Non-Full Sbox Linearization: Applications to Collision Attacks on Round-Reduced Keccak

Ling Song[1,2,4], Guohong Liao[3,1], and Jian Guo[1]

[1] Nanyang Technological University, Singapore
[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, China
[3] South China Normal University, China
[4] Data Assurance and Communication Research Center, Chinese Academy of Sciences, China

{songling.alpha,ntu.guo,liaogh.cs}@gmail.com

Abstract. The Keccak hash function is the winner of the SHA-3 competition and became the SHA-3 standard of NIST in 2015. In this paper, we focus on practical collision attacks against round-reduced Keccak hash function, and two main results are achieved: the first practical collision attacks against 5-round Keccak-224 and an instance of 6-round Keccak collision challenge. Both improve the number of practically attacked rounds by one. These results are obtained by carefully studying the algebraic properties of the nonlinear layer in the underlying permutation of Keccak and applying linearization to it. In particular, techniques for partially linearizing the output bits of the nonlinear layer are proposed, utilizing which attack complexities are reduced significantly from the previous best results.

Keywords: Keccak, `SHA-3`, hash function, collision, non-full linearization, adaptive.

1 Introduction

The Keccak hash function [4] was a submission to the `SHA-3` competition [19] in 2008. After four years of evaluation, it was selected as the winner of the competition in 2012. In 2015, it was formally standardized by the National Institute of Standards and Technology of the U.S. (NIST) as Secure Hash Algorithm-3 [23]. The `SHA-3` family contains four main instances of the Keccak hash function with fixed digest lengths, denoted by Keccak-d with $d \in \{224, 256, 384, 512\}$, and two eXtendable-Output Functions (XOFs) `SHAKE128` and `SHAKE256`. To promote the analysis of the Keccak hash function, the Keccak designers proposed versions with lower security levels in the Keccak Crunchy Crypto Collision and Pre-image Contest (the Keccak challenge for short) [2], for which the digest lengths are 80 and 160 bits for preimage 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 attracted intensive cryptanalysis from the community [1,9,10,11,12,13,14,15,16,18,21]. In this paper, we mainly focus on the collision resistance of Keccak hash function, in particular those collision attacks with practical complexities. In collision attacks, the aim is to find two distinct messages which lead to the same hash digest. Up to date, the best practical collision attacks against Keccak-224/256 is for 4 out of 24 rounds due to Dinur et al.'s work [10] in 2012. These 4-round collisions were found by combining a 1-round connector and a 3-round differential trail. The same authors gave practical collision attacks for 3-round Keccak-384/512, and theoretical collision attacks for 5/4-round Keccak-256/384 in [11] using internal differentials. Following the work of Dinur et al., Qiao et al. [21] further introduced 2-round connectors by adding a fully linearized round to the 1-round connectors, and gave practical collisions for 5-round `SHAKE128` and two 5-round instances of the Keccak collision challenge, as well as collision attack against 5-round Keccak-224 with theoretical complexities. To the best of our knowledge, there exists neither practical collision attacks against 5-round Keccak-224/256/384/512, nor solution for any 6-round instances of the Keccak collision challenge.

Our contributions. We develop techniques of non-full linearizaion for the Keccak Sbox, upon which two major applications are found. Firstly, improved 2-round connectors are constructed and actual collisions are consequently found for 5-round Keccak-224. Secondly, we extend the connectors to 3 rounds, and apply it to Keccak$[1440, 160, 6, 160]$ — a 6-round instance of the Keccak collision challenge, which leads to the first 6-round real collision of Keccak.

These results are obtained by combining a differential trail and a connector which links the initial state of Keccak and the input of the trail. Our work benefits from two observations on linearization of the Keccak Sbox, which are necessary for building connectors for more than one round. One is to linearize part (not all) of the output bits of a non-active Sbox, at most 2 binary linear equations over the input bits are needed. The other is that, for an active Sbox 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 Sbox, two linear equations of input bits are required, as noted by Dinur et al. in [10]. Therefore, for both non-active and active Sboxes, 2 or less equations can be used to linearize part of the output bits. In this paper, we call it *non-full linearization*. When all output bits of an Sbox need to be linearized, *at least* three equations of input bits are required as shown in [21]. So, the non-full linearization saves degrees of freedom on Sboxes where it is applicable. With this in mind, we apply techniques of non-full linearization to the first round permutation of Keccak-224, and successfully construct a 2-round connector with a much larger solution space, which brings the collision attack complexity against 5-round Keccak-224 from 2^{101} down to practise. Applying techniques of non-full linearization to the second round, 3-round connectors are constructed for Keccak for the first time. Furthermore, adaptive constructions for connectors are proposed to save degrees of freedom, and applied to KECCAK[1440,160,6,160]. In adaptive 3-round connectors, non-full linearization of the second round actually does not consume any degree of freedom, but rather it divides the solution space into subspaces of smaller sizes. This guarantees that sufficiently many message pairs that bypass the first three rounds can be generated such that a colliding pair following the latter 3-round differential trail can be found eventually.

Results obtained in this paper are listed in Table 1, compared with the best previous practical collision attacks and related theoretical attacks.

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

Target	n_r Rounds	Complexity	Reference
KECCAK-512	3	Practical	[11]
KECCAK-384	3	Practical	[11]
KECCAK-256	4	Practical	[10]
`SHAKE128`	5	Practical	[21]
KECCAK-224	4	Practical	[10]
	5	2^{101}	[21]
	5	Practical	Sect. 6
KECCAK$[1440, 160, 160]$	5	Practical	[21]
	6	$2^{70.24}$	[21]
	6	Practical	Sect. 7

Organization. The rest of the paper is organized as follows. In Section 2, a brief description of the KECCAK family is given, followed by some notations to be used in this paper. The framework of our collision attacks is sketched in Section 3. We propose techniques of non-full linearization in Section 4. Section 5 presents GPU implementation of KECCAK for searching differential trails and collisions. Section 6 and Section 7 are applications to 5-round KECCAK-224 and KECCAK$[1440, 160, 6, 160]$, respectively. We conclude the paper in Section 8.

2 Description of Keccak

2.1 The sponge function

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

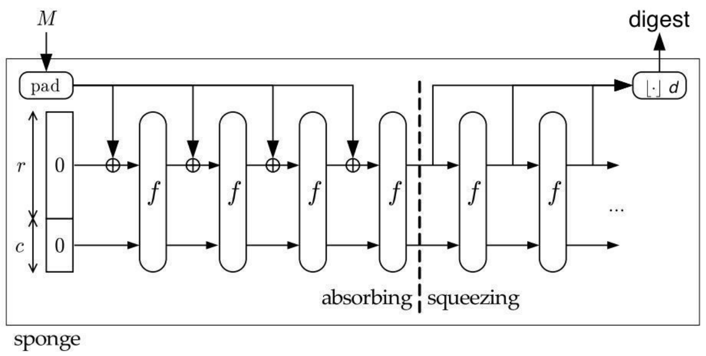

Figure 1: Sponge Construction [3].

2.2 The Keccak hash function

The Keccak hash function follows the sponge construction. The underlying permutation of Keccak is chosen from a set of seven Keccak-f permutations, denoted by Keccak-$f[b]$, where $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ is the width of the permutation in bits. The default Keccak employs Keccak-$f[1600]$. The 1600-bit state can be viewed as a 3-dimensional $5 \times 5 \times 64$ array of bits, denoted as $A[5][5][64]$. Let $0 \leq i, j < 5$, and $0 \leq k < 64$, $A[i][j][k]$ represents one bit of the state at position (i, j, k). Defined by the designers of Keccak, $A[*][j][k]$ is called a row, $A[i][*][k]$ is a column, and $A[i][j][*]$ is a lane.

The Keccak-$f[1600]$ permutation has 24 rounds, each of which consists of five mappings $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$.

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

$$\rho: \ A[i][j][k] \leftarrow A[i][j][(k + T(i,j))\%64], \text{where } T(i,j) \text{ is a predefined constant}$$

$$\pi: \ A[i][j][k] \leftarrow A[i'][j'][k], \text{where } \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix}.$$

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

$$\iota: \ A \leftarrow A + RC_{i_r}, \text{where } RC_{i_r} \text{ is the round constants for } i_r\text{-th round.}$$

Here, '+' denotes XOR and '·' denotes logic AND. As ι plays no essential role in our attacks, we will ignore it in the rest of the paper unless otherwise stated.

2.3 Instances of Keccak and SHA-3

There are four instances Keccak-d of the Keccak sponge function, where c is chosen to be $2d$ and $d \in \{224, 256, 384, 512\}$. To promote cryptanalysis against Keccak, the Keccak design team also proposed versions with lower security levels in the Keccak challenge, where $b \in \{1600, 800, 400, 200\}$, $(d = 80, c = 160)$ for preimage challenge and $(d = 160, c = 160)$ for collision challenge. In this paper, we follow the designers' notation Keccak$[r, c, n_r, d]$ for the instances in the challenge, where r is the rate, $c = b - r$ is the capacity, d is the digest size, and n_r is the number of rounds the underlying permutation Keccak-f is reduced to.

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

The SHA-3 standard adopts the four Keccak instances with digest lengths $224, 256, 384$, and 512. The only difference is the padding rule. In SHA-3 standard, the message is appended '01' first. After that, the multi-rate padding is applied. In this paper, we only fucus on collision attacks against 5-round Keccak-224 and Keccak$[1440, 160, 6, 160]$.

2.4 Notations

In this paper, only one-block padded messages are considered for collision attacks, i.e., we choose message M such that $\overline{M} = M||10^*1$ is one block. According to the multi-rate padding rule, the minimal number of padded bits is 2 while the minimal number of fixed padding bit p is 1. The first three mappings θ, π, ρ of the round function are linear, and we denote their composition by $L \triangleq \pi \circ \rho \circ \theta$. The nonlinear layer χ applying to each row is called an Sbox, denoted by $S(\cdot)$. The differential distribution table (DDT) is a 2-dimensional 32×32 array, where all differences are calculated with respect to bitwise XOR. δ_{in} and δ_{out} are used to denote the input and output difference of an Sbox. Then DDT $(\delta_{in}, \delta_{out})$ is the size of the solution set $\{x \mid S(x) + S(x + \delta_{in}) = \delta_{out}\}$. Let $AS(\alpha)$ denote the number of active Sboxes in the state α.

3 The Collision Attack Framework

This section gives an overview of the framework of our collision attacks, and describes our motivations after a brief review of previous works.

In our attacks, as well as two previous related works [10, 21], an n_{r_1}-round connector and a high probability n_{r_2}-round differential trail are combined to find collisions for $(n_{r_1} + n_{r_2})$-round Keccak. Here, an n_{r_1}*-round connector* is defined as a certain procedure which produces message pairs $(\overline{M}_1, \overline{M}_2)$ satisfying three requirements.

(1) The last $(c + p)$-bit difference of the initial state is zeros;

(2) The last $(c+p)$-bit value of the initial state is fixed;
(3) The output difference after n_{r_1} rounds should be fixed and equal to the input difference of the differential trail.

Given an n_{r_2}-round differential, there are two stages of our $(n_{r_1}+n_{r_2})$-round attack, as illustrated in Fig. 2 below:

– *Connecting stage.* Construct an n_{r_1}-round connector and get a subspace of messages bypassing the first n_{r_1} rounds.
– *Brute-force searching stage.* Find a colliding pair following the n_{r_2}-round differential trail from the subspace by brute force.

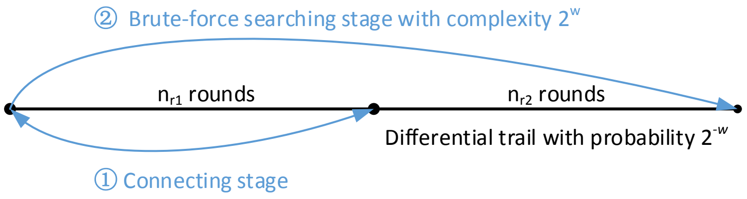
② Brute-force searching stage with complexity 2^w
n_{r1} rounds
n_{r2} rounds
Differential trail with probability 2^{-w}
① Connecting stage

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

We use χ_i to represent the nonlinear layer χ at round i. Then the first n_{r_1} rounds of Keccak can be denoted as

$$\chi_{n_{r_1}-1} \circ L \circ \cdots \circ \chi_0 \circ L.$$

For the differential trail, we denote the differences before and after i-th round by α_i and α_{i+1}, respectively. Let $\beta_i = L(\alpha_i)$, then an n_{r_2}-round differential trail starting from the n_{r_1}-th round is of the following form

$$\alpha_{n_{r_1}} \xrightarrow{L} \beta_{n_{r_1}} \xrightarrow{\chi} \alpha_{n_{r_1}+1} \xrightarrow{L} \cdots \alpha_{n_{r_1}+n_{r_2}-1} \xrightarrow{L} \beta_{n_{r_1}+n_{r_2}-1} \xrightarrow{\chi} \alpha_{n_{r_1}+n_{r_2}}.$$

For the sake of simplicity, a differential trail can also be represented with only β_i's or α_i's. Additionally, let the weight $w_i = -\log_2 \Pr(\beta_i \to \alpha_{i+1})$. For the last round, since only the Sboxes related to the digest matter, we denote the weight and difference for those Sboxes as $w^d_{n_{r_1}+n_{r_2}-1}$ and $\alpha^d_{n_{r_1}+n_{r_2}}$, respectively.

3.1 Dinur et al.'s one-round connector

In [10], collisions of 4-round Keccak-224 and Keccak-256 are found by combining 1-round connectors and 3-round differential trails. The 1-round connector is implemented by a procedure called *target difference algorithm* which converts the construction of a 1-round connector to solving a system of linear equations. An important property used in the target difference algorithm is as follow.

Property 1. [10] Given a pair of input and output difference $(\delta_{in}, \delta_{out})$ of a Keccak Sbox such that $\texttt{DDT}(\delta_{in}, \delta_{out}) \neq 0$, the set of values $V = \{v \mid S(v) + S(v + \delta_{in}) = \delta_{out}\}$ forms an affine subspace.

Note that, any i-dimensional affine subspace of $\{0, 1\}^5$ can be deduced from $(5 - i)$ linear equations. Now, given an output difference of the first round (or the input difference of a 3-round differential trail), the target difference algorithm proceeds in two phases by adding certain linear equations.

1. Choose a subspace of input differences for each active Sbox which are required to be consistent with the $(c + p)$-bit initial difference. As noted in [10], for any non-zero output difference of a Keccak Sbox, the set of possible input differences include at least five 2-dimensional affine subspaces.
2. Choose a subspace of input values for each active Sbox which are required to be consistent with the $(c+p)$-bit initial value by selecting an input difference from the difference subspace obtained in the previous phase.

Once a consistent system of linear equations is obtained after processing all active Sboxes, a 1-round connector succeeds and the first round now can be fulfilled automatically if messages are chosen from the solution space of the system.

3.2 Qiao et al.'s two-round connector

In [21], 5-round collisions are found by combining 2-round connectors and 3-round differential trails. These 5-round collisions directly benefit from the 2-round connectors in which the first round is fully linearized. It was noted in [21] that affine subspaces of dimension up to 2 could be found such that the Sbox can be linearized.

Any affine subspace of dimension 2 requires 3 linear equations to be defined. Therefore, at least $\frac{b}{5} \times 3$ degrees of freedom are needed to linearize one full round. Note that the total number of available degrees of freedom is at most $b - (c + p)$. Hence, when the capacity is relatively small, i.e., $c < \frac{2b}{5}$ (omitting the small p), linearization of one full round is possible. Once the first round is linearized, the constraints (linear equations over the values) for the Sbox in the first round and in the second round can be united to construct 2-round connectors.

However, linearizing a full round consumes too many degrees of freedom , which leads to very small message subspaces or even makes the 2-round connector fail. To save degrees of freedom, differential trails which impose least possible conditions to the 2-round connector are more desirable. To this end, a dedicated search strategy was used [21] to find suitable differential trails of up to 4 rounds.

3.3 Directions for improvements

It can be seen that both Dinur et al.'s original 1-round connectors and Qiao et al.'s 2-round connectors are constructed by processing a system of linear equations. A side effect of these methods, especially linearizing a full round, is a quick reduction of freedom degrees. On the other hand, connectors are possible only when there are sufficient degrees of freedom. Furthermore, the message space returned by the connector needs to be large enough, otherwise no collision can be found. For example, in the collision attack of 5-round KECCAK-224 from [21], a 2-round connector was constructed successfully, however the obtained message space has a dimension of only 2 which is far from being sufficient to find a colliding pair following the 3-round differential trail.

In [21], a 2-round connector was also constructed successfully for KECCAK$[1440, 160, 6, 160]$, and returned a subspace with large enough messages that bypass the first two rounds. However, the complexity of the brute-force stage is $2^{70.24}$, which leaves the attack against KECCAK$[1440, 160, 6, 160]$ impractical.

In order to find practical collisions for both 5-round KECCAK-224 and KECCAK$[1440, 160, 6, 160]$, these remaining problems in the previous work need be solved. There are two directions to this end. The first is to save degrees of freedom and to consume only when necessary. The second is to spend more effort in faster implementations of KECCAK, for finding differential trails which impose less conditions to the connector, as well as speeding up the brute-force stage.

These are our starting point of this paper. The next four sections elaborate on our effort in these two directions which finally results in practical collisions on 5-round KECCAK-224 and KECCAK$[1440, 160, 6, 160]$.

4 Non-full Sbox Linearization

In this section, techniques of non-full linearization are proposed to save degrees of freedom. For convenience, we introduce the techniques in the context of 2-round connectors, even though they can be applied to 3-round connectors or potentially connectors of even more rounds.

4.1 Two Observations

In the construction of a 2-round connector, there are two systems of linear equations, E_M and E_z, which are generated using Property 1. E_M is over the input value x of the nonlinear layer χ_0 of the first round, while E_z is over the input value z of the nonlinear layer χ_1 of the second round. In order to unite these two systems of linear equations to get a 2-round connector, the nonlinear layer χ_0 between them should be linearized. However, the question is whether all Sboxes of χ_0 must be fully linearized? We show below that the answer is no.

Let the output value of χ_0 be y. Then E_z can be re-expressed over y as E_y since $L \cdot (y + RC_0) = z$, where RC_0 is the round constant for the first round. Due to the diffusion of L, E_y is usually denser than E_z. Let $u = (u_0, u_1, \cdots, u_{b-1})$ be a flag vector where $u_i = 1$ $(0 \leq i < b)$ if y_i is involved in E_y, otherwise $u_i = 0$. Let $U = (U_0, U_1, \cdots, U_{\frac{b}{5}-1})$ where $U_i = u_{5i}u_{5i+1}u_{5i+2}u_{5i+3}u_{5i+4}$, $0 \leq i < \frac{b}{5}$. According to the definition, $0 \leq U_i < 2^5$. For the i-th Sbox of χ_0, if U_i is not zero, a.k.a. some bits of the corresponding Sbox are involved in the equation system, this Sbox should be linearized for the union of the two systems of equations. Note that, it requires at least 3 equations to fully linearize an Sbox. However, the aim of linearization is to unite the two systems of linear equations, which does not necessarily require a full linearization of all Sboxes.

With this intuition in mind, below we show two observations of the Keccak Sbox which explain the background for the non-full linearization.

Observation 1 *For a non-active* Keccak *Sbox, when $U_i \neq 31$,*

a. if $U_i = 0$, it does not require any linearization;
b. if $U_i \in \{\mathtt{01}, \mathtt{02}, \mathtt{04}, \mathtt{08}, \mathtt{10}, \mathtt{03}, \mathtt{06}, \mathtt{0C}, \mathtt{11}, \mathtt{18}\}$ (numbers in typerwritter font are hexadecimals), at least 1 equation should be added to E_M to linearize the output bit(s) of the Sbox marked by U_i;
c. otherwise, at least 2 equations should be added to E_M to linearize the output bits of the Sbox marked by U_i.

This observation comes from the algebraic relation between the input and output of χ. Suppose the 5-bit input of the Sbox 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 Sbox 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 $U_i = \mathtt{01}$ 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, assume the value of x_1 is fixed to be 0 or 1. When $x_1 = 0$, we have $y_0 = x_0 + x_2$; otherwise $y_0 = x_0$. When $U_i = \mathtt{0F}$, it maps to 4 output bits y_0, y_1, y_2, y_3 and they 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 $U_i = \mathtt{1F}$, 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).

For the nonlinear layer χ_0 of the first round, most Sboxes are active and many of them have a DDT value of 8. As noted in [21], to fully linearize those Sboxes with DDT of 8, three equations should be added to E_M for each of them. However, Observation 2 shows that two equations may be enough, and thus 1 bit degree of freedom could be saved.

Observation 2 *For a 5-bit input difference δ_{in} and a 5-bit output difference δ_{out} such that $\mathtt{DDT}(\delta_{in}, \delta_{out}) = 8$, 4 out of 5 output bits are already linear if the input is chosen from the solution set $V = \{x \mid \mathtt{S}(x) + \mathtt{S}(x + \delta_{in}) = \delta_{out}\}$.*

Take $\mathtt{DDT}(\mathtt{01}, \mathtt{01}) = 8$ as an example (see Table 6 of [21]). The solution set is $V = \{\mathtt{10}, \mathtt{11}, \mathtt{14}, \mathtt{15}, \mathtt{18}, \mathtt{19}, \mathtt{1C}, \mathtt{1D}\}$. We rewrite these solutions in 5-bit stings where the right most bit is the LSB as follows.

$$\begin{aligned}
\texttt{10} &: \mathbf{1}00\mathbf{0}0\\
\texttt{11} &: \mathbf{1}00\mathbf{0}1\\
\texttt{14} &: \mathbf{1}01\mathbf{0}0\\
\texttt{15} &: \mathbf{1}01\mathbf{0}1\\
\texttt{18} &: \mathbf{1}10\mathbf{0}0\\
\texttt{19} &: \mathbf{1}10\mathbf{0}1\\
\texttt{1C} &: \mathbf{1}11\mathbf{0}0\\
\texttt{1D} &: \mathbf{1}11\mathbf{0}1
\end{aligned}$$

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

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

Therefore, if the only nonlinear bit y_1 is not involved in E_y, these two equations $x_1 = 0$ and $x_4 = 1$ are enough for the union. Note that, given the input difference and the output difference, these two equations are used to restrict the input value from $\{0, 1\}^5$ to the solution set and have already been included in E_M.

4.2 How to choose β_1

In both previous works [10, 21], those β_1s are chosen such that all Sboxes of $\alpha_1 = L^{-1}(\beta_1)$ are active. This is reasonable since a fully active α_1 makes it easy to find a β_0 that is compatible with α_1 and $(c + p)$-bit zero initial difference. Additionally, if full linearization is applied to every Sbox of χ_0, non-active Sboxes have no advantage over active Sboxes in saving degree of freedoms.

Now non-full linearizations are to be applied. The observations in this section demonstrate that for an Sbox less than 3 equations may be enough for the union. It is likely that non-active Sboxes have advantage over active Sboxes. To extensively exploit the non-full linearization for a larger solution space, it is better to have more non-active Sboxes. Moreover, it is interesting to note that once β_1 is chosen, we can not only calculate the number of non-active Sboxes $\#nonact$ of the first round, but also the number of non-active Sboxes which require only 1 or 2 equations for the union. Those non-active Sboxes which require only 1 equation for the union are more interesting. Let the number of them be $\#save$. Large $\#nonact$ and $\#save$ probably lead to large message subspaces that bypass the first two rounds. However, too many non-active Sboxes will slow down the 2-round connector finding program. This problem will be further discussed when techniques of non-full linearization are applied to concrete instances in latter sections.

5 GPU Implementation of Keccak

In this section, techniques for GPU implementation of Keccak are introduced to improve our computing capacity over CPU implementations. 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 usages: searching for differential trails among larger spaces and bruteforce search of collisions from differential trails with lower probability.

5.1 Overview of the 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. So GPU computing is widely used to accelerate compute-intensive applications nowadays. 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.

CUDA is a general purpose parallel computing architecture and programming model that is used in Nvidia GPUs [20]. One of programming interfaces of CUDA is CUDA C/C++ which is based on standard C/C++. Here, we mainly focus CUDA C++.

5.2 Existing implementations and our implementations

Guillaume Sevestre [22] 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 [8], 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 shared memory. One disadvantage of this strategy is bank conflict — concurrent access to shared memory of the same bank by threads from the same warp will be forced to be sequential. Besides, there are two open-source softwares providing GPU implementations of Keccak: ccminer (ref. `http://ccminer.org`) and hashcat (ref. `https://hashcat.net`) in CUDA and OpenCL, respectively.

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

5.3 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 A.2.

Table 2: 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 2 lists the performance. 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 bruteforce 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. Both versions did not include the data transfer time. It can be seen that GTX1070 can be 2^8 times faster than a CPU core. The source codes of these two versions are available freely via `http://team.crypto.sg/Keccak_GPU_V1andV2.zip`.

5.4 Search for differential trails

We follow the strategies proposed in [21] for searching differential trails. Specifically, special differences (explained more in Appendix B) before χ of the third round β_3 are first generated by KeccakTools [6], and then extended one-round forward to check the validity for d-bit collisions. For those β_3s which are possible for collision, we extend them one round backward, and calculate the number of active Sbox AS in the extended round. A trail with small AS is desirable for connectors.

Note that all extensions should be traversed. Given a β_3, suppose there are C_1 possible one-round forward extensions and C_2 one round backward extensions. These two numbers are determined by the active Sboxes of β_3. If the number of active Sboxes is AS, then roughly $C_1 = 4^{AS}$ and $C_2 = 9^{AS}$ according to the `DDT` referred from Table 6 in [21]. In the search for 3-round trails of Keccak-224, C_2 is the dominant time complexity, while for 4-round trails of Keccak$[1440, 160, 6, 160]$, we start from (β_3, β_4) generated by KeccakTools, and C_1 is almost as large as C_2.

With the help of the GPU implementation, the β_3s generated by KeccakTools where $C_2 \le 2^{35}$ are traversed for finding differential trails for Keccak-224 with

AS as small as possible, and (β_3, β_4) where $C_1 \leq 3^{36}$ are explored for finding 4-round trails for Keccak$[1440, 160, 6, 160]$ with $w_3+w_4+w_5^d$ as small as possible. As a comparison, the search for differential trails in [21] only covers β_3 and (β_3, β_4) with C_1, C_2 being less than 2^{30}. In summary, the best 3-round differential trail we obtained for Keccak-224 has $AS = 81$, and the best 4-round differential trail for Keccak$[1440, 160, 6, 160]$ holds with $w_3 + w_4 + w_5^d = 52$. These two trails are used in our collision attacks in the following two sections respectively. More details of the searching algorithm are given in Appendix B.

6 Application to 5-Round Keccak-224

In this section, techniques for non-full linearization are applied to 5-round Keccak-224. Firstly, the best 3-round differential trail we found for Keccak-224 is described. With this differential trail, an improved 2-round connector using non-full linearizations is constructed and it outputs sufficient message pairs among which collisions of 5-round Keccak-224 are found with real examples.

6.1 3-Round differential trail

The information of the best 3-round differential trail we obtain is listed in Table 3 and the trail itself is displayed in Table 7. Specifically, the weight of χ_1 is 187. Once the 2-round connector succeeds and outputs an sufficiently large message space, the complexity for searching a collision is 2^{48} and can be reduced to $2^{45.62}$ if multiple trails of last two rounds are taken into account. In brief, this trail imposes 187 equations to the 2-round connector and requires a solution space of size at least $2^{45.62}$. As shown in the table, our trail is better than the one used in [21] which imposes a bit more equations to the 2-round connector.

Table 3: Differential trails for collision attacks against Keccak-224.

No.	$AS(\alpha_2$-β_2-β_3-$\beta_4^d)$	w_1-w_2-w_3-w_4^d	$w_2 + w_3 + w_4^d$	Reference
1	85- 9-10-2	190-25-20-3	48	[21]
2	81-10-10-1	187-26-20-2	48	This paper

6.2 Improved 2-round connector

In order to extensively exploit the non-full linearization, large *#nonact* and *#save* would be beneficial. However, too many non-active Sboxes may make it difficult or impossible to find β_0s that are compatible with the $(c + p)$-bit zero initial difference, and further make it difficult for the 2-round connector to succeed. To find a balance, values for $\#nonact$ and $\#save$ are heuristically explored. Finally, we set $10 < \#nonact \leq 30$ and $\#save \geq 16$.

Our improved 2-round connector is given as follows and the steps are visualized in Fig. 3.

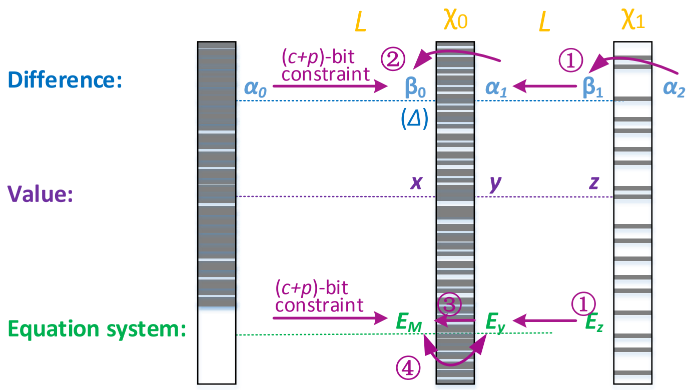

Figure 3: Visualized 2-round connector.

The 2-Round Connector for Keccak-*224.*
Inputs: 449-bit fixed initial value, α_2, two bound variables bnd_1, bnd_2.
Outputs: Difference Δ, a subspace of messages.

1. Randomly choose a possible input difference β_1 of χ_1 according to α_2 such that the differential $\beta_1 \rightarrow \alpha_2$ has the best probability. Calculate $\alpha_1 = L^{-1}(\beta_1)$ and $\#nonact$ of α_1. Construct a system of linear equations E_z over the values of the second χ using Property 1. Derive E_y from E_z using L, RC_0. Calculate U and $\#save$. If $10 < \#nonact \leq 30$ and $\#save \geq 16$, go to Step 2, otherwise repeat this step.
2. Launch Dinur et al.'s target difference algorithm with β_1 and 449-bit fixed initial value. Once the algorithm succeeds, the input differences for the first two rounds are fixed and a system of linear equations E_M over the input x of χ_0 that defines a subspace is obtained, and move to Step 3. If this step fails bnd_1 times, go to Step 1, otherwise repeat this step.
3. Partially linearize the first round according to Observation 1 and 2 by adding equations to E_M. Once succeed, a smaller subspace defined by the updated E_M and the corresponding partial linear mapping of the first χ is obtained, and move to Step 4, otherwise repeat this step.
4. Unite E_M and E_y using the partial linear mapping of χ_0. Once a consistent system is obtained, go to Step 5. If this step fails bnd_2 times, go to Step 1, otherwise go to Step 3.

5. A 2-round connector is constructed successfully. Check the size of the solution space of the resulted equation system. If the size of the solution space is less than 2^{46}, go to Step 1; otherwise output difference Δ and the solution space.

6.3 Experiments and results

Our 2-round connector succeeds in 15 core hours. The obtained subspace of messages has a size of 2^{55}, larger than the required size of 2^{46}. The number of non-active Sboxes of χ_0 is 29 and $\#save = 16$. Among the non-active Sboxes, no Sbox has $U_i = 0$, and seven Sboxes require 2 equations for the union. Among the 105 active Sboxes with DDT entry 8, 26 of them are exempted from adding an extra equation to E_M. These results confirm that the non-full linearization does save some degrees of freedom and both observations contribute to a larger message subspace that bypasses the first two rounds.

After the 2-round connector succeeds, from the message space returned by the connector, a brute-force search is needed to find a colliding message pair which follows the differential trail in latter 3 rounds. The brute-force search is implemented in CUDA and the search is done on an NVIDIA GeForce GTX1070 graphic card. The first collision is found in 21 minutes, which corresponds to $2^{39.90}$ message pair evaluations in the brute-force stage[5]. The actual complexity is smaller than expected by a non-negligible factor. This may be due to the possibility that there are some other differential trails missing from our collision probability calculation, or we might be just lucky. We give one instance of collision in Table 6.

7 Applications to Keccak[1440, 160, 6, 160]

In this section, 3-round connectors are firstly introduced to attack more rounds of Keccak practically. Since one more round is covered by the connector, hence one less round needs to be fulfilled probabilistically in the bruteforce stage, resulting in lower complexities for the bruteforce search stage. This idea leads to a practical attack against Keccak[1440, 160, 6, 160]. In the following, the differential trail used in our attack is described first, and details of 3-round connectors and experiments are given afterwards.

7.1 4-Round differential trail

Four-round differential trails are searched and used in the attack against Keccak [1440, 160, 6, 160]. The first round of the trail is covered by the connector. Namely, $\beta_2 \to \alpha_3$ is included as the last round of 3-round connector. Thus the weight of the last three rounds, namely $w_3 + w_4 + w_5^d$, determines the time complexity for the brute-force searching stage. To make the attack practical, $w_3 + w_4 +$

[5] Our experiment shows $2^{29.6}$ pairs of 5-round Keccak could be evaluated per second on NVIDIA GeForce GTX1070 graphic card.

w_5^d should be as small as possible. So in the search for differential trails for KECCAK$[1440, 160, 6, 160]$, our major goal is to find a 4-round trail with minimal $w_3 + w_4 + w_5^d$, which is different from the goal of searching trails for 5-round KECCAK-224. The best 4-round trail we obtained using GPU is listed in Table 4. The exact differential trail is shown in Table 9. The time complexity for the brute-force stage is 2^{52} which can be reduced to $2^{51.14}$ if we consider multiple trails starting from the same β_4. The weight of the third round is 25, indicating 25 linear equations of this round should be added to the whole equation system by surmounting the barrier of χ_1.

Table 4: Differential trails for collision attacks against KECCAK$[1440, 160, 6, 160]$.

No.	$AS(\alpha_2$-β_2-β_3-β_4-$\beta_5^d)$	w_1-w_2-w_3-w_4-w_5^d	$w_3 + w_4 + w_5^d$	Reference
1	145-6-6-10-14	340-15-12-22-23	57	[21]
2	127-9-8- 8-10	292-25-18-18-16	52	This paper

7.2 Adaptive 3-round connector

To construct 3-round connectors, a 2-round connectors is constructed first. Here, full linearizations are applied to χ_0 in the first round, since almost all ($1595 \sim 1600$) output bits of χ_0 are involved in the equation system of latter two rounds due to the diffusion of the linear layer L. Suppose the resulted equation system of the 2-round connector over the first two rounds is E_M. Then equations for the third round are added to E_M adaptively to get 3-round connectors.

Note that, the first three rounds of KECCAK permutation is represented as

$$\chi_2 \circ L \circ \chi_1 \circ L \circ \chi_0 \circ L$$

by omitting the ι. Let the input and output of χ_0 be x and y, the input and output of χ_1 be z and y', and the input of χ_2 be z', as shown in Fig. 4. Suppose the system of equations E_M returned by the 2-round connector is

$$A \cdot x = t_0.$$

The full linear map of χ_0 is also returned and expressed as

$$L_{\chi_0} \cdot x + t_1 = y.$$

That is to say, $x = L_{\chi_0}^{-1} \cdot (y + t_1)$. Since $z = L \cdot (y + RC_0)$, now E_M can be re-expressed over z as follow.

$$\begin{aligned} A \cdot x &= A \cdot L_{\chi_0}^{-1} \cdot (y + t_1) \\ &= A \cdot L_{\chi_0}^{-1} \cdot (L^{-1} \cdot z + RC_0 + t_1) \\ &= t_0. \end{aligned}$$

Let $A' = A \cdot L_{\chi_0}^{-1} \cdot L^{-1}$ and $t_0' = t_0 + A \cdot L_{\chi_0}^{-1} \cdot (RC_0 + t_1)$. Then an equivalent equation system E_M' of E_M is obtained as

$$A' \cdot z = t_0'. \tag{1}$$

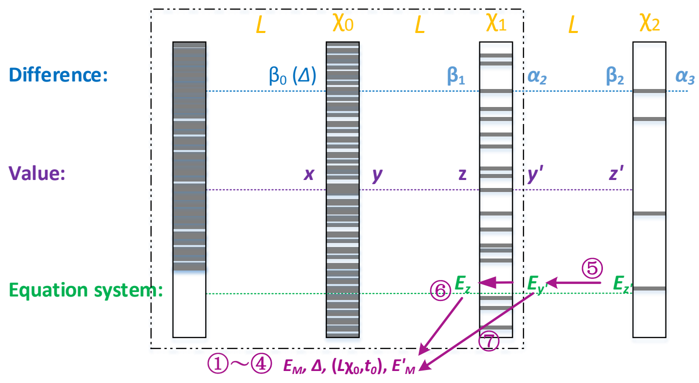

Figure 4: Visualized 3-round connector.

With E_M', equations of the third round, i.e., χ_2, now can be processed in the following way. Suppose the equation system $E_{z'}$ constructed using Property 1 for χ_2 is

$$D \cdot z' = t_4.$$

Since $z' = L \cdot (y' + RC_1)$, then $E_{z'}$ can be re-expressed as $E_{y'}$ over y', i.e.,

$$D \cdot L \cdot (y' + RC_1) = t_4. \tag{2}$$

Now to combine E_M' and $E_{y'}$, a linear map between z and y' is needed. Suppose using techniques of non-full linearization a couple of equations E_z,

$$B \cdot z = t_2$$

linearize y' as

$$L_{\chi_1} \cdot z + t_3 = y'. \tag{3}$$

By stacking E_M' and E_z, we get

$$\begin{bmatrix} A' & t_0' \\ B & t_2 \end{bmatrix} \tag{4}$$

Check the consistency of system (4). If it is consistent, then the linear map (3) is valid, otherwise it is not valid. If the linear map (3) is valid, the equation system (2) for the third round now can be united, since

$$\begin{aligned} D \cdot L \cdot (y' + RC_1) &= D \cdot L \cdot (L_{\chi_1} \cdot z + t_3 + RC_1) \\ &= t_4. \end{aligned}$$

If the consistency of the following system (5) holds, then the 3-round connector succeeds, and returns a subspace of z and β_1.

$$\begin{bmatrix} A' & t'_0 \\ B & t_2 \\ D \cdot L \cdot L_{\chi_1} & t_4 + D \cdot L \cdot (t_3 + RC_1) \end{bmatrix} \tag{5}$$

Special Sboxes of χ_1. The 3-round connector for Keccak[1440, 160, 6, 160] may not return a sufficiently large solution space due to a great consumption of degrees of freedom for linearizing χ_1, so multiple 3-round connectors are needed. Whether a 3-round connector succeeds or not depends on the consistency of (5). Note that, if (4) is consistent, (5) is consistent with high probability. However, (4) is consistent with a low probability. This is because $E_{z'}$ has a few equations, while E_z has much more. Take Trail 2 as an example, E_z has 146 equations, while $E_{z'}$ has only 25.

To make the 3-round connector succeed faster, E_z is scrutinized in depth. For an Sbox of χ_1 that should be linearized for uniting E_z with E'_M, let the 5-bit input be $z_0z_1z_2z_3z_4$ and the 5-bit output $y'_0y'_1y'_2y'_3y'_4$. Suppose the value of z_0 is to be fixed to partially linearize χ_1. There are two cases for z_0. The first case is that the value of z_0 has not been fixed in E'_M. In this case both values (0 or 1) for z_0 are valid for the linearization of χ_1. The other case is that z_0 has already been fixed in E'_M. Then only the value that is consistent with E'_M is valid for the linearization. For the latter case, this Sbox is defined to be a *special Sbox*. Our idea is to spot all special Sboxes of χ_1 and always choose the valid linearization for them. For the rest Sboxes, any linearization is valid. In this way, (4) is always consistent.

For Trail 2, 125 Sboxes of χ_1 require to be linearized. The number of special Sboxes is 19. So for the rest 106 Sboxes, any linearization is valid and can be used to successfully construct sufficiently many 3-round connectors.

Algorithm of adaptive 3-round connectors. In adaptive 3-round connectors, full linearizations are applied to χ_0, while non-full linearizations are used for χ_1. Each time the algorithm outputs a subspace of messages by solving (5). More subspaces of messages can be obtained by replacing the linearization of χ_1 with an unused one.

The Adaptive 3-Round Connector
Inputs: 161-bit fixed initial value, α_3, β_2 and α_2
Outputs: initial difference Δ and β_1, multiple subspaces of messages.

1. Apply *The 2-Round Connector* using the 161-bit fixed initial value and α_2. When the 2-round connector succeeds, it returns E_M, Δ, β_1 and the linear map (L_{χ_0}, t_1) with which the equivalent system E'_M can be derived.
2. Construct $E_{z'}$ using β_2 and α_3. Then deduce $E_{y'}$ from $E_{z'}$. Calculate U' for $E_{y'}$. Now the bits of y' that need to be linearized are known. Spot special Sboxes by trying all linearizations for each Sbox whose output bits are

marked by U'. After that, a list of special Sboxes and a corresponding valid linearization are obtained. Initialize a list structure for all Sboxes of χ_1 that are marked by U'. Each Sbox is a node on the list structure. For special Sboxes, the node has only one choice for the linearization, while for other Sboxes, the node contains multiple choices for the linearization.

3. Use the current linearization (L_{χ_0}, t_3) to deduce a united equation system (5). If the system (5) is consistent, solve this system, return a solution space and β_1 and go to Step 4; otherwise, shift the pointer of the list to the next linearization, go to Step 3.
4. Check whether more messages are needed or not. If yes, shift the pointer of the list to the next linearization, go to Step 3; otherwise, exit.

In brief, in 3-round adaptive connectors, the freedom degrees for linearizing the second round are reused and hence not consumed. Thus, multiple solution spaces can be generated successively if one is not enough.

7.3 Experiments and results

The 3-round adaptive connector is applied to Trail 2 in our experiments. In the first step, the 2-round connector succeeds in 4.5 core hours and returns an E_M with 174 degrees of freedom. Every time Step 4 outputs a subspace of messages of size $2^{32} \sim 2^{35}$ which bypass the first three rounds. In order to find one colliding pair, at least $2^{51.14}$ pairs of messages are required. This could be achieved by repeating Step $3 \sim 4$ for $2^{16.14} \sim 2^{19.14}$ 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[6]. An example of collision is given in Table 8.

8 Conclusions

In conclusion, we proposed two major types of techniques for saving degrees of freedom in constructing connectors: non-full linearizations and adaptive connectors. Techniques of non-full linearization avoid unnecessary consumption of degrees of freedom, and its application directly leads to practical collision attacks against 5-round Keccak-224. Adaptive connectors are constructed in an adaptive way that some degrees of freedom are reused, hence not consumed. By combining techniques of non-full linearization and adaptive connectors, 3-round connectors are constructed successfully, resulting in a practical collision attack against Keccak$[1440, 160, 6, 160]$.

These two types of techniques significantly save degrees of freedom. Therefore, one potential future work is to apply these techniques to other Keccak instances which have a tighter budget of freedom degrees, such as Keccak$[240, 160, 5, 160]$.

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

Acknowledgement. The authors would like to thank anonymous reviewers of CRYPTO 2017 for their helpful comments and suggestions. Part of this work was supported by the National Key Basic Research Program of China (2013CB834203) the National Natural Science Foundation of China (Grants 61472417, 61472415, 61402469, 61672516, and 61572028), the Project of Science and Technology of Guangdong (2016B010125002), and the Natural Science Foundation of Guangdong (No. 2015A030313630, 2014A030313439).

References

1. Aumasson, J.P., Meier, W.: 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. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak Crunchy Crypto Collision and Pre-image Contest. http://keccak.noekeon.org/crunchy_contest.html
3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic Sponge functions. Submission to NIST (Round 3) (2011), http://sponge.noekeon.org/CSF-0.1.pdf
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak Reference. http://keccak.noekeon.org (January 2011), version 3.0
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak SHA-3 Submission. Submission to NIST (Round 3) 6(7) (2011)
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: KeccakTools. http://keccak.noekeon.org/ (2015)
7. Canteaut, A. (ed.): Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers, Lecture Notes in Computer Science, vol. 7549. Springer (2012)
8. Cayrel, P.L., Hoffmann, G., Schneider, M.: GPU Implementation of the Keccak Hash Function Family. In: International Conference on Information Security and Assurance. pp. 33–42. Springer (2011)
9. Daemen, J., Assche, G.V.: Differential propagation analysis of keccak. In: Canteaut [7], pp. 422–441
10. Dinur, I., Dunkelman, O., Shamir, A.: New attacks on keccak-224 and keccak-256. In: Canteaut [7], pp. 442–461
11. Dinur, I., Dunkelman, O., Shamir, A.: Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. In: Moriai, S. (ed.) Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8424, pp. 219–240. Springer (2013)
12. Dinur, I., Dunkelman, O., Shamir, A.: Improved practical attacks on round-reduced keccak. J. Cryptology 27(2), 183–209 (2014)
13. Dinur, I., Morawiecki, P., Pieprzyk, J., Srebrny, M., Straus, M.: Cube Attacks and Cube-Attack-Like Cryptanalysis on the Round-Reduced Keccak Sponge Function. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. LNCS, vol. 9056, pp. 733–761. Springer (2015)
14. Duc, A., Guo, J., Peyrin, T., Wei, L.: Unaligned rebound attack: Application to keccak. In: Canteaut [7], pp. 402–421

15. Guo, J., Liu, M., Song, L.: Linear Structures: Applications to Cryptanalysis of Round-Reduced Keccak. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. LNCS, vol. 10031, pp. 249–274 (2016)
16. Jean, J., Nikolic, I.: Internal Differential Boomerangs: Practical Analysis of the Round-Reduced Keccak-f Permutation. In: Leander, G. (ed.) Fast Software Encryption - FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers. LNCS, vol. 9054, pp. 537–556. Springer (2015)
17. Murthy, G.S.: Optimal Loop Unrolling for GPGPU Programs. Ph.D. thesis, The Ohio State University (2009)
18. Naya-Plasencia, M., Röck, A., Meier, W.: Practical analysis of reduced-round keccak. In: Bernstein, D.J., Chatterjee, S. (eds.) Progress in Cryptology - INDOCRYPT 2011 - 12th International Conference on Cryptology in India, Chennai, India, December 11-14, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7107, pp. 236–254. Springer (2011)
19. NIST: SHA-3 COMPETITION. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html (2007-2012)
20. Nvidia, C.: CUDA C PROGRAMMING GUIDE. Nvidia Corporation 120(18) (2011)
21. Qiao, K., Song, L., Liu, M., Guo, J.: New Collision Attacks on Round-Reduced Keccak. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017, Paris, France, April 30 - May 4, 2017, Proceedings, Part III. LNCS, vol. 10212, pp. 216–243 (2017)
22. Sevestre, G.: Implementation of Keccak Hash Function in Tree Hashing Mode on Nvidia GPU (2010), http://hgpu.org/?p=6833
23. 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), http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
24. Volkov, V.: Better Performance at Lower Occupancy. In: Proceedings of the GPU technology conference, GTC. vol. 10. San Jose, CA (2010)

A GPU Implementation

A.1 Techniques of 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, texture memory provides better performance than global memory, and it reduces memory traffic as well. So we can bind input data and some frequent accessed read-only data with texture memory.

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

Instruction-level parallelism. From [24], hashcat, and ccminer, we see that forcing adjacent instructions independent gives better performance. Without prejudice to the functions of the program, we can adjust the order of instructions to improve the efficiency of the operations. In addition, loop unrolling [17] is also a good practice to obtain ILP.

A.2 Hardware specification sheet of GPU

Table 5: 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

B Algorithm for Searching Differential Trails

Before the description of our algorithm for searching differential trails, we introduce more notations which are mainly defined by the designers of Keccak. A state s is in the *Column Parity kernel* (CP-kernel) if $s = \theta(s)$ [5], which means θ acts as an identity and dose not diffuse any bit of the state. The differential trail in the CP-kernel has a number of rounds at most 2, as studied in [9, 14, 18]. Also, an n-round *trail core* (suppose starting from Round 0) is defined with $n-1$ consecutive β_i's, $(\beta_1, \cdots, \beta_{n-1})$, which contains a set of n-round trails

$\alpha_0 \xrightarrow{L} \beta_0 \xrightarrow{\chi} \alpha_1 \xrightarrow{L} \beta_1 \cdots \xrightarrow{L} \beta_{n-1} \xrightarrow{\chi} \alpha_n$ where the first round is of the minimal weight determined by $\alpha_1 = L^{-1}(\beta_1)$, and α_n is compatible with β_{n-1}. In the collision attack of 5-round Keccak-224, actually a 4-round trail core is needed even though the first round is covered by the 2-round connector. In the attack of Keccak$[1440, 160, 6, 160]$, a 5-round core is required and the first two rounds of the trail are covered by 3-round connectors. We list below the steps for finding 4-round cores for Keccak-224, and then describe the difference for Keccak$[1440, 160, 6, 160]$.

– Generate β_3 such that $\alpha_3 = L^{-1}(\beta_3)$ lies in CP-kernel, and that there exists a compatible α_3 in CP-kernel, using TrailCoreInKernelAtC of KeccakTools [6] where the parameter *aMaxWeight* is set to be 64. The number of such β_3 we obtained is 2347.
– For each β_3, if $C_1 \leq 2^{36}$, we traverse all possible α_4, compute β_4, and check whether the collision is possible for β_4. If yes, keep this β_3 and record this forward extension, otherwise, discard this β_3.
– For remaining β_3, if $C_2 \leq 2^{35}$, try all possible β_2 which are compatible with $\alpha_3 = L^{-1}(\beta_3)$, and compute $AS(\alpha_2)$ where $\alpha_2 = L^{-1}(\beta_3)$. If $AS(\alpha_2) \leq 86$, check whether this trail core $\beta_2, \beta_3, \beta_4$ is practical for the collision attack.

Using this algorithm, the best 4-round trail core we found for Keccak-224 has $AS(\alpha_2) = 81$ and $w_2 + w_3 + w_4^d = 48$. In the case of Keccak$[1440, 160, 6, 160]$, trails with one more round are searched, so the second step is adapted as follow.

– For each β_3, extend forwards for one round using KeccakFTrailExtension of KeccakTools [6] with weight up to 45. As a result, 43042 two-round cores are generated. Then for each generated two-round core, if $C_1 \leq 2^{36}$ for β_4, traverse all possible α_5 and compute β_5. Check whether there exists a α_6 such that $\alpha_6^d = 0$. If yes, record the three-round core $\beta_3, \beta_4, \beta_5$, otherwise, discard the β_3. In total, there are only 11 β_3s left for the 160-bit collision.

The best 5-round trail core we found for Keccak$[1440, 160, 6, 160]$ has $AS(\alpha_2) = 127$ and $w_3 + w_4 + w_5^d = 52$. In order to estimate the complexity for the brute-force stage accurately, we consider all possible trails which are possible for the collision and start from the same β_3 for Keccak-224 (β_4 for Keccak$[1440, 160, 6, 160]$).

C Differential Trails and Collisions

In this section, we give details of differential trails used in our attacks and the obtained collisions. The 1600-bit state is displayed as a 5×5 array, ordered from left to right, where '|' acts as the separator; each lane is denoted in hexadecimal using little-endian format; '0' is replaced with '-' for differential trails.

Table 6: Collision for 5-round Keccak-224

	F49A78F0E0CBB2C0	997CF6C13F9F5E37	091EF2AE68CA026C	787A6189D311D2AB	F410786AB060476E
	A56E341B9175DDBD	ED9381C907F7DEFD	EAF49557D1F449F4	BBFDC0C22F0ED3C6	A5FCE33236960AAE
M_1	192598A5E0B275ED	DA7C4363F554A4AE	85B14515A3040D1B	2C5E5C7DDC7E43C3	A900385251BB4F77
	DB530E201E571450	A9C981793A78152F	C55991AC63389C0F	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	C316798E019C8ECB	2EFDF516C6322BEA	B9FE8432A626B2B2	4EEA0858AF5684C2	1793DC9B8BE1EFF0
	DDF791B683238A70	E43E484F5F767DB3	6AE5AD63D1FD51DC	57C509C21AF67220	AF14D053F09C4E6C
M_2	44E594BA9943900F	F2995743C285D101	00C055CA1502459A	013AD29EE0FFB76B	8A9B6A7750956AFF
	D200A9BD2E38993F	54583BF0DAF4D84D	E9784271C6556FFF	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
digest	9F78D9AAD557721B	8DA633A88E5FA089	97403614B9152D9D	0E1F496F	

Table 7: Differential trail used in the collision attack of 5-round Keccak-224. The total probability is $2^{-45.62}$ considering multiple trails of the last two rounds. After collisions were obtained, we found that the trail obtained by cyclically rotating this one 22 bits to the right has a better probability of $2^{-44.59}$.

	-----------8----	--22------------	--2--------8----	--2-------------	--22------8-----	
	----------------	----------------	----------------	----------------	----------------	
β_2	-----------8----	----------------	-----------8----	-----------8----	----------8-----	2^{-26}
	-------2--------	----------------	----------------	--2----2--------	----------------	
	-------2--------	---2------------	----------------	----------------	----------------	
	----------------	----------------	4---------------	-------------4--	----------------	
	-----------2----	----------------	----------------	----------------	----------------	
β_3	---4------------	----------------	----------------	----------------	---8------------	2^{-20}
	---4------------	----------------	----------------	----------------	----------------	
	-----------2----	----------------	4---------------	-------------4--	---8------------	
	----------------	----------------	----------------	----------------	---------------2	
	------4---------	----------------	--2-------------	----------------	-8--------------	
β_4	----------------	----------------	----------------	----------------	-------8--------	2^{-2}
	----------------	--2-------------	----------------	----------------	---------------4	
	1---------------	----------------	---------4------	---------8------	----------------	

Table 8: Collision for the challenge instance Keccak$[1440, 160, 6, 160]$

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

Table 9: Differential trail used in the collision attack of Keccak[1440, 160, 6, 160]. The total probability is $2^{-76.14}$ considering multiple trails of last two rounds. The probability of last three rounds is $2^{-51.14}$.

β_2	`----------------|-----8----------|-----8------4---|----------------|------------4---`
	`-----------2---8|-----------2----|---------------8|-----------2---8|-----------2----`
	`----------------|-----8----------|----------------|----------------|-----8------4---`
	`-----------2----|----------------|-----8----------|-----------2---8|----------------`
	`----------------|----------------|----------------|----------------|----------------`
		2^{-25}

β_3	`----------------|----------------|----------------|---------1------|----------------`
	`----------------|----------------|----------------|---------1------|----------------`
	`----------------|----------------|----------------|----------------|----------------`
	`-----2----------|--2-------------|--2-------------|-4--------------|----------------`
	`-----2----------|-4--------------|--2-------------|-4--------------|----------------`
		2^{-18}

β_4	`----------------|----------------|----------------|------------8---|----------------`
	`--1-------------|----------------|----------------|----------------|---4------------`
	`----------------|----------------|----------------|----------------|-8--------------`
	`----------------|----------------|----------------|--------------1-|---4------------`
	`----------------|------------8---|----------------|-----------4----|----------------`
		2^{-18}

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