_M then
                Update L_χ0, t_{L_χ0};
            else
                Add this S-box to l_sb;
return l_sb,L_χ0, t_{L_χ0};
```

system of equations E'_M on z, *i.e.*, the input value of the second round. It can be considered that the first round disappeared. Next, given E'_M, a 2-round connector is to construct for the second and the third rounds where the second round is partially linearized. Once the second 2-round connector succeeds, its solutions will bypass the first three rounds. That is to say, a 3-round connector is obtained.

Due to a great consumption of degree of freedom, the solution space of a 3-round connector may have a dimension smaller than the weight of the following differential, making it hard to find a real collision with one solution space. However, this problem can be solved by constructing the second 2-round connectors repeatedly. In these 3-round connectors, the degrees of freedom for linearizing the second round can be reused and hence are not consumed. Thus, multiple solution spaces can be generated successively if one is not enough. Such successively generated 3-round connectors are called *adaptive* 3-round connectors, for which a detailed description can be found in Section 7.2 of [26].

4.5 Analysis of Degrees of Freedom

Let the degree of freedom of the final E_M for the connector be $\mathtt{DF}$. In our collision attacks, $\mathtt{DF}$ is a key factor on success. A solution space with $\mathtt{DF}$ larger than the weight of the differential trail is possible to suggest a message pair with digest collision. After linearizing the first round, the degree of freedom is $\sum_{i=0}^{b/5-1} \mathtt{DF}_i^{(1)}$ in which $\mathtt{DF}_i^{(1)}$ is the degree of freedom of 5-bit input space of the i-th S-box in the first round and calculated according to rules in Table 2. This concerns not only the transition from β_0 to α_1, but also the S-box linearizations.

Table 2: Value of $\mathtt{DF}_i^{(1)}$

	$\mathtt{DDT}(\delta_{in}, \delta_{out})$=32	$\mathtt{DDT}(\delta_{in}, \delta_{out})$=2	$\mathtt{DDT}(\delta_{in}, \delta_{out})$=4	$\mathtt{DDT}(\delta_{in}, \delta_{out})$=8
$\mathtt{DF}_i^{(1)*}$	$2 \sim 5$	1	2	2 or 3

* δ_{in} and δ_{out} are the input and output differences of the i-th S-box in the first round. The exact value of $\mathtt{DF}_i^{(1)}$ is based on whether the output bits of the i-th S-box influence ΔS_I, *i.e.*, the value of U_i.

The constraints on the initial state reduce $(c + p)$ degrees of freedom for the pre-fixed $(c + p)$-bit. Another decrement on the degree of freedom is due to the constraints on the input values of the S-box layer in the second round. The definition of $\mathtt{DF}_i^{(2)}$, the degree of freedom of 5-bit input values to S-boxes in the second round, is

$$\mathtt{DF}_i^{(2)} = \begin{cases} 1, & \mathtt{DDT}(\delta_{in}, \delta_{out}) = 2, \\ 2, & \mathtt{DDT}(\delta_{in}, \delta_{out}) = 4, \\ 3, & \mathtt{DDT}(\delta_{in}, \delta_{out}) = 8, \\ 5, & \mathtt{DDT}(\delta_{in}, \delta_{out}) = 32, \end{cases} \tag{9}$$

where δ_{in} and δ_{out} are the input and output differences of the i-th S-box in the second round. For the i-th S-box in the second round, we add $(5 - \mathtt{DF}_i^{(2)})$ equations to E_M and suppose to deduce the degree of freedom by this amount.

Therefore, the degree of freedom of the final E_M of our 2-round connector is estimated as

$$\mathtt{DF} = \sum_{i=0}^{b/5-1} \mathtt{DF}_i^{(1)} - (c + p) - \sum_{i=0}^{b/5-1} (5 - \mathtt{DF}_i^{(2)}). \tag{10}$$

Large $\mathtt{DF}$ benefits our search for collisions in rounds beyond the second round and sufficiently large $\mathtt{DF}$s (say > 140) may allow 3-round connectors.

4.6 How to choose β_2, β_1 and β_0

Choosing β_1 in the 2-round connector. Recall that we randomly choose compatible input differences β_1 according to ΔS_I (α_2) until the 2-round connector succeeds. As the number of active S-boxes in the second round is large

enough (range from tens to hundreds in our attacks), there is a huge number of compatible choices of β_1 so that we can only choose those β_1 such that $\beta_1 \to \alpha_2$ is of the best probability for the given α_2. This allows a speedup of 2-round connectors, as well as an increment of DF. Interestingly, the differential probability of $\beta_1 \to \alpha_2$ does not need to be high because this transition is covered in our 2-round connector.

Choosing β_0 in the 2-round connector. So far we have not given details on how β_0 can be selected. We follow Dinur *et al.*'s work [10] in a more general way to uniquely determine β_0, namely, the difference before χ layer of the first round. Specifically, the so-called "target difference algorithm" is used which consists of a difference phase and a value phase.

Given ΔS_I, we have randomly chosen a compatible input difference β_1. We then build two equation systems E_Δ and E_M accordingly. E_Δ is on differences of the message pairs (specifically, on β_0) and E_M is on the value of one message (specifically, on the value of the message before χ_0, *i.e.*, x as shown in Fig. 3). The initialization of E_Δ should abide by (1) the constraints implied by the $(c+p)$-bit pre-fixed value that the corresponding difference should be 0, and (2) the input difference bits of non-active S-boxes in the first round equal to 0. The initialization of E_M should abide by the pre-fixed value of the last $c+p$ bits. These rules are easy to be implemented as the variable vector x is the image of the initial vector by an invertible linear mapping. Therefore, in the initialization we equate the corresponding bits to their enforced values in E_Δ and E_M.

For E_Δ, we add additional equations so that its solution, *i.e.*, β_0 is compatible with α_1. Though the obvious way is to equate the 5 input difference bits to a specific value for each active S-box in the first round, this will restrict the solution space significantly. As suggested in [10], we chose one of the 2-dimensional affine subsets of input differences instead of a specific value for each active S-box. This is based on the fact that given any nonzero 5-bit output difference to a Keccak S-box, the set of possible input differences contains at least five 2-dimensional affine subspaces. After a consistent E_Δ system is constructed, the solution space is an affine subspace of candidates for β_0. Then we continue to maintain E_Δ by iteratively adding two additional equations to uniquely specify a 5-bit input difference for the active S-boxes. For each active S-box, once the specific input difference is determined, we add equations to E_M system to enforce every active 5-bit of x (input bits to active S-box) to an affine subspace corresponding to the uniquely determined δ_{in} and δ_{out}. In this way, we always find a compatible β_0 for α_1 that fulfills the constraints imposed by the $(c+p)$-bit pre-fixed value.

Choosing β_2 in the 3-round connector. β_2 is chosen before running the 3-round connector. Its value fulfills two requirements: (1) given $\alpha_2 = L^{-1}(\beta_2)$, a 2-round connector for the first two rounds can be obtained with a sufficiently large degree of freedom; and (2) $\beta_2 \to \alpha_3$ is of high probability where α_3 is the input difference of the following differential trail. While constructing 3-round connectors, specifically the second 2-round connector, β_2 is fixed and does not

change. In this way, the second 2-round connector can be constructed efficiently thanks to the high probability of $\beta_2 \to \alpha_3$.

5 Search for Differential Trails with GPU

In this section, we elaborate on searching differential trails of Keccak. The idea of searching differential trails greatly benefits from previous differential analysis of Keccak [9, 14, 19, 21]. The associated techniques developed in the previous works are reviewed first, followed by the considerations in finding differential trails that fit in our attacks. Subsequently, the exact searching algorithm that provides differential trails for practical collision attacks against Keccak$[640, 160, 5, 160]$, 5-round `SHAKE128`, 5-round `SHA3`-224, 5-round `SHA3`-256, Keccak$[1440, 160, 5, 160]$, and Keccak$[1440, 160, 6, 160]$ is illustrated. To speed up the searching process, we introduce GPU implementations of Keccak. The differential trails to be used in our collision attacks are listed at the end of this section.

5.1 Differential Properties of Keccak

In this subsection, we recall special properties of the linear and nonlinear layer of Keccak round function which have been identified in previous works. In the following paragraphs, we follow the notations from [9].

Key properties used in differential analysis. The concept of *column parity*, defined in [4], is the most critical property of θ operation. Formally, the *column parity* (or parity for short) $P(A)$ of a value (or a difference) A is defined as the parity of the columns of A, *i.e.*, $P(A)[i][k] = \Sigma_j A[i][j][k]$. A column is even, if its parity is 0, otherwise it is odd. A state is in Column Parity kernel (CP-kernel for short) if all of its columns are even. θ adds a pattern, called the θ-effect, to the state. The θ-effect of a state A is $E(A)[i][k] = P(A)[i-1][k] + P(A)[i+1][k-1]$. Thus the effect of θ depends only on column parities. Given a state A in CP-kernel, the Hamming weight of A remains unchanged after θ. Another interesting property is that θ^{-1} diffuses much faster than θ. To be precise, a single bit difference can propagate into half state bits through θ^{-1} on average.

Regarding the nonlinear operation S-box, given a non-zero input difference, all compatible output differences occur with the same probability. Particularly, for input differences with one active bit, the S-box acts as identity with probability 2^{-2}. However, given an output difference of the S-box, the probabilities of compatible input differences vary even though the best probability is determined. When the best probability is only 2^{-3}, there are multiple input differences achieving this. Therefore, given an output difference of χ, usually there are multiple input differences fulfilling the best probability.

Representation of trails and their weights. As in previous sections, we denote the differences before and after the i-th round by α_{i-1} and α_i, respectively. Let $\beta_i = L(\alpha_i)$, then an n-round differential trail starting from the 1-st round is of the following form

$$\alpha_0 \xrightarrow{L} \beta_0 \xrightarrow{\chi} \alpha_1 \xrightarrow{L} \cdots \alpha_{n-1} \xrightarrow{L} \beta_{n-1} \xrightarrow{\chi} \alpha_n.$$

For simplicity, a trail can also be represented with only β_i's or α_i's.

The weight of a differential $\beta \to \alpha$ over a function f with domain $\{0,1\}^b$ is defined as

$$w(\beta \to \alpha) = b - \log_2|\{x : f(x) \oplus f(x \oplus \beta) = \alpha\}|.$$

In other words, the weight of a differential $\beta \to \alpha$ is equal to $-\log_2 \Pr(\beta \to \alpha)$. α and β are compatible when $\Pr(\beta \to \alpha) > 0$, otherwise the weight of $\beta \to \alpha$ is undefined.

The weight $w(\beta_i \to \alpha_{i+1})$ is denoted by w_i, and thus the weight of a trail is the sum of the weights of round differentials that constitute the trail. In addition, $\#\mathrm{AS}(\alpha)$ is used to represent the number of active S-boxes in a state difference α. According to the properties of χ, given β_i the weight of $(\beta_i \to \alpha_{i+1})$ is determined; also, given β_i the minimum reverse weight of $(\beta_{i-1} \to L^{-1}(\beta_i))$ is fixed.

As in [4], $n-1$ consecutive β_i's, say $(\beta_1, \cdots, \beta_{n-1})$ is called an n-round **trail core** which defines a set of n-round trails $\alpha_0 \xrightarrow{L} \beta_0 \xrightarrow{\chi} \alpha_1 \xrightarrow{L} \beta_1 \cdots \xrightarrow{L} \beta_{n-1} \xrightarrow{\chi} \alpha_n$ where the first round is of the minimal weight determined by $\alpha_1 = L^{-1}(\beta_1)$, and α_n is compatible with β_{n-1}. The first step of mounting collision attacks against n-round Keccak is to find good $(n-1)$-round trail cores.

5.2 Requirements for Differential Trails

Good trail cores are those satisfying all the requirements which will be explained as follows. Firstly, the difference of the output needs to be zero, *i.e.*, $\alpha^d_{n_r} = 0$ ($\alpha^d_{n_r}$ represents the first d bits of α_{n_r}).

Secondly, the consumption of degree of freedom must be within budget. With the definition of weight, Eq. (10), *i.e.*, the estimation of DF of our 2-round connectors can be represented in an alternative way

$$\mathtt{DF} = \sum_{i=0}^{b/5-1} \mathtt{DF}_i^{(1)} - (c+p) - w_1. \tag{11}$$

The first term of the formula depends on the output bits of the first round that need to be linearized and DDT entries of active S-boxes as depicted in Table 2. Empirically, the inputs of most S-boxes are confined to 2-dimensional subsets. Therefore, we heuristically set $\frac{b}{5} \times 2$ as a threshold for the first term in (11), and denote a threshold of the first two terms in (11) for further search conditions by

$$\mathtt{TDF} = \frac{b}{5} \times 2 - (c+p).$$

Note that this is a **pessimistic estimation** of the first two terms. The actual value is usually higher due to the technique of non-full S-box linearization and the fact that there may exist dependencies in the system of equations used for constructing connectors.

In order to mount collision attacks against KECCAK$[r, c, n_r, d]$ with methods described in Section 4, it is necessary that

$$\texttt{TDF} > w_1 + \cdots + w_{n_r-2} + w^d_{n_r-1} \tag{12}$$

where $w^d_{n_r-1}$ is the portion of w_{n_r-1} that relates to the digest. The trail searching phase is performed to provide ΔS_I for the connector. However, the conditions for a good trail core is restrained by results of the connector, *i.e.*, the degree of freedom of the solution space. So we take (12) as a heuristic condition for searching good trail cores which are promising for collision attacks.

Thirdly, the collision attack should be practical. Note that after we obtain a subspace of message pairs that fulfill the first two or three rounds, the complexity of searching a collision is 2^w, where $w = w_{n_r-3} + \cdots + w^d_{n_r-1}$ is the weight of the differential trail over the last three rounds. To make our attacks practical, we restrict w to be small enough, say 55.

We summarize the requirements for differential trails as follows and list TDFs for different variants of KECCAK$[r, c, n_r, d]$ in Table 3.

(1) $\alpha^d_{n_r} = 0$, *i.e.*, the difference of output must be zero.
(2) $\texttt{TDF} > w_1 + \cdots + w^d_{n_r-1}$, *i.e.*, the degree of freedom must be sufficient;
(3) $w = w_{n_r-3} + \cdots + w^d_{n_r-1} \leq 55$, the complexity of finding a collision should be practical.

Table 3: TDFs of different versions of KECCAK$[r, c, n_r, d]$

KECCAK$[r, c, n_r, d]$	TDF	Remarks
KECCAK$[1440, 160, 5, 160]$	479	KECCAK contest
KECCAK$[1344, 256, 5, 256]$	378	SHAKE128
KECCAK$[\ 640, 160, 5, 160]$	159	KECCAK contest
KECCAK$[1440, 160, 6, 160]$	479	KECCAK contest
KECCAK$[1152, 448, 5, 224]$	188	SHA3-224
KECCAK$[1088, 512, 5, 256]$	124	SHA3-256

5.3 Searching Strategies

In 5-round collision attacks on KECCAK, we are to combine a 2-round connector and a 3-round differential trail. Given α_2, *i.e.*, the input difference of the differential trail over the last three rounds, the minimal weight of the second round w_1

is determined. Note that, the second round is covered by the connector, and the smaller w_1, the fewer constraints imposed to the connector. In order to make it practical to construct 2-round connectors, we aim to find 3-round differential trails that impose a least possible number of constraints to the connector. Namely, w_1 should be as small as possible. Actually, this means we need to find good 4-round differential trails cores. Similarly, in 6-round collision attacks, a 3-round connector is combined with a 3-round differential trail. As shown in Section 4.4, $\beta_2, \beta_3(\alpha_3)$ are known and fixed in the 3-round connector, so 5-round trails cores are needed, whose first two rounds will be covered by the connector.

Differential trails with two rounds in CP-kernel. The differential trails with two rounds in CP-kernel are searched to fulfill the requirements discussed above. The 2-round trail cores, denoted by (β_3), that ensure α_3 and at least one choice of α_4 in CP-kernel are the starting point of our algorithm for searching differential trails. A detailed analysis of the reason to do so is postponed to Appendix C.1. By extensively making use of the KeccakTools [5] developed by the Keccak Team, we generate such 2-round trails cores, based on which trail cores of 4 or 5 rounds for our collision attacks are obtained it the following way.

Algorithm for searching differential trails. We sketch below our steps for finding 4-round differential trail cores of Keccak, upon which 5-round collision attacks are mounted, as illustrated in Fig. 4.

1. Using KeccakTools, find special β_3's with a low Hamming weight, say 10.
2. For every β_3 obtained, traverse all possible α_4 using a tree structure, compute $\beta_4 = L(\alpha_4)$ and test whether there exists a compatible α_5 where $\alpha_5^d = 0$ (Requirement (1)). If so, keep this β_3 and record its forward extension, otherwise discard it.
3. For the remaining β_3's, traverse all possible β_2's which are compatible with $L^{-1}(\beta_3)$'s. For the trail core $(\beta_2, \beta_3, \beta_4)$, check Requirement (2) and (3). If these two requirements are satisfied, then output $(\beta_2, \beta_3, \beta_4)$.

We provide a description in more details in Appendix C.2. To mount collision attacks on 6-round Keccak, 5-round differential trail cores are needed. In this case, we just extend forward the 4-round trail cores by one more round, as shown in Fig. 5.

5.4 GPU Implementations of Keccak

Techniques for GPU implementations of Keccak are introduced to improve our computing capacity. While one could expect a speed of order 2^{21} Keccak-f evaluations per second on a single CPU core, we show in this section this number could increase to 2^{29} per second on NVIDIA GeForce GTX1070 graphic card. The significant speedup will benefit us in two stages: searching for differential trails among larger spaces and brute-force search of collisions following differential trails with lower probability.

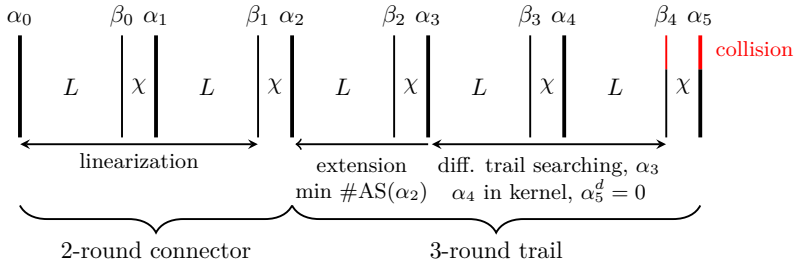

Figure 4: Search for differential trails and collision attacks on 5-round KECCAK.

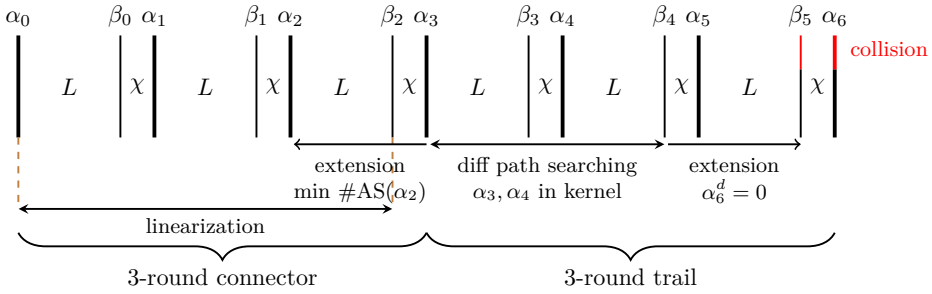

Figure 5: Search for differential trails and collision attacks on 6-round KECCAK.

Overview of GPU and CUDA. GPUs (Graphics Processing Unit) are intended to process the computer graphics and image originally. With more transistors for data processing, a GPU usually consists of thousands of smaller but efficient ALUs (Arithmetic Logic Unit), which can be used to process parallel tasks efficiently. CUDA is a general purpose parallel computing architecture and programming model that is used in NVIDIA GPUs [23]. One of the programming interfaces of CUDA is CUDA C/C++ which is based on standard C/C++. Here, we mainly focus on CUDA C++.

To better understand the techniques for implementations, some basic concepts are introduced. From the view of hardware architecture, a GPU is comprised of several SMs (Streaming Multiprocessors), which determine the parallelization capability of GPU. In Maxwell architecture, each SM owns 128 SPs (streaming processors) — the basic processing units. Warp is the basic execution unit in SM and each warp consists of 32 threads. All threads in a warp execute the same instructions at the same time. Each thread will be mapped into a SP when it is executed.

Existing implementations and our implementations. Guillaume Sevestre [25] implemented KECCAK in a tree hash mode, the nature of which allows each thread to run a copy of KECCAK. Unfortunately, there are no implementation details given. In [7], Pierre-Louis Gayrel *et al.* implemented KECCAK-f[1600]

with 25 threads that calculate all 25 lanes in parallel in a warp and these threads cooperate via the shared memory. One disadvantage of this strategy is bank conflict — concurrent access to the shared memory of the same bank by threads from the same warp will be forced to be sequential. Besides, there are two open-source softwares providing GPU implementations of KECCAK: ccminer (ref. http://ccminer.org) and hashcat (ref. https://hashcat.net) in CUDA and OpenCL, respectively.

Having learnt from the existing works and codes, we implemented KECCAK following two different strategies: one thread for one KECCAK or one warp for one KECCAK. From experimental results, we find that one thread for one KECCAK gives a better number of KECCAK-f evaluations per second. So we adopt this strategy in this paper. More detailed techniques of implementation optimization are introduced in Appendix B.1.

Benchmark. With all the optimization techniques in mind, we implemented KECCAK-f[1600] in CUDA, and have it tested on NVIDIA GeForce GTX1070 and NVIDIA GeForce GTX970 graphics cards. The hardware specifications of GTX1070 and GTX970 are given in Appendix B.2.

Table 4: Benchmark of our KECCAK implementations in CUDA

Target	KECCAK-f evaluations per second	GPU
KECCAK-f[1600]$v1$	$2^{28.90}$	GTX1070
KECCAK-f[1600]$v2$	$2^{29.24}$	GTX1070
KECCAK-f[1600]$v1$	$2^{27.835}$	GTX970
KECCAK-f[1600]$v2$	$2^{28.37}$	GTX970

Table 4 presents the performance, where KECCAK-f[1600]$v1$ and KECCAK-f[1600]$v2$ are our implementations used to search for differential trails and to find real collisions in the brute-force stage, respectively. The difference between the two versions is: KECCAK-f[1600]$v1$ copies all digests into global memory, and KECCAK-f[1600]$v2$ only copies the digest into global memory when the resulted digest equals to a given digest value. For both versions we did not include the data transfer time. It can be seen that roughly GTX1070 can be 2^8 times faster than a CPU core. The source codes of these two versions are available freely via http://team.crypto.sg/Keccak_GPU_V1andV2.zip.

5.5 Searching results

Some of the best differential trail cores we obtained with the help of GPU are listed in Table 5. As can be seen, Trail cores No. 1∼3 are all suitable for collision attacks against KECCAK[1440,160,5,160] and the 5-round SHAKE128. Trail core

No. 4 is sufficiently good for collision attacks against Keccak[640, 160, 5, 160]. To mount collision attacks on 5-round SHA3-224, 5-round SHA3-256 and Keccak[1440,160,6,160], Trail core No. 3 and 5 are used respectively. Trail core No. 6 is slightly different from the other cores as it leads to an inner collision of the last 160 state bits for Keccak[1440,160,6,160]. Details of these differential trail cores are provided in Appendix D.

Table 5: Differential trail cores of Keccak, where $160^{\dashv}$ stands for an inner collision on the last 160 bits of the state.

No.	$r+c$	$\#AS(\alpha_2\text{-}\beta_2\text{-}\beta_3\text{-}\beta_4^d)$	$w_1\text{-}w_2\text{-}w_3\text{-}w_4^d$	$w_2{+}w_3{+}w_4^d$	d	Trail info.
1	1600	102- 8- 8-2	240-19-16-4	39	256	
2	1600	88- 8- 7-0	195-21-15-0	36	256	Table 9
3	1600	59-10- 9-0	127-24-19-0	43	256	
4	800	38- 8- 8-0	85-20-16-0	36	160	Table 10
No.	$r+c$	$\#AS(\alpha_2\text{-}\beta_2\text{-}\beta_3\text{-}\beta_4\text{-}\beta_5^d)$	$w_1\text{-}w_2\text{-}w_3\text{-}w_4\text{-}w_5^d$	$w_3{+}w_4{+}w_5^d$	d	Trail info.
5	1600	127-9-8-8-10	292-25-18-16-16	50	160	Table 11
6	1600	135-8-7-8-15	317-17-14-16-20	50	$160^{\dashv}$	Table 12

6 Experiments and Results

6.1 Summary of Collision Attacks

In this section, we employ 4-round (5-round) trail cores in Table 5 to mount collision attacks against 5-round (6-round) Keccak. Recall that our collision attack consists of a connecting phase and a brute-force searching stage. Let T_c denote the time consumed by the connecting stage and T_b by the brute-force searching stage. After the connecting stage, a message space with DF degrees of freedom is returned by the connector. If DF is greater than the weight w of the differential trail over the last rounds beyond the connector, a colliding pair will be found in the brute-force searching stage with a very high probability.

Table 6 summarizes seven collision attacks we obtained (for six Keccak variants), and the corresponding timings, DFs of the returned message space and w of the differential trail. With the attacks on Keccak[1440, 160, 5, 160], Keccak[640, 160, 5, 160] and Keccak[1440, 160, 6, 160], we provide the first solutions to three instances of Keccak contest [2]. The source codes for verifying the seven collisions are available via http://team.crypto.sg/VerifyCollisions.zip.

Collisions of 5-round instances. Take the first target in Table 6 as an example. We apply Trail core No. 2 to the collision attack on Keccak[1440, 160, 5, 160].

Table 6: Experimental details of our collision attacks.

Target $[r, c, d]$	n_r	Trail Core	T_c	DF	w	T_b	Collision
Keccak[1440,160,160]	5	No. 2	9.6s$^{\wr}$	162	36	2.48h	Table 13
Keccak[640,160,160]	5	No. 4	0.5h	56	35	2.67h	Table 14
SHAKE128	5	No. 1	0.4h	94	39	0.5h	Table 15
SHA3-224	5	No. 3	11.7h	83	36.70*	29h	Table 16
SHA3-256	5	No. 3	428.8h	37	36.70*	45.6h	Table 17
Keccak[1440,160,160]	6	No. 5	4.5h	174†	47.81*	112h‡	Table 18
Keccak[1440,160,160$^{\dashv}$]	6	No. 6	7.5h	145†	48.83*	60200h	Table 19

$^{\wr}$ Usual computers evaluate around $2^{20} \sim 2^{22}$ full Keccak evaluations per second.
* The weight takes multiple trails of the last two rounds into consideration.
† DF of the message space returned by the 2-round connector for the first two rounds.
‡ The actual time using three NVIDIA GeForce GTX970 GPUs while elsewhere it refers to the time when one CPU core is used.

After solving the 2-round connector in 9.6 seconds, the resulting solution space has a degree of freedom DF of 162 which is sufficiently larger than 36, *i.e.*, the weight of the differential trail over the remaining 3 rounds. The time for searching a collision is 2.48 core hours. We give one example of collisions in Table 13. The attacks on other 5-round targets work similarly.

Collision on Keccak[1440, 160, 6, 160]. Our collision attacks on 6-round instances exploit 3-round connectors to the first three rounds, where a 2-round connector for the first two rounds is constructed first, based on which adaptive 3-round connectors are constructed. The attack on Keccak[1440, 160, 6, 160] uses Trail core No. 5. The 2-round connector of the first two rounds succeeds in 4.5 core hours and returns a message space with 174 degrees of freedom. Every time the second round connector outputs a subspace of messages (*i.e.*, solution of 3-round connectors) with DF $\in [40, 45]$. In order to find one colliding pair, at least $2^{47.81}$ pairs of messages are required. This could be achieved by repeating the second round connector for $2^{2.81} \sim 2^{7.81}$ times. By running our CUDA implementation on three NVIDIA GeForce GTX970 GPUs, the first collision is found in 112 hours, which equals to $2^{49.07}$ message pair evaluations[1]. An example of collision is given in Table 18.

Inner collision on the last 160-bit of Keccak[1440, 160, 6, 160] When an inner collision on the last 160 bits of the state is obtained (*i.e.*, a collision on capacity bits), we could then construct state collisions of messages of any block length more than 2 by choosing the next message block properly. Specifically, we

[1] Our experiment shows $2^{28.87}$ pairs of 5-round Keccak could be evaluated per second on NVIDIA GeForce GTX970 graphic card.

choose certain values for the next message block such that the r-bit difference of the state is cancelled after absorption. Thus, in the next permutation the whole states become identical. This property maintains if all subsequent message blocks are identical, and in such situations we can obtain a collision on the final digest with certainty. With this in mind, a 6-round collision attack on the last 160 bits of KECCAK[1440, 160, 6, 160] is mounted. Similar to the normal collision attack, a 3-round connector is constructed based on Trail core No. 6. The size of the solution space we obtained is around $2^{58} \sim 2^{64}$ which exceeds the data complexity required. To compare with the efficiency of GPU implementations, this collision is obtained with 172 CPU cores consuming approximately 60200 core hours in total. The first collision occurs after enumerating approximately $2^{47.4}$ message pairs. An example of a colliding pair is given in Table 19.

6.2 Technical Analysis of the Results

Choice of β_1. In our 2-round connector, β_1 is randomly chosen such that $\beta_1 \to \alpha_2$ is of the best probability, except KECCAK[1440,160,5,160] for which the connector randomly chooses any compatible β_1. The `DF` of the message space returned by the connector for KECCAK[1440,160,5,160] is 162. In comparison, the first 2-round connector for KECCAK[1440,160,6,160] returns a message space with `DF` $= 174$, even though its weight of the second round w_1 which is covered by the connector is much larger than that of KECCAK[1440,160,5,160], as can be seen in Table 5. This confirms the effect of the choice of β_1 on the resulted `DF`.

S-box linearization. Full S-box linearizations are enforced in 2-round connectors for KECCAK[1440, 160, 5, 160], KECCAK[640, 160, 5, 160] and `SHAKE128`, as well as the first 2-round connector for KECCAK[1440, 160, 6, 160]. Namely, the first round is fully linearized no matter whether it is necessary or not. For `SHA3-224`, `SHA3-256`, S-boxes of the first round are fully linearized only when necessary, otherwise non-full S-box linearizations are used. It is the same case for S-boxes in the second round while constructing the second round connector for KECCAK[1440, 160, 6, 160].

It is worth noting that our collision attacks on 5-round `SHA3-256` and KECCAK[1440, 160, 6, 160] are not possible without the techniques of non-full S-box linearization. Take the 2-round connector for `SHA3-256` as an example. While partially linearizing the first round, (besides the S-boxes with `DDT` entry 2 or 4) there are 26 S-boxes in the first round exempted from consuming extra degrees of freedom due to techniques of non-full S-box linearization. Finally, our 2-round connector returns a space of messages with `DF` almost as large as the weight of the differential trail. Moreover, in this space of messages, we find only one collision. This in turn proves the usefulness and effectiveness of the non-full S-box linearization techniques.

7 Conclusion

We observed that connectors of KECCAK can be extended to multiple rounds using S-box linearizations. In this paper, two types of S-box linearization are proposed, *i.e.*, full S-box linearization and non-full S-box linearization. By linearizing S-boxes in the first round, we extend Dinur *et al.*'s 1-round connector to two rounds. Further, by applying linearization to the first two rounds, 3-round connectors became possible. We combined our connectors with suitable differential trails searched with dedicated strategies, and then obtained seven practical collisions on six round-reduced `SHA-3` and KECCAK variants, including 5-round `SHAKE128`, 5-round `SHA3-224`, 5-round `SHA3-256`, a 6-round instance of KECCAK contest and a variant of it. So far, these are the best collision attacks on round-reduced `SHA-3` and KECCAK contest instances.

Acknowledgments

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Strategic Capability Research Centres Funding Initiative, NTU Research Grant M4080456 and M4082123, and Ministry of Education Singapore Grant M4012049. Guohong Liao is partially supported by the National Natural Science Foundation of China (Grants No. 61572028). Guozhen Liu is partially supported by the State Scholarship Fund (No. 201706230141) organized by China Scholarship Council. Meicheng Liu is partially supported by the National Natural Science Foundation of China (Grants No. 61672516). Kexin Qiao and Ling Song are partially supported by the National Natural Science Foundation of China (Grants No. 61802399, 61802400, 61732021 and 61772519), the Youth Innovation Promotion Association CAS, and Chinese Major Program of National Cryptography Development Foundation (Grant No. MMJJ20180102).

Bibliography

[1] J.-P. Aumasson and W. Meier. Zero-Sum Distinguishers for Reduced Keccak-f and for the Core Functions of Luffa and Hamsi. *rump session of Cryptographic Hardware and Embedded Systems-CHES*, 2009, 2009.

[2] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak Crunchy Crypto Collision and Pre-image Contest. http://keccak.noekeon.org/crunchy_contest.html.

[3] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Cryptographic Sponge Functions. *Submission to NIST (Round 3)*, 2011. http://sponge.noekeon.org/CSF-0.1.pdf.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak Reference. http://keccak.noekeon.org, January 2011. Version 3.0.

[5] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. KeccakTools. http://keccak.noekeon.org/, 2015.

[6] A. Canteaut, editor. *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*. Springer, 2012.

[7] P.-L. Cayrel, G. Hoffmann, and M. Schneider. GPU Implementation of the Keccak Hash Function Family. In *International Conference on Information Security and Assurance*, pages 33–42. Springer, 2011.

[8] J. Daemen. *Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis.* PhD thesis, Doctoral Dissertation, March 1995, KU Leuven, 1995.

[9] J. Daemen and G. V. Assche. Differential Propagation Analysis of Keccak. In Canteaut [6], pages 422–441.

[10] I. Dinur, O. Dunkelman, and A. Shamir. New Attacks on Keccak-224 and Keccak-256. In Canteaut [6], pages 442–461.

[11] I. Dinur, O. Dunkelman, and A. Shamir. Collision Attacks on Up to 5 Rounds of SHA-3 Using Generalized Internal Differentials. In S. Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 219–240. Springer, 2013.

[12] I. Dinur, O. Dunkelman, and A. Shamir. Improved Practical Attacks on Round-Reduced Keccak. *J. Cryptology*, 27(2):183–209, 2014.

[13] I. Dinur, P. Morawiecki, J. Pieprzyk, M. Srebrny, and M. Straus. Cube Attacks and Cube-Attack-Like Cryptanalysis on the Round-Reduced Keccak Sponge Function. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *LNCS*, pages 733–761. Springer, 2015.

[14] A. Duc, J. Guo, T. Peyrin, and L. Wei. Unaligned Rebound Attack: Application to Keccak. In Canteaut [6], pages 402–421.

[15] J. Guo, J. Jean, I. Nikolic, K. Qiao, Y. Sasaki, and S. M. Sim. Invariant Subspace Attack Against Midori64 and The Resistance Criteria for S-box Designs. *IACR Trans. Symmetric Cryptol.*, 2016(1):33–56, 2016.

[16] J. Guo, M. Liu, and L. Song. Linear Structures: Applications to Cryptanalysis of Round-Reduced Keccak. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *LNCS*, pages 249–274, 2016.

[17] J. Jean and I. Nikolic. Internal Differential Boomerangs: Practical Analysis of the Round-Reduced Keccak-f Permutation. In G. Leander, editor, *Fast Software Encryption - FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *LNCS*, pages 537–556. Springer, 2015.

[18] S. Kölbl, F. Mendel, T. Nad, and M. Schläffer. Differential Cryptanalysis of Keccak Variants. In M. Stam, editor, *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*, volume 8308 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2013.

[19] S. Mella, J. Daemen, and G. V. Assche. New Techniques for Trail Bounds and Application to Differential Trails in Keccak. *IACR Trans. Symmetric Cryptol.*, 2017(1):329–357, 2017.

[20] G. S. Murthy. *Optimal Loop Unrolling for GPGPU Programs*. PhD thesis, The Ohio State University, 2009.

[21] M. Naya-Plasencia, A. Röck, and W. Meier. Practical Analysis of Reduced-Round Keccak. In D. J. Bernstein and S. Chatterjee, editors, *Progress in Cryptology - INDOCRYPT 2011 - 12th International Conference on Cryptology in India, Chennai, India, December 11-14, 2011. Proceedings*, volume 7107 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2011.

[22] NIST. SHA-3 Competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html, 2007-2012.

[23] C. Nvidia. CUDA C Programming Guide. *Nvidia Corporation*, 120(18), 2011.

[24] K. Qiao, L. Song, M. Liu, and J. Guo. New Collision Attacks on Round-Reduced Keccak. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 216–243, 2017.

[25] G. Sevestre. Implementation of Keccak Hash Function in Tree Hashing Mode on Nvidia GPU. 2010.

[26] L. Song, G. Liao, and J. Guo. Non-full Sbox Linearization: Applications to Collision Attacks on Round-Reduced Keccak. In J. Katz and H. Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 428–451. Springer, 2017.

[27] The U.S. National Institute of Standards and Technology. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions . Federal Information Processing Standard, FIPS 202, 5th August 2015.
[28] V. Volkov. Better Performance at Lower Occupancy. In *Proceedings of the GPU technology conference, GTC*, volume 10. San Jose, CA, 2010.

A Linearizable Affine Subspaces

Table 7: The 80 2-dimensional linearizable affine subspaces of KECCAK S-box

{0, 1, 4, 5}	{2, 3, 6, 7}	{0, 1, 8, 9}	{4, 5, 8, 9}	{0, 2, 8, A}
{1, 2, 9, A}	{0, 3, 8, B}	{1, 3, 9, B}	{2, 3, A, B}	{6, 7, A, B}
{0, 1, C, D}	{4, 5, C, D}	{8, 9, C, D}	{4, 6, C, E}	{5, 6, D, E}
{4, 7, C, F}	{5, 7, D, F}	{2, 3, E, F}	{6, 7, E, F}	{A, B, E, F}
{0, 2, 10, 12}	{8, A, 10, 12}	{1, 3, 11, 13}	{9, B, 11, 13}	{0, 4, 10, 14}
{1, 5, 10, 14}	{2, 4, 12, 14}	{0, 4, 11, 15}	{1, 5, 11, 15}	{3, 5, 13, 15}
{10, 11, 14, 15}	{0, 6, 10, 16}	{2, 6, 12, 16}	{3, 7, 12, 16}	{4, 6, 14, 16}
{C, E, 14, 16}	{1, 7, 11, 17}	{2, 6, 13, 17}	{3, 7, 13, 17}	{5, 7, 15, 17}
{D, F, 15, 17}	{12, 13, 16, 17}	{10, 11, 18, 19}	{14, 15, 18, 19}	{0, 2, 18, 1A}
{8, A, 18, 1A}	{10, 12, 18, 1A}	{11, 12, 19, 1A}	{10, 13, 18, 1B}	{1, 3, 19, 1B}
{9, B, 19, 1B}	{11, 13, 19, 1B}	{12, 13, 1A, 1B}	{16, 17, 1A, 1B}	{8, C, 18, 1C}
{9, D, 18, 1C}	{A, C, 1A, 1C}	{8, C, 19, 1D}	{9, D, 19, 1D}	{B, D, 1B, 1D}
{10, 11, 1C, 1D}	{14, 15, 1C, 1D}	{18, 19, 1C, 1D}	{8, E, 18, 1E}	{A, E, 1A, 1E}
{B, F, 1A, 1E}	{4, 6, 1C, 1E}	{C, E, 1C, 1E}	{14, 16, 1C, 1E}	{15, 16, 1D, 1E}
{9, F, 19, 1F}	{A, E, 1B, 1F}	{B, F, 1B, 1F}	{14, 17, 1C, 1F}	{5, 7, 1D, 1F}
{D, F, 1D, 1F}	{15, 17, 1D, 1F}	{12, 13, 1E, 1F}	{16, 17, 1E, 1F}	{1A, 1B, 1E, 1F}

B GPU Implementation

B.1 Techniques for GPU implementation optimization

The techniques commonly used to optimize the CUDA program include memory optimizations, execution configuration optimizations, and instruction-level parallelism (ILP).

Memory optimizations. Usually registers have the shortest access latency compared with other memory, so keeping data in registers as much as possible improves the efficiency in general. However, dynamically indexed arrays cannot be stored in registers, so we define some variables for the 25 lanes by hand in order to have them stored in registers. Constant memory is a type of read-only memory. When it is necessary for a warp of threads read the same location of memory, constant memory is the best choice. So we store 24 round constants on it. When the threads in a warp read data which is physically adjacent to each other, the texture memory provides better performance than global memory, and it reduces memory traffic as well. So we can bind input data and some frequent accessed read-only data with texture memory.

Execution configuration. With resources like registers and shared memory limited in each graphic card, the number of threads running in each block will affect the performance since too many threads running in parallel will cause a shortage of registers and shared memory allocated to each thread, while too few parallel threads reduce the overall performance directly. According to our experiments, one block with 128 threads gives the best performance.

Instruction-level parallelism. From [28], hashcat, and ccminer, we see that forcing adjacent instructions independent gives better performance. Without prejudice to the functions of the program, we can adjust the order of instructions to improve the efficiency of the operations. In addition, loop unrolling [20] is also a good practice to obtain ILP.

B.2 Hardware specification of GPU

Table 8: The hardware specification sheet of GTX1070 and GTX970

	GTX1070	GTX970
Core Clock Rate	1645 MHz	1228 MHz
Multiprocessors	16	13
Regs Per Block	65536	65536
Total Global Memory	8105.06 MiB	4036.81 MiB
Bus Width	256 bits	256 bits
Memory Clock Rate	4004 MHz	3505 MHz
L2 Cache Size	48 KiB	48 KiB
Shared Memory Per Block	48 KiB	48 KiB
Total Constant Memory	64 KiB	64 KiB

C Details of Differential Trail Search

C.1 Analysis of the Starting Point of the Search

The following paragraphs describe how specific attributes of differential trails are settled down. We take the 5-round KECCAK collision situation as an example to explain why those kinds of trails are necessary.

Searching from light β_3's. Our initial goal is to find collisions for 5-round KECCAK. To facilitate a 5-round collision of KECCAK, we need to find 4-round differential trails satisfying the three requirements mentioned in Section 5.2. However, it is difficult to meet all of them simultaneously even though each of them can be fulfilled solely.

We explain as follows. Since we aim for practical attacks, $w_2 + w_3 + w_4^d$ must be small enough, say 55. That is to say, the last three rounds of the trail must be light and sparse. When we restrict a 3-round trail to be lightweight and extend it backwards for one round, we almost always unfortunately get a heavy state α_2 (usually $\#AS(\alpha_2) > 120$) whose weight may exceed the `TDF`. We take Keccak-224 as an example. The `TDF` of Keccak-224 is 191, which indicates $\#\mathrm{AS}(\alpha_2) < 92$ as the least weight for an S-box is 2. For a lightweight 3-round trail, it satisfies Requirement (1) occasionally. The greater d is, the fewer trails satisfy Requirement (1).

With these requirements in mind, we search for 4-round differential trail cores from light middle state differences β_3's. From light β_3's we search forwards and backwards, and check whether Requirement(1) and (2) are satisfied respectively; once these two requirements are satisfied, we compute the weight $w_2 + w_3 + w_4^d$ for brute force, hoping it is small enough for practical attacks.

α_3, α_4 in CP-kernel. The designers of Keccak show in [4] that it is not possible to construct 3-round low weight differential trails which stay in CP-kernel. However, 2-round differential trails in CP-kernel are possible, as studied in [9, 14, 21].

We restrict α_3 in CP-kernel. If $\rho^{-1} \circ \pi^{-1}(\beta_3)$ is outside the CP-kernel and sparse, say 8 active bits, the active bits of $\alpha_3 = L^{-1}(\beta_3)$ will increase due to the strong diffusion of θ^{-1} and the sparseness of β_3. When $\#AS(\alpha_3) > 11$, the complexity for searching backwards for one β_3 is greater than $2^{34.87}$ which is too time-consuming. On the other hand, we had better also confine α_4 to the CP-kernel. If not, the requirement $\alpha_{n_r}^d = 0$ may not be satisfied. As can be seen from the lightest 3-round trail for Keccak-f[1600] [14], after θ the none-zero difference bits are diffused among the state making a 224-bit collision impossible (a 160-bit collision is still possible). So our starting point is special β_3 which makes sure $\alpha_3 = L^{-1}(\beta_3)$ lies in CP-kernel, and for which there exists a compatible α_4 in CP-kernel. Fortunately, such kind of β_3's can be obtained with KeccakTools [5].

C.2 Detailed Algorithm for Searching Differential Trails

In this section, we describe more at length about the algorithm for finding differential trails. Firstly, light β_3's, namely, 2-round in CP-kernel trail cores are generated with KeccakTools [5], and then extended one round forwards and backwards respectively to find suitable 4-round trail cores. Note that all extensions should be traversed. Given a β_3, suppose there are C_1 possible one-round forward extensions and C_2 one-round backward extensions. These two numbers are mainly determined by the active S-boxes of β_3. If the number of active S-boxes is AS, then roughly $C_1 \geq 4^{AS}$ and $C_2 \geq 9^{AS}$ according to the `DDT`. In the search for 4-round trail cores, C_2 is the dominant time complexity, while for 5-round trail cores of Keccak$[1440, 160, 6, 160]$, we start from (β_3, β_4) generated by KeccakTools, and C_1 is almost as large as C_2.

- Generate β_3 such that $\alpha_3 = L^{-1}(\beta_3)$ lies in CP-kernel, and that there exists a compatible α_4 in CP-kernel, using TrailCoreInKernelAtC of KeccakTools [5] where the parameter *aMaxWeight* is set to be 60. We obtain more than 3000 such cores.
- For each β_3, if $C_1 \leq 2^{36}$, we traverse all possible α_4, compute β_4, and check whether the collision is possible for β_4. If yes, keep this β_3 and record this forward extension, otherwise, discard this β_3.
- For remaining β_3, if $C_2 \leq 2^{35}$, try all possible β_2 which are compatible with $\alpha_3 = L^{-1}(\beta_3)$, and compute $AS(\alpha_2)$ where $\alpha_2 = L^{-1}(\beta_3)$. If $AS(\alpha_2) \leq 110$, check whether this trail core $(\beta_2, \beta_3, \beta_4)$ is practical for the collision attack.

To find a 5-round trail core for Keccak$[1440, 160, 6, 160]$, we adapt the second step as follows.

- For each β_3, extend forwards for one round using KeccakFTrailExtension of KeccakTools [5] with weight up to 45. For each generated 2-round core (β_3, β_4), if $C_1 \leq 2^{36}$ for β_4, traverse all possible α_5 and compute β_5. Check whether there exists an α_6 such that $\alpha_6^d = 0$. If yes, record the three-round core $\beta_3, \beta_4, \beta_5$, otherwise, discard the β_3.

D Differential Trails

In this section, we give details of differential trails of Keccak mentioned in Section 5. Actually, we present trail cores. For example, a 4-round tail core $(\beta_2, \beta_3, \beta_4)$ consisting of three state differences represents a set of 4-round differential trails

$$\alpha_1 \xrightarrow{L} \beta_1 \xrightarrow{\chi} \alpha_2 \xrightarrow{L} \beta_2 \xrightarrow{\chi} \alpha_3 \xrightarrow{L} \beta_3 \xrightarrow{\chi} \alpha_4 \xrightarrow{L} \beta_4 \xrightarrow{\chi} \alpha_5$$

where α_5 is compatible with β_4 and $\beta_1 \to \alpha_2$ is of the least weight determined by β_2. In our collision attacks on 5-round (6-round) Keccak, 4-round (5-round) trail cores are needed.

The 1600-bit state is displayed as a 5×5 array, ordered from left to right, where '|' acts as the separator; each lane is denoted in hexadecimal using little-endian format; '0' is replaced with '-' for differential trails.

E Instances of Collisions

In this section, we give instances of collisions against Keccak$[1440, 160, 5, 160]$, Keccak$[640, 160, 5, 160]$, Keccak $[1440, 160, 6, 160]$, 5-round `SHAKE128`, 5-round `SHA3-224`, 5-round `SHA3-256` respectively. Note that we denote two colliding messages with M_1, M_2.

Table 9: Trail core No.1 $\sim$ 3 used in the collision attacks

Trail core No. 1, used in the collision attack of 5-round SHAKE128		
β_2	----------|----------24|----------2-|----------|---------- ----------|----------4|----------|----------|---------4------ ----------|------4---------|----------2-|----------|---------- ----------|----------|----------|----------|---------- ---------4------|------4---------|---------4------|---------4------|---------4------	2^{-19}
β_3	----------|----4----------|---1----------|----------|-----1---------- ----------|----4----------|----------|----------|---------- ----------8|----------|----------|----------|---------- ----------|----------|---1----------|----------|---------- ----------8|----------|----------|----------|-----1----------	2^{-16}
β_4	----------|----------|----------|----------|2-4--------- ----------|----------8|----------4-|---------8------|---------- ----------|----------|----------|--8----------|----------2----- ----------4-8|----------4----|-1----------|8----------|---------- ----4----------|----------|---------4-----|----------|---1----------	2^{-4}

Trail core No. 2, used in the collision attack of Keccak[1440, 160, 5, 160]		
β_2	----------|----------|----------|----------|---------- -------1-------|----------8----|----------1--|--------1----1--|---------- -------1-------|----------|-1----------|-1----------|-1---------- ----------|----------8----|----------|----------|-1---------- ----------|----------|----------1--|----------1--|----------	2^{-21}
β_3	----------|8----------|----------|----------|---------- ----------|----------|--------8-------|----------1|---------- ----------|----------|----------|----------1|---------- ----------|----------1|----------|----------|----------1 ----------|8----------|--------8-------|----------|----------	2^{-15}
β_4	----------|----------|----------|----------|---------- ----------|----------|----------|----2----------|--------1------- ----------1|------2---------|---------2------|----------|---------- ----------|----------|----------4--|----------|---------- ----------|--8----------|----------|----------|----------2	1

Trail core No. 3, used in the collision attack of 5-round SHA3-224 and 5-round SHA3-256. The probability is $2^{-36.70}$ considering multiple trails of the last two rounds.		
β_2	----------1|----------|----------4|----------|---------- ----------4|----------4|----------4|----------2----|---------- 2----------|----------|----------2-----|----------|---------- 2----------|----------|----------|----------2----|---------- ---------2----1|----------|----------2-----|----------|----------1	2^{-24}
β_3	----------1|----------|----------1|------4---------|---------- ----------|----------|----------1|----------|----------4---- ----------|----------1--|----------|----------|----------4---- ----------|----------|----------|----------|---------- ----------1|----------1--|----------|------4---------|----------	2^{-19}
β_4	----------|----------|----------|----------|---------- --------1------4|------4---------|----------|----------|---------- ------8---------|----------4-|----------|----------|----------4---- ----------|------1---------|----------4----|----------|-1------4------- 4----------|----------|-2----------|----------|-----1-------4--	1

Table 10: Trail core No. 4, used in the collision attack of KECCAK[640,160,5,160]. β_4 has two choices.

β_2	--------|-----8--|-----A--|-----8--|-----8-- --------|--------|--------|--------|-------- 2-------|2-------|--------|2-------|-------- 2-------|--------|-----8--|-----8--|-1------ --------|--------|-----2--|--------|-1------	2^{-20}	β_4^1	--------|--------|--------|--------|-------- 1------4|-4------|--------|--------|-------- --------|------4-|-2------|--------|-2------ -------2|-----8--|--------|--------|4------- 4-------|--------|--------|--------|--------	1
β_3	--------|--------|--------|-------1|------4- ------8-|--------|-------1|--------|------4- --------|--------|--------|-------1|-------- --------|--------|--------|--------|-------- ------8-|--------|--------|------4-|--------	2^{-16}	β_4^2	--------|--------|--------|--------|-------- 1-------|-4------|--------|--------|-------- --------|------4-|-2------|--------|-2------ -------2|-----8--|--------|--------|4------- 4-------|2-------|--------|--------|--------	1

Table 11: Trail core No. 5, used in the collision attack of KECCAK[1440, 160, 6, 160]. The total probability is $2^{-72.81}$ considering multiple trails of last two rounds. The probability of last three rounds is $2^{-47.81}$.

β_2	---------------|-----8---------|-----8------4---|---------------|-----------4--- ----------2---8|----------2----|--------------8|----------2---8|----------2---- ---------------|-----8---------|---------------|---------------|-----8------4--- ----------2----|---------------|-----8---------|----------2---8|--------------- ---------------|---------------|---------------|---------------|---------------	2^{-25}
β_3	---------------|---------------|---------------|--------1------|--------------- ---------------|---------------|---------------|--------1------|--------------- ---------------|---------------|---------------|---------------|--------------- -----2----------|--2------------|--2------------|-4-------------|--------------- -----2----------|-4-------------|--2------------|-4-------------|---------------	2^{-18}
β_4	---------------|---------------|---------------|-----------8---|--------------- --1------------|---------------|---------------|---------------|---4----------- ---------------|---------------|---------------|---------------|-8------------- ---------------|---------------|---------------|-------------1-|---4----------- ---------------|-----------8---|---------------|-----------4---|---------------	2^{-16}
β_5	-8-------------|-------1--------|48-1---1--------|---------2-2----|-------12--4---C -----8---1------|------48-1---34-|4---------------|-------2--------|---4-------12--4 --2--------1----|-9-------24--8--|----------2-----|34-------48-1---|-----------2--- ----24--8--18---|--------81------|4---------------|-------48-1---12|--1---------4-- ---8-------24--8|---8-8----------|-24--8-418------|------1---------|--4--------2----	2^{-16}

Table 12: Trail core No. 6, used in the last 160-bit collision attack of KECCAK[1440, 160, 6, 160]. The total probability is $2^{-65.83}$ considering multiple trails of last two rounds. The probability of last three rounds is $2^{-48.83}$.

β_2	-2-------------|----8----------|---------------|---------2-----|---------2----- -2-------------|---------------|---------------|---------------|--------------- ---------------|----8----------|-----------4---|---------------|--------------- ---------------|---------------|-----------4---|---------------|---------2----- ---------------|---------------|---------------|---------------|---------------	2^{-17}
β_3	-2-------------|---------------|-2-------------|---------------|--------------- ---------------|---------------|---------------|---------------|--------------- ---1-----------|---------------|---------------|-------2-------|--------------- ---1-----------|--------2------|-2-------------|-------2-------|--------------- ---------------|---------------|---------------|---------------|---------------	2^{-14}
β_4	---------------|---------------|-----------1--|---4-----------|--------------- ---------------|---------------|---8-----------|---------------|--------------- ---------------|---------------|--4------------|---------------|--------------- ---------------|---------------|---------------|----1--------1--|--------------- --8-------------|---------------|---------------|---------2------|---------------	2^{-16}
β_5	---------------|-------1-2------|--1----841----2-|--------2---29--|-4----8---4---41 ----------14C1--|----2---1---1841|---------------|-------2-4------|--29-4----8---4- --2-4-----------|5--8---1----8---|-----------29-2-|841----2---1---1|------------2-- --1----8---C2-8-|---------------|4-8-----------1|18---2---18---A4|---1C81----2---- --52-8---1----C-|----A4-8--------|----8---C2-8---1|---------------|--4-8-----------	2^{-20}

Table 13: Collision for the contest instance KECCAK[1440, 160, 5, 160]

M_1	FE04ABD5B6CB82B0|61CC2361C8649AFA|8192015AE5A4CD17|18CA3F02D74C2A1E|2FD4F65B9B8954B7 C3E83AFE2554E96C|D40EA8CEA0A5D897|13F77204F374C0FA|C50800E587690A4A|A818DB7455FC0EEC D165BE70A75C79BD|A5E719D73D740232|3129EFB27EE7D766|CBC2BA892C29908E|67723AF6944A3147 5BC2E4ECA7B4EA3A|1BCF8762A0E3E233|18A28C2474950C18|DB52DE89434D26B2|A642C6B06F111CAC 7866F86D04A61505|B2B98F0702FE80C1|00000000F8514DD0|0000000000000000|0000000000000000
M_2	E8AC1434F0FEBE8E|85FDE653956EF372|7B1CEE0DB775BFC0|F80D568E8D0A97CC|9E3E49ADCFFC18BF 8DC76DBCC537ED14|AD6CCD45B0990F18|A9CF7A6399115588|970C9BC4ED512782|08A0B2CA4DDD83BF 9F13F101BF5BE7EC|7B533F4848528805|21F6E6F54C30B798|0BDA3E2038C5FAA6|84808A532E7E1407 8D2DE036C7989316|2778E5A9A1717231|4CC28A17BA036E68|42DA44B9D77DE6F6|56C2556B467D1BBC AD99AD3F868AD505|2F4EB3D410EC316E|0000000092856EAD|0000000000000000|0000000000000000
digest	0887B202077B082F|E6D2B9B1BA9B51AC| 7BF7B1AF

Table 14: Collision for the contest instance KECCAK[640, 160, 5, 160].

M_1	297DB73F|CE5FB46D|63EFD5AB|AB75DBB2|020119E7 06927773|A645A6A4|68E6E3F8|15282462|633AAB83 96C7A5FB|5E4CBEB5|92614C96|DD9647DA|D4B0094F 4C68376F|D3B63751|6286AB56|DE577A52|9003EA0F 00000000|00000000|00000000|00000000|00000000	M_2	5B150BCB|C0F3F2FC|5907B5A5|22736DC3|914CF0C5 87477D63|A675A649|8BBEA96F|52EB8AE3|19402D41 D9FB4CC3|669FD630|D8C9FC71|57558554|0662F64A 64B4B5C5|7F12BF56|2BEADBF0|F6207B10|F2FD9787 00000000|00000000|00000000|00000000|00000000
digest	F90B5ABA|7430682D|85668C62|66E1B0AD|B052AC35		

Table 15: Collision for 5-round SHAKE128

M_1	0A3E44EBE62104A0|1E8617C352E80FBC|B69A38114369962F|1237F5EEA8045DAB|D4144AC64E22044C 1240A93D79FCCB2E|C8C63A830CBACFFC|B36B34C0E1719824|F94803ECC5586680|ACF133FE29839CAD CA5F88F260DEFAA7|972FE7E882A4AB03|D11344BE12431A54|814488EBAE68F93B|56D10CF0251FAED3 77A665FCC5F52D9F|D50EF69FAB128ACC|87F3F1816E740894|770D4D55489234B0|737134B1243F3A3D FE0E2AE7F23D8E40|0000000000000000|0000000000000000|0000000000000000|0000000000000000
M_2	0CDDD5D25A8BD7CA|F71D259EE445A4D8|EB84F177D51C9D45|0A70C1FC50024C24|7108096E3F024F63 3B8D1EEBC5E9150E|EADCC9FB19824E75|B8A97CB74697BAAD|5988E2CD64063AD1|FB55123185E2E4A4 E74FF74033CA1486|915F016B41BDAF6B|145441AAC9EFA342|D9A609CF15E6C626|5609C4F58F5DEE0A AB4E178C43BA8687|3774B01D78F2ABE4|AA35E3D371664594|A26EAD50F73069A7|DE4E25F8A0F8E928 FE431BE34F8371D8|0000000000000000|0000000000000000|0000000000000000|0000000000000000
digest	9D2E953AD7C6A939|326F59A68A6016EF|A71EAFEE371700D7|3C463D5D098D9B76

Table 16: Collision for 5-round SHA3-224

M_1	BA651BB077352C22|020878C86F47A777|A68806B74E37F538|FC14E1158751958F|38364A327DE40704 F24920D7259B9A3B|98E0602DA4CF65CB|A0CD3329C8DC6E96|08050DBA7BFFB106|3B74AE3D29AA1F74 C72E0606BA4A9E5C|E7E83E0C5A8D05EE|78E2095685EEABBC|114AD7006B5B4905|2FFD9CF449950256 EAB168AEC88D6917|294C016FD4B8C424|EA95CDFF2F0B0939|0000000000000000|0000000000000000 0000000000000000|0000000000000000|0000000000000000|0000000000000000|0000000000000000
M_2	DF3E25D454129D90|743A8BB224F5D398|803A36FD69A1590C|26E9CC61513C3B2C|52777F2E82CBABF9 0ED8318F4F31BE85|AE55CC288D95514A|E57052E3B305CCDE|3CC6422BDC7AE40D|F4D560227B5E03BB A88F4E0FF407024C|4102F764F58A1C70|68D868FC2E2EE351|554ED55BF9A456A7|BCD9BE774A516ED6 C1C74E8F2CF70106|DAC93A2C7E587301|E4294D74AD68A5F6|0000000000000000|0000000000000000 0000000000000000|0000000000000000|0000000000000000|0000000000000000|0000000000000000
digest	79D8E749B6633429|8006840F0FA7F0B1|4FF9CCCF110BD0D4|FB35AD94

Table 17: Collision for 5-round SHA3-256

M_1	FECA67BD2D3F021A|BD10A64A4C2B774F|F8EF6FF82DD21FC7|6F4BA4D964A78764|0F4FD1C92A24BC6E FB4B8C0A11C64088|EDA7B9EBC05F50A8|0A71DD08E7F1EB5B|5342D2AE78A8BFB5|6591A9B0CC2E7CE9 52A3DD827F4EF6DC|9D89B18362B80DE4|FEA719A1875BFFF7|49A2B95AD7B7D147|B23784B72EB9260A 187AEFD07295FD59|EE806366EF9D09FF|0000000000000000|0000000000000000|0000000000000000 0000000000000000|0000000000000000|0000000000000000|0000000000000000|0000000000000000
M_2	16F97050842C2D17|A731EE935A43480A|6D8E356BDBD7CBE9|D62C0B356FFA158A|4FAD968080C7F8C8 7C83B8E1C61BC5AB|7E3FCA22B5E29305|5888D4DBE848C840|236DE21CCEF77B8A|69D59EF589070E60 E87FCD2BF2C6CCE1|B1E28B821FD93ABC|AD5D6FB1860CB45C|AB8FC7D1015975D5|24C6B737EE96CC23 D3BFB5957965A447|EE31D3F5269F254F|0000000000000000|0000000000000000|0000000000000000 0000000000000000|0000000000000000|0000000000000000|0000000000000000|0000000000000000
digest	65017C2E8B6040B4|344FF8BB933B4BD6|C6A3F13368BE2003|AB427B4B33435ACB

Table 18: Collision for the contest instance Keccak$[1440, 160, 6, 160]$

M_1	DA27ABE5B7EC359D|328A2AB4CD0E256A|00DBDEECA184390E|3843F66481C745F4|DDF83BEF39D4F594 46BA2A960272C97A|8CC8CE3E13185558|2D7C6CC662546532|4D8DCDC25DC7F4B8|574252F43F85BF94 BDCFA2D6B04CBDEE|208D7A02168A7596|AFE7C652F0A68792|467C04748D85916F|F1BFEAF63C4B97C3 C2B0AAEA35887CD4|72A3D23F9D84434D|97A5D9A090590B61|BBE1EC62DBD4327E|64284BCB9BE462C5 8843CBC8B55E106A|DD3DD96A1AC48100|00000000E9151D67|0000000000000000|0000000000000000
M_2	5A0C640730278910|32C1A7D724790C0B|8BCE75C46404A83A|7FCE23E92ECE7E31|1BEE08F9F932C785 3969BA55EB6B17F9|E82948B06C21C6A8|AF42ACEF22202C1F|A9C1BD90BF96FB60|0F98E27C36B57BDA A02B26453D88C70F|5EC5F74DC919C7E6|31391D7A23A3C8DD|C0BECDAD0AC7F275|14FA28F6B2C9D390 69F67EEAEF258217|159B7FEDCED37178|DA89C2B0291CCA7D|7BDDE79F989414AE|3088CBE192E15B4B 138617865C48CEA9|2A917CE5E3AD1374|0000000098425E60|0000000000000000|0000000000000000
digest	602133DD97109089|611B5125914B0F05|532B96C0

Table 19: Collision on the last 160 bits of KECCAK$[1440, 160, 6, 160^{\dashv}]$

M_1	6FDB2A20F8992431|2087B01388592A01|D114527BE00F3C5A|2CCAE47DF6025BBB|276C0E5A2B64FD12
	F7C10ACF1A83B66B|9558BC260011465E|06C34BD57E6F00CA|81BF6D44927DA86A|F1A1EE2D0F329641
	78367D0B0C0A12BF|A932C8A9906D8577|78116644C91F9867|0510A128067B884D|63F5D9D184C7D6E2
	02BEA4FBB2F9DA4A|CED47DD4D59E41F8|DDA6E9AC3E28B45A|FD12804E77348FB8|557A955C5CD88D02
	96E21F07B25C1F2E|104767C63F7C2AB4|00000000C60B57C1|0000000000000000|0000000000000000
M_2	033C171CD7A65128|45F8C2FB4052939B|55A9BD5D6B4533A6|69DF5701873D16E7|ADC98EEA28325D2C
	8EF732C6DAED91BE|1A7AF5636039685D|84915390F1F832B6|92677C0DFB5DB15B|998A4A254F9C88E0
	F810289754AA0C71|28ACE4F7585781D4|9400E644963033D7|3432AEC4E5231119|23E1FD9183D58A0C
	CFAA789629A11457|E7681880B18D4DA6|6A2C40394F8E1172|E9562446427C4AAC|D97ABBDB097B07E2
	73D4472B79731977|37EE2EC25F6706B6|00000000AF85B4FB|0000000000000000|0000000000000000
$160^{\dashv}$	|8FA6D263 |1BF0161E867C399C|DE5348CD351076D5