TIDAL: Practical Collisions on State-Reduced Keccak Variants

Sahiba Suryawanshi, Dhiman Saha, Shashwat Jaiswal

`de.ci.phe.red` Lab
Department of Electical Engineering and Computer Science
Indian Institute of Technology Bhilai, India
`(sahibas,dhiman,shashwatj)@iitbhilai.ac.in`

Abstract. An important tool that has contributed to collision search on Keccak/SHA3 is the Target Difference Algorithm (TDA) and its internal differential counterpart Target Internal Difference Algorithm (TIDA), which were introduced by Dinur *et al.* in separate works in FSE 2012 and 2013 respectively. These algorithms provide an ingenious way of extending the differential trails by one round and exploiting the affine subspaces generated due to the low algebraic degree of the Keccak S-box. The current work introduces TIDAL, which can extend TIDA by one more round capitalizing on linearization techniques introduced by Guo *et al.* in JoC. This approach requires increment consistency checks, which is also improved in this work. The TIDAL strategy, in conjunction with a deterministic internal differential trail, has been applied to Keccak variants up to 400-bit state-size and leads to practical collision attacks for most of them up to 5 rounds. In particular collisions have been confirmed for 4-round Keccak[136, 64] with a complexity of 2^{20} and on 6-round of Keccak[84,16] with a complexity of 2^5. Further, this work completely characterizes all collision attacks on state-reduced variants, showcasing that TIDAL covers most space up to 5 rounds. As state and round-reduced Keccak variants are used to realize the internal states of many crypto primitives, the results presented here generate a significant impact. Finally, it shows new directions for the long-standing problem of state-reduced variants being difficult to be attacked.

1 Introduction

Collision search is one of the fundamental problems that provide insight into the strength of a cryptographic hash function. The latest hash standard SHA3 and its parent submission to the SHA3 competition Keccak have been one of the most extensively studied hash algorithms. Collision search for SHA3 has evolved in various directions but most of the effort has been concentrated on attacking the variants in the SHA3 standard and to be more precise, on the maximum size permutation. This had lead to the belief that state-reduced or so-called smaller variants of Keccak are particularly difficult to attack and the same has been acknowledged by the designers as well. In FSE 2021, Boissier *et al.* made an effort to target state-reduced variants. However, the authors themselves acknowledge that complexities observed for up to 2-round have been impractical and *'even two rounds required a strong effort'*. The current work tries to set up a framework to deal with state-reduced variants and bridge gap by drastically reducing the collision search complexity to practical limits for most of the variants up to 5 rounds.

Previous Work: Collisions are the holy grail of any hash function analysis and KECCAK/`SHA3` have been no exceptions. In 2011, Naya-Plasencia *et al.* gave the first practical 2-round collision and 3-round near-collision on `SHA3-224` and `SHA3-256`, using a *double kernel* to find the differential path [17]. Later in 2012, Dinur *et al.* gave practical 4-round and 5-round near-collision attacks on the same variants [7]. In the attack, they used a 3-round high probability trail and a 1-round connector derived using what they referred to as the *Target Difference Algorithm* (TDA). In 2013, the same group of authors gave a practical 3-round and theoretical 4-round collision on `SHA3-384` and `SHA3-512`, and increased the number of rounds for `SHA3-224` and `SHA3-256` to 5-round by mounting squeeze attacks [8], which leverage the idea of internal differential cryptanalysis. In 2017, Qiao *et al.* extended Dinur's TDA by 1 more round by applying linearization leading to a 6-round collision attack [19]. In the same year, Song *et al.* improved Qiao's work by using non-full S-box linearization [21] and saving the degrees of freedom providing practical collision attacks on many variants which included a 6-round collision on KECCAK[1440,160]. In 2020, Guo *et al.* [11] gave a 5-round collision on `SHAKE-128`, `SHA3-224`, and `SHA3-256` and a 6-round collision on KECCAK-p[1440, 160] and KECCAK-p[640, 160]. They extended the 1-round connector provided by Dinur *et al.*, to upto three rounds and developed an approach to find suitable differential trails compatible with the connector. Nevertheless, their technique does not apply to larger capacity sizes like `SHA3-384` and `SHA3-512` because of insufficient degree of freedom. In FSE 2022 [13], Huang *et al.* overcame the limitations faced by earlier works for states with larger capacities and reported a practical 4-round collision attack on `SHA3-384`. They used a 2-block message in place of the 1-block message and used SAT solver instead of linearization to improve their result. Also, in place of linearization, they have used SAT solver to improve their result. The analysis of `SHA3` is also being explored from a quantum computing perspective. In a quantum setting [12], Guo *et al.* gave a 6-round collision on `SHAKE-128`, `SHA3-224` and `SHA3-256` using an SAT-based toolkit to search the differential trail. The earliest attempt of attacking a state-reduced variant is attributed to Kölbl *et al.* [15] who reported collisions on the KECCAK variants instantiating the 800-bit permutation. Recently, the interest in state-reduced versions has been renewed by Boissier *et al.* mounting collision attacks on two rounds of the 200 and 400-bit version of KECCAK permutation using algebraic and linearization techniques [6].

The current work attempts to explore the collision search problem with regards to the smaller i.e., state-reduce versions of the KECCAK-p permutation particularly looking at 100, 200, 400-bits versions. This is motivated by two conditions. The first one being the recent work by Boissier *et al.* and their *inability* to penetrate more than 2-round for upto 200-bit variants. This difficulty has also been acknowledged by the designers and is attributed to the faster diffusion in a smaller state [3]. The second reason is the fact that many authenticated lightweight ciphers like CAESAR [1] candidate Ketje [4] and NIST LWC [2] Finalists ISAP [9] and Elephant [10] prefer to instantiate state-reduced permutations for realizing their internal states. This inspires us to systematically investigate how collision search problem pans out for the smaller states. For a thorough analysis we come up with a general framework to capture the way states are reduced, classifying them as `Type-I` which preserve the ratio of rate and capacity of the `SHA3` standard and `Type-II` for fixed capacity size of 160 conforming to KECCAK Crunchy Contest. To put things in context, it can be noted that the variant attacked by Kölbl *et al.* is `Type-I` while out of the attacks reported by Boissier *et al.* on three variants, two are of `Type-I` and the remaining one is of `Type-II`. We revisit the idea of squeeze-attacks that capitalize on the ways internal differentials evolve in the KECCAK state. In particular, we give an algorithm to extend the *Target Internal Difference Algorithm (`TIDA`)* of Dinur *et al.* by one more round which allows us to generate a subspace of two-round symmetric states. This is coupled with a deterministic 1.5 round internal differential trail which leads to a 5-round squeeze attack giving us practical collisions for most of the state-reduced `Type-I` KECCAK variants. The `TIDA` approach requires the incremented consistency check throughout an evolving system of linear equations. Therefore, the proposed approach utilizes data from earlier iterations while performing an incremental linear equation system's consistency check. As a result, it requires lesser operations than the traditional method used in `TDA`/`TIDA`.

Our Contributions: The current work constitutes a comprehensive treatment for plugging the gap in the collision attack space that has been existing in the KECCAK cryptanalysis literature for the sub-800 permutations size zone. The major contributions are outlined below with a summary furnished in Table 1.

- Revisiting the `TIDA` strategy to understand the way it differs from `TDA`
- Equip an approach to handle incremental consistency check operation efficiently.
- Introducing `TIDAL`, a strategy to extend basic `TIDA` for another round adapting state-of-the-art linearization techniques
- Mounting squeeze attacks leveraging `TIDAL` to get most efficient collisions on $2, 3, 4, 5$ rounds of many state-reduced KECCAK variants
- Characterization of the collision attack space for state-reduced KECCAK variants utilizing a general framework to capture the way states are reduced

Organization The rest of the paper is organized as follows. Section 2 gives a brief description of the `SHA3`, squeeze attack, Internal Differential Cryptanalysis of KECCAK, Target Difference Algorithm, and TDA-Connector. In Section 3, we re-introduce the `TIDA` algorithm and give an algorithm for the same. Also, provide an approach to improve the consistency of check operation. `TIDAL`, which extends the `TIDA` by one more round, is introduced in Section 4. We also provide degrees of freedom and experimental support for it in this section. The notion

Table 1: Summary of the results. Here DoF stands for Degree of Freedom

State Reduce Keccak variant	**DoF**	**Generic Collision Complexity**	**TIDAL Collision Complexity**			
			Number of Rounds			
			2	**3**	**4**	**5**
Keccak-p[144, 56]	2^{42}	2^{28}	2^{20}	2^{20}	2^{20}	-
Keccak-p[136, 64]	2^{34}	2^{32}	2^{20}	2^{20}	2^{20}	2^{31}
Keccak-p[288, 112]	2^{86}	2^{56}	2^{40}	2^{40}	2^{40}	-
Keccak-p[272,128]	2^{70}	2^{64}	2^{40}	2^{40}	2^{40}	2^{62}

of our attack devises in Section 5, along with the experimental result, which supports our attack. Comparisons of best results for different variants of state reduce Keccak are given in Section 6. Finally, concluding remarks are provided in Section 7.

2 Preliminaries

2.1 Keccak Internal State and the SPONGE Mode of Operation

The Keccak structure follows the SPONGE construction that maps a variable length input to a fixed or variable length output using a fixed-length permutation. The internal state is b bits wide where $b = c + r$ with c being the capacity and r the rate. Here $b \in \{25, 50, 100, 200, 400, 800, 1600\}$. The internal state is three-dimensional and can be visualized as an array of 5×5 slices where number of slices (l_s) vary as per the permutation size, $l_s = \log_2(\frac{b}{25})$. The nomenclature of the state is captured in Figure 1. The number of iterations (n_r) of the round function is governed by $n_r = 12 + 2 \times l_s$. Absorb and squeeze are the two phases of the SPONGE construction. Initially, the message is processed in the absorption phase, and then the squeeze phase generates the message digest. The message M is broken into multiple blocks of size r. The last block requires a 10*1 padding. The round function is $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$. The working of the sub-operations are as follows :

– θ **(Theta)**: θ mapping is a linear operation that provides diffusion. In this mapping $A[x, y, z]$ XORed with parities of neighboring 2 columns in the following manner:

  $$A[x, y, z] = A[x, y, z] \oplus P[(x-1) \bmod 5, *, z] \oplus P[(x+1) \bmod 5, *, (\bmod 64)]$$

  Here $P[x, *, z]$ is parity of a column that can be calculated as :

  $$P[x, *, z] = \bigoplus_{j=0}^{4} A[x, j, z]$$

– ρ **(Rho)**: ρ is also a linear operation in which inter-slice dispersion happens. Each lane rotates bitwise in this operation by predefined offset values. These rotation offsets for each lane are distinct, as shown in the Table 2. Here column and row represent y axis and x asix values respectively.

  $$A[x, y, z] = A[x, y, z_{\lll t}] \; for \; x, y = 0, ...4$$

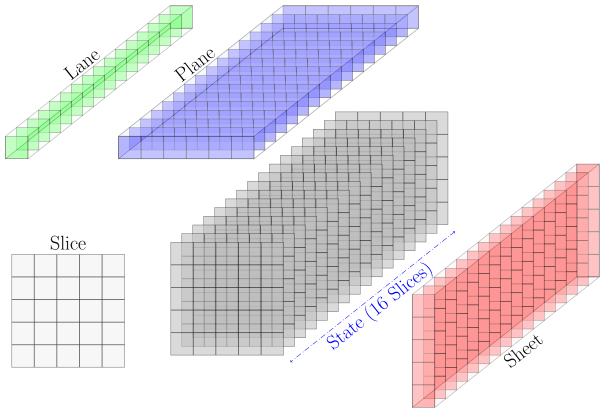


Fig. 1: Nomenclature of KECCAK internal state parts w.r.t KECCAK-p[400]

Here $\lll$ is a bitwise rotation

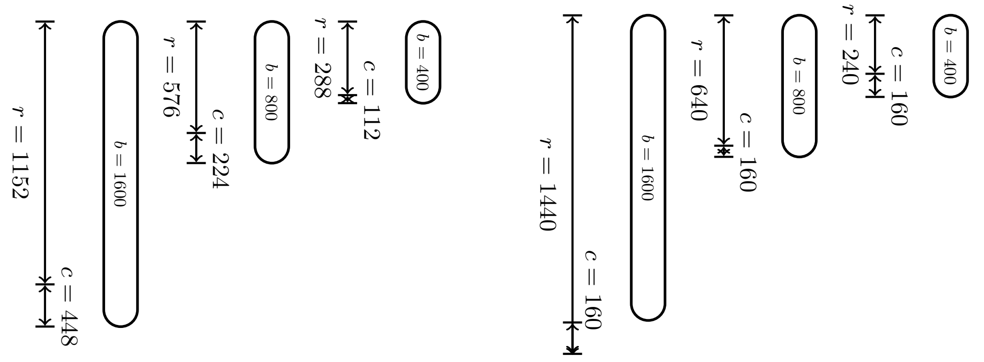
Table 2: Offset values of rotation for each lane(ρ)

4	18	2	61	56	14
3	41	45	15	21	8
2	3	10	43	25	39
1	36	44	6	55	20
0	0	1	62	28	27
y/x	0	1	2	3	4

– π **(Pi)**: This is another linear operation that breaks horizontal and vertical alignment. In this operation, the permutation happens on slices by interchanging lanes as:
$$A[y,(2x+3y)\ mod\ 5,z] = A[x,y,z] \text{ for } x,y = 0,...4, z = 0,...63$$
– χ **(Chi)**: χ is the only non-linear operation with degree two that operates on rows independently as:
$$A[x,y,z] = A[x,y,z] \oplus (\sim A[x+1,y,z]) \wedge A[x+2,y,z]$$
– ι **(iota)**: A unique RC add to lane $A[0,0]$ depend on round number.
$$A[0,0,*] = A[0,0,*] \oplus RC$$

Keccak Variants The current work focuses on state-reduced variants classifying them into `Type-I` and `Type-II` based on how the reduction affects the ratio of rate and capacity parameters of `SPONGE`. If the state-size is reduced in proportion to the standard version then we call it `Type-I`. For example `SHA3-224` uses rate (r) of 1152 and capacity (c) of 448 for KECCAK-p[1600] permutation.

When reduced to 800 bits this becomes $[r = 576, c = 224]$ while for 400 bits this becomes $[r = 288, c = 112]$. On the other had, `Type-II` variants maintain a fixed capacity ($c = 160$) for any state-size. This variant particularly fits the Crunchy contest specification where the hash-size is fixed at 160 bits.



(a) `Type-I` Variant (b) `Type-II` Variant

Fig. 2: `Keccak` state-reduced variant classification. `Type-I` variant shown here is based on `SHA3`-224

Squeeze Attack The philosophy of the squeeze attack is about achieving faster collisions in a target subset. It was mounted by Dinur *et al.* in FSE 2013 by leveraging the generalized internal differentials of the `Keccak` state. The idea is to map many `Keccak` inputs into a relatively small subset of possible outputs. The trick is to be able to do so with a surprisingly **large** probability. The success of the attack is generally premised in forcing the output to conform to some very specialized form. For a random permutation generating such inputs that map to the target subset actually lead to complexities greater than the *Birthday Bound (BB)* and are hence unusable. Herein lies the crux of the analysis which aims to exploit the non-random properties of `Keccak`-p permutation and in turn achieve an advantage over BB. In the squeeze attacks mounted by the authors in FSE 2013, they capitalized the internal symmetry of the `Keccak` state to generate faster collisions in a special target subset. In doing so the authors explored the idea of *Internal Differential Cryptanalysis* which is illustrated next.

Internal Differential Cryptanalysis of Keccak Internal differentials capture the internal difference between parts of the internal state and study the evolution of these differences across different rounds in a manner similar to a classical differential attack. The only difference is that a first order internal differential requires a single message/state while the classical counterpart requires two. The primary idea is to leverage the internal relations of a state to mount attacks and was first explored by Peyrin [18] in Crypto 2010. In the context of Keccak the idea was generalized by Dinur *et al.* in FSE 2013 to find collisions for up to 5-round. Kuila *et al.* [16] later extended it to find practical distinguishers on Keccak-p for up to 6-round. Nikolic and Jean [14] used the same idea to devise internal differential boomerang distinguishers. Further works that exploit the internal symmetry of Keccak state involve the SymSum distinguisher by Saha *et al.* [20] and its extension by Suryawanshi *et al.* [22]. All these work use translation invariance property of four out of five sub-operations of the Keccak round function which was already reported by the designers [5]. The current work is another effort in this direction that leverages the internal differential counterpart of the *Target Difference Algorithm* introduced by Dinur *et al.* in FSE 2012.

Target Difference Algorithm [7] The Target Difference Algorithm, as the name suggests, tries to generate conforming message pairs that, after one round of Keccak, deterministically produce a target difference. It takes a target difference (α_1) as input which is essentially the input difference of a differential trail and outputs the message pair (m_1, m_2) such that $R_1(m_1) \oplus R_1(m_2) = \alpha_1$. The basic idea of this algorithm is to generate a system of linear equations to capture all constraints induced due to α_1 and the static constraints due to the capacity and padding part of the input state. If that equation system has a solution, TDA outputs the message and fails otherwise, implying that there is no guarantee of getting the solution for arbitrary target differences. Thus, it is a heuristic algorithm. This algorithm has 2 phases: the *difference* phase and the *value* phase. In the difference phase, the system of equations is generated based on the state difference and its solution provides the state difference at the start of the first round. Like the difference phase, the value phase involves a set of equations based on absolute values, the solution of which gives the actual value of the state at the start of the first round.

TDA Connector [11] Guo *et al.* introduced a `TDA` connector that extends `TDA` up to 2 and 3 rounds by linearizing the S-boxes of first and second rounds respectively. Like `TDA`, the input to the connector is the target difference $\varDelta T$ against which it generates a system of equations whose solution is message pair (M_1, M_2). This pair under two rounds (R_2) of Keccak satisfies the output difference $\varDelta T$ as $\varDelta T = R_2(M_1) \oplus R_2(M_2)$. By limiting the input of the S-box to a specified affine subspace, linearization S-box's output is achieved. For connectors, linearization for S-boxes requires more constraints thereby reducing the degrees of freedom. To minimize the constraints, authors proposed non-full linearization and provided some observations highlighting the properties of solution subspace (Refer Appendix B) that facilitate this.

Target Internal Difference Algorithm [8] Dinur *et al.* [8] has given a 5-round collision attack on `SHA3-256`, in which they extend the trail using `TIDA`.

TIDA is similar to TDA, but TIDA is an approach that connects the internal differential trail (the initial internal difference of trail is at the output of the first round) to the initial state of the KECCAK. The initial internal difference of the trail is the internal target difference, and the algorithm outputs messages in which the internal difference after one round would be the internal target difference.

3 Revisiting the Target Internal Difference Algorithm

Our motivation to revisit the TIDA technique stems from our struggle while implementing it from our knowledge of the implementation of TDA. As per the authors of TIDA, its extension from TDA is straightforward and their rendition of TIDA is minimal. However, our experience shows that one needs to have a detailed account of the scheme in order to understand the nuances while implementing the same. This motivation leads to the following algorithmic depiction of our recreation of the algorithm. We have to emphasize that we are not aware if the original authors of TIDA did follow the same steps as outlined below.

Let E_{Δ} and E_M are the system of equations for differences and values. Then, the variables $\Delta_X = \{\Delta_{x_1}, \Delta_{x_2}, \ldots, \Delta_{x_{800}}\}$ and $X = \{x_1, x_2, \ldots, x_{1600}\}$ represent the state of difference and actual values after the linear layer (L). As we are taking internal differences, for standard SHA3, $|\Delta_X| = 800$ and $|X| = 1600$. The introductory procedure of TIDA is as follows.

1. Add initial equations for capacity and padding.
   (a) Add equations in E_{Δ} for capacity i.e. $L^{-1}(\Delta_x) == 0$
   (b) Add equations in E_m for capacity and padding, i.e. $L^{-1}(x) == 0$ and $L^{-1}(x) == p$ [here, we require only half the equations for the capacity part. Another half will be taken care of by equations added in E_{Δ}].
   (c) Substitute the $\Delta_{x_{i_j}}$ to zero for i^{th} in-active S-box in E_{Δ} where j is 5 bit of S-box.
   (d) Add equations to E_m for in-active S-box to define the differences between 2 halves.

Algorithm 1: Target Internal Difference Algorithm (`TIDA`)

```
   Input  : $\gamma_0$: Target Difference
   Output: 'No Solution' or $M$ such that $C_1 \oplus C_2 = \gamma_1$ where $R_1(M) = C$ and $C = C_1||C_2$
   ―――――――――――――――― Difference Phase ――――――――――――――――
1  $E_\Delta \leftarrow \{\}$                                        ▷ Empty System
2  $\Delta_X \leftarrow \{\Delta_{x_1}, \Delta_{x_2}, \cdots, \Delta_{x_{800}}\}$   ▷ { Allocating difference variables after linear layer.
                                                                       Note that state-size is halved.
                                                                       This is the first deviation from TDA
3  $exp_\Delta = L^{-1}(\Delta_X)$
4  $E_\Delta \leftarrow E_\Delta \cup (exp_\Delta(c) = 0)$              ▷ { Adding equations w.r.t capacity only
                                                                       Note padding constraints are not needed
                                                                       This is the another deviation from TDA
5  foreach $i^{th}$ inactive S-box with input difference variables $(\Delta_{x_{i_1}}, \cdots \Delta_{x_{i_5}})$ do
     $E_\Delta \leftarrow E_\Delta \cup \{\Delta_{x_{i_1}} = 0, \cdots, \Delta_{x_{i_5}} = 0\}$
6  if $E_\Delta$ inconsistent then output "Fail" and exit()
7  foreach active S-box do Initialize IDSL[1] and store it in IDSD
8  for iteration $<$ Threshold do
9      foreach active S-box in IDSD do
10         Retrieve output difference $\delta^{out}$
11         Select a 2D affine subspace ($S$) from IDSL
12         $E_\Delta \leftarrow E_\Delta \cup$ (3 Affine Equations for $S$)
13         if $E_\Delta$ inconsistent then
14             if All 2D affine subspaces are not exhausted then Goto Step 11
15             else
16                 Change the IDSD order by bringing the failed S-box at first position
17                 Goto Step 9 (Next Iteration)
18             end
19         end
20     end
       ―――――――――――――――― Value Phase ――――――――――――――――
21     $E_M \leftarrow \{\}$                                         ▷ Empty System
22     $X \leftarrow \{x_1, x_2, \cdots, x_{1600}\}$               ▷ Allocating state variables after linear layer
23     $exp_M = L^{-1}(X)$
24     $E_M \leftarrow E_M \cup \{exp_\Delta(c'||p) = 0||\mathbf{p}\}$    ▷ { Adding eqn. for half of the capacity part
                                                                       This is another deviation from TDA
25     foreach $i^{th}$ inactive S-box with input state variables $(x_{i_1}, \cdots x_{i_5})$ do
         $E_M \leftarrow E_M \cup \{(x_{i_1} \oplus x_{i_1+32} = 0), (x_{i_2} \oplus x_{i_2+32} = 0), \cdots, (x_{i_5} \oplus x_{i_5+32} = 0)\}$
26     if $E_M$ is not consistent then output "Fail" and exit()
27     foreach $i^{th}$ active S-box in IDSD do
28         Retrieve output difference $\delta_i^{out}$
29         Choose one $\delta_{i_j}^{in}$ as per affine subspace stored in $E_\Delta$
30         $E_M \leftarrow E_M \cup$ {Lin. Eqns. for affine subspace from soutions of $(\delta_{i_j}^{in}, \delta^{out})$}
31         if $E_M$ is consistent then
32             $E_M \leftarrow E_M \cup \{x_{i_j} \oplus x_{i_j \oplus 32} = \delta_{i_j}^{in}, 1 \le j \le 5\}\}$ ▷ Another deviation from TDA
33         end
34         else Goto Step 29
35     end
36 end
37 if iteration $<$ Threshold then output $M$
38 else 'No solution' and exit()
```

2. For each active S-box, add a 2-dimensional affine subspace of input differences. After adding a set of equations in E_Δ for each S-box, check the consistency of the system of equations.

[1] IDSL and IDSD are data structures defined by Dinur *et al.* [7] to store list of input differences and the order in which these lists are stored for the entire state.

3. For each active S-box, fix one of the input differences from the affine subspace and add an n-dimensional affine subspace of the solution subspace in E_m.
   (a) After adding a set of equations for each S-box, check the consistency of the system of equations.
   (b) For a consistent system of equations, add equations in E_Δ for the selected input difference

In the algorithm, we have reduced some constraints that will improve the complexity. The detailed approach for `TIDA` is given in Algorithm 1. It is interesting to note that, despite its introduction in FSE 2013, Algorithm 1 is the first detailed description of `TIDA` available in literature.

Input difference subset list (IDSL) stores the input difference subsets in a sorted manner for each of the t active S-boxes. Suppose δ_{out} is the output difference for a specific S-box, then we compare two input difference subsets $\{\delta_1, \delta_2, \delta_3, \delta_4\}$ and $\{\delta_5, \delta_6, \delta_7, \delta_8\}$ in such a way that the equations below hold.

$$DDT(\delta_1, \delta_{out}) \geq DDT(\delta_2, \delta_{out}) \geq DDT(\delta_3, \delta_{out}) \geq DDT(\delta_4, \delta_{out}) \geq 0$$
$$DDT(\delta_5, \delta_{out}) \geq DDT(\delta_6, \delta_{out}) \geq DDT(\delta_7, \delta_{out}) \geq DDT(\delta_8, \delta_{out}) \geq 0$$

We start by comparing the sizes of the largest subspace, if $DDT(\delta_1, \delta_{out}) \geq DDT(\delta_5, \delta_{out})$, we prefer the input difference subset with the larger size. If their size is the same, we compare the next two i.e. $DDT(\delta_2, \delta_{out}) \geq DDT(\delta_6, \delta_{out})$ and so on to choose accordingly. The input difference subset data structure (IDSD) contains the IDSLs. Each element in IDSL has a pointer to an input difference subset that points to the corresponding input difference subsets. Each active S-box has a single entry in the IDSD, which is then arranged by the IDSD order (which may differ from the natural order of the S-boxes). The initial IDSD order is selected randomly and shuffled during the execution of the Algorithm1.

Improving Consistency Check in `TIDA` The incremental consistency check, which we refer to as lines 9–20 and 27–35 of algorithm 1, adds some restrictions to the system of equations for each S-box and verifies their correctness continuously. Therefore, we apply the Gauss elimination approach to examine that the equation is consistent, which calls for translating elementary row operations into their Echelon form. The same fundamental row operations must be continually converted into their Echelon form when new equations are added to the system of equations. As a result, it overdue the operations. We know that the Echelon form is the same as a system of linear equations. Hence, adding new linear equations directly to the given system of equations is equivalent to adding them to the row echelon form (REF) of the given system of equations. As a result, while new operations are being added, the system of equations' past-due operations on the same equations will be reduced.

Experimental Verification: For the proof-of-concept, we use the `Type-I` variant Keccak-p[72,28] with a width of 100, rate = 72, and capacity = 28. For target internal difference $\varDelta T$ of size 50, `TIDA` returns conforming state M of size 100, which has internal difference α_0 of size 50 such that $M = m_1||m_2$ and $m_1 \oplus m_2 = \alpha_0$ as shown in Table 3.

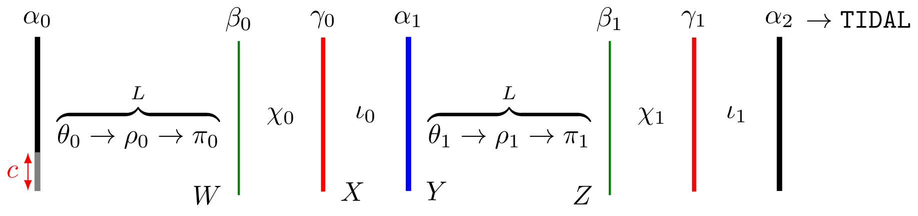
Table 3: Showing the input message M has an internal difference α_0 return by TIDA for target difference $\varDelta T$ of Keccak-p[72,28]

γ_0					$\alpha_0 = m_1 \oplus m_2$					$M = m_1\|\|m_2$				
01	00	00	01	00	00	11	01	00	00	0101	0110	1011	1111	1111
10	00	00	01	00	00	11	00	10	00	0101	0011	0000	1000	0101
10	00	00	01	00	00	00	11	00	00	1010	0000	1001	1010	0000
10	00	00	00	00	11	00	00	00	00	1001	1111	1111	0000	0000
10	00	00	01	10	00	00	00	00	00	0000	0000	0000	0000	0000

4 TIDAL: Extending TIDA using Linearization

In this section, we introduce `TIDAL` which is a mechanism to extend `TIDA` idea by one more round. The fundamental idea of `TIDAL` is to convert the two rounds of Keccak-p permutation into a system of linear equations and is inspired from Guo *et al.*'s `TDA` Connector. However, the technique needs to be adapted as we need to handle internal differences and hence also incorporate the effect of round constants. One round of Keccak-p permutation can be expressed as $\iota \circ \chi \circ \rho \circ \pi \circ \theta$, where χ is the only non-linear function. Given, a target difference $\varDelta T_i$, the solution of the linear system of equations outputted by `TIDAL` will give a message subspace $\mathcal{M}$ such that $\forall M \in \mathcal{M}, R_2(M) = C = C_1||C_2$, where $C_1 \oplus C_2 = \varDelta T_i$ and $R_2(\cdot)$ represents two rounds of Keccak-p. We can trivially generate linear equations for the four linear mappings of the Keccak round function. However, we need to modify the S-box linearization strategy given by Guo *et al.* [11] for the non-linear map χ to handle internal differences. For convenience of understanding, we denote the individual *internal* differences of the state before $L(= \rho \circ \pi \circ \theta)$, χ and ι of the i^{th} round as α_i, β_i and γ_i respectively. Thus, for i^{th} round, the state difference propagation can be visualized as $\alpha_i \xrightarrow{L} \beta_i \xrightarrow{\chi} \gamma_i \xrightarrow{\iota} \alpha_{i+1}$. The actual values of the state corresponding to state differences of β_0, γ_0, α_1 and β_1 are denoted by W, X, Y and Z respectively. Bit-level state variables are $(w_j \in W), (x_j \in X), (y_j \in Y)$ and $(z_j \in Z)$ where $(1 \leq j \leq 25b), b$ being the width of Keccak state. This entire setting used for `TIDAL` is given in the Fig. 3.

`TIDAL` strategy takes the input $\varDelta T_i = \alpha_2$, and has three parts: Main Linearization, Basic Linearization and Pre-process procedures. The Algorithmic rendition of these procedures is captured in Algorithms 2, 3 and 4 respectively. For the rest of the discussion, please refer Fig. 3. In the main procedure using $\varDelta T_i$ as input, we first calculate the state difference γ_1 in the second round `XOR`ing



α_0
β_0
γ_0
α_1
β_1
γ_1
$\alpha_2 \rightarrow$ TIDAL
L
$\theta_0 \rightarrow \rho_0 \rightarrow \pi_0$
χ_0
ι_0
L
$\theta_1 \rightarrow \rho_1 \rightarrow \pi_1$
χ_1
ι_1
c
W
X
Y
Z

Fig. 3: Initial 2-round of Keccak-p. Here α_i, β_i and γ_i represent intermediate internal difference of the state after the sub-operations of the round function.

the specific round-constant[2]. This is a deterministic step. Now, we select the output difference δ_{out} for each active S-box in γ_1 and randomly select the input difference δ_{in} from the Difference Distribution Table (DDT) of Keccak S-box such that $\mathrm{DDT}(\delta_{in}, \delta_{out}) \neq 0$. Altogether, these input differences δ_{in} will constitute the state difference β_1. From β_1 to α_1 is again a deterministic step as $\alpha_1 = \theta^{-1} \circ \pi^{-1} \circ \rho^{-1}(\beta_1)$. Then, we calculate the γ_0 by XORing the rounds constant of first round to α_1. Once we have γ_0, we can apply TIDA (Algorithm 1) to obtain the state difference β_0 and a system of equations E_M. It can be noted that here we use a slight variation of Algorithm 1 for TIDA as we need the system of equations but not the actual message. We obtain α_0 from β_0 in the same way as stated above. After retrieving the state internal differences ($\alpha_0, \beta_0, \gamma_0, \alpha_1, \beta_1$ and γ_1) and system of equations E_M for the first round, we need to add more constraints so that the new system of equations will linearize the χ layer of the first round. The claim is that the solution of this system of equations will satisfy the target internal difference (ΔT_i) after the second round.

We need to add some constraints to capture the input difference δ_{in} from γ_1 to β_1 for each active S-box in γ_1. The solution subspace generates an affine set for each possible transition $\delta_{in} \rightarrow \delta_{out}$ and ensure that this transition happens which would otherwise be probabilistic. The set of constraints for every active S-box due to the affine set mentioned above can be presented as the following system of linear equations (1). Here, $Z = \rho \circ \pi \circ \theta \circ \iota(X) = L_1 \cdot X$ (Refer Fig. 3). As the transition between γ_0 to β_1 is linear, we can re-express Equation (1) as Equation (2). We already have a system of linear equations E_M at β_0 (as retrieved from TIDA), which is represented as Equation (3).

$$B \cdot Z = t_b \qquad (1) \qquad\qquad A_1 \cdot W = t_{A_1} \qquad (3)$$
$$B \cdot L_1 \cdot X = t_b \qquad (2) \qquad\qquad A_2 \cdot W = t_{A_2} \qquad (4)$$

Here A_1 has the constraints corresponding to the affine sets between γ_0 and β_0 similar to the affine set between γ_1 and β_1 along with extra constraints as given in Step 32 of Algorithm 1. Additionally, A_2 is Equation (4) enforces the conditions

[2] Note that for algorithms involving internal difference like TIDA and TIDAL, round-constants play a vital role. However, they can be ignored for TDA.

Algorithm 2: The Main Procedure of TIDAL

```
Input  : $\alpha_2$: Target Difference after 2 rounds
Output: "Fail" or $E_M$ such that solution of $E_M = L(M)$ where
        $C_1 \oplus C_2 = \alpha_1$, $R_2(M) = C$ and $C = C_1||C_2$
1  $\gamma_1 \leftarrow \alpha_2 \oplus rc_2$                    ▷ $rc_2 \rightarrow$ round constant of second round
2  Populate $\beta_1$ by selecting compatible $\delta_{in}$ from $\gamma_1$
3  $\alpha_1 \leftarrow L^{-1}(\beta_1)$
4  $\gamma_0 \leftarrow \alpha_1 \oplus rc_1$                    ▷ $rc_1 \rightarrow$ round constant of first round
5  $\beta_0, \alpha_0$ and $E_M \leftarrow$ TIDA($\gamma_0$)                    ▷ Refer Algorithm 1
6  Obtain matrix $B$ and vector $t_b$ from $\gamma_1, \beta_1$              ▷ Refer Equation (1)
7  Compute flag variable $U$
8  while counter < Threshold do
9      Execute Basic Linearization Procedure $(E_M, \beta_0, \alpha_1, U) \leftarrow$ Algorithm 3
10     if Algorithm 3 succeeds then
11         $E_M \leftarrow E_M \cup \{B.L_1.(L_{\chi_0}.W + t_{L_{\chi_0}}) = t_b\}$
12         if Eqn is consistent then return $E_M$
13 return "Fail"
```

for capacity and padding part. It is worth noting that we require conditions for only **half** of the capacity bits as stated earlier in Step 24 of Algorithm 1. Our final aim is to express the complete system of linear equations in terms of the state variable W. Equation (4) and (3) conform to that. To restate Equation (2) in terms of W, we need to linearize the χ layer of first round. It is interesting to note that it is sufficient to linearize only those variables in X which appear in Equation (2). Thus partial linearization is sufficient for restating Equation (2) in terms of W as already observed by Guo *et al.* [11]. For partial linearization, first, we must find those bits of W that participate in Equation (1). We modify the technique devised by Guo *et al.* to take into account the relations induced between the two halves of the state.

We use a flag $U = (U_0, U_1, \ldots, U_{b/2*(5-1)})$ where $U_i = \{U_{(i,1)}, U_{(i,2)}, \ldots, U_{(i,5)}\}$, when $U_{(i,j)} = 1$, which means at least one of $(x_i, x_{i+25*b/2})$ participates in Equation (1). This step is a *deviation from TDA Connector* as we to keep track of the bit $x_{i+25*b/2}$ at the symmetric position corresponding to bit x_i. To compute the value of $U_{(i,j)}$, we need to look at the coefficients of x_i and $x_{i+25*b/2}$ in Equation (2). If any of them has a coefficient value of 1, then we set $U_{(i,j)} = 1$ otherwise $U_{(i,j)} = 0$. Depending on the value of $U_{(i,j)}$, partial linearization will happen due to Observation 3 (Refer Appendix B).

The rest of procedure follows TDA Connector strategy and is restated below for the sake of completeness.

When $U_i = 11111$, we require linearization of all 5 bits of i^{th} S-box.

- Case 1: If DDT value is 2 or 4 by Observation 2, the solution subspace is already linear.
- Case 2: If DDT value is 8 by Observation 2 we need to randomly select one of the six linearizable affine subspaces.

Algorithm 3: The Basic Linearization Procedure of `TIDAL`

```
input  : $E_M, \beta_0, \alpha_1, U$
output: $E_M, (L_{\chi 0}, t_{L_{\chi 0}})$: matrix and vector for linearizing $\chi_0$
1  $l_{sb}, L_{\chi 0}, T_{L_{\chi 0}} \leftarrow$ Pre-Process Procedure $(E_M, \beta_0, \alpha_1, U)$    ▷ Refer Algorithm 4
2  foreach S-box in $l_{sb}$ do
3      Initialize an empty list $l_{lin}$ of set of equations
4      if S-box is active then Add corresponding LAS equations to $l_{lin}$
5                                        ▷ LAS ← Linearizable Affine Subspaces
6      else Add equations to $l_{lin}$ according to $U$
7      while for untested set in $l_{lin}$ do
8          Randomly choose an untested set of equations
9          if set of equations are consistent with $E_M$ then
10             Add them to $E_M$
11             Update $(L_{\chi 0}, t_{L_{\chi 0}})$ for that S-box
12             Update $(L_{\chi 0}, t_{L_{\chi 0}})$ for other half of state      ▷ Deviation from TDA
                 Connector
13         end
14         else if all sets have been tested then
15             return "Fail"
16         end
17     end
18 end
19 return $E_M, (L_{\chi 0}, t_{L_{\chi 0}})$
```

Algorithm 4: The Pre-process Procedure of `TIDAL`

```
input  : $\beta_0, \alpha_1, U$
output: $\begin{cases} l_{sb} : \text{List of S-Box,} \\ (L_{\chi 0}, t_{L_{\chi 0}}) : \text{matrix and vector for linearizing } \chi_0 \end{cases}$
1  Initialize an empty list $l_{sb}$, matrix $L_{\chi 0}$, and vector $t_{L_{\chi 0}}$
2  foreach S-box do
3      from $\beta_0, \alpha_1$ obtain $\delta_{in}, \delta_{out}$ respectively
4      if $U_i > 0$ then
5          if $DDT(\delta_{in}, \delta_{out}) = 2$ or $4$ then
6              Update $(L_{\chi 0}, t_{L_{\chi 0}})$ for that S-box
7              Update $(L_{\chi 0}, t_{L_{\chi 0}})$ for other half of state      ▷ Deviation from TDA
                 Connector
8          end
9          else if $DDT(\delta_{in}, \delta_{out}) = 8$ then add S-box to $l_{sb}$
10         else if S-box inactive then add S-box to $l_{sb}$
11     end
12 end
```

- Case 3: If DDT value is 32 by Observation 1 we need to randomly select one of the 80 linearizable affine subspaces.

When $U_i \neq 11111$, we require partial linearization of i^{th} S-box.

- Case 1: If DDT value is 2 or 4 by Observation 2, the solution subspace is already linear.
- Case 2: If the DDT value is 8, one of 5 has one of the five-bit will be non-linear. Suppose that for i^{th} S-box j^{th} bit is non-linear, then if $U_{(i,j)} = 0$, we do not require linearization; otherwise, we randomly select one of the six linearizable affine subspaces.
- Case 3: If DDT value is 32 by Observation 3, we require partial linearization, depending on how many bits are required for linearization.

We add new equations to E_M for inactive S-boxes and active S-boxes with DDT value 8 as Equation (5). After linearizing the S-box, it can expressed as Equation (6). By substituting the value of X in Equation (6), we get Equation (7).

$$A_3 \cdot W = t_{A_3} \tag{5}$$

$$L_{\chi_0} \cdot W + t_{L_{\chi_0}} = X \tag{6}$$

$$B \cdot L_1 \cdot (L_{\chi_0} \cdot W + t_{L_{\chi_0}}) = t_b \tag{7}$$

We add Equations (7) and (5) to E_M (Recall that E_M already has Equations (3) and (4)), and the solutions of E_m should satisfy $\varDelta T_i$ after two rounds of Keccak. TIDAL provides a way to produce message-subspaces that can deterministically lead to a given target internal difference after two rounds. In the next subsection we discuss the degrees of freedom of TIDAL. Subsequently we show how we can combine TIDAL with squeeze attacks to produce practical collisions on state-reduced Keccak up to 5 rounds.

4.1 Degrees of Freedom for TIDAL

We can only control $r-2$ bits of Keccak state by taking 2-bit fixed padding (i.e., 11 in binary). Thus the maximum number of messages we can generate is 2^{r-2}. The required number of random messages to achieve the desired internal difference of $\varDelta$ is $2^{-|\varDelta|}$. By the notion of the internal difference, $|\varDelta|$ is equal to half the size of the state. Thus, for standard SHA3, $|\varDelta|$ is 800. Suppose the size of the state-reduced Keccak variant is S, then $|\varDelta| = \frac{S}{2}$. The maximum number of messages that satisfy the specified target difference is $2^{r-2-\frac{S}{2}}$ and hence the degrees of freedom are $2^{26}, 2^{50}, 2^{54}$ and 2^{102} for state-reduced variants Keccak-p[128, 72], Keccak-p[152, 48], Keccak-p[526, 144] and Keccak-p[304, 96], respectively. The dimension $\mathcal{D}$ of the solution-space of TIDAL depends on the number of equations in E_M corresponding to capacity, padding, active and inactive S-boxes. We can express the dimension of the solution-space as below.

$$\mathcal{D} = \sum_{i=1}^{s} \mathcal{D}_{1,i} - \left(\frac{c}{2} + p + \sum_{j=1}^{s} \mathcal{E}_{2,j} \right)$$

Here, c and p denote constraints of initial state for capacity, and padding, respectively. $\mathcal{D}_{1,i}$ represents the dimension of the solution-space of the i^{th} S-box of the first round and $\mathcal{E}_{2,j}$ denotes the equations corresponding to the j^{th} S-box of the second round. For the first and second rounds, the total dimensions is given by $\sum_{i=1}^{s} \mathcal{D}_{1,i}$ and the total number of equations is given by $\sum_{i=1}^{s} \mathcal{E}_{2,j}$ over the all s S-boxes in the internal difference. The values of $\mathcal{D}_{1,i}$ and $\mathcal{E}_{2,j}$ are given below.

$$\mathcal{D}_{1,i} = \begin{cases} 1, & \text{DDT}(\delta_{in}, \delta_{out}) = 2 \\ 2, & \text{DDT}(\delta_{in}, \delta_{out}) = 4 \\ 2 \sim 3, & \text{DDT}(\delta_{in}, \delta_{out}) = 8 \\ 2 \sim 5, & \text{DDT}(\delta_{in}, \delta_{out}) = 32^{\dagger} \end{cases} \qquad \mathcal{E}_{2,j} = \begin{cases} 4, & \text{DDT}(\delta_{in}, \delta_{out}) = 2 \\ 3, & \text{DDT}(\delta_{in}, \delta_{out}) = 4 \\ 2, & \text{DDT}(\delta_{in}, \delta_{out}) = 8 \\ 0, & \text{DDT}(\delta_{in}, \delta_{out}) = 32^{\dagger} \end{cases}$$

Here, δ_{in} and δ_{out} represents the input and output differences of each S-box. As a result of partial linearization of first round, dimension of S-box deviates, which is better than complete linearization increases and is thus an improvement over complete linearization.

4.2 Experimental Verification

For an arbitary target internal difference α_2, `TIDAL` returns a message subspace $\mathcal{M}$ such that $\forall M \in \mathcal{M}, M = m_1 || m_2$ and $m_1 \oplus m_1 = \alpha_0$ which satisfies α_2 after 2-round. In our experiment, we have taken α_2 as zero state (signifying zero internal difference) for a state-reduced variant Keccak-p[72,28], with a width of 4 (state size = 100), rate = 72, and capacity = 28. Table 4 shows a message satisfying α_2 as a zero state after 2-round.

Table 4: Input message M with internal difference α_0 generated by `TIDA` for target difference α_2 for Keccak-p[72,28]

α_2					$\alpha_0 = m_1 \oplus m_2$					$M = m_1 \| m_2$				
00	00	00	00	00	00	11	10	11	01	0000	1100	1000	1001	1011
00	00	00	00	00	10	10	00	10	01	1000	1000	0101	0010	1110
00	00	00	00	00	01	10	00	11	10	1011	1000	1010	0011	0111
00	00	00	00	00	01	00	00	00	00	0100	1111	1111	0000	0000
00	00	00	00	00	00	00	00	00	00	0000	0000	0000	0000	0000

5 Finding Collisions using TIDAL

This section summarises our attacks, which have three stages, as depicted in Fig. 4. The first stage executes the `TIDAL` connector with *zero* target internal difference, which *extends* the trails for 2-round backward. The second stage has

[†]For inactive S-box

deterministic trail that starts with a symmetric state (zero internal difference). The third stage uses the squeeze attack technique developed by Dinur *et al.* to cover forward 1.5 rounds and relies on the facts that χ has low diffusion and needs the allocation of 2 variables for each bit-difference in the output difference of the deterministic trail. The χ operation maps 5 bits of input of a row to itself. The $n-$bit output digest requires $n_p = \lceil \frac{n}{5 \cdot l_s} \rceil$ number of planes where l_s denotes the lane-size. In terms of the internal difference, each plane has $n_s = \frac{l_s}{2}$ number of S-boxes each of which can have 2^5 possible values. Therefore, an n-bit hash can actually have $2^{5 \cdot n_s \cdot n_p}$ possible values instead of 2^n. Suppose the hamming weight of the output difference of the deterministic trail is hw. As stated earlier, this would imply $2 \cdot hw$ variable allocations. Then in this *squeezed* hash-space by Birthday Paradox, the number of messages required to find a collision is $2^{(5 \cdot n_s \cdot n_p + 2 \cdot hw)/2}$. The generic collision complexity being $2^{\frac{n}{2}}$, we have a collision attack whenever $(2^{(5 \cdot n_s \cdot n_p + 2 \cdot hw)/2} < 2^{n/2})$.

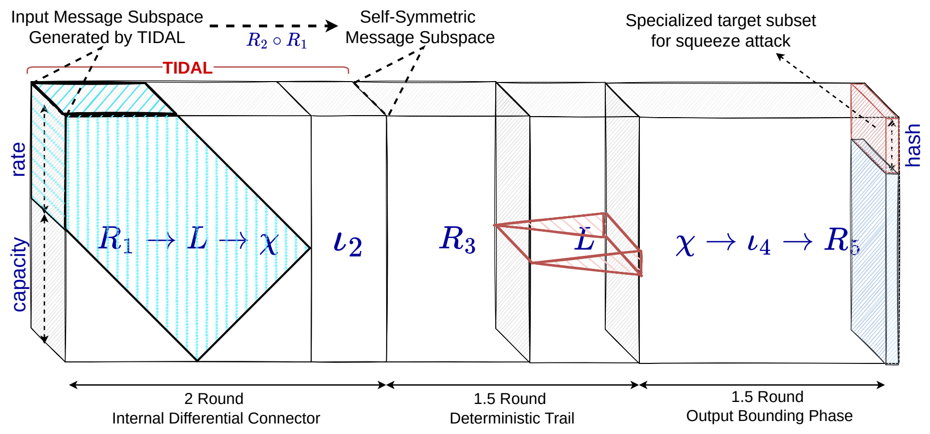


Fig. 4: `TIDAL` Flow for 5-Round Collision Search

The complete procedure for collision search is provided in Algorithm 5, which runs `TIDAL` and generates the required messages according to collision complexity. For this attack `TIDAL` outputs a message subspace on which when we apply Keccak for 2-round, and *it always gives a symmetric state.* This is the crux of all attacks that follow. Due to `TIDAL`, we are able to augment the squeeze attack strategy shown by Dinur *et al.* to cover extra rounds. After generating sufficient messages, run Keccak for pre-determined number of rounds and search for collision.

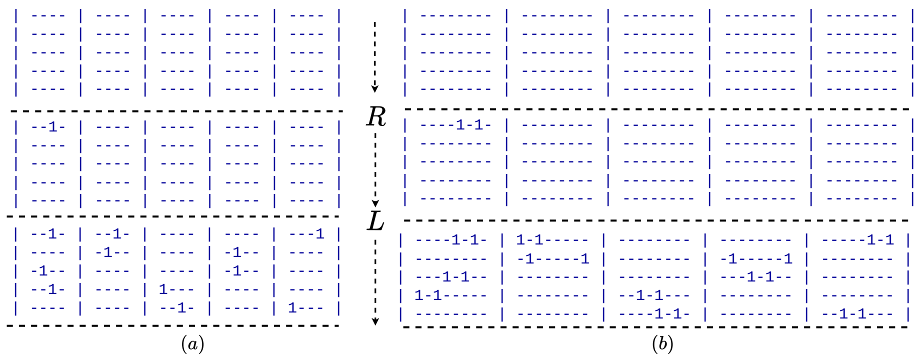


Fig. 5: Deterministic trail for state-reduce KECCAK variants with state size 200 and 400 starting from round 3

Algorithm 5: TIDAL Collision Search

```
Input  : { S ← State-size C ← Capacity H ← hash-length
         { R_n ← #Collision-Rounds R_S ← Starting round
Output: “’Fail” or Colliding Message Pair
1  Initialize, $\alpha_2 \leftarrow$ All zero state        ▷ Target zero internal difference
2  Compute collision complexity $\mathcal{C}$              ▷ Size of Squeezed hash-space
3  Initialize message space $\mathcal{M} \leftarrow \{\}$
4  Execute TIDAL ($\alpha_2$)   ▷ Refer Algorithm 2, Appropriate round constants
     must be picked from Table 5
5  if Algorithm 2 “Fail” then return “Fail”
6  else
7      foreach $m \in \mathcal{M}$ do
8          Run KECCAK[$S$] for $R_n$ rounds starting from $R_s$
9          Stores all hashes in a list $\mathcal{H}_l$
10     end
11     Search for collision in $\mathcal{H}_l$
12     return Colliding message pairs
13 end
```

5.1 Using Null-Space for Generating Conforming Message Subspace

The TIDAL strategy need not be invoked for individually generating conforming input messages. We can invoke it once to retrieve the entire message subspace of conforming messages as per the dimension of E_M using a well-known strategy. To do so we use the basis vector of the null-space, which corresponds to the vector set for a homogeneous system of equations $A \cdot x = 0$. For instance, if x_1 and x_2 are two solutions to the system of equations $A \cdot x = b$, then we have $A \cdot x_1 = b$ and $A \cdot x_2 = b$. Thus $A \cdot x_1 - A \cdot x_2 = b - b = 0$, is in the null-space.

As a result, given a single solution x_s and a null basis vector x_b, we can produce all solutions in the message subspace. The same idea allows us to generate all solutions from the message subspace of E_M using one solution returned by `TIDAL` and the null-space basis vectors.

5.2 A note on Round Constants

The above-explained collision search algorithm can apply to any consecutive rounds starting from any round. Here we are stressing on the starting round because of the impact of the round constant, which we can see in the Table 5. We can see that for different widths of KECCAK, the hamming weight of internal difference after ι is different. Table 5 shows some of the rounds. For all 24 rounds see Table 8 in Appendix C. In some cases, ι does not affect states of specific size and rounds. For example, for state sizes of 32, 16 and 8, the round constant of round $4, 22$ and 4 respectively does not affect the internal state difference after ι operation for those rounds. Thus, we start our trail according to the round constants internal difference hamming weight. It is possible to reduce the complexity of finding collisions or increase the number of rounds with the same complexity by changing the starting round.

Table 5: This table shows some of the round constant for state reduced KECCAK along with the hamming weight of internal difference due to round constant for different state size. Here, $L_s(n)$ and n_r represent size of lane is n and round

n_r	Round Constant									
	$L_s(64)$	HW	$L_s(32)$	HW	$L_s(16)$	HW	$L_s(8)$	HW	$L_s(4)$	HW
3	0x800000000000808a	5	0x0000808a	4	0x808a	2	0x8a	1	0xa	0
4	0x8000000080008000	1	0x80008000	0	0x8000	1	0x00	0	0x0	0
22	0x8000000000008080	3	0x00008080	2	0x8080	0	0x80	1	0x0	0

5.3 Experimental Verification

We use a trail with probability one to find a 4-round collision on state-reduce KECCAK. For KECCAK-p[336,64], the rate is 336, thus degree of freedom is $2^{r-2-200} = 2^{134}$. Since we require one plane for output digest, thus, the output space is of size 2^{5*8*1}, and the probability of trail is 1. Thus we need to try for 2^{20} messages. Moreover, after applying the `TIDAL`, the dimension of the solution space is 2^{38}, which is sufficient to choose random 2^{20} messages from the message space given by `TIDAL`. We have verified the above argument and got collision with complexity 2^{20} given in Table 6. Interestingly, we can find a collision for states of size 100 up to 6 rounds with same complexity i.e. 2^5 because of the round constant of the third and fourth rounds. It is happening because the internal difference for the third and fourth round constant is zero, as depicted in the Table 5. We found collision on KECCAK[84,16] for 6-round given in the Appendix A.

Table 6: 4-round outer collision for KECCAK-p[336,64] with hash `8FA5 0BAB B5B5 2E25 303F`

	CDD9	A2F3	B8D8	D74F	5EC6		6E7A	2E7F	E888	1F87	9A02
	DAAE	BEAE	0942	0408	E6DC		E89C	2131	E9A2	A4A8	3C06
M_1	31B4	D390	70E8	ECE3	5152	M_2	33B6	1F5C	CC54	2A25	E8EB
	78A0	366A	80FA	E672	E9E8		37EF	FFA3	2258	33A7	7978
	C07E	0000	0000	0000	0000		CD73	0000	0000	0000	0000

Table 7: 4-round inner collision for KECCAK-p[334,64]

	BDA9	BFEE	C8A8	D941	2FB7		3226	BEEF	6B0B	57CF	E37B
	E692	2030	A9E2	868A	BF85		FC88	B1A1	C883	0C00	6C56
M_1	BD38	4605	029A	ECE3	2221	M_2	2CA9	C586	37AF	3E31	B9BA
	E139	376B	443E	B125	8081		22FA	B7EB	116B	3EAA	3031
	D866	0000	0000	0000	0000		DB65	0000	0000	0000	0000
	B63C	11B1	0104	E4EA	8007		153F	3191	ABAF	4E49	389D
	FAFB	F5F4	8383	F1B0	B9B8		5D5D	A9A9	2160	FCBD	4141
Out_1	3226	FCF8	1703	B2A6	3024	Out_2	8591	F7E7	EFEF	C4C4	D8CC
	FE76	C6C6	EFC7	D2D2	F171		43CB	F7DF	0C24	B616	0606
	F9FB	545C	212B	2921	7C54		ADAF	545C	212B	2921	7C54

5.4 A note on Inner Vs Outer Collisions

The collision occurs at the rate part of the state in the outer collision, whereas in an inner collision, the collision occurs in the inner part of the state (capacity part of the state). We can get state collision after getting the inner collision by choosing the next input block of the message pair. Our attack focuses on state-reduced KECCAK where the length of the hash is equal to the capacity (n=c). In our attack with 1 probability trail, the output space we require to find the inner collision will be $2^{(5\cdot n_s \cdot n_p + 2\cdot hw)/2}$, by Birthday Paradox, where n_p is the number of planes needed for c bit, n_s is the total number of slices in internal difference and hw is the hamming weight of the state difference at the end of trail. Thus, our attack can apply to inner and outer collisions with the same complexity for all state reduced KECCAK variants. The is the example of inner collision with complexity 2^{20} for KECCAK-p[334,64] is given in Table 7.

6 Comparative Analysis

The `TIDAL` strategy allows us to find new collision attacks on state-reduced variants. For an existing trail, `TIDAL` is always better than `TIDA` whereas in the absence of a probabilistic trail, `TIDAL` can work with zero or low hamming weight deterministic trails which is not applicable for `TIDA`. Thus `TIDAL` can be seen as an improvement over `TIDA`, not only from the perspective of penetrating more rounds but also in terms of compatibility with deterministic trails. The same applies while comparing `TIDAL` with the `TDA` Connector strategy. While `TDA` Connector requires extensive GPU based trail search [11], `TIDAL` can work with deterministic trails albeit with higher (but still practical) complexities.

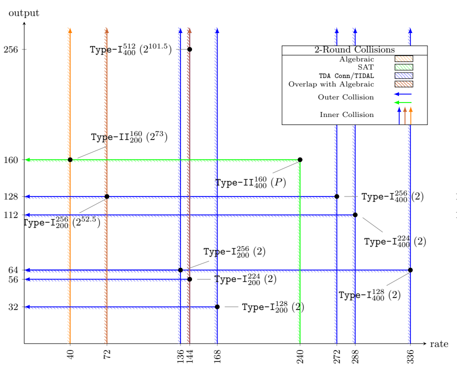
On the state-reduced KECCAK, numerous attempts have been made to find a collision. We demonstrate many approaches that can be used to find collisions on

TYPE-I and TYPE-II variants and highlight the best result among them. Fig. 6a, 6b, 6c and 6d show a summary of the best results of the 2, 3, 4, and 5-round collision attacks on states with sizes 200 and 400. The figures have x-axis for the rate and y-axis for the output size. Each dot in the above mentioned figures show a collision found for an offered rate and output size. The vertical arrows indicate an inner collision that can trivially transform into an outer collision. Similarly, horizontal arrows parallel to the x-axis represent outer collisions. In this case, the state-reduced version of SHA3 $-i$ with state size j has collision complexity k and is represented by the $\texttt{Type-x}_j^i(k)$ where Type-x can be either TYPE-I or TYPE-II. Blue indicates where TDA-connector and TIDAL can apply. All the results with complexity (2) are given by TDA-connector, whereas TIDAL yields the best results for the remaining. Although TDA-connector and TIDAL have the same applicability, TIDAL works best for higher rounds in absence of trails. In Fig.6a, the orange line indicates the attack done by Boissier et al. [6], which uses the combination of linearization and algebraic techniques for collision attack. This is an inner collision attack with complexity $2^{73}, 2^{52.5}$ and $2^{101.5}$ for Keccak-p[40,160], Keccak-p[72,128] and Keccak-p[144,256], respectively. Green line indicates the result given by Pawel Morawiecki, which uses a SAT solver to attack the TYPE-II variant Keccak-p[160,240] in practical time. Lastly, the brown line shows that Algebraic, TDA-connector, and TIDAL are all applicable for some variants and the best result is also shown among them. To understand it better, let us take the entry of Fig. 6a, $\texttt{Type-II}_{400}^{160}(P)$ shows that a collision can be found in practical time (P) on state-reduced variants (TYPE-II) with state size 400. There is only a horizontal arrow, indicating that only outer collision exists for this variant. Similarly, in Fig. 6a, $\texttt{Type-I}_{200}^{256}(2)$ illustrates that we can find collision with complexity (2) on the state-reduced variant of SHA3-256 (TYPE-I) with state size 200, i.e., Keccak-p[144,56]. There are vertical and horizontal arrows for this variant, which means both inner and outer collisions exist for this variant. In same manner all entries of the figures can be interpreted.

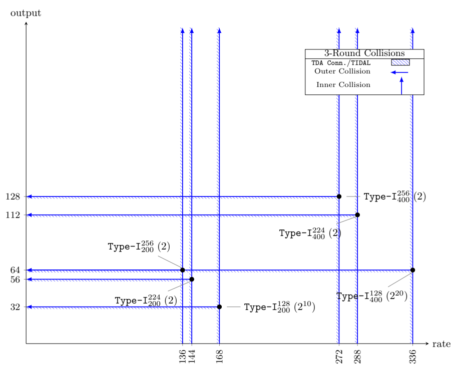
In Fig. 6c, green line indicates a collision attack by Stefan *et al.* [15], which applies to one of the crunchy contest variants, Keccak-p[160,240], for 4-rounds. It uses a standard differential attack with a 4-round differential trail. Although the TDA-connector applies to the variants as shown in Fig.6c, because of the lack of trails for the TDA-connector, the best result so far is given by the collision attack using TIDAL. This attack uses a 0.5-round trail for a 4-round collision. In Fig. 6b and 6d, we have not found any other collision attack on state-reduced Keccak. In Fig. 6b, the TDA connector with one round trail obtained the best result with a complexity of 2. Nevertheless, we cannot find all variants' TDA connector trails. Thus, for those, TIDAL provides the best results. Similarly, for 5-round, TIDAL has the best results using 1.5-round trails with a probability of 1.

7 Conclusion

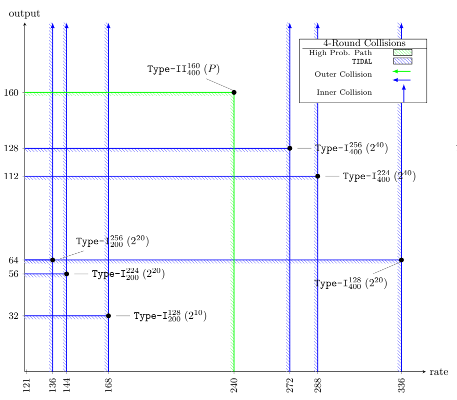
In this work, we systematically investigated the collision attack space for state-reduced Keccak variants. The variants have been classified as Type-I and Type-II based on the way the ratio of rate and capacity is reduced when defining the



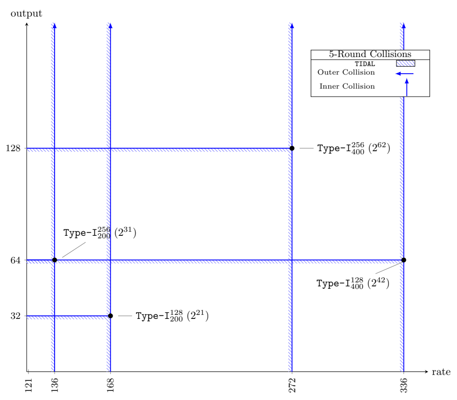
(a) 2-Round Collision Attack Space



(b) 3-Round Collision Attack Space



(c) 4-Round Collision Attack Space



(d) 5-Round Collision Attack Space

Fig. 6: Collision attack space overview for $2 \sim 5$ Rounds. It can be noted that `TIDAL` is able to cover many of the `Type-I` variants

state reduction. A new strategy `TIDAL` has been proposed that can produce states that after 2 rounds of Keccak-p lead to a given target difference. Using this, self-symmetric states are produced that form the input difference to a deterministic internal differential trail. This combination leads to an extension of squeeze attacks proposed by Dinur *et al.* in FSE 2013, and this work explores their applicability on state-reduced variants for the first time. Practical collisions are observed for many variants, which have been verified by simulating in MATLAB. Comparative analysis is depicted, showcasing the new strategy's power in penetrating higher rounds for `Type-I` variants. Existing works by Boissier *et al.* and Kölbl *et al.* apply to a few state-reduced variants but leave a gap for various others. This work addresses filling the void of the long-standing empty space for collisions on state-reduced variants, which had been previously deemed difficult. This work also improved the `TIDA` proposed by Dinur *et al.*, which can be employed on the TDA approach, where it reduces the additional work done while finding a consistent system that, as a result, gives the required message space.

Acknowledgment

Leading provider of IT services, consulting, and business solutions Tata Consultancy Services (TCS) is a supporter of the research. The first author equips financial support through the TCS Research Scholarship Program (TCS RSP).

References

1. CAESAR: competition for authenticated encryption: security, applicability, and robustness (2014), http://competitions.cr.yp.to/caesar.html
2. NIST Lightweight cryptography project (2015), https://csrc.nist.gov/Projects/lightweight-cryptography/email-list
3. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak SHA-3 submission in NIST. Submission to NIST (Round 3) (2011), http://keccak.noekeon.org/Keccak-submission-3.pdf
4. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V., Keer, R.V.: The Ketje authenticated encryption scheme (2016), https://keccak.team/ketje.html.
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The keccak reference. Submission to NIST (Round 2) **3**(30), 320–337 (2011)
6. Boissier, R.H., Noûs, C., Rotella, Y.: Algebraic collision attacks on keccak. IACR Trans. Symmetric Cryptol. **2021**(1), 239–268 (2021). https://doi.org/10.46586/tosc.v2021.i1.239-268, https://doi.org/10.46586/tosc.v2021.i1.239-268
7. Dinur, I., Dunkelman, O., Shamir, A.: New attacks on keccak-224 and keccak-256. In: 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, pp. 442–461. Springer (2012). https://doi.org/10.1007/978-3-642-34047-5_25, https://doi.org/10.1007/978-3-642-34047-5_25
8. 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). https://doi.org/10.1007/978-3-662-43933-3_12, https://doi.org/10.1007/978-3-662-43933-3_12
9. Dobraunig, C., Eichlseder, M., Mangard, S., Mendel, F., Mennink, B., Primas, R., Unterluggauer, T.: Isap v2. 0 (2020), https://isap.iaik.tugraz.at/
10. Dobraunig, C., Mennink, B.: Elephant v1. (2019)
11. Guo, J., Liao, G., Liu, G., Liu, M., Qiao, K., Song, L.: Practical collision attacks against round-reduced SHA-3. J. Cryptol. **33**(1), 228–270 (2020). https://doi.org/10.1007/s00145-019-09313-3, https://doi.org/10.1007/s00145-019-09313-3
12. Guo, J., Liu, G., Song, L., Tu, Y.: Exploring SAT for cryptanalysis: (quantum) collision attacks against 6-round SHA-3. IACR Cryptol. ePrint Arch. p. 184 (2022), https://eprint.iacr.org/2022/184

13. Huang, S., Ben-Yehuda, O.A., Dunkelman, O., Maximov, A.: Finding collisions against 4-round SHA3-384 in practical time. IACR Cryptol. ePrint Arch. p. 194 (2022), `https://eprint.iacr.org/2022/194`
14. Jean, J., Nikolic, I.: Internal differential boomerangs: Practical analysis of the round-reduced keccak- f f permutation. In: FSE. Lecture Notes in Computer Science, vol. 9054, pp. 537–556. Springer (2015)
15. Kölbl, S., Mendel, F., Nad, T., Schläffer, M.: Differential cryptanalysis of keccak variants. In: Stam, M. (ed.) Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8308, pp. 141–157. Springer (2013). `https://doi.org/10.1007/978-3-642-45239-0_9`, `https://doi.org/10.1007/978-3-642-45239-0_9`
16. Kuila, S., Saha, D., Pal, M., Chowdhury, D.R.: Practical distinguishers against 6-round keccak-f exploiting self-symmetry. In: Pointcheval, D., Vergnaud, D. (eds.) Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8469, pp. 88–108. Springer (2014). `https://doi.org/10.1007/978-3-319-06734-6_6`, `https://doi.org/10.1007/978-3-319-06734-6_6`
17. 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). `https://doi.org/10.1007/978-3-642-25578-6_18`, `https://doi.org/10.1007/978-3-642-25578-6_18`
18. Peyrin, T.: Improved differential attacks for ECHO and grøstl. In: Rabin, T. (ed.) Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6223, pp. 370–392. Springer (2010). `https://doi.org/10.1007/978-3-642-14623-7_20`, `https://doi.org/10.1007/978-3-642-14623-7_20`
19. 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 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10212, pp. 216–243 (2017). `https://doi.org/10.1007/978-3-319-56617-7_8`, `https://doi.org/10.1007/978-3-319-56617-7_8`
20. Saha, D., Kuila, S., Chowdhury, D.R.: Symsum: Symmetric-sum distinguishers against round reduced SHA3. IACR Trans. Symmetric Cryptol. **2017**(1), 240–258 (2017)
21. Song, L., Liao, G., Guo, J.: Non-full sbox linearization: Applications to collision attacks on round-reduced keccak. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 428–451. Springer (2017). `https://doi.org/10.1007/978-3-319-63715-0_15`, `https://doi.org/10.1007/978-3-319-63715-0_15`
22. Suryawanshi, S., Saha, D., Sachan, S.: New results on the symsum distinguisher on round-reduced SHA3. In: AFRICACRYPT. Lecture Notes in Computer Science, vol. 12174, pp. 132–151. Springer (2020)

A Collision on 6-round

We found a collision for states of size 100 up to 6-round with same complexity with 4-round because of round-constant of the third and fourth rounds, the conforming input states and hash is given below. Hash : `0 E 2 4 2`

6-round collision with hash 0 E 2 4 2			
M_1	5 D D 1 2 B 2 0 0 0 9 F 6 D 6 E 9 6 0 8 F 0 0 0 0	M_2	5 8 7 1 2 E D 0 5 5 9 0 9 2 6 E 3 6 5 7 F 0 0 0 0

B The observations that help in S-box linearization [11]

Observation 1 *[11] Out of the entire 5-dimensional input space,*

1. *there are totally 80 2-dimensional linearizable affine subspaces.*
2. *there does not exist any linearizable affine subspace with dimension 3 or more.*

Observation 2 *[11] Given a 5-bit input difference δ_{in} and a 5-bit output difference δ_{out} such that $DDT(\delta_{in}, \delta_{out}) \neq 0$, i.e., the solution set $V = \{x : S(x) + S(x + \delta_{in}) = \delta_{out}\}$ is not empty, we have*

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

Observation 3 *[11] For a non-active Keccak S-box, when U_i is not 11111,*

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

C Effect on hamming weight of round constants

Table 8: This table shows all round constant for state reduced Keccak along with the hamming weight of internal difference due to round constant for different state size. Here, $L_s(n)$ and n_r represent size of lane is n and round

n_r	Round Constant									
	$L_s(64)$	HW	$L_s(32)$	HW	$L_s(16)$	HW	$L_s(8)$	HW	$L_s(4)$	HW
1	0x0000000000000001	1	0x00000001	1	0x0001	1	0x01	1	0x1	1
2	0x0000000000008082	3	0x00008082	3	0x8082	1	0x82	2	0x2	1
3	0x800000000000808a	5	0x0000808a	4	0x808a	2	0x8a	1	0xa	0
4	0x8000000080008000	1	0x80008000	0	0x8000	1	0x00	0	0x0	0
5	0x000000000000808b	5	0x0000808b	5	0x808b	3	0x8b	2	0xb	1
6	0x0000000080000001	2	0x80000001	2	0x0001	1	0x01	1	0x1	1
7	0x8000000080008081	3	0x80008081	2	0x8081	1	0x81	2	0x1	1
8	0x8000000000008009	4	0x00008009	3	0x8009	3	0x09	2	0x9	2
9	0x000000000000008a	3	0x0000008a	3	0x008a	3	0x8a	1	0xa	0
10	0x0000000000000088	2	0x00000088	2	0x0088	2	0x88	0	0x8	1
11	0x0000000080008009	4	0x80008009	2	0x8009	3	0x09	2	0x9	2
12	0x000000008000000a	3	0x8000000a	3	0x000a	2	0x0a	2	0xa	0
13	0x000000008000808b	6	0x8000808b	4	0x808b	3	0x8b	2	0xb	1
14	0x800000000000008b	5	0x0000008b	4	0x008b	4	0x8b	2	0xb	1
15	0x8000000000008089	5	0x00008089	4	0x8089	2	0x89	1	0x9	2
16	0x8000000000008003	4	0x00008003	3	0x8003	3	0x03	2	0x3	2
17	0x8000000000008002	3	0x00008002	2	0x8002	2	0x02	1	0x2	1
18	0x8000000000000080	2	0x00000080	1	0x0080	1	0x80	1	0x0	0
19	0x000000000000800a	3	0x0000800a	3	0x800a	3	0x0a	2	0xa	0
20	0x800000008000000a	2	0x8000000a	3	0x000a	2	0x0a	2	0xa	0
21	0x8000000080008081	3	0x80008081	2	0x8081	1	0x81	2	0x1	1
22	0x8000000000008080	3	0x00008080	2	0x8080	0	0x80	1	0x0	0
23	0x0000000080000001	2	0x80000001	2	0x0001	1	0x01	1	0x1	1
24	0x8000000080008008	2	0x80008008	1	0x8008	2	0x08	1	0x8	1