

# Practical Collision Attacks against Round-Reduced SHA-3\*

Jian Guo<sup>1</sup>, Guohong Liao<sup>2</sup>, Guozhen Liu<sup>3,1</sup>,  
Meicheng Liu<sup>4</sup>, Kexin Qiao<sup>5</sup>, and Ling Song<sup>1,4</sup>(✉)

<sup>1</sup> Division of Mathematical Sciences, School of Physical and Mathematical Sciences,  
Nanyang Technological University, Singapore

guojian@ntu.edu.sg

<sup>2</sup> South China Normal University, China

liaogh.cs@gmail.com

<sup>3</sup> School of Cyber Security, Shanghai Jiao Tong University, China

liuguozhen@sjtu.edu.cn

<sup>4</sup> Institute of Information Engineering, Chinese Academy of Sciences, China

{liumeicheng, songling}@iie.ac.cn

<sup>5</sup> Beijing Unionpay Card Technology Co., Ltd., China

qiaokexin@bctest.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] 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.

---

\* 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.

# 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, denoted by namely, SHA3- $d$  (corresponding to  $d$ ) with  $d \in \{224, 256, 384, 512\}$  being the digest size in bits, and ~~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 ~~preimage~~ ~~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, ~~it has received~~ 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 ~~224~~KECCAK[448]/~~256~~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 ~~384~~KECCAK[768]/~~512~~KECCAK[1024], and theoretical collision attacks (with complexities beyond the reach of practical resources) on 5/4-round ~~256~~KECCAK[512]/~~384~~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 ~~224~~KECCAK[448]/~~256~~KECCAK[512]/ ~~384~~KECCAK[768]/~~512~~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/~~256~~, 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 ~~only~~ 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 state 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

129 suitable differential trails of ~~4 and 5~~ 3 and 4 rounds. Our GPU implementations  
 130 confirmed the correctness of this idea, and successfully found desirable differential  
 131 trails for our collisions attacks.

132 Results obtained in this paper are listed in Table 1, compared with the best  
 133 previous practical collision attacks and related theoretical attacks. In brief, we  
 134 obtain actual collisions on three 5-round instances of SHA-3, *i.e.*, SHAKE128, and  
 135 SHA3-224/~~256~~, SHA3-256, and three instances of KECCAK contest. The number  
 136 of practically attacked rounds of KECCAK instances now is increased to 6. For  
 137 the instance of contest KECCAK[1440, 160, 6, 160], ~~a special collision attack an~~  
 138 inner collision [3] on the last 160-bit of the output state is mounted to construct  
 139 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
<del>-512</del> KECCAK[1024]	3	Practical <del>†</del>	[11]
<del>-384</del> KECCAK[768]	3	Practical	
<del>-384</del> KECCAK[768]	4	$2^{147}$	
<del>-256</del> KECCAK[512]	5	$2^{115}$	
<del>-256</del> KECCAK[512]	4	Practical	[10, 12]
<del>-224</del> KECCAK[448]	4	Practical	
KECCAK[1440, 160, 160]	4	Practical	
SHAKE128	5	Practical <sup>†</sup>	
SHA3-224	5	Practical	Sect. 6
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] <sup>‡</sup>	6	Practical	

<sup>†</sup> For exact complexities, refer to Table 6.

<sup>‡</sup>  $160^{-1}$  corresponds to the last 160-bit collision rather than the general digest collision.

140

141 **Organization.** The rest of the paper is organized as follows. In Section 2, a  
 142 brief introduction to the SHA-3 hash function is given. Section 3 first presents  
 143 the notations used in this paper and subsequently provides a synopsis of the  
 144 collision attacks against SHA-3 and KECCAK instances. The properties of S-  
 145 box linearization and non-full linearization are illustrated in Section 4, followed  
 146 by detailed explanations over how the 2-round and 3-round connectors are  
 147 constructed. Section 5 outlines the search for differential trails as well as the  
 148 highly efficient GPU implementations of KECCAK. In Section 6, the experimental

149 results of collision attacks are given. We conclude the paper in Section 7. Details  
 150 of the differential trails and actual collisions are postponed to the Appendix.

## 151 2 Description of SHA-3

### 152 2.1 The Sponge Function

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

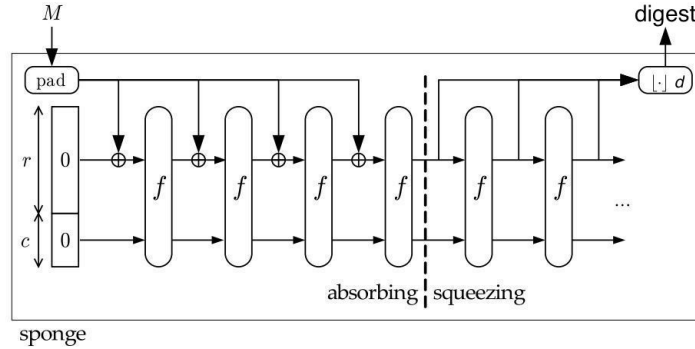


Figure 1: Sponge Construction [3]

### 166 2.2 The Keccak Hash Function

167 The KECCAK hash function follows the sponge construction. The underlying  
 168 permutation of KECCAK is chosen from a set of seven KECCAK- $f$  permutations,  
 169 denoted by KECCAK- $f[b]$ , where  $b \in \{25, 50, 100, 200, 400, 800, 1600\}$  is the width  
 170 of the permutation in bits. The default KECCAK employs KECCAK- $f[1600]$ . The  
 171 1600-bit state can be viewed as a 3-dimensional  $5 \times 5 \times 64$  array of bits, denoted  
 172 as  $A[5][5][64]$ . Let  $0 \leq i, j < 5$ , and  $0 \leq k < 64$ ,  $A[i][j][k]$  represents one bit of  
 173 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 ~~24 rounds~~  $12 + 2l$  rounds ( $l = \log_2 \frac{b}{25}$ ), each of which consists of five mappings  $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$ .

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

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

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

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

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

Here, ~~'+' denotes exclusive-or and '.' stands for logic AND~~ the additions and multiplications between the terms are in GF(2). As  $\iota$  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  ~~$\text{KECCAK}[c]$~~   $\triangleq \text{KECCAK}[r = 1600 - c, c]$ , where the ~~digest length is  $d \in \{224, 256, 384, 512\}$  and the capacity  $c$  is twice as large as  $c \in \{448, 512, 768, 1024\}$  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 ~~preimage-pre-image~~ contest and  $(d = 160, c = 160)$  for collision contest. In this paper, we follow the designers' notation  $\text{KECCAK}[r, c, n_r, d]$  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  $\bar{M}$  is  $M \parallel 10^*1$ .

The SHA-3 standard adopts the four KECCAK instances  ~~$d, d \in \{224, 256, 384, 512\}$~~  and names them ~~SHA3- $d$ . The only difference is the padding rule. In 224, SHA3- $d$  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. Another padding rule is applied to For SHAKE where, a four-bit suffix '1111' is appended 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.

213

$c$	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
$d^\perp$	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, <i>e.g.</i> , $\chi_i$ denotes the $\chi$ layer at the $i$ -th round for $i = 0, 1, 2, \dots$ .
$L$	Composition of $\theta, \rho, \pi$ and its inverse denoted by $L^{-1}$
$RC_i$	Round constant for the $i$ -th round, $i = 0, 1, 2, \dots$
$R^i(\cdot)$	KECCAK permutation reduced to the first $i$ rounds
$S(\cdot)$	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,
DDT	Differential distribution table, and $\text{DDT}(\delta_{in}, \delta_{out}) =  \{x : S(x) + S(x + \delta_{in}) = \delta_{out}\} $ , where $ \cdot $ denotes the size of a set.
$\alpha_i$	Input difference of the $i$ -th round function, $i = 0, 1, 2, \dots$
$\beta_i$	Input difference of $\chi$ in the $i$ -th round, $i = 0, 1, 2, \dots$
$w_i$	Weight of the $i$ -th round, $w_i, i = 0, 1, 2, \dots$
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.
$M_1    M_2$	Concatenation of strings $M_1$ and $M_2$

215

### 3.2 Overview of the Attack

216

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 Figure 2. Let  $\Delta S_I$  and  $\Delta S_O$  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

223

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

224

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

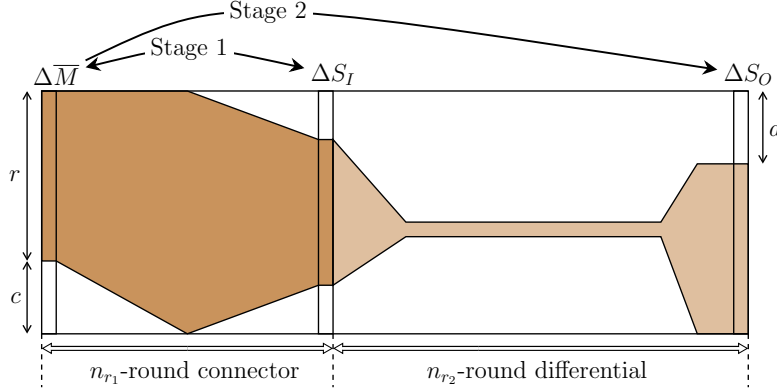
225

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:

229

- *Stage 1 — Connecting stage:* Construct an  $n_{r_1}$ -round connector and get a subspace of messages bypassing the first  $n_{r_1}$  rounds.

230



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

- *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  $\Delta S_I$ . 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.



## 259 4 S-box Linearization and Connectors

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

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

### 273 4.1 S-box Linearization

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

280 **Definition 1 (Linearizable affine subspace).** *Linearizable affine subspaces*  
 281 *are affine input subspaces on which S-box substitution is equivalent to a linear*  
 282 *transformation. If  $V$  is a linearizable affine subspace of an S-box operation  $S(\cdot)$ ,*  
 283 *then  $\forall x \in V, S(x) = A \cdot x + b$ , where  $A$  is a  $5 \times 5$  matrix and  $b$  is a 5-bit constant*  
 284 *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:

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

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

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

290 **Observation 1** *Out of the entire 5-dimensional input space,*  
 291 *a. there are totally 80 2-dimensional linearizable affine subspaces, as listed in*  
 292 *Table 7 in Appendix A.*  
 293 *b. there does not exist any linearizable affine subspace with dimension 3 or more.*

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

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

299 **Observation 2** *Given a 5-bit input difference  $\delta_{in}$  and a 5-bit output difference*  
 300  *$\delta_{out}$  such that  $\text{DDT}(\delta_{in}, \delta_{out}) \neq 0$ , i.e., the solution set  $V = \{x : \mathbf{S}(x) + \mathbf{S}(x + \delta_{in}) =$*   
 301  *$\delta_{out}\}$  is not empty, we have*  
 302 *a. if  $\text{DDT}(\delta_{in}, \delta_{out}) = 2$  or 4, then  $V$  is a linearizable affine subspace.*  
 303 *b. if  $\text{DDT}(\delta_{in}, \delta_{out}) = 8$ , then there are six 2-dimensional subsets  $V_i \subset V, i =$*   
 304  *$0, 1, \dots, 5$  such that  $V_i (i = 0, 1, \dots, 5)$  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  $\text{DDT}(01, 01)$ , i.e., with both input and output differences being 01, is  $\{10, 11, 14, 15, 18, 19, 1C, 1D\}$  and the six 2-dimensional linearizable affine subspaces from it are

$$\begin{aligned} &\{10, 11, 14, 15\}, \{10, 11, 1C, 1D\}, \{14, 15, 18, 19\}, \\ &\{10, 11, 18, 19\}, \{18, 19, 1C, 1D\}, \{14, 15, 1C, 1D\}. \end{aligned}$$

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

## 309 4.2 Non-full S-box Linearization

310 Suppose we are to construct a connector of two rounds. Given the input difference  
 311  $\Delta S_I$  of the differential trail, i.e., the output difference of the connector [α<sub>2</sub>](#), it  
 312 may be the case that some output bits of the S-boxes in the first round of the  
 313 connector do not influence  $\Delta S_I$ . The S-boxes with such output bits do not need  
 314 full linearizations. That is, it is enough to restrict the input to certain subsets  
 315 such that only the output bits of the first round affecting  $\Delta S_I$  are to be expressed  
 316 with a linear relation with input bits. Such linearization is called *non-full S-box*  
 317 *linearization* and may contribute to a lower consumption of degree of freedom,  
 318 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  $\chi_0$  (the  $\chi$  mapping of the first round) be  $x$  and  $y$  respectively. Let  $u = (u_0, u_1, \dots, u_{b-1})$  be a flag vector where  $u_i = 1$  ( $0 \leq i < b$ ) if  $y_i$  influences  $\Delta S_I$ , otherwise  $u_i = 0$ . Let  $U = (U_0, U_1, \dots, U_{b/5-1})$  where  $U_i = u_{5i}u_{5i+1}u_{5i+2}u_{5i+3}u_{5i+4}$ ,  $0 \leq i < b/5$ . According to the definition,  $0 \leq U_i \leq 32$ . For the  $i$ -th S-box of  $\chi_0$ , if  $U_i$  is not zero (00000), aka. some bits of the corresponding S-box will influence  $\Delta S_I$ , this S-box should be fully or partially linearized. The value of  $u$  is determined by  $\beta_1$  and  $\alpha_2$  ( $\Delta S_I$ ), 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  $U_i \neq 31$   $U_i$  is not 11111,

- a. if  $U_i = 0$  ( $U_i = 00000$ ), it does not require any linearization;
- b. if  $U_i \in \{01, 02, 04, 08, 10, 03, 06, 0C, 11, 18\}$  (in hex)  $U_i \in \{00001, 00010, 00100, 01000, 10000, 00011, 00010, 01\}$ , 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  $\chi$ . Suppose the 5-bit input of the S-box is  $x_0x_1x_2x_3x_4$  and the 5-bit output  $y_0y_1y_2y_3y_4$ . 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  $U_i = 01$  ( $U_i = 00001$ ) 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  $U_i = 0F$  ( $U_i = 01111$ ), 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  $U_i = 4F$  ( $U_i = 11111$ ), 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.

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

Take  $DDT(01, 01) = 8$  as an example (see Table 6 of [24]). The solution set is  
 $V = \{10, 11, 14, 15, 18, 19, 1C, 1D\}$ . We re-write these solutions in [the following](#)  
 5-bit ~~strings where the right most bit is the LSB as follows~~[strings where the bits](#)  
[are written as  \$x\_4x\_3x\_2x\_1x\_0\$ .](#)

10 : 10000, 11 : 10001, 14 : 10100, 15 : 10101,  
 18 : 11000, 19 : 11001, 1C : 11100, 1D : 11101.

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

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

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

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

### 377 4.3 2-Round Connector

378 The core idea of our 2-round connector is to convert the problem to that  
 379 of solving a system of linear equations. Note the first two rounds of KEC-  
 380 CAK permutation can be expressed as  ~~$\chi_1 \oplus L_1 \oplus \chi_0 \oplus L_0$~~  [\(omitting the  \$\iota\$ \), as depicted in Fig. 3.](#) Using the techniques discussed above,  $\chi_0$   
 381 which has fixed input and output differences can be (partially) linearized,  
 382 *i.e.*, the first three operations  ~~$L_1 \oplus \chi_0 \oplus L_0$~~  [become \(partially\) linear.](#)  
 383 For convenience, the differential transition of the  $i$ -th round is denoted  
 384 by  ~~$\alpha_{i-1} \xrightarrow{L_{i-1}} \beta_{i-1} \xrightarrow{\chi_{i-1}} \alpha_i$~~  [\(omitting the  \$\iota\$ \), as depicted in Fig. 3.](#) where  $i = 1, 2, \dots$ . We will give  
 385 details of the method how input and output differences of  $\chi_0$ , *i.e.*,  $\beta_0$  and  $\alpha_1$ , are  
 386 selected later. Now, we show how the  $\chi_1$  can be inverted by adding more linear  
 387 equations of constraints.

In the setting of a 2-round connector, the output difference of  $\chi_1$ , *i.e.*,  $\alpha_2$  is  
 given as  $\Delta S_I$  — the input difference of the differential trail. It is not necessary  
 that all S-boxes of  $\chi_1$  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

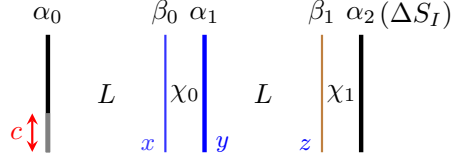


Figure 3: The first two rounds of KECCAK permutation

difference with a non-zero number of solutions which is called *compatible* input difference. Formally, given the output difference  $\delta_{out}$ , for the KECCAK S-box, its compatible input differences are from the set  $\{\delta_{in} : \text{DDT}(\delta_{in}, \delta_{out}) \neq 0\}$ . As noted previously in [4, 8, 10], for any pair of  $(\delta_{in}, \delta_{out})$ , the solution set  $V = \{x : S(x) + S(x + \delta_{in}) = \delta_{out}\}$  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  $\text{DDT}(03, 02)$  is the 2-dimensional affine subspace  $\{14, 17, 1C, 1F\}$  which can be formulated by the following three linear equations as the constraints:

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

389 It is important to note, under the  $i$  linear equations or set  $V$ ,  $\delta_{in}$  propagates to  
 390  $\delta_{out}$  deterministically. Hence, each active S-box in  $\chi_1$  is inverted by a choice of  
 391 compatible input difference together with the corresponding  $i$  linear equations  
 392 on the input values. Once the input difference and linear equations for all active  
 393 S-boxes of  $\chi_1$  are enforced and fulfilled together with the linearization of the  
 394 first round, solutions of the 2-round connector are found. Note that a compatible  
 395 input difference of  $\chi_1$  is a choice of  $\beta_1$ , and  $\alpha_1$  can be uniquely determined by  
 396 the relation  $\alpha_1 = L^{-1}(\beta_1)$ . In the remaining part of this subsection, more details  
 397 on the implementation of this idea are given.

398 **Generation of linear equations.** As depicted in Figure 4Fig. 3, the variables  
 399 of the equation system are the bit values before the first  $\chi$  $\chi_0$  denoted by  
 400 vector  $x$ . Additionally,  $y$  and  $z$  are bit vectors of intermediate values for further  
 401 interpretation where  $y$  represents the output after the first  $\chi$  $\chi_0$  and  $z$  the bits  
 402 before the second  $\chi$  $\chi_1$ . The main task is to derive all constraints on differences  
 403 and affine subspaces into that on the variables  $x$ . Suppose  $\beta_1$  and  $\beta_0$  (details will  
 404 be given in Section 4.6) are fixed, and  $\Delta S_I$  (*aka.*  $\alpha_2$ ) is given, we show how the  
 405 system of equations could be set up.

406 With the input difference  $\beta_1$  and output difference  $\Delta S_I$  $\alpha_2$  of  $\chi_1$ , all the  
 407 linear equations on the input affine subspaces of the active S-boxes in the second  
 408 round can be derived and expressed as

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

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

412 A similar procedure can be done for input affine subspaces of the first round  
 413 given  $\beta_0$  and  $\alpha_1$ , and the corresponding linear equations on  $x$  is denoted by

$$414 \quad A_1 \cdot x = t_{A_1}. \quad (2)$$

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

$$417 \quad A_2 \cdot x = t_{A_2}. \quad (3)$$

418 where  $A_2$  is a submatrix of  $L^{-1}$  and  $t_{A_2}$  is the preset value. Apart from Eq.  
 419 (2) and (3), some extra equations are imposed to linearize  $\chi_0$ . To derive these  
 420 equations, we need to specify the flag vector  $u$  identifying the output bits of  $\chi_0$   
 421 which affect  $\Delta S_I$  and should be linearized. In fact, the value of  $u$  is determined  
 422 by Eq. (1) which can be re-expressed as

$$423 \quad B \cdot L \cdot (y + RC_0) = t_B, \quad (4)$$

424 since  $z = L \cdot (y + RC_0)$ . Then  $u_i = 1$  if  $y_i$  appears in Eq. (4) with a non-zero  
 425 ~~coefficients~~coefficient, otherwise  $u_i = 0$ . Recall that  $U = (U_0, U_1, \dots, U_{b/5-1})$   
 426 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,  
 427 S-boxes with DDT entry being 2 or 4 are already linearized due to the fact that  
 428 the input is confined to the solution set  $V = \{x : S(x) + S(x + \delta_{in}) = \delta_{out}\}$  by  
 429 linear equations in (2). So we only need to handle non-active S-boxes and active  
 430 S-boxes with DDT entry 8.

431 *Full S-box linearization.*  ~~$U_i = 31$~~   $U_i = 11111$  indicates that fully linearizing the  
 432  $i$ -th S-box is required.

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

438 *Non-full S-box linearization.*  ~~$U_i \leftarrow 31$~~   $U_i \neq 11111$  indicates that only part of the  
 439 output bits of the  $i$ -th S-box needs to be linearized.

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

446 We denote these extra linear equations for linearizing  $\chi_0$  by

$$447 \quad A_3 \cdot x = t_{A_3}. \quad (5)$$

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

$$449 \quad y = L_{\chi_0} \cdot x + t_{L_{\chi_0}}, \quad (6)$$

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

$$451 \quad B \cdot L \cdot L_{\chi_0} \cdot x + B \cdot L \cdot (t_{L_{\chi_0}} + RC_0) = t_B. \quad (7)$$

452 We merge the four systems of linear equations on  $x$  together, *i.e.*, Eq. (2),  
453 (3), (5) and (7) and denote the resulted system of linear equations by  $E_M$ :

$$454 \quad A \cdot x = t_A. \quad (8)$$

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

456 **Algorithm of the 2-round connector.** Since Eq. (3) remains static, and  
457 given differences  $(\beta_0, \alpha_1, \beta_1, \alpha_2)$ , Eq. (2) and (1) are determined, the core part is  
458 the generation of Eq. (5), *i.e.*, linearizing  $\chi_0$ , which is sketched in the procedure  
459 *basicLinearization* (see Alg. 2). The procedure *mainLinearization* (see Alg. 1)  
460 invokes *basicLinearization* and generates the final system of linear equations  $E_M$   
461 whose solutions are exactly the solutions of the 2-round connector.

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

468 In *basicLinearization*, a procedure named *preProcess* is called on Line 1 which  
469 is described more at length in Alg. 3. *preProcess* identifies all possible S-boxes of  
470  $\chi_0$  whose output bits marked by  $U$  are already linear under the initial  $E_M$ , so  
471 the number of equations in Eq. (5) will be minimized. In fact, the conditions of  
472 Line 11 and 16 would never hold if  $E_M$  does not contain Eq. (3), and they hold  
473 more likely for large Eq. (3), *i.e.*, large  $c + p$ . Therefore, *preProcess* is crucial to  
474 the success of solving KECCAK instances with a relatively large capacity, such as  
475 SHA3-256.

476 Note that Alg. 1 and 2 do not succeed all the time. To overcome this problem,  
477 we repeat random picks of compatible input differences  $\beta_1$  ( $\beta_0$  changes accordingly)  
478 for a given  $\Delta S_I$ , until the main procedure succeeds.

#### 479 4.4 3-Round Connector

480 We construct 3-round connectors upon a 2-round ~~connectors~~connector. Since  
481 almost all output bits of the first round affect  $\alpha_3$  (the difference after the

---

**Algorithm 1:** mainLinearization.

---

**Input:**  $E_M, \beta_0, \alpha_1, \beta_1$ , and  $\alpha_2$   
**Output:** updated  $E_M$

- 1 Derive Eq. (1) from  $(\beta_1, \alpha_2)$ ;
- 2 Derive Eq. (4) and  $U$ ;
- 3 Initialize a counter  $cnt$  to a large integer;
- 4 **while**  $cnt > 0$  **do**
  - // Linearize  $\chi_0$  and obtain  $(L_{\chi_0}, t_{\chi_0})$  and update  $E_M$ .*
  - 5 **if**  $basicLinearization(E_M, \beta_0, \alpha_1, U)$  **succeeds** **then**
    - 6 Derive Eq. (7) using  $(L_{\chi_0}, t_{L_{\chi_0}})$  output by  $basicLinearization$ ;
    - 7 **if** Eq. (7) is consistent with  $E_M$  **then**
      - 8 Add Eq. (7) to  $E_M$ ;
      - 9 Output  $E_M$ ;
      - 10 **return** success;
  - 11  $cnt = cnt - 1$ ;
- 12 **return** ~~failure~~;

---

---

**Algorithm 2:** basicLinearization.

---

**Input:**  $E_M, \beta_0, \alpha_1$ , and  $U$   
**Output:** updated  $E_M, L_{\chi_0}, t_{\chi_0}$

- 1  $l_{sb}, L_{\chi_0}, t_{L_{\chi_0}} = preProcess(\beta_0, \alpha_1, U)$ ; *// Generate the primary  $L_{\chi_0}, t_{L_{\chi_0}}$  and a list of S-boxes that need further processing.*
- 2 **for** each S-box  $sb$  in  $l_{sb}$  **do**
  - 3 Initialize an empty list  $l_{in}$  of set of equations;
  - 4 **if**  $sb$  is active **then**
    - 5 Load to  $l_{in}$  the 6 sets; *// Each set has 1 equation.*
  - 6 **else**
    - 7 Load to  $l_{in}$  all sets of least possible equations that allow linearization of output bits marked by  $U$ ;
  - 8 **while** There is any untested set in  $l_{in}$  **do**
    - 9 Randomly choose an untested set of equations;
    - 10 **if** The set of equations are consistent with  $E_M$  **then**
      - 11 Add it to  $E_M$  and update  $L_{\chi_0}, t_{L_{\chi_0}}$ ;
      - 12 Break;
    - 13 **else**
      - 14 **if** All sets have been tested **then**
        - 15 Output “No solution in  $basicLinearization$ ”;
        - 16 **return** failure;
- 17 Output  $E_M, L_{\chi_0}, t_{\chi_0}$ ; *//  $E_M$  is updated with Eq. (5).*
- 18 **return** ~~failure~~ **success**;

---



---

**Algorithm 3:** preProcess.

---

**Input:**  $\beta_0, \alpha_1, U$   
**Output:** A list  $l_{sb}$  of S-boxes that need further processing,  $L_{\chi_0}, t_{L_{\chi_0}}$

```
1 Initialize an empty list  $l_{sb}$ , a matrix  $L_{\chi_0}$  and a vector  $t_{L_{\chi_0}}$ ;  
2 for  $i = 0 \rightarrow b/5 - 1$  do  
3   if  $U_i > 0$  then  
4     Extract  $\delta_{in}, \delta_{out}$  of the  $i$ -th S-box from  $\beta_0, \alpha_1$ ;  
5     if  $DDT(\delta_{in}, \delta_{out}) = 2, 4$  then  
6       Update  $L_{\chi_0}, t_{L_{\chi_0}}$ ;  
7     if  $DDT(\delta_{in}, \delta_{out}) = 8$  then  
8       if The only nonlinear output bit is not marked by  $U_i$  then  
9         Update  $L_{\chi_0}, t_{L_{\chi_0}}$ ;  
10      else  
11        if Among the 6 2-dimensional linearizable subspaces, some one is  
12          already specified in  $E_M$  then  
13            Update  $L_{\chi_0}, t_{L_{\chi_0}}$ ;  
14          else  
15            Add this S-box to  $l_{sb}$ ;  
16      if the S-box is not active then  
17        if Any set of equations for linearizing the output bits marked by  $U_i$  is  
18          already contained in  $E_M$  then  
19            Update  $L_{\chi_0}, t_{L_{\chi_0}}$ ;  
20        else  
21          Add this S-box to  $l_{sb}$ ;  
22 return  $l_{sb}, L_{\chi_0}, t_{L_{\chi_0}}$ ;
```

---

third round), full linearizations are enforced on the first nonlinear layer  $\chi_0$  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 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 ~~2-round~~ connector be DF. In our collision attacks, DF is a key factor on success. A solution space with 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  $\sum_{i=0}^{b/5-1} \text{DF}_i^{(1)}$  in which  $\text{DF}_i^{(1)}$  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  $\beta_0$  to  $\alpha_1$ , but also the S-box linearizations.

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

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

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

The constraints on the initial state reduce  $(c + p)$  degrees of freedom for the ~~prefixed-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  $\text{DF}_i^{(2)}$ , the degree of freedom of 5-bit input values to S-boxes in the second round, is

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

where  $\delta_{in}$  and  $\delta_{out}$  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  $(5 - \text{DF}_i^{(2)})$  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

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

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

## 523 4.6 How to choose $\beta_2$ , $\beta_1$ and $\beta_0$

524 **Choosing  $\beta_1$  in the 2-round connector.** Recall that we randomly choose  
 525 compatible input differences  $\beta_1$  according to  $\Delta S_I(\alpha_2)$  until the 2-round connector  
 526 succeeds. As the number of active S-boxes in the second round is large enough  
 527 (range from tens to hundreds in our attacks), there is a huge number of compatible  
 528 choices of  $\beta_1$  so that we can only choose those  $\beta_1$  such that  $\beta_1 \rightarrow \alpha_2$  is of the best  
 529 probability for the given  $\alpha_2$ . This allows a speedup of 2-round connectors, as well  
 530 as an increment of DF. Interestingly, the differential probability of  $\beta_1 \rightarrow \alpha_2$  does  
 531 not need to be high because this transition is covered in our 2-round connector.

532 **Choosing  $\beta_0$  in the 2-round connector.** So far we have not given details on  
 533 how  $\beta_0$  can be selected. We follow Dinur *et al.*'s work [10] in a more general way  
 534 to uniquely determine  $\beta_0$ , namely, the difference before  $\chi$  layer of the first round.  
 535 Specifically, the so-called “target difference algorithm” is used which consists of  
 536 a difference phase and a value phase.

537 Given  $\Delta S_I$ , we have randomly chosen a compatible input difference  $\beta_1$ . We  
 538 then build two equation systems  $E_\Delta$  and  $E_M$  accordingly.  $E_\Delta$  is on differences  
 539 of the message pairs (specifically, on  $\beta_0$ ) and  $E_M$  is on ~~values~~ the value of one  
 540 message (specifically, on the value of the message before  $\chi_0$ , i.e.,  $x$  as shown in  
 541 Fig. 3). The initialization of  $E_\Delta$  should abide by (1) the constraints implied by  
 542 the  $(c + p)$ -bit ~~prefixed~~ pre-fixed value that the corresponding difference should  
 543 be 0, and (2) the input difference bits of non-active S-boxes in the first round  
 544 equal to 0. The initialization of  $E_M$  should abide by the ~~prefixed~~ pre-fixed value  
 545 of the last  ~~$(c + p)$ -bit~~  $c + p$  bits. These rules are easy to be implemented as the  
 546 variable vector  $x$  is ~~an invertible linear mapping the image~~ of the initial vector  
 547 ~~by an invertible linear mapping~~. Therefore, in the initialization we equate the  
 548 corresponding bits to their enforced values in  $E_\Delta$  and  $E_M$ .

549 For  $E_\Delta$ , we add additional equations ~~to enforce that  $\alpha_1$  is possibly deduced~~  
 550 ~~from so that its solution, i.e.,  $\beta_0$  is compatible with  $\alpha_1$~~ . Though the obvious way  
 551 is to equate the 5 input difference bits to a specific value for each active S-box in  
 552  ~~$\beta_0$  the first round~~, this will restrict the solution space significantly. As suggested  
 553 in [10], we chose one of the 2-dimensional affine subsets of input differences  
 554 instead of a specific value for each active S-box. This is based on the fact that  
 555 given any nonzero 5-bit output difference to a KECCAK S-box, the set of possible  
 556 input differences contains at least five 2-dimensional affine subspaces. After a  
 557 consistent  $E_\Delta$  system is constructed, the solution space is an affine subspace of  
 558 candidates for  $\beta_0$ . Then we continue to maintain  $E_\Delta$  by iteratively adding two  
 559 additional equations to uniquely specify a 5-bit input difference for the active  
 560 S-boxes. For each active S-box, once the specific input difference is determined,  
 561 we add equations to  $E_M$  system to enforce every active 5-bit of  $x$  (input bits to  
 562 active S-box) to an affine subspace corresponding to the uniquely determined  
 563  $\delta_{in}$  and  $\delta_{out}$ . In this way, we always find a compatible  $\beta_0$  for  $\alpha_1$  that fulfills the  
 564 constraints imposed by the  $(c + p)$ -bit ~~prefixed~~ pre-fixed value.

565 **Choosing  $\beta_2$  in the 3-round connector.**  $\beta_2$  is chosen before running the  
566 3-round connector. Its value fulfills two requirements: (1) given  $\alpha_2 = L^{-1}(\beta_2)$  as  
567 ~~AS~~<sub>T</sub>, a 2-round connector for the first two rounds can be obtained with a  
568 sufficiently large degree of freedom; and (2)  $\beta_2 \rightarrow \alpha_3$  is of high probability where  
569  $\alpha_3$  is the input difference of the following differential trail. While constructing  
570 3-round connectors, specifically the second 2-round connector,  $\beta_2$  is fixed and  
571 does not change. In this way, the second 2-round connector can be constructed  
572 efficiently thanks to the high probability of  $\beta_2 \rightarrow \alpha_3$ .

## 573 5 Search for Differential Trails with GPU

574 In this section, we elaborate on searching differential trails of KECCAK. The  
575 idea of searching differential trails greatly benefits from previous differential  
576 analysis of KECCAK [9, 14, 19, 21]. The associated techniques developed in  
577 the previous works are reviewed first, followed by the considerations in finding  
578 differential trails that fit in our attacks. Subsequently, the exact searching algo-  
579 rithm that provides differential trails for practical collision attacks against KECCAK  
580 [640, 160, 5, 160], 5-round SHAKE128, 5-round SHA3-224/256, and, 5-round  
581 SHA3 256, KECCAK[1440, 160, 5, 160], and KECCAK[1440, 160, 6, 160] is illustrated.  
582 To speed up the searching process, we introduce GPU implementations of KECCAK.  
583 The differential trails to be used in our collision attacks are listed at the  
584 end of this section.

### 585 5.1 Differential Properties of Keccak

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

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

600 Regarding the nonlinear operation S-box, given a non-zero input difference, all  
601 compatible output differences occur with the same probability. Particularly, for in-  
602 put differences with one active bit, the S-box acts as identity with probability  $2^{-2}$ .  
603 However, given an output difference of the S-box, the probabilities of compatible  
604 input differences vary even though the best probability is ~~determined~~<sub>determined</sub>.

605 When the best probability is only  $2^{-3}$ , there are multiple input differences achiev-  
 606 ing this. Therefore, given an output difference of  $\chi$ , usually there are multiple  
 607 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  $\alpha_{i-1}$  and  $\alpha_i$ , respectively. Let  $\beta_i = L(\alpha_i)$ , then an  $n$ -round differential trail starting from the 1-st round is of the following form

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

608 For simplicity, a trail can also be represented with only  $\beta_i$ 's or  $\alpha_i$ 's.

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

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

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

612 ~~The weight of the  $i$ -th round differential~~ The weight  $w(\beta_i \rightarrow \alpha_{i+1})$  is denoted  
 613 by  ~~$w_{i-1}$  where  $i$  starts from 1~~  $w_i$ , and thus the weight of a trail is the sum of the  
 614 weights of round differentials that constitute the trail. In addition,  $\#AS(\alpha)$  ~~are~~ is  
 615 used to represent the number of active S-boxes in a state difference  $\alpha$ . According  
 616 to the properties of  $\chi$ , given  $\beta_i$  the weight of  $(\beta_i \rightarrow \alpha_{i+1})$  is determined; also,  
 617 given  $\beta_i$  the minimum reverse weight of  $(\beta_{i-1} \rightarrow L^{-1}(\beta_i))$  is fixed.

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

## 623 5.2 Requirements for Differential Trails

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

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

$$630 \quad DF = \sum_{i=0}^{b/5-1} DF_i^{(1)} - (c + p) - w_1. \quad (11)$$

631 The first term of the formula depends on the output bits of the first round that  
 632 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

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

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.

[Secondly, In order](#) to mount collision attacks against  $\text{KECCAK}[r, c, n_r, d]$  with methods described in Section 4, it is necessary that

$$\text{TDF} > w_1 + \dots + w_{n_r-2} + w_{n_r-1}^d \quad (12)$$

where  $w_{n_r-1}^d$  is the portion of  $w_{n_r-1}$  that relates to the digest. The trail searching phase is performed to provide  $\Delta S_I$  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  $w = w_{n_r-3} + \dots + w_{n_r-1}^d$  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  $\text{KECCAK}[r, c, n_r, d]$  in Table 3.

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

**Table 3:** TDFs of different versions of  $\text{KECCAK}[r, c, n_r, d]$

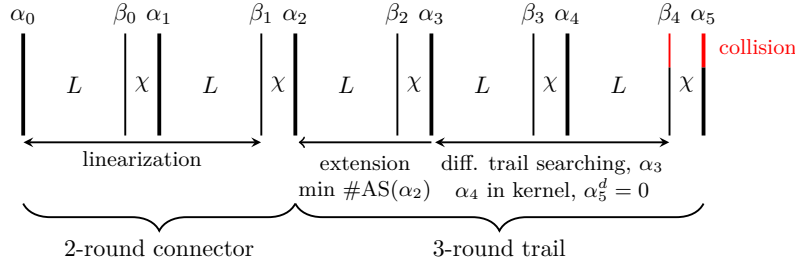
$\text{KECCAK}[r, c, n_r, d]$	TDF	Remarks
$\text{KECCAK}[1440, 160, 5, 160]$	479	KECCAK contest
$\text{KECCAK}[1344, 256, 5, 256]$	378	SHAKE128
$\text{KECCAK}[640, 160, 5, 160]$	159	KECCAK contest
$\text{KECCAK}[1440, 160, 6, 160]$	479	KECCAK contest
$\text{KECCAK}[1152, 448, 5, 224]$	188	SHA3-224
$\text{KECCAK}[\textcolor{red}{1152,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

### 5.3 Searching Strategies

**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  $(\beta_3)$ , that ensure  $\alpha_3$  and at least one choice of  $\alpha_4$  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 differential trails of 4 or 5 rounds for our collision attacks are obtained in the following way.

**Algorithm for searching differential trails.** We sketch below our steps for finding 4-round differential trail cores ~~for-of KECCAK and, upon which 5-round collision attacks are mounted, as illustrated in Fig. 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 trails by one more round, ~~as shown in Fig. 5.~~

1. Using KeccakTools, find special  $\beta_3$ 's with a low Hamming weight, say 10.
2. For every  $\beta_3$  obtained, traverse all possible  $\alpha_4$  using a tree structure, compute  $\beta_4 = L(\alpha_4)$  and test whether there exists a compatible  $\alpha_5$  where  $\alpha_5^d = 0$  (Requirement (1)). If so, keep this  $\beta_3$  and record its forward extension, otherwise discard it.
3. For the remaining  $\beta_3$ 's, traverse all possible  $\beta_2$ '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  $(\beta_2, \beta_3, \beta_4)$ .



**Figure 4:** ~~Collision Search for differential trails and collision attacks on 5-round KECCAK. The 6-round attack is obtained by extending the differential trail one round forward and covering the third round by the connector. That is to say, we need to search 5-round trail cores where the differential trail of the last three rounds is fulfilled probabilistically in the brute-force searching stage.~~

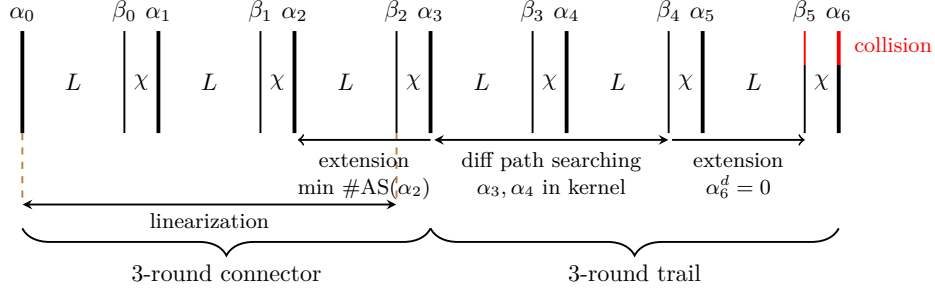


Figure 5: Search for differential trails and collision attacks on 6-round KECCAK.

#### 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.

**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: ccmminer (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](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 that 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/~~256~~, 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 ~~collisions-an inner collision~~ of the last 160 state bits for KECCAK[1440,160,6,160], ~~rather than a general digest collision~~. Details of these differential trail cores are provided in Appendix D.

**Table 5:** Differential trail cores of KECCAK, where  $160^\dagger$  stands for an inner collision on the last 160 bits of the state.

No.	$r + c$	$\#AS(\alpha_2-\beta_2-\beta_3-\beta_4^d)$	$w_1-w_2-w_3-w_4^d$	$w_2+w_3+w_4^d$	$d$	Trail info.
1	1600	102- 8- 8-2	240-19-16-4	39	256	Table 9
2	1600	88- 8- 7-0	195-21-15-0	36	256	
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.	$r + c$	$\#AS(\alpha_2-\beta_2-\beta_3-\beta_4-\beta_5^d)$	$w_1-w_2-w_3-w_4-w_5^d$	$w_3+w_4+w_5^d$	$d$	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	$160^\dagger$	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.cryptosg/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]. After solving the 2-round connector in 9.6 seconds, the ~~resulted~~ resulting solution

**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	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 <sup>†</sup>	47.81*	112h <sup>‡</sup>	Table 18
KECCAK[1440,160,160 <sup>-1</sup> ]	6	No. 6	7.5h	145 <sup>†</sup>	48.83*	60200h	Table 19

\* The weight takes multiple trails of the last two rounds into consideration.

<sup>†</sup> DF of the message space returned by the 2-round connector for the first two rounds.

<sup>‡</sup> The actual time using three NVIDIA GeForce GTX970 GPUs while elsewhere it refers to the time when one CPU core is used.

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<sup>1</sup>. An example of collision is given in Table 18.

**Collision-Inner collision on the last 160-bit of Keccak[1440, 160, 6, 160]**  
When ~~a~~-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 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

<sup>1</sup> Our experiment shows  $2^{28.87}$  pairs of 5-round KECCAK could be evaluated per second on NVIDIA GeForce GTX970 graphic card.

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  $\beta_1$ .** In our 2-round connector,  $\beta_1$  is randomly chosen such that  $\beta_1 \rightarrow \alpha_2$  is of the best probability, except KECCAK[1440, 160, 5, 160] for which the connector randomly chooses any compatible  $\beta_1$ . 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  $\beta_1$  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 SHA3-224/256, 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 ~~message~~ 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,

831 *i.e.*, full S-box linearization and non-full S-box linearization. By linearizing S-boxes  
 832 in the first round, we extend Dinur *et al.*'s 1-round connector to ~~one of~~ two rounds.  
 833 Further, by applying linearization to the first two rounds, 3-round connectors  
 834 became possible. We combined our connectors with suitable differential trails  
 835 searched with dedicated strategies, and then obtained seven practical collisions  
 836 on six round-reduced SHA-3 and KECCAK variants, including 5-round SHAKE128,  
 837 5-round SHA3-224/~~256~~, 5-round SHA3-256, a 6-round instance of KECCAK contest  
 838 and a variant of it. So far, these are the best collision attacks on round-reduced  
 839 SHA-3 and KECCAK contest instances.

## 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](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.



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

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

## 960 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

## 961 C Details of Differential Trail Search

### 962 C.1 Analysis of the Starting Point of the Search

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

966 **Searching from light  $\beta_3$ 's.** Our initial goal is to find collisions for 5-round  
967 KECCAK. To facilitate a 5-round collision of KECCAK, we need to find 4-round  
968 differential trails satisfying the three requirements mentioned in Section 5.2.  
969 However, it is difficult to meet all of them simultaneously even though each of  
970 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  $\alpha_2$  (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  $\#AS(\alpha_2) < 92$  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  $\beta_3$ 's. From light  $\beta_3$ '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.

**$\alpha_3, \alpha_4$  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  $\alpha_3$  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  $\theta^{-1}$  and the sparseness of  $\beta_3$ . When  $\#AS(\alpha_3) > 11$ , the complexity for searching backwards for one  $\beta_3$  is greater than  $2^{34.87}$  which is too time-consuming. On the other hand, we had better also confine  $\alpha_4$  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  $\theta$  the non-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  $\beta_3$  which makes sure  $\alpha_3 = L^{-1}(\beta_3)$  lies in CP-kernel, and for which there exists a compatible  $\alpha_4$  in CP-kernel. Fortunately, such kind of  $\beta_3$ '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  $\beta_3$ '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  $\beta_3$ , 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  $\beta_3$ . 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  $(\beta_3, \beta_4)$  generated by KeccakTools, and  $C_1$  is almost as large as  $C_2$ .

- 1013 • Generate  $\beta_3$  such that  $\alpha_3 = L^{-1}(\beta_3)$  lies in CP-kernel, and that there exists  
1014 a compatible  $\alpha_4$  in CP-kernel, using `TrailCoreInKernelAtC` of KeccakTools  
1015 [5] where the parameter *aMaxWeight* is set to be 60. We obtain more than  
1016 3000 such cores.
- 1017 • For each  $\beta_3$ , if  $C_1 \leq 2^{36}$ , we traverse all possible  $\alpha_4$ , compute  $\beta_4$ , and check  
1018 whether the collision is possible for  $\beta_4$ . If yes, keep this  $\beta_3$  and record this  
1019 forward extension, otherwise, discard this  $\beta_3$ .
- 1020 • For remaining  $\beta_3$ , if  $C_2 \leq 2^{35}$ , try all possible  $\beta_2$  which are compatible with  
1021  $\alpha_3 = L^{-1}(\beta_3)$ , and compute  $AS(\alpha_2)$  where  $\alpha_2 = L^{-1}(\beta_3)$ . If  $AS(\alpha_2) \leq 110$ ,  
1022 check whether this trail core  $(\beta_2, \beta_3, \beta_4)$  is practical for the collision attack.

1023 To find a 5-round trail core for KECCAK[1440, 160, 6, 160], we adapt the second  
1024 step as follows.

- 1025 • For each  $\beta_3$ , extend forwards for one round using `KeccakFTrailExtension`  
1026 of KeccakTools [5] with weight up to 45. For each generated 2-round core  
1027  $(\beta_3, \beta_4)$ , if  $C_1 \leq 2^{36}$  for  $\beta_4$ , traverse all possible  $\alpha_5$  and compute  $\beta_5$ . Check  
1028 whether there exists an  $\alpha_6$  such that  $\alpha_6^d = 0$ . If yes, record the three-round  
1029 core  $\beta_3, \beta_4, \beta_5$ , otherwise, discard the  $\beta_3$ .

## 1030 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

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

1031 where  $\alpha_5$  is compatible with  $\beta_4$  and  $\beta_1 \rightarrow \alpha_2$  is of the least weight determined  
1032 by  $\beta_2$ . In our collision attacks on 5-round (6-round) KECCAK, 4-round (5-round)  
1033 trail cores are needed.

1034 The 1600-bit state is displayed as a  $5 \times 5$  array, ordered from left to right, where  
1035 ‘|’ acts as the separator; each lane is denoted in hexadecimal using little-endian  
1036 format; ‘0’ is replaced with ‘-’ for differential trails.

## 1037 E Instances of Collisions

1038 In this section, we give instances of collisions against KECCAK[1440, 160, 5, 160],  
1039 KECCAK[640, 160, 5, 160], KECCAK [1440, 160, 6, 160], 5-round SHAKE128, 5-round  
1040 SHA3-224/~~256~~, 5-round SHA3-256 respectively. Note that we denote two colliding  
1041 messages with  $M_1, M_2$ .

**Table 9:** Trail core No.1 ~ 3 used in the collision attacks

Trail core No. 1, used in the collision attack of 5-round SHAKE128						
$\beta_2$	----- -----24 -----2 ----- ----- -----	$2^{-19}$				
	----- -----4 ----- ----- -----4					
	-----4----- -----2 ----- ----- -----					
	-----4----- -----4-----4----- -----4-----4					
$\beta_3$	----- -----4-----1----- ----- -----1-----	$2^{-16}$				
	----- -----4----- ----- ----- -----					
	-----8 ----- ----- ----- -----					
	----- -----1----- ----- -----1-----					
$\beta_4$	----- ----- ----- 2-4-----	$2^{-4}$				
	-----8-----4-----8----- -----					
	-----4-8 -----4-----1-----8-----2-----					
	-----4----- -----4----- -----1-----					
Trail core No. 2, used in the collision attack of KECCAK[1440, 160, 5, 160]						
$\beta_2$	----- ----- ----- ----- ----- -----	$2^{-21}$				
	-----1-----8-----1-----1----- -----					
	-----1----- -----1----- -----1----- -----					
	----- -----8----- ----- -----1-----					
$\beta_3$	-----8----- ----- -----1----- -----	$2^{-15}$				
	----- -----8----- -----1----- -----					
	----- -----1----- ----- -----1					
	-----8-----8----- ----- -----					
$\beta_4$	----- ----- -----2----- -----1-----	1				
	-----1-----2-----2----- ----- -----					
	----- -----4----- ----- -----					
	-----8----- ----- -----2-----					
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.						
$\beta_2$	-----1----- -----4 ----- ----- -----	$2^{-24}$				
	-----4-----4 -----4 -----2----- -----					
	2----- -----2----- -----2----- -----					
	2-----2-----1-----2----- -----1					
$\beta_3$	-----1----- -----1 -----4----- -----	$2^{-19}$				
	----- ----- -----1 -----4----- -----					
	----- -----1----- ----- -----4-----					
	-----1-----1----- -----4----- -----					
$\beta_4$	-----1-----4-----4----- -----4-----	1				
	-----8----- -----4----- ----- -----					
	----- -----1----- -----4----- -----1-----4-----					
	4----- -----2----- ----- -----1-----4-----					

**Table 10:** Trail core No. 4, used in the collision attack of KECCAK[640,160,5,160].  $\beta_4$  has two choices.

$\beta_2$	----- -----8-- -----A-- -----8-- -----8-- ----- ----- ----- ----- ----- 2----- 2----- ----- 2----- ----- 2----- ----- -----8-- -----8-- 1----- ----- ----- -----2-- ----- 1-----	$2^{-20}$	$\beta_4^1$	----- ----- ----- ----- ----- 1-----4 -4----- ----- ----- ----- ----- -----4- -2----- ----- 2----- -----2 -----8-- ----- ----- 4----- 4----- ----- ----- ----- -----	1
$\beta_3$	----- ----- ----- -----1 -----4- -----8- ----- -----1 ----- -----4- ----- ----- ----- -----1 ----- ----- ----- ----- ----- ----- -----8- ----- ----- -----4- -----	$2^{-16}$	$\beta_4^2$	----- ----- ----- ----- ----- 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  $2^{-47.81}$ .

$\beta_2$	----- -----8-- -----8--4-- -----4-- -----2--8 -----2-- -----8 -----2--8 -----2-- ----- -----8----- ----- -----8--4-- -----2-- ----- -----8----- -----2--8 ----- ----- ----- ----- ----- -----	$2^{-25}$
$\beta_3$	----- ----- ----- -----1----- ----- ----- ----- ----- -----1----- ----- -----2----- -----2----- -----2----- 4----- ----- -----2----- 4----- -----2----- 4----- -----	$2^{-18}$
$\beta_4$	----- ----- ----- -----8-- ----- 1----- ----- ----- -----4----- ----- ----- ----- ----- -----8----- ----- ----- ----- ----- -----1--4----- ----- ----- -----8-- ----- -----4-- -----	$2^{-16}$
$\beta_5$	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  $2^{-48.83}$ .

$\beta_2$	-2----- ---8----- -----2----- -----2-----	$2^{-17}$
	-2----- ----- ----- ----- -----	
	----- ---8----- -----4--- ----- -----2-----	
	----- ----- -----4--- ----- -----	
$\beta_3$	-2----- ----- -----2----- ----- -----	$2^{-14}$
	----- ----- ----- ----- -----	
	--1----- ----- -----2----- ----- -----	
	--1----- -----2----- -----2----- -----2----- -----	
$\beta_4$	----- ----- -----1-- ---4----- -----	$2^{-16}$
	----- ----- ---8----- ----- -----	
	----- ----- ---4----- ----- -----	
	----- ----- ----- ---1-----1-- -----	
$\beta_5$	--8----- ----- -----2----- ----- -----	$2^{-20}$
	----- -----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--	
$\beta_5$	--1---8---C2-8- ----- 4-8----- 1 18---2---18---A4 ---1C81---2---	$2^{-20}$
	--52-8---1---C- ---A4-8----- ---8---C2-8---1 ----- ---4-8-----	
	----- ----- ----- ----- -----	
	----- ----- ----- ----- -----	

**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 08A0B2CA4DD83BF 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	$M_2$	5B150BCB C0F3F2FC 5907B5A5 22736DC3 914CF0C5 87477D63 A675A649 8BBA96F 52EB8AE3 19402D41 D9FB4CC3 669FD630 D8C9FC71 57558554 0662F64A 64B4B5C5 7F12BF56 2BEADBF0 F6207B10 F2FD9787 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
$M_2$	0CDD5D25A8BD7CA 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 EA95CDDF2F0B0939 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 FB4B8COA11C64088 EDA7B9EBC05F50A8 0A71DD08E7F1EB5B 5342D2AE78A8BF5 6591A9B0CC2E7CE9 52A3DD827F4EF6DC 9D89B18362B80DE4 FEA719A1875BFFF7 49A2B95AD7B7D147 B23784B72EB9260A 187AEFD07295FD59 EE806366EF9D09FF 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
$M_2$	16F97050842C2D17 A731EE935A43480A 6D8E356BDBD7CBE9 D62C0B356FFA158A 4FAD968080C7F8C8 7C8B8E1C61BC5AB 7E3FCA22B5E29305 5888D4DBE848C840 236DE21CCEFF77B8A 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 COBECADAOAC7F275 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<sup>-1</sup>]

$M_1$	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 D97ABDB097B07E2 73D4472B79731977 37EE2EC25F6706B6 00000000AF85B4FB 0000000000000000 0000000000000000
160 <sup>-1</sup>	8FA6D263 1BF0161E867C399C DE5348CD351076D5