IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2019, No. 4, pp. 407–437. DOI:10.13154/tosc.v2019.i4.407-437

New Techniques for Searching Differential Trails in Keccak

Guozhen Liu[2,1], Weidong Qiu[2] and Yi Tu[1]

[1] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore, tuyi0002@e.ntu.edu.sg
[2] School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, liuguozhen@sjtu.edu.cn,qiuwd@sjtu.edu.cn

Abstract. Keccak-f is the permutation used in the NIST SHA-3 hash function standard. Inspired by the previous exhaustive differential trail search methods by Mella *et al.* at ToSC 2017, we introduce in this paper new algorithms to cover 3-round trail cores with propagation weight at least 53, up from the previous best weight 45. To achieve the goal, the concept of ideal improvement assumption is proposed to construct *theoretical representative* of subspaces so as to efficiently cover the search space of 3-round trail cores with at least one out-Kernel α state. Of particular note is that the exhaustiveness in 3-round trail core search of at least one out-Kernel α is only experimentally verified. With the knowledge of all 3-round trail cores of weight up to 53, lower bounds on 4/5/6-round trails are tightened to 56/58/108, from the previous 48/50/92, respectively.

Keywords: SHA3 · Keccak-f · Differential Trail Search · Exhaustive Search · Lower Bound of Differential Trail

1 Introduction

1.1 The birth of Keccak

Following the breakthrough of cryptanalysis against a series of standards such as MD5 [WY05], SHA-0 [WYY05b], and SHA-1 [WYY05a], the U.S. National Institute of Standards and Technology (NIST) organized the SHA-3 competition (2008–2012), which received 64 submissions. Keccak [BDPV11] designed by Bertoni *et al.* won the competition in 2012, and was later formally adopted as the Secure Hash Algorithm 3 (SHA-3) standard in 2015 [The15]. The idea is to have a new standard with different design principle from SHA-1 and SHA-2, hence in the very unlikely event of similar issues to SHA-1 found on SHA-2, there will be a backup solution SHA-3.

The SHA-3 family has four main instances with fixed digest lengths, namely, SHA3-224, SHA3-256, SHA3-384, and SHA3-512, which correspond to Keccak$[c] \triangleq$ Keccak$[r = 1600 - c, c]$ where $c \in \{448, 512, 768, 1024\}$ are the capacity sizes in bits. There are also two extendable-output functions named SHAKE128 and SHAKE256, which support digest of variable sizes and collision resistance of level up to 128 and 256 bits, respectively. All hash functions of the SHA-3 family share the same underlying permutation function Keccak-f of 1600-bit internal state. To promote cryptanalysis, the designers of Keccak also proposed variants with capacity of 160 bits, as well as the underlying Keccak-f permutation of sizes in $\{200, 400, 800, 1600\}$, under *the Keccak Crunchy Crypto Collision and Pre-image Contest* [BDH+]. The same Keccak-f permutation (or its variant) could be used to construct authenticated encryption such as Keyak [BDP+16b] and Ketje [BDP+16a],

Received: 2019-06-01, Revised: 2019-09-01, Accepted: 2019-11-01, Published: 2020-01-31

message authentication code such as KMAC [The16], as well as key derivation function such as Kravatte [DHVV18].

1.2 The security status of Keccak

Since the birth, Keccak has received a significant amount of cryptanalysis from the public research community, and generally the results can be divided by the following security notions. So far, there is no attack threatening the security of Keccak, and actually all attacks work on Keccak variants reduced to at most 9 out of the total 24 rounds, leaving a large security margin of more than 62% of the rounds.

Collision Resistance is regarding the sponge hash functions based on Keccak-f and its variants. In 2012, Dinur *et al.* [DDS12] found then the best practical collision attacks against Keccak[448]/Keccak[512] reduced to 4 rounds. The results was later furnished in the journal version [DDS14b]. These 4-round collisions were generated by combining a 1-round connector and a 3-round differential trail. The same authors presented practical collision attacks on 3-round Keccak[768]/Keccak[1024], and theoretical collision attacks (with complexities beyond the reach of practical resources) on 5/4-round Keccak[512]/Keccak[768] in [DDS14a] using internal differentials. Qiao *et al.* [QSLG17] extended the idea of connector from 1 round to 2 rounds, extending the collision attacks to 5 rounds. Song, Liao, and Guo [SLG17] further extended the connectors to 3 rounds and found the first practical collision against one 6-round variant in the Keccak collision contest. These results were then summarized in [GLL+19], where practical collisions of 5-round SHAKE128, SHA3-224, and SHA3-256 were presented. Up to date, there exists no attack against any SHA-3 variants reduced to 6 rounds or more.

Preimage Resistance is also regarding the sponge hash functions. In 2016, Guo, Liu, and Song [GLS16] proposed the *linear structure* and found preimage attacks up to 4 rounds, including practical solutions to the 4-round Keccak[1440, 160] and the 3-round Keccak[1440, 160] and Keccak[640, 160] of the Keccak preimage contest. This was later extended by Li *et al.* [LSLW17] to the so-called *cross-linear structure*, and found its application in improving the complexities of some attacks, including the first practical preimage on 3-round Keccak[240, 160]. Further complexity improvements were found by Li and Sun [LS19]. In summary, the best preimage attacks, in terms of number of attacked rounds, are for no more than 4 rounds.

Key Recovery is regarding the keyed constructions such as authenticated encryption schemes, message authentication codes, and key derivation functions. The most effective attacks so far against the Keccak-f-based keyed constructions are cube attacks [DLWQ17, DMP+15, HWX+17, LBDW17, SGSL18, BDL+19], taking advantage of the low algebraic degree 2 (or its inverse 3) of the Keccak round function. Among them, the best attack works for up to 9 rounds.

Distinguishers. There are also distinguishers against the permutation Keccak-f itself, under models like limited-birthday [DGPW12] and zero-sum [AM09, GLS16]. While the limited-birthday distinguisher is based on differentials, which worked for up to 8 rounds with complexities bounded by 2^{512}, zero-sum distinguisher takes advantage of the low algebraic degrees of the Keccak round function and its inverse and worked for 11 rounds practically and 14 rounds with complexity 2^{129}.

1.3 Differential trail search

Differential trails are a key part of the security analysis against symmetric-key primitives, *e.g.*, they are the key factors deciding the effectiveness, in terms of rounds and attack complexities, of the attacks for collisions and differential-based distinguishers. However, unlike AES, there is no known tight bound for differential trials of KECCAK-f due to its weak aligned round function. The weight W of the best differential trails refers to the smallest value $\log_2(1/p)$, where p is the differential probability. W, for a given number of rounds, can be lower bounded by some theoretical reasoning such as counting the minimum number of active Sboxes in all differential trails, and upper bounded by differentials actually found. There have been continuous efforts in closing the gap between the lower and upper bounds for KECCAK-f, mostly by finding effective ways to enumerate all possible differentials falling in the gap.

It is trivial to see $W[r=1]=2$ and $W[r=2]=8$ (the minimum weight for 1-round and 2-round differential trails). The KECCAK designers [BDPV11] made the first effort in providing a tighter bounds, and $W[r=3]$ is proven to be 32 — a tight bound by Daemen and Van Assche in [DV12] with a matching differential trail found by Duc *et al.* in [DGPW12]. A more sophisticated technique was proposed by Mella-Daemen-Van Assche in [MDV17] to exhaustively search 3-round differential trails up to weight 45, which was used to give tighter bounds for differential trails of 4, 5, and 6 rounds.

1.4 Our contributions

Following the direction of [MDV17], this paper focuses on exhaustively enumerating 3-round differential trails of KECCAK-f to higher weight. The search space dramatically increases with the weight, hence the main difficulty to overcome here is to find an efficient way to quickly eliminate differential trails out of the target range, and hence to reduce the search space into one small enough for exhaustive search in practice. Two main new techniques are proposed in this paper: idealized differentials to enable early abortion and a careful division of the subcases.

Depending on whether the state falls in the *Column-Parity Kernel* (Kernel for short), the two consecutive rounds of the differential are divided into 4 cases, *i.e.*, $|K|K|$, $|K|N|$, $|N|K|$, and $|N|N|$[1]. In searching 3-round trail cores with $|K|K|$ feature, rather than enumerating in-Kernel α_1 by recursively adding orbitals as done in [DV12], we construct candidate α_1 based on β_1 structures that ensure the resulting α_2's are in-Kernel. For in-Kernel α_1 of m orbitals, through $\pi \circ \rho$, by examining how the $2m$ bits are distributed to β_1 slices, the compatible in-Kernel α_2 are enumerated. With such candidate β_1 structures, α_1 are constructed with data complexity 2^{42}, leading to exhaustive coverage of 3-round trail cores of weight no greater than 53.

In searching 3-round trail cores with at least one out-Kernel α, *i.e.*, 3-round trail cores with $|N|K|$, $|N|N|$ or $|K|N|$ feature, the search space of out-Kernel states α is divided into subspaces V_α derived from α. Each subspace has a *theoretical representatives* in terms of the number of active rows of all 3-round trail cores. The *theoretical representatives* are generated under the *ideal improvement assumption* which are deduced to construct the theoretically optimal state of the subspace. Therefore the subspace can be efficiently pruned with constraints on its *theoretical representative*. With this new search strategy, 3-round trail cores of weight up to 53 can be covered. Note that the exhaustive search with threshold weight 53 is only experimentally proved of which the technical details will be illustrated in Section 5.

With the help of the exhaustively searched 3-round trail cores of weight up to 53, lower bound of propagation weight of 4/5/6-round trails is tightened to 56/58/108 from the previous best result 48/50/92 in [MDV17], respectively.

Paper Organization. The paper is organized as follows. In Section 2, brief introduction of KECCAK-$f[1600]$ as well as propagation properties of linear operations used in deducing search algorithms are given. In Section 3, the basic concepts of differential trail cores and how they are classified in exhaustive search are illustrated. The search strategies of $|K|K|$,

[1] K (resp. N) denotes the state is in (resp. out) Kernel.

$|N|K|$ and $|N|N|$, and $|K|N|$ 3-round trail cores are explained in Section 4, 5 and 6 respectively. In Section 7, the valid 3-round trail cores searched with our new techniques are provided and discussed. We conclude this work in Section 8. Some additional details of the analysis are postponed to Appendix.

2 Keccak and Propagation Properties

In this section, a brief introduction to Keccak-f[1600] is given at first. Some frequently-used concepts, notations, and properties used for deducing the search strategies are presented as well.

2.1 Keccak

Keccak [BDPV11] is a family of sponge functions with permutations Keccak-f[b] where $b \in \{25, 50, 100, 200, 400, 800, 1600\}$. We study the differential propagation of Keccak-f[1600] which is the SHA-3 permutation.

The 1600 bits of its state are organized in a 3-dimensional array of $5 \times 5 \times 64$ size. Each bit is denoted by a (x, y, z) coordinate where $0 \leq x \leq 4$, $0 \leq y \leq 4$ and $0 \leq z \leq 63$. To better study the round function, structures that contain partial bits of the state are introduced as shown in Fig 1. A group of 5 bits that share the same y (resp. x) coordinate is called a *row* (resp. *column*), denoted by (x, z) (resp. (y, z)), while a *lane* consists of 64 bits of the same (x, y) coordinates. The 5×5 bits of the same z coordinate form a *slice*. Other structures include *plane* and *sheet*.

For structures composed of columns, i.e., column, slice, sheet and state, ***parity*** is defined. Parity, denoted by p, is the XOR of all 5 bits of a column. Thus parity of a column, i.e., $p(x, z)$ is a bit. If $p(x, z) = 0$ (or $p(x, z) = 1$), there are *even* (or *odd*) number of active bits in column (x, z). The column that has an odd (or even) number of active bits is called an *odd* (or *even*) column. For a slice, its parity $p(z)$ is a 1×5 row and for a state a, the parity $p(a)$ is a 5×64 plane.

The round function of Keccak-f[1600] consists of 5 operations, i.e., the linear components θ, ρ, π and ι, and the non-linear component χ.

- θ adds a pattern to each bit position (x, y, z). The pattern which is defined as θ-*effect* in [BDPV11,DV12,MDV17] is the XOR of the parity of column $(x-1, z)$ and column $(x+1, z-1)$.
- ρ rotates bits on lane (x, y) with an offset defined by (x, y).
- π rearranges the $5 \times 5 = 25$ bits on one slice.
- ι adds a constant to each round. It is always omitted in the analysis as it has no influence on differential propagation.
- χ is a 5-bit S-box. It is the only non-linear operation in Keccak-f[1600].

2.2 Propagation properties of linear layer operations

The linear layer of one round is composed of three components, i.e., θ, ρ, and π, denoted by $\lambda = \pi \circ \rho \circ \theta$. Propagation properties of those operations are described in the following paragraphs.

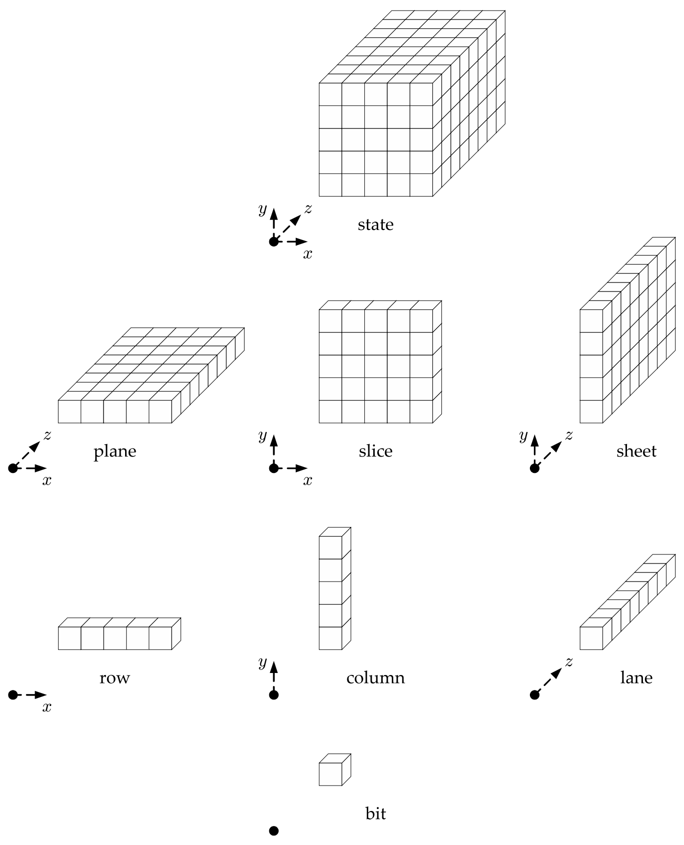


Figure 1: Different structures of round state.

2.2.1 Propagation property of θ

Out of the four linear components, only θ operation may alter the number of active bits. Given a state α if its parity $p(\alpha) = \mathbf{0}$ where $p(\mathbf{0})$ is a 5×64 zero vector, θ acts as an identity function, i.e., there is no θ-*effect* on the state. State with zero parity is called in-Kernel, denoted by K. Otherwise, it is out-Kernel, denoted by N. The number of active bits of in-Kernel states is preserved through θ.

However, if $p(\alpha)$ is a nonzero vector, the θ-*effect* really complicates the propagation of active bits. To better understand θ-*effect* of out-Kernel states, works [BDPV11, DV12, MDV17] introduced *run* to reorganize the nonzero parity. If the coordinates of a group of *odd* columns satisfy that $z_{i+1} = (z_i + 1) \mod 64$ and $x_{i+1} = (x_i + 3) \mod 5$, where (x_i, z_i) and (x_{i+1}, z_{i+1}) are any continuous *odd* columns, this collection of *odd* columns form a *run*. According to the definition of θ operation, no matter how many *odd* columns are in a *run*, it only affects two columns. The affected columns whose active bits will be flipped through θ are called *affected* columns. Parity thus can be partitioned by a series of *runs*. Fig. 2[2] shows two parities of 1-run and 2-run as well as the affected columns.

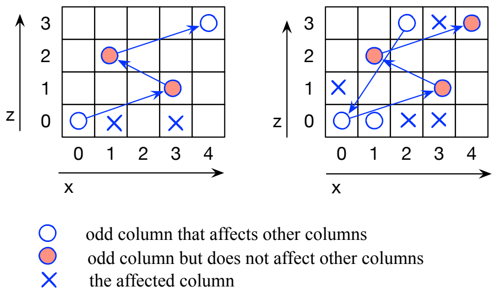

z
x
0
1
2
3
4
odd column that affects other columns
odd column but does not affect other columns
the affected column

Figure 2: Examples of a 1-run and a 2-run parity.

On the other hand, we can construct parities by listing all the possible combinations of *runs*. In later differential trail search sections, parities of 1-run, 2-run and 3-run are enumerated. For any out-Kernel difference, adding two active bits to a column (if it is possible), its parity remains unchanged. Practically, a group of 2 active bits that are in the same column is called an *orbital*. For in-Kernel differences, its active columns contain several orbitals.

Given a k-run parity of states α, *parity-bare* states are introduced to represent all states that share the same parity out-Kernel in regarding to differential probability of $\chi^{-1}(\alpha)$ and $\chi(\beta)$. Here α are out-Kernel input differences of θ and $\beta = \lambda(\alpha)$ is the input difference of χ. As introduced in [MDV17], of all the out-Kernel differences under the parity, *parity-bare* states have the least number of active bits in differences α and β, i.e., their hamming weight $||\alpha|| + ||\beta||$ is minimal. $||\cdot||$ denotes the number of active bits of a structure which is also called the *hamming weight*. Given a k-run parity, by adding orbitals to its *parity-bare* states all out-Kernel differences under the parity are obtained.

2.2.2 $\pi \circ \rho$-*effect*

While the θ-*effect* flips bits on affected rows of the out-Kernel states, ρ and π are only permutations of bit positions. The $\pi \circ \rho$-*effect* is the joint effect of ρ and π operations imposed on the active bits of state after θ-*effect*. For example, under the Eq. 1 model, α is transformed to β^θ through θ-*effect*, and β^θ is permuted to β through $\pi \circ \rho$-*effect*.

$$\alpha \xrightarrow{\theta} \beta^\theta \xrightarrow{\rho} \beta^\rho \xrightarrow{\pi} \beta \tag{1}$$

[2]In this example $z = 4$ rather than 64.

$\pi \circ \rho$-*effect* considers the bit positions before and after the associated linear functions. For example, given a bit of β^θ, the corresponding bit coordinates, row index, column index, and slice index at β of the specific bit are carefully examined. Similarly, given a bit of β, the origin bit coordinates, row index, column index, and slice index at β^θ are checked as well.

When α is in-Kernel θ becomes an identity function. In other words, there is only $\pi \circ \rho$-*effect* at such cases which simplifies the propagation of the complex λ.

The $\pi \circ \rho$-*effect* is used to bypass $\pi \circ \rho$. Given β^θ, one does not have to operate $\pi \circ \rho$ on the whole state. Rather, with only the active bits, the active columns, rows, and slices are easily obtained which is more advantageous when α is in-Kernel. Generally, $\pi \circ \rho$-*effect* has the following use scenarios in the subsequent trail search strategy.

- When α is in-Kernel, the number of active rows, i.e., Row_β is obtained through combining the row index of each bit.
- When α is out-Kernel with orbitals added to the non-affected columns, the number of added rows $AddedRow_\beta$, is also obtained by combining the added row index with the previous row index set.
- Given β, if the corresponding α is supposed to be in-Kernel, check the origin column index of β bits to verify whether it has in-Kernel α.

$\pi \circ \rho$-*effect* is widely used to efficiently count the number of active rows in the trail search strategy.

2.2.3 ρ property

In Section 4, a property of ρ explained in Lemma 1 is used in searching 3-round $|K|K|$ trail cores.

Lemma 1. *For any in-Kernel α with m orbitals, through $\pi \circ \rho$, there are at most m bits in any active β slices.*

Proof. From the definition of ρ operation, it is direct that the 25 bits of any α slices must be distributed to 25 different β slices and vice versa. As the m orbitals are at most placed in m slices the number of active bits of any β slices must not exceed m. □ □

3 3-Round Differential Trail Cores

Differential trail cores and the classification of 3-round trail cores are introduced in this section. Most of the concepts are brought from previous works [BDPV11, DV12, MDV17].

3.1 3-round trail cores

A 3-round differential trail, denoted by $(\alpha_0, \alpha_1, \alpha_2)$ (or $(\beta_0, \beta_1, \beta_2)$) is shown in Eq. 2, where α_i and β_i are input difference and output difference of λ respectively. It contains three χ layers with differential propagation probability $P_3 = P(\beta_0) + P(\beta_1) + P(\beta_3)$. In our analytic context, *propagation weight*, i.e., $w_3 = w(\beta_0) + w(\beta_1) + w(\beta_2)$, is extensively used to represent differential probability of trails.

$$\alpha_0 \xrightarrow{\lambda} \beta_0 \xrightarrow{\chi} \alpha_1 \xrightarrow{\lambda} \beta_1 \xrightarrow{\chi} \alpha_2 \xrightarrow{\lambda} \beta_2 \xrightarrow{\chi} \alpha_3 \tag{2}$$

Comparing to 3-round trails, 3-round trail cores (α_1, α_2) (or (β_1, β_2)) shown in Eq. 3 contain only two differences rather than three. Given any α_1, the minimal χ-compatible propagation weight $w(\beta_0)$ which is formally denoted by $w^{rev}(\alpha_1)$, is obtained simply by checking the differential distribution table. Therefore a 3-round trail core represents a set of 3-round trails among which the minimal propagation weight is $w_3 = w^{rev}(\alpha_1) + w(\beta_1) + w(\beta_2)$.

$$\alpha_1 \xrightarrow{\lambda} \beta_1 \xrightarrow{\chi} \alpha_2 \xrightarrow{\lambda} \beta_2 \tag{3}$$

Similarly, 2-round trail cores are denoted by (α_1) or (α) for general. We also simply call a 2-round trail core a α state in introducing ideas and methods. In previous works [DV12, MDV17], 2-round trail cores $(\alpha_1)/(\alpha_2)$ (or $(\beta_1)/(\beta_2)$) are exhaustively searched and extended forward or backward by one round to obtain 3-round trail cores. The search space is so huge that only 3-round trail cores with propagation weight no greater than a threshold weight T_3 are covered.

To exhaustively generate 3-round trail cores with a greater threshold weight T_3, namely the **valid** 3-round trail cores, we follow the previous rule of first collecting possible 2-round trail cores and then extending them forward or backward by one round. A 2-round trail core α_1 (resp. α_2) is called **valid** when there exists at least one χ-compatible α_2 (resp. α_1) that satisfies T_3.

Additionally, the number of active rows is used to evaluate the propagation weight of 3-round trail cores as the best differential probability (or *propagation weight*) of an active row is 2^{-2} (or 2). For example, the number of active rows of **valid** 3-round trails must satisfy that $Row_3 \leq \lceil \frac{T_3}{2} \rceil$ where $Row_3 = Row_{\alpha_1} + Row_{\alpha_2} + Row_{\beta_2}$. The number of active rows of a difference state α is denoted by Row_α.

3.2 Classification of 3-round trail cores

According to whether α_1 and α_2 are in-Kernel or not (refer to Eq. 3), the 3-round trail cores (α_1, α_2) can be categorized into four groups, i.e., $|K|K|$, $|N|K|$, $|N|N|$ and $|K|N|$ where K (resp. N) represents the state pattern α is in-Kernel or out-Kernel. Dedicated search strategies are proposed to cover the groups of 3-round trail cores with different characteristics, i.e.,

- the trail cores in the $|K|K|$ group are separately searched (in Section 4) due to the special characteristic that both α_1 and α_2 are in-Kernel;
- all the other trail cores with at least one out-Kernel α are searched under the same strategy of different methods. More specifically,
  - the trail cores in $|N|K|$ and $|N|N|$ groups are searched (in Section 5) with the same method;
  - while the trail cores in the $|K|N|$ group are covered (in Section 6) with a slightly different algorithm of similar ideas.

In summary, there are three kinds of 3-round trail cores of which each will follow a specific search algorithm.

4 Generating |K|K| Trail Cores

In this chapter, we introduce the methods used to generate valid 3-round trail cores (α_1, α_2) of $|K|K|$ feature. With the constraint that α_2 is in-Kernel, a strategy of constructing and filtering *target* α_1 that ensures both α_1 and α_2 in-Kernel is illustrated. Combining with another constraint on Row_3, satisfactory α_1 states are collected and extended forward by one round.

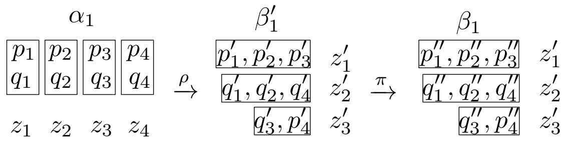

α_1
p_1 q_1 p_2 q_2 p_3 q_3 p_4 q_4
z_1 z_2 z_3 z_4
$\xrightarrow{\rho}$
β_1'
p_1', p_2', p_3' z_1'
q_1', q_2', q_4' z_2'
q_3', p_4' z_3'
$\xrightarrow{\pi}$
β_1
p_1'', p_2'', p_3'' z_1'
q_1'', q_2'', q_4'' z_2'
q_3'', p_4'' z_3'

Figure 3: A candidate β_1 structure example $\{3, 3, 2\}$ of α_1 with 4 orbitals.

4.1 Generating target α_1

As α_1 is in-Kernel, it has m orbitals distributed at different slices where $m \in \{1, 2, \cdots\}$. At β_1, the $2m$ bits will be distributed to k slices through $\pi \circ \rho$. As $\pi \circ \rho$ are linear operations, α_1 and β_1 are linearly equivalent. Thus α_1 can be represented by a k-tuple $\{n_1, n_2, \cdots, n_k\}$ where n_i is the number of active bits of the i-th β_1 slice $\beta_1|_{z_i}$. In this way, in-Kernel α_1 are divided into disjoint subsets with k-tuple $\{n_1, n_2, \cdots, n_k\}$ that depicts bit distribution pattern of the corresponding β_1. For example, for α_1 of 4 orbitals, there are four k-tuples, i.e., $\{2, 2, 2, 2\}$, $\{2, 2, 4\}$, $\{2, 3, 3\}$ and $\{4, 4\}$.

Definition 1. Valid β_i slice. An active slice $\beta_i|_z$ that contains at least one χ-compatible in-Kernel slice $\alpha_{i+1}|_z$ through χ operation is called a **valid** slice. There are at least 2 active bits in a valid β_1 slice.

Definition 2. Candidate β_1 structure. Any target α_1 must consist of a group of valid β_1 slices that contain $2m$ bits in total, i.e., $\sum_{i=1}^{i=k} ||\beta_1|_{z_i}|| = 2m$. The combination of the number of bits of valid β_1 slices $\bigcup_{i=1}^{i=k} ||\beta_1|_{z_i}||$ is called a *candidate β_1 structure* of target α_1.

To ensure α_2 in-Kernel, all β_1 slices must be **valid** (refer to Def. 1). The target α_1 must belong to some candidate β_1 structure (defined to Def. 2). For example, the candidate β_1 structures of α_1 with 3 orbitals is $\{\{3, 3\}, \{2, 2, 2\}\}$. To be specific, for any target α_1 of 3 orbitals, through $\pi \circ \rho$, its β_1 either consists of two valid slices each of 3 bits, or of 3 valid slices each of 2 bits.

Generating target α_1 includes constructing all α_1 under candidate β_1 structures and filtering them with constraints imposed by in-Kernel characteristic of both α_1 and α_2. With an example shown in Fig. 3, the whole process is explained at length. The two bits of an orbital at α_1 are denoted by p_i, q_i where $i \in \{1, 2, 3, 4\}$. Note that the notations of bits and orbitals are used just for clear description. The bits of one orbital have no distinction in themselves, so are the four orbitals.

Generating target α_1 under candidate β_1 structure. To efficiently construct α_1, all slice patterns of valid β_1 slices with s bits where $s \in \{2, 3, 4, 5, 6\}$, are stored in a lookup table in advance.

1. Through $\pi \circ \rho$, the candidate β_1 structure $\{3, 3, 2\}$ determines how the 8 bits of the 4 orbitals are organized at β_1. More specifically, three bits p_1, p_2, and p_3 from three distinct orbitals are mapped to the same slice $\beta_1|_{z_1'}$. One bit q_4 of the fourth orbital is mapped to slice $\beta_1|_{z_2'}$ with q_1 and q_2. The left two bits q_3 and p_4 are mapped to another valid $\beta_1|_{z_3'}$ slice.
2. Under the candidate β_1 structure, for any valid pattern of slice $\beta_1|_{z_1'}$ chosen from the lookup table, p_1'', p_2'' and p_3'' are determined. Accordingly, q_1, q_2 and q_3 can be decided from the α_1 orbital relations. With a determined q_3, p_4'' can be decided through the relations imposed by the validity of the slice $\beta_1|_{z_3'}$. Similarly, q_4 can be decided with α_1 orbital relation. When all the 8 bits of the 4 orbitals are determined, an α_1 is constructed.

3. For any constructed α_1, filter it with the constraint imposed by the candidate β_1 structure. To fulfill this β_1 structure, the ρ-offsets of the 8 bits must satisfy

$$\begin{aligned} z'_1 &= z_1 + \text{offset}[p_1] \\ &= z_2 + \text{offset}[p_2], \\ &= z_3 + \text{offset}[p_3] \end{aligned}$$

$$\begin{aligned} z'_2 &= z_1 + \text{offset}[q_1] \\ &= z_2 + \text{offset}[q_2], \\ &= z_4 + \text{offset}[q_4] \end{aligned}$$

$$\begin{aligned} z'_3 &= z_3 + \text{offset}[q_3] \\ &= z_4 + \text{offset}[p_4], \end{aligned}$$

   which can be further abstracted with

$$\begin{aligned} v_1 &= v_2 \\ v_3 &= v_1 + v_4, \end{aligned}$$

   where $v_i = (\text{offset}[q_i] - \text{offset}[p_i]) \bmod 64$ with $i \in \{1, 2, 3, 4\}$. Here offset$[q_i]$ and offset$[p_i]$ stand for the ρ-offset value of the bit position q_i and p_i. With this filter the α_1 states that strictly follow the β_1 structure can be obtained.

4. For any remaining α_1 states, filter it with constraints imposed by α_2 in-Kernel. In this case, we should further check whether slice $\beta_1|_{z'_2}$ is valid or not.

5. The constructed α_1 states which pass the above two filters are the target α_1.

Totally, candidate β_1 structures of α_1 with up to 6 orbitals are constructed and filtered. The associated analytical models are listed at Appendix A.

Remark 1. Although the methods for constructing target α_1 of different candidate β_1 structures are similar, the above steps are exclusive for $\{3, 3, 2\}$ structure. In practice, different candidate β_1 structures exhibit slightly different features that inspire dedicated treatments. Especially, even for the same structure, there might be several distinct sub-structures that show diverse bit organizations at β_1. For example, as shown in Table 1 and Fig. 10, candidate structure $\{3, 3, 2, 2\}$ consists of three different sub-structures. The five α_1 orbitals are distributed to two 3-bit β_1 slices and two 2-bit β_1 slices through $\pi \circ \rho$-*effect*. In sub-structure $\{3, 3, 2, 2\}(a)$, the three bits of the second 3-bit β_1 slice come from the first, second, and third α_1 orbital while in sub-structure $\{3, 3, 2, 2\}(b)$, the three bits come from the first, second and forth α_1 orbital.

When all target α_1 are collected, the satisfactory α_1 states are generated by further checking whether the requirement on Row_3, i.e., $Row_3 = Row_{\alpha_1} + Row_{\beta_1} + Row_{\beta_2} \leq \lceil \frac{T_3}{2} \rceil$ is met or not. Extending the satisfactory α_1 forward by one round, all $|K|K|$ trails (α_1, α_2) are successfully obtained.

4.2 Exhaustive search algorithm of $|K|K|$ trail cores

Rather than enumerate state α_1, the techniques used in this work construct in-Kernel α_1 from candidate β_1 structures. The |K|K| Trail Cores Search Method consists of three phases: α_1 *Construction*, *Filtering*, and *Trail Extension* as shown in Algorithm 1.

Phase 1. α_1 Construction. First prepare all the theoretical candidate β_1 structures for in-Kernel α_1 of m orbitals. For each candidate β_1 structure, start from the β_1 slice with the maximum number of active bits. To be precise, the (x, y, z) coordinates of active bits from the starting β_1 slice are determined, whose pairing bits of the same orbital in α_1 are known as well. In the end, coordinates of the $2m$ bits at α_1 are determined either from the valid β_1 slices or from the orbital relations at α_1.

Phase 2. Filtering. When α_1 are constructed according to the bit relations determined by the theoretically deduced β_1 structures, further check whether the m orbitals actually follow the structures or not. Filter α_1 with constraints that guarantee the m orbitals are mapped to valid β_1 slices. The constraints can be expressed by 1) relationship equations of the offsets relations $v_i = (\text{offset}[q_i] - \text{offset}[p_i])\%64$ with $i \in \{1, 2, ..., m\}$ of the m orbitals and 2) direct valid β_1 slice patterns stored in lookup tables. Store the α_1 that pass the filtering phase.

Phase 3. Trail Extension. Through the filtering phase, all target α_1 are collected. To generate valid 3-round trail cores of $|K|K|$ feature, target α_1 states are extended forward by one round with the constraint on Row_3 which is set as $\lceil \frac{T_3}{2} \rceil$. The trail extension methods are the same as in [MDV17] whose time complexity is neglected.

Algorithm 1: |K|K| Trail Cores Search Method

```
Input: The candidate $\beta_1$ structure
Output: The valid $|K|K|$ 3-round trail cores
 1: while (not last slice pattern of valid $\beta_1$ slices) do
 2:     (Phase 1 $\alpha_1$ Construction:)
 3:     Construct $\beta_1|_{z'_1}$
 4:     Apply $\rho^{-1} \circ \pi^{-1}$ to get corresponding bits in $\alpha_1$
 5:     Construct the related untried orbitals in $\alpha_1$
 6:     Using the existing orbitals to construct the other bits in $\beta_1|_{z'_i i>1}$
 7:     Get the constructed $\alpha_1$ structure
 8:     (Phase 2 Filter:)
 9:     if $\alpha_1$ passes the offsets filter
10:         if $\beta_1$ valid and the weight requirement met
11:             satisfactory $\alpha_1$ are obtained
12:         end if
13:     else
14:         go to line 4
15:     end if
16: end while
```

The time complexity of each candidate β_1 structure is decided by how α_1 are constructed. Take trail cores with α_1 of 3 orbitals whose candidate β_1 structures are $\{\{2,2,2\},\{3,3\}\}$as an example. For β_1 structure $\{3,3\}$, firstly, there are $1000 = 2^{9.97}$ valid slices. For one bit of a valid slice, there are 4 candidate orbitals at α_1. Thus for each valid slice of 3 bits, there are 4^3 candidate α_1 patterns. Therefore, the time complexity of $\{3,3\}$ is $2^{15.97}$.

The theoretical time complexities of all the associated analytical models are listed at Table 1. In practical programming, the searching space can be pruned by checking whether some simple constraints are met or not.

With a total complexity of around 2^{42}, $|K|K|$ 3-round trail cores of weight up to 53 are exhaustively generated. The time unit is the process of filtering a constructed α_1 state. All the cases in Table 1 are searched, and the experimental results are displayed in Section 7.

Table 1: Time complexity of $|K|K|$ trail core search.

# orbitals in α_1	Candidate β_1 structure	#time complexity of a case	#time complexity in totall
3 orbitals	1. $\{3,3\}$	$2^{15.97}$	$2^{16.23}$
	2. $\{2,2,2\}$	$2^{13.64}$	
4 orbitals	1. $\{4,4\}$	$2^{21.06}$	$2^{23.49}$
	2. $\{3,3,2\}$	$2^{19.97}$	
	3. $\{4,2,2\}$	$2^{21.06}$	
	4. $\{2,2,2,2\}$	$2^{15.64}$	
5 orbitals	1. $\{4,4,2\}$	$2^{25.06}$	$2^{29.97}$
	2. $\{5,3,2\}$	$2^{20.42}$	
	3. $\{4,3,3\}$	$2^{29.17}$	
	4. $\{4,2,2,2\}$	$2^{27.64}$	
	5. a) $\{3,3,2,2\}$	$2^{25.61}$	
	6. b) $\{3,3,2,2\}$	$2^{25.55}$	
	7. c) $\{3,3,2,2\}$	$2^{25.55}$	
	8. $\{2,2,2,2,2\}$	$2^{25.64}$	
6 orbitals	1. $\{6,6\}$	$2^{29.30}$	$2^{42.21}$
	2. $\{6,4,2\}$	$2^{29.30}$	
	3. $\{6,3,3\}$	$2^{29.30}$	
	4. $\{6,2,2,2\}$	$2^{29.30}$	
	5. a) $\{5,3,2,2\}$	$2^{32.74}$	
	6. b) $\{5,3,2,2\}$	$2^{32.74}$	
	7. a) $\{4,4,2,2\}$	$2^{31.06}$	
	8. b) $\{4,4,2,2\}$	$2^{31.64}$	
	9. a) $\{4,3,3,2\}$	$2^{24.70}$	
	10. b) $\{4,3,3,2\}$	$2^{24.28}$	
	11. c) $\{4,3,3,2\}$	$2^{35.58}$	
	12. a) $\{3,3,3,3\}$	$2^{37.94}$	
	13. b) $\{3,3,3,3\}$	$2^{41.11}$	
	13. c) $\{3,3,3,3\}$	$2^{41.11}$	
	14. a) $\{4,2,2,2,2\}$	$2^{30.70}$	
	15. b) $\{4,2,2,2,2\}$	$2^{30.64}$	
	16. c) $\{4,2,2,2,2\}$	$2^{29.06}$	
	17. a) $\{3,3,2,2,2\}$	$2^{29.61}$	
	18. b) $\{3,3,2,2,2\}$	$2^{29.55}$	
	19. c1) $\{3,3,2,2,2\}$	$2^{31.19}$	
	20. c2) $\{3,3,2,2,2\}$	$2^{29.55}$	
	21. d1) $\{3,3,2,2,2\}$	$2^{31.94}$	
	22. d2) $\{3,3,2,2,2\}$	$2^{29.55}$	
	23. $\{2,2,2,2,2,2\}$	$2^{29.97}$	

Discussion.

- With this algorithm, all 3-round trail cores of 6 orbitals and part trails of 7 orbitals are obtained. But we cannot cover all cases of 7 orbitals as theoretically there are too many candidate structures and the time complexity increases vigorously.
- The limitation of this exhaustive search algorithm is that it is oriented to orbitals. All 2-round trail cores of $m \leq 6$ orbitals are completely covered. The minimal propagation weight of 3/4/5/6-orbital $|K|K|$ trails are 33/39/48/53 respectively. As $|K|K|$ trails of 7 orbitals cannot be covered and there is no theoretical proof for a satisfactory lower bound, say 55 or 56 as well, we claim exhaustiveness over 53 which is experimentally proved.
- The weights of 7-orbital trails must be greater than 53. Let's assume the minimal weight of 7-orbital trails is 53. Compared to 6-orbital trails, the 7-th orbital must share rows with two other orbitals at α_1. In other words, for the 7-orbital trails of minimal weight 53, there is an α_1 slice that has 3 orbitals. Otherwise, the total weight of the 7-orbital trails will exceed 53. On the other hand, the 3 orbitals at the α_1 slice will be distributed to 6 slices at β_1. Thus, the structure of the 7-orbital trails are either $\{2,2,2,2,2,4\}$ or $\{2,2,2,2,3,3\}$. For either of the structures, the previous 6-orbital structure is $\{2,2,2,2,2,2\}$ which implies a weight increase at β_1. Therefore, compared to the 6-orbital trails, there must be weight increase for the 7-orbital trails.

Based on the above considerations, exhaustive search under threshold weight 53 are claimed. Without caring exhaustive search, 3-round trail cores of 3, 4, 5, 6 orbitals are generated. Those paths can be used in collision attack, preimage attack, and differential distinguisher attack. Particularly, in [GLL+19] one of the newly obtained 3-round trail cores of 5 orbitals are used to successfully identify a 5-round collision on SHA3-256.

5 Generating |N|K| and |N|N| Trail Cores

To obtain valid 3-round trail cores (α_1, α_2) with at least α_1 out-Kernel, valid 2-round trail cores α_1 that allow the existence of valid 3-round trail cores are collected and extended forward by one round. As in [MDV17], *parity-bare* states which represent all out-Kernel α_1 under the same parity are enumerated to generate valid 2-round trail cores. A propagation weight T_2 is set to list the *parity-bare* states of candidate parities.

In this section, we illustrate primary properties used to develop the *ideal improvement assumption* under which the *theoretical representatives* of subspaces are generated. Based on those concepts, out-Kernel α_1 are efficiently covered through *viability check* of subspace V_{α_1}. Valid 3-round trail cores can be obtained through extending all the valid α_1 forward by one round. Ultimately, the strategy of generating valid 3-round trail cores of $|N|K|$ and $|N|N|$ feature is presented.

5.1 Propagation property of out-Kernel α_1

With observations on differential propagation of out-Kernel α_1, β_1 does exhibit properties that can be exploited in the searching phase. Lemma 2 illustrates the elemental properties used to develop exhaustive search algorithms of $|N|K|$ and $|N|N|$ trails. The analytical model is shown in Eq 4.

$$\alpha_1 \xrightarrow{\lambda} \beta_1 \xrightarrow{\chi} \alpha_2 \xrightarrow{\lambda} \beta_2 \tag{4}$$

Definition 3. Given any difference α (or β), a **group** is comprised of l contiguous active slices of which the neighbors are nonactive, denoted by $\alpha|_g$ or $\beta|_g$.

Lemma 2. *Given a group pattern $\beta_1|_g$, assuming χ-compatible is not considered, the corresponding groups $\alpha_2|_g$ that have the minimal propagation weight at β_2 must be in-Kernel when $l \leq 6$.*

Proof. When checking out-Kernel state α_1 which is either *parity-bare* state or state generated by adding orbitals to *parity-bare* state, through λ operation, the corresponding β_1 is composed of several group patterns, i.e., $\beta_1 \coloneqq \bigcup_{k=1}^{n} \beta_1|_{g^k}$, where n is the number of groups.

For each β_1 group $\beta_1|_g$, there are a set of χ-compatible groups $\alpha_2|_g$ which are either in-Kernel or out-Kernel. Experimentally, the more *runs* the parity of χ-compatible groups $\alpha_2|_g$ have, the more active bits they have which incur heavier propagation weight at β_2.

For any group $\beta_1|_g$, without considering χ-compatibility, suppose it has in-Kernel $\alpha_2|_g$ groups. For any β_1 slice of group $\beta_1|_g$, there are a number of in-Kernel α_2 slices. The assumed in-Kernel α_2 slices are prepared following the rule that

- if a β_1 slice only has one active row, all α_2 slices of one orbital that have one bit in the corresponding active row are counted in;
- if a β_1 slice has multiple active rows, all the α_2 slices that have the least number of orbitals which are distributed over the corresponding active rows are counted in.

On the other hand, suppose any group $\beta_1|_g$ has optimal out-Kernel $\alpha_2|_g$ groups, i.e., the group patterns under 1-run parity which corresponds to the active β_1 rows. When all the in-Kernel and out-Kernel $\alpha_2|_g$ groups are prepared, compare the corresponding propagation weight at β_2.

It is experimentally proved that there is **conditional monotonic relation** between the number of active bits $||\beta_2^\theta|_g||$[3] and the number of active rows Row_{β_2} when $l \leq 6$. The

[3] β^θ is introduced in Eq 1.

propagation weight of the supposed optimal out-Kernel patterns is larger than that of the assumed in-Kernel patterns.

□ □

Lemma 3. *Without considering χ-compatibility, given any β_1, the ideal α_2 of minimal propagation weight at β_2 must be in-Kernel.*

Proof. The n-group β_1 can be reorganized according to the decrease of the length of each group, i.e., when checking propagation weight $w(\beta_2)$, the group with the largest length is first studied. For each group $\beta_1|_{g^k}$, there are two kinds of χ-compatible groups $\alpha_2|_{g^k}$, i.e., in-Kernel and out-Kernel where $k \in \{1, \cdots, 6\}$. Following Lemma 2, for all the β_1 group patterns, without considering χ-compatibility, prepare the corresponding in-Kernel and out-Kernel $\alpha_2|_{g^k}$ pattern.

The computation of propagation weight of $w(\beta_2)$ starts from the first group g^1. According to Lemma 2 the in-Kernel states will lead to lower weight. When the second group g^2 is involved, there might be overlaps on rows with g^1 which greatly complicates the whole situation. To address this problem, an experiment of exhaustively adding all assumed in-Kernel and out-Kernel $\alpha_2|_{g^2}$ to in-Kernel $\alpha_2|_{g^1}$ and computing the associated propagation weight is conducted. The length of group g^2 is at most 5. It shows that the in-Kernel $\alpha_2|_{g^2}$ always have less propagation weight. When taking the third, fourth and etc groups into consideration, as the group length decreases, in-Kernel $\alpha_2|_{g^k}$ patterns promise lower total $w(\beta_2)$.

Based on the above analysis, it is concluded that without considering χ-compatibility, given any β_1, the ideal α_2 of minimal propagation weight at β_2 must be in-Kernel.

□ □

Recall that the ultimate goal is to search 3-round trail cores that satisfy $w^{rev}(\alpha_1) + w(\beta_1) + w(\beta_2) \leq T_3$. Given any α_1 (or β_1), the propagation weight $w^{rev}(\alpha_1)$ and $w(\beta_1)$ are direct. Combining the newly introduced property Lemma 3, $w(\beta_2)$ can be predicted under some assumptions which will be explained in the following section.

5.2 The ideal improvement assumption

To search valid 2-round trail cores, it is essential to predict how the out-Kernel α propagates forward (or backward). To clearly explain the strategy and methods, definitions are given in advance.

Definition 4. The derivative subspace of the original 2-round trail cores. Given any out-Kernel state α, by adding n orbitals to it, where $n \in \{0, 1, 2, \cdots\}$, a set of out-Kernel states (denoted by α') that have the same parity is generated. Such a set is called the *derivative subspace* V_α of the **original** out-Kernel state α.

Instinctively, if α_1 (or β_1) can predict or evaluate the 3-round propagation weight of all out-Kernel states in its derivative subspace to some extent, valid 2-round trail cores can be efficiently constructed. According to Def. 4, each *parity-bare* state α_1 represents a derivative subspace of 2-round trail cores under specific parities. However it is impossible to decide whether there exist valid 2-round trail cores in the subspace directly from the *parity-bare* state.

From another perspective, the out-Kernel states α_1 are classified with parities p under which each *parity-bare* state stands for a derivative subspace V_{α_1}. The search space of out-Kernel 2-round trail cores is divided into disjoint derivative subspaces of the *parity-bare* states. A naive method to generate valid 2-round trail cores from each subspace V_{α_1} is to examine all α'_1 (where $\alpha'_1 \in V_{\alpha_1}$) with the constraint on propagation weight, i.e., $w^{rev}(\alpha'_1) + w(\beta'_1) \leq T_3 - 2$. The naive plan is obviously impractical.

On the other hand, if there exists some kind of representative 2-round trail core of V_{α_1}, the derivative subspace as well as the whole search space can be efficiently covered. With Observation 1, neither the original 2-round trail core α_1 nor other constructed patterns α_1' can effectively represent V_{α_1} in terms of Row_3 where $Row_3 = Row_{\alpha_1} + Row_{\beta_1} + Row_{\beta_2}$ is the number of active rows of 3-round trail cores. Based on Observation 2 where the decrement of Row_3 is possible by adding orbitals to α_1, the *theoretical representative* α_1^{τ} (refer to Def. 6) is proposed to represent the subspace. The *ideal improvement assumption* of original 2-round trail core α_1 based on which the *theoretical representative* of V_{α_1} is defined will be illustrate in this section.

Observation 1. *Given any derivative subspace V_{α_1}, comparing to the original α_1, the constructed 2-round trail cores α_1' have a larger number of active rows (i.e., Row_{α_1}+Row_{β_1}). However, α_1' may have less active rows in β_2 (i.e., Row_{β_2}) than α_1 due to the round operations. Therefore while α_1 is not valid it is possible to exist valid α_1' in subspace V_{α_1}.*

Observation 2. *Given a subspace V_{α_1}, when α_1 is not valid it is possible that there exist some valid α_1', which implies adding orbitals to α_1 really decreases Row_3 or more precisely Row_{β_2}. At this case, although adding orbitals to α_1 may increase Row_{α_1} or Row_{β_1}, the decrement on Row_{β_2} can compensate the cost so that a valid α_1' is possible.*

Definition 5. Common and **preferred** slice. A β_1 slice $\beta_1|_z$ is **common** if none of its χ-compatible slices $\alpha_2|_z$ is in-Kernel. Otherwise $\beta_1|_z$ is called a **preferred** slice.

The ideal improvement of α_1 in terms of Row_3. According to the analysis in Section 5.1, any α_1 that has common β_1 slices (refer to Def. 5) can be improved with a better Row_{β_2} by transforming common slices preferred. Here the *"improvement"* or the later *"improve"* means the decrement of the number of active rows, or more precisely a lower propagation weight. Each common slice can be improved by adding a specific number of bits to it which introduces the same number of orbitals to α_1. For example, a slice of only one bit is a common slice. To improve it we can add one active bit to the bit positions of the same column as the previous active bit. For each of the added orbitals, through the $\pi \circ \rho$-*effect*, the other bit is distributed either to an active slice or to an empty slice. In the most ideal case, the other bit of the added orbital also improves a common slice to be preferred. Actually, given an original 2-round trail core α_1 of V_{α_1}, an *theoretically optimal state* in regarding to Row_3 can be deduced with the following three assumptions.

β_1 is ideally improved to obtain optimal Row_{β_2}. Assume each common slice $\beta_1|_z$ is transformed preferred by only adding one bit to it. The other bit of the corresponding orbital improves another common slice simultaneously. If there are an odd number of common slices, the impact of the other bit of the left orbital is simply not considered. With this assumption, no new β_1 slice is introduced while the least number of orbitals are taken to improve α_1. When all common slices $\beta_1|_z$ become preferred, the improved β_1' has an optimal Row_{β_2} according to the conclusion shown in 5.1.

Row_{β_1} is counted with the least increase of #row. For each common slice $\beta_1|_z$, there are several kinds of improvements. For example, a bit of an unactive row and a bit of an active row may both improve a common slice. When counting Row_{β_1}, choose the one with the least increase of #row among all the possible improvements. With this method, Row_{β_1} is minimal under the circumstance of improving common $\beta_1|_z$.

Row_{α_1} is counted with the least increase of #row. When counting the new Row_{α_1} after improving β_1, choose the added orbital combinations with the least #row increase.

The optimal state is called the *theoretical representative* α_1^r of V_{α_1} (defined in below 6). As shown in Lemma 4, comparing to all 2-round trail cores in V_{α_1}, *theoretical representative* has the minimal propagation weight w_3^r, where $w_3^r = w^{rev}(\alpha_1^r) + w(\beta_1^r) + w(\beta_2^r)$.

Definition 6. Theoretical representative of derivative subspace. Under the *ideal improvement assumption*, given any out-Kernel 2-round trail core α, the ideally generated state is called *theoretical representative* of the derivative subspace V_α, denoted by α_1^r.

Lemma 4. *The* theoretical representative α_1^r *of a derivative subspace* V_{α_1} *represents all the 2-round trail cores, i.e, both* α_1 *and* $\alpha_1' \in V_{\alpha_1}$*, in terms of* Row_3.

Proof. When α_1 has no common slices $\beta_1|_z$, the *theoretical representative* α_1^r is identical to α_1. When α_1 has at least one common slice, α_1^r has the minimal Row_3 according to the *ideal improvement assumption*. Besides, as α_1^r is ideally improved from α_1 it naturally has a lower Row_3 than the other α_1' except those α_1' that follow the ideal improvement. Therefore, the *theoretical representative* always provide the possibly optimal Row_3 for all 2-round trail cores in V_{α_1}. □ □

Remark 2. The *theoretical representatives* of subspaces do not necessarily exist in practice, i.e., $\alpha_1^r \notin V_{\alpha_1}$ is acceptable, as the ideal assumption may not comply with the actual $\pi \circ \rho$*-effect.*

5.3 The 3-round trail core search

The *theoretical representative* provides a criteria to evaluate whether there are valid 2-round trail cores in the subspace. As shown in Lemma 4, if the *theoretical representative* of V_{α_1} is valid, there possibly are valid 2-round trail cores in the subspace. On the other hand, if the *theoretical representative* is not valid, there definitely are not any valid 2-round trail cores in the subspace. In this section, we explain at length how to generate valid 2-round trail cores with the *ideal improvement assumption*. We first illustrate the viability check of a derivative subspace V_{α_1}.

5.3.1 Generating viable 2-round trail cores

Given a subspace V_{α_1}, the process of constructing its *theoretical representative* with the *ideal improvement assumption* and checking the validity of the *theoretical representative* is called **viability check**. A subspace that passes the viability check is **viable** (defined below). For any viable subspaces, the following problem is to precisely identify the valid 2-round trail cores. In the following sections, we use viable subspace V_α and viable state α (or 2-round trail cores) indiscriminately. The viability check of a subspace V_α equals to the viability check of a state α.

Definition 7. Viable 2-round trail cores. The 2-round trail cores α of which the *theoretical representatives* α^r of its derivative subspace can pass the viability check, i.e., $w_3^r \leq T_3$, are called **viable**. The derivative subspace of a viable 2-round trail core is also called a viable subspace.

Viability check only shows whether it is possible that there exist valid 2-round trail cores in the subspace. Actually, the viable subspaces do not necessarily promise valid 2-round trail cores. With Lemma 5, the valid 2-round trail cores are obtained by checking the validity of viable trail cores.

Lemma 5. *The valid 2-round trail cores must be in the space of* ***viable*** *2-round trail cores.*

Proof. With the previous illustration, there are definitely not any valid 2-round trail cores in a non-viable subspace. In a viable subspace, following the definition of viable 2-round trail core (Def. 7), the valid 2-round trail cores must also satisfy the constraints of the possibly valid trail cores. Therefore, the space of viable states contains all the valid 2-round trail cores. □ □

The strategy used in locating all viable 2-round trail cores in a viable subspace is introduced in the following paragraph.

Locate all viable 2-round trail cores of a viable subspace V_{α_1}. Given a viable subspace V_{α_1}, the 2-round trail cores in it contain the original α_1 as well as a great number of α_1' which are constructed by adding orbitals to α_1. To collect all viable 2-round trail cores of a viable derivative subspace, a strategy of adding one orbital to viable original states each time is widely used.

- The original 2-round trail core α_1 are viable which could possibly be valid. Viable 2-round trail cores are stored for further trail extension.
- As for the 2-round trail cores α_1' in V_{α_1}, rather than enumerate all α_1', only α_1' with one orbital added to α_1 are checked at first. If the sub-subspace $V_{\alpha_1'}$ is viable, the corresponding α_1' is a viable 2-round trail core. Otherwise, the sub-subspace can be safely discarded. Thus α_1' with only one orbital added to α_1 is covered. Similarly, to cover α_1'' with two orbitals added to α_1,add one orbital to the previously collected viable α_1' and check the viability of the new derivative subspaces $V_{\alpha_1''}$. In this way, every time the viable 2-round trail cores with n orbitals added to α_1 are collected, add one orbital to them and check the viability of the newly generated subspaces. This process will terminate when none of states generated by adding one orbital to the newly obtained viable 2-round trail cores can pass the viability check.

The *parity-bare* states of all candidate parities are enumerated to operate the viability check after which a set of viable 2-round trail cores are collected. With the method presented above, we generate the viable 2-round trail cores in each of of the viable derivative subspaces. In the end, all viable 2-round trail cores of the search space are collected.

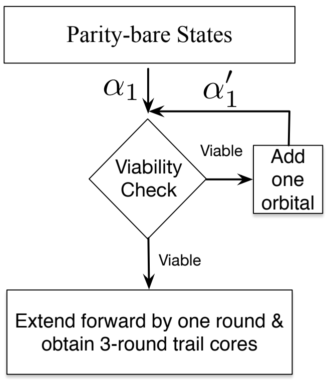


Figure 4: The process of 3-round trail core search for trails with $|N|K|$ and $|N|N|$ feature.

5.3.2 Search strategy of 3-round trail cores

Up to now, the *ideal improvement assumption* and the *theoretical representatives* of derivative subspaces are introduced with which the search space of 2-round trail cores can be efficiently covered and filtered out. All viable 2-round trail cores are collected through the viability check of subspaces. At this part, we give a comprehensive description on how to generate valid 3-round trail cores with the techniques explained in the previous sections.

1. Prepare parities p of 1-run, 2-run and even 3-run, and enumerate the associated *parity-bare* states. With rough estimation on the propagation weight of α_1 and β_1, only states under parities of at most 3-runs can meet the threshold weight.
2. Given any *parity-bare* state α_1, check the viability of its derivative subspace V_{α_1}.
3. Given any viable subspace V_{α_1}, collect all the *viable 2-round trail cores* in the subspace.
4. Extend all viable 2-round trail cores forward by one round with threshold weight T_3 to obtain all valid 3-round trail cores.

The exhaustiveness of the search space. To fully cover the search space of $|N|K|$, $|N|N|$ case and $|K|N|$ case, i.e., the 2-round trail cores, we enumerate the *parity-bare* states under all candidate parities as in [MDV17]. To be more specific, candidate parities of 1-run, 2-run, and 3-run are prepared.

- For 1-run parities, the length is up to 7. The time complexity of checking the viability of the *parity-bare* states is around $5 \times (2^4)^9 = 2^{38.3}$. It shows that there are no viable states for runs of length greater than 5.
- For 2-run case, parities of lengths $(1,1)$ and $(1,2)$ are listed where $(1,1)$ (or $(1,2)$) represents the lengths of the two runs are both 1 (or 1 and 2). The time complexity of checking the viability of the *parity-bare* states are around $2^{10.6} \times (2^4)^6 = 2^{34.6}$ and $2^{11.6} \times (2^4)^7 = 2^{39.6}$ respectively.

  For 2-run parities of lengths $(2,3)$, $(2,2)$ and $(1,3)$, we only prepare those with affected odd columns (as shown in the first example of Fig 5) of which the time complexity are less than 2^{35}. In fact, for most of the states under those parities, especially for those of more than 30 active bits in α and β^θ, the weights of the 2-round trail cores already exceed the threshold weight T_3. Thus we do not need to conduct the viability check for those states.

  And there are no viable states under $(2,3)$, $(2,2)$ and $(1,3)$ parities.

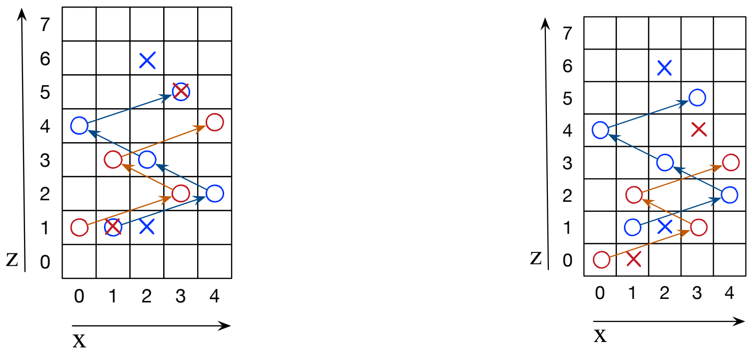


Figure 5: Examples of 2-run parities with and without affected odd columns

- For 3-run parities of length (1,1,1), only those with affected odd columns are prepared. The time complexity of checking the viability of the *parity-bare* states under those parities are around 2^{36}. The weights of all 2-round trail cores under those parities are greater than the threshold weight T_3 implying there are no viable states under 3-run parities.

With the increase on lengths of the runs, the number of active bits in α and β^θ increases accordingly. To be specific, wherever the positions of affected columns, the 2-run (or 3-run) parities have $2 \times 2 \times 5 + 2 \times \sum_{i=1}^{k} n_i$ (or $3 \times 2 \times 5 + 2 \times \sum_{i=1}^{k} n_i$) active bits in α and β^θ where k is the number of odd columns minus the number of affected odd columns (i.e., the number of unaffected odd columns), and n_i is the number of active bits of the i-th unaffected odd column. While a greater number of active bits does not always mean a heavier weight, it is so when the number of active bits are small enough.

Although for different parities the active bits are placed at different positions, the increase of active bits at α and β^θ will not decrease the weights under this context. Thus with the experimental verification, the longer the lengths of runs, the heavier the weights under those parities. In total, we prepare as many candidate parities as possible. When there is no viable states under a parity pattern, e.g., the (1,1,1) 3-run parities, we just quit and do not check other cases under this category that have more active bits in α and β^θ. In this way, all parities that may contain viable states are covered.

Figure 4 shows the complete process. The enumeration of *parity-bare* states and trail extension methods are directly taken from [MDV17]. In practice, 3-round trail cores with $T_3 = 53$ are generated with a time complexity of around 2^{40} whose time unit is the viability check of a 2-round trail core. The search results are summarized in Section 7.

5.4 The list of assumptions used to exhaustively cover the search space

The exhaustive search algorithms of 3-round trail cores with $|N|N|$, $|N|K|$ and $|K|N|$ features are based on several assumptions, i.e., the ideal improvement assumption and the assumptions used to fully cover the search space. To clarify the logic behind, we summarize them in this section.

1. **The assumptions used to fully cover the search space** According to the search strategy, the search space is divided into subspaces derived from *parity-bare* states which are generated under parities. Parities are organized by *runs*, for example, 1-run, 2-run and 3-run parities. As discussed in Section 5.3.2, the number of active bits of α and β^θ of *parity-bare* states is $m \times 2 \times 5 + 2 \times \sum_{i=1}^{k} n_i$, where m is the number of runs of parities, k is the number of unaffected odd columns, and n_i is the number of active bits of the i-th unaffected odd column. To enumerate all candidate parities, we have the following assumptions.

    - The more *runs* of a parity, the larger the number of active bits of α and β^θ of *parity-bare* states under the parity.
    - With the increase of length and number of *runs*, the number of active bits increases accordingly. Under this context, it is experimentally verified that the increase of the number of active bits will not decrease the propagation weight $w^{rev}(\alpha) + w(\beta)$.
    - Therefore, when enumerating parities organized by *runs*, if all the *parity-bare* states of one parity are not viable, we quit checking other parities of longer length or more *runs*.

    Starting from 1-run parity, we enumerate all possible parities with these assumptions.

2. **The ideal improvement assumption.** As described in Section 5.2, *the ideal improvement assumption* is used to determine whether a subspace is viable or not. In other words, with this assumption we decide whether it is possible to exist valid 3-round

trail cores in a subspace. To that end, an ideal representative of the subspace in terms of 3-round propagation weight is built upon the assumption that all the most ideal cases happen at the same time when improving the original 2-round trail core.

6 Generating |K|N| Trail Cores

$|K|N|$ trail core search shares the same strategy with $|N|K|$ and $|N|N|$ trail core search. Concepts and definitions illustrated in Section 5 are equally applied to $|K|N|$ trail core search. In searching 3-round trail cores, the principal technique is to compute Row_3 of *theoretical representatives* of *derivative* subspaces V_{α_2}, i.e., the viability check. In this section, the *ideal improvement assumption* under the $|K|N|$ situation is introduced to construct *theoretical representatives.* With the dedicated ideal compensating assumption, the algorithm of counting Row_3 is explained in detail. Finally, the general process of generating $|K|N|$ 3-round trail cores with threshold propagation weight T_3 is provided.

6.1 The ideal improvement assumption in compensating α_2

3-round trail cores with $|K|N|$ feature require α_1 in-Kernel. With the analytic model shown in Eq. 5 where $\lambda'^{-1} :=\rho^{-1}\circ\pi^{-1}$, the *theoretical representatives* of subspaces V_{α_2}[4] are generated and checked. In each subspace V_{α_2}, suppose there are at least one 2-round trail cores that have in-Kernel α_1, i.e., either α_2 or states generated by adding orbitals to α_2 have in-Kernel α_1. Basically, it is assumed that the least number of orbitals are added to α_2 to ensure *theoretical representative* with in-Kernel α_1. Additionally, the added orbitals also impose the least number of row increase on $Row_{\alpha_2} + Row_{\beta_2}$. Some definitions are given before introducing the ideal assumptions.

$$\alpha_1 \xleftarrow{\lambda'^{-1}} \beta_1 \xleftarrow{\chi^{-1}} \alpha_2 \xrightarrow{\lambda} \beta_2 \tag{5}$$

Definition 8. Orbital and Non-Orbital bit and row of difference β_1. Given a difference β_1, among all the active rows, through $\rho^{-1}\circ\pi^{-1}$, if one bit of an active row $\beta_1|_r^i$ is distributed to the same **column** at α_1 with another bit from another active row $\beta_1|_r^j$, the bit is called an *orbital bit.* If an active row $\beta_1|_r$ contains at least one orbital bit, it is called an *orbital row.* Otherwise, it is a *non-orbital* row.

Definition 9. Sharing bit and Sharing row of difference β_1. Given a difference β_1, among all the active rows, through $\rho^{-1}\circ\pi^{-1}$, if one bit of an active row $\beta_1|_r^i$ is distributed to the same **row** at α_1 with another bit from another active row $\beta_1|_r^j$, the bit is called a *sharing bit.* If an active row $\beta_1|_r$ contains at least one sharing bit, it is called a *sharing row.*

Definition 10. Compensating row of a non-orbital row. For any non-orbital β_1 row, there are around $5\times 4 = 20$ α_2 rows that can form α_1 orbitals with it. Such α_2 rows are called the compensating rows of the non-orbital row.

The process of adding orbitals to α_2 to make the related non-orbital β_1 rows orbital is called *compensating* the non-orbital rows. Similar to the concept of *improving* a common slice in the last chapter, here *compensating* a row means decreasing the total number of active rows Row_3.

[4]The *theoretical representative* α_2^r of V_{α_2} promises the minimal propagation weight w_3^r of the subspace, where $w_3^r = w^{rev}(\alpha_1^r) + w(\beta_1^r) + w(\beta_2^r)$.

The ideal compensating assumption. Given an out-Kernel state α_2, to efficiently determine whether there are valid 2-round trail cores in its *derivative* subspace, the validity of the *theoretical representative* ought to be checked. The assumptions used to compute Row_3 of the *theoretical representative* of V_{α_2} are listed below.

Superset of β_1 active row. To ensure α_1 in-Kernel with the minimal cost of orbitals added to α_2, without considering the compatibility of χ^{-1}, assume all the $(2^5 - 1)$ nonzero β_1 row value are valid. With this assumption, active rows at β_1 are classified into two categories, i.e., orbital and non-orbital rows (refer to Def 8). The orbital rows naturally form orbitals at α_1 through $\rho^{-1} \circ \pi^{-1}$, while the non-orbital rows need to be compensated to be orbital rows by adding orbitals to α_2. The α_1 orbitals that correspond to orbital β_1 rows through $\pi \circ \rho$ are called *natural orbitals* as there is no need to add extra orbitals to α_2 to make such α_1 orbitals possible. Comparatively, the α_1 orbitals that corresponds to non-orbital β_1 rows are called *compensating orbitals* because such β_1 rows need to be compensated to be orbital at α_1 (through adding orbitals to α_2).

For any active rows of α_2, as the superset of the corresponding β_1 row is considered, the χ^{-1} layer can be bypassed.

The optimal Row_{α_1}. Practically, α_1 of the *theoretical representative* is comprised of a set of *natural* orbitals and *compensating* orbitals. When selecting the combination of α_1 orbitals from both orbital and non-orbital β_1 rows, it is possible that the involved β_1 rows are also sharing rows (refer to Def 9). Therefore, there are cases that either the *natural* orbitals or the *compensating* orbitals share some α_1 rows. In that case the α_1 orbitals of sharing rows are always selected. Under the above assumption, the optimal Row_{α_1} of the *theoretical representative* can be obtained.

The least increase on the number of rows to $Row_{\alpha_2} + Row_{\beta_2}$. To compensate the non-orbital β_1 rows, orbitals are added to α_2. At this phase, whatever α_2 orbitals are chosen, suppose the optimal α_1 generated in the last step always holds. The orbitals added to α_2 must at first compensate all the non-orbital β_1 rows. In other words, the added α_2 orbitals at least lie in one compensating row (refer to Def 10). Then the finally chosen orbital combination must have the least influence on α_2 and β_2 in regard of the number of active rows. The number of active rows of α_2 and β_2 of the *theoretical representative* is computed with this assumption.

The ideal compensating assumption described above ensures a *theoretical representative* of subspace V_{α_2} with $|K|N|$ feature. To check the viability of V_{α_2}, the ideal Row_3 of the *theoretical representative* is computed with the optimal Row_{α_1} and $Row_{\alpha_2} + Row_{\beta_2}$. We introduce the detailed algorithm used in computing Row_3 of *theoretical representative* under the ideal improvement assumption in the next section.

6.2 Counting Row_3 of *theoretical representatives*

Row_3 of the *theoretical representative* stands for all states in subspace V_{α_2} in regard to the constraints imposed by $|K|N|$ feature and threshold propagation weight T_3. If $Row_3 > \lfloor \frac{T_3}{2} \rfloor$, there is no valid 2-round trail cores in V_{α_2}. The subspace can be safely discarded. The search space can be pruned effectively in this way.

With the ideal assumption, Row_3 is the sum of Row_{α_1} and $Row_{\alpha_2} + Row_{\beta_2}$. The exact algorithm is listed below.

Counting Row_{α_1}. According to the previous analysis, there are two kinds of orbitals in α_1, i.e., the natural orbitals that come from the active β_1 rows of original α_2, and the compensating orbitals that are generated through adding orbitals to α_2. In counting Row_{α_1}, natural orbitals and compensating orbitals are considered separately.

- **Natural orbitals.** The natural α_1 orbitals that correspond to β_1 orbital rows which have only one orbital bit must be counted in computing Row_{α_1}. Some orbital β_1 rows contain multiple orbital bits which implies they correspond to multiple α_1 orbitals. In that case, if any of the α_1 orbitals has been counted, the other α_1 orbitals of the β_1 rows can be discarded. Otherwise, if none of the associating α_1 orbitals have been counted, choose the one with the least increase on Row_{α_1}. With this rule, the theoretically optimal #row of natural orbitals is obtained.

- **Compensating orbitals.** The compensating α_1 orbitals of a non-orbital β_1 row might share α_1 rows with natural orbitals or with other compensating orbitals. In that case, always choose the compensating orbital that shares rows with natural orbitals over the one that shares rows with other compensating orbitals. If the non-orbital row has no sharing row, i.e., its compensating orbitals neither share rows with natural orbitals nor with other compensating orbitals, simply increase #row with 2.

Counting $Row_{\alpha_2} + Row_{\beta_2}$ with added orbitals. As explained earlier, when considering $Row_{\alpha_2} + Row_{\beta_2}$ suppose the theoretically optimal Row_{α_1} always holds whatever the orbital combinations are added to α_2. At α_2, orbitals with compensating rows are added to α_2 to make the non-orbital β_1 rows orbital. The combinations of added α_2 orbitals with the least increase on #row to α_2 and β_2 are chosen.

In compensating the non-orbital β_1 rows, there are theoretical cases that one α_2 compensating row compensates multiple non-orbital β_1 rows. There are also theoretical cases that one α_2 slice contains several compensating rows. In general, there are some situations to be considered when choosing the added orbitals to α_2.

- If there are α_2 slices in which the number of compensating α_2 rows is less than that of the compensated non-orbital β_1 rows, such compensating α_2 orbitals always have the least increase on the number of active rows to $Row_{\alpha_2} + Row_{\beta_2}$. If such best case exists, the related α_2 rows will be chosen over all the other compensating rows.

- There are cases that an α_2 slice contains multiple compensating rows each of which compensates a different non-orbital β_1 row. If the corresponding non-orbital β_1 rows can not be compensated in the best situation, it should be compensated under this circumstance.

- In the worst case, there is only one compensating α_2 row that can only compensate one non-orbital β_1 row in a α_2 slice. If a non-orbital β_1 row cannot be compensated with the last two cases, it has to be compensated in this way. The corresponding α_2 orbitals of the compensating α_2 rows will be counted in computing $Row_{\alpha_2} + Row_{\beta_2}$.

At last, compute the #row of α_2 and β_2 with the orbitals of compensating rows added to α_2 by the above rule.

The key technique in our search strategy is to determine Row_3 of the *theoretical representative* of V_{α_2}. With the above algorithm of computing Row_{α_1} and $Row_{\alpha_2} + Row_{\beta_2}$, the complete process of exhaustively searching 3-round trail cores of $|K|N|$ characteristic is illustrated in next part.

6.3 Search strategy of |K|N| 3-round trail cores

Following the same idea in Section 5, the search space of all out-Kernel α_2 is divided into a series of subspaces V_{α_2} where the original states α_2 are the *parity-bare* states under candidate parities. With ideal improvement assumption, *viability check* is conducted to the subspaces to collect viable α_2. Ultimately all viable α_2 are collected and extended backward by one round to generate valid 3-round trail cores of $|K|N|$ feature. Depicted in Fig 6, the complete search steps are introduced in the following paragraphs.

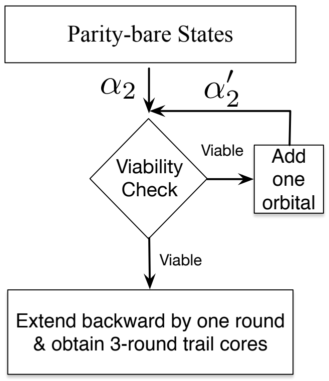


Figure 6: The process of 3-round trail core search for trails with $|K|N|$ feature.

1. To cover the space of out-Kernel α_2, prepare parities p of 1-run, 2-run and 3-run. Enumerate *parity-bare* states under candidate parity.
2. Conduct *viability check* to subspaces V_{α_2} where α_2 are *parity-bare* states. The viability check is to check the validity of the *theoretical representative* of the derivative subspace by checking whether $Row_3 \leq \lceil \frac{T_3}{2} \rceil$ is met or not.
3. For each viable derivative subspace V_{α_2}, by adding one orbital each time and checking the viability accordingly, collect all the viable 2-round trail cores in the subspace. The method is the same as in Section 5, i.e.,
   (a) For all viable α_2, add one orbital to it to construct new 2-round trail cores α'_2. Conduct *viability check* to new generated $V_{\alpha'_2}$ and collect the viable α'_2 accordingly.
   (b) Repeat the above process on the freshly viable α'_2 until all the new 2-round trail cores that are constructed from adding one orbital to the current viable 2-round trail core cannot pass the *viability check.*
4. Extend all viable α_2 backward by one round to generate valid 3-round trail cores with $|K|N|$ feature.

In our experiments, 3-round trail cores of weight up to 53 are exhaustively searched with a time complexity of 2^{45}. Similar to $|N|K|$ and $|N|N|$ case, parities of at most 3-runs are prepared after rough estimation on propagation weight. The search results are reported in Section 7.

7 Search Result And Bounds for Trails Covering More Rounds

7.1 Summary of the search result

According to the methods and algorithms introduced in Section 4, 5 and 6, 3-round trail cores with threshold propagation weight T_3 are exhaustively searched. More specifically,

Table 2: Summarized result of 3-round trail cores search.

	$\|K\|K\|$	$\|N\|K\|$	$\|N\|N\|$	$\|K\|N\|$
T_3	53	53	53	53
Time Complexity	2^{42}	2^{40}	2^{40}	2^{45}
Minimal Weight	35	46	48	32

$|K|K|$ trail cores, $|N|K|$ and $|N|N|$ trail cores, and $|K|N|$ trail cores of propagation weight up to 53 are generated with time complexity 2^{42}, 2^{40}, and 2^{45} respectively. The results of exhaustive 3-round trail core search are summarized in Table 2.

The propagation weight distribution of 3-round trail cores under distinct categories are listed in Fig 7. There is a trend that when α_1 is in-Kernel it is more likely to propagate to valid 3-round trail cores.

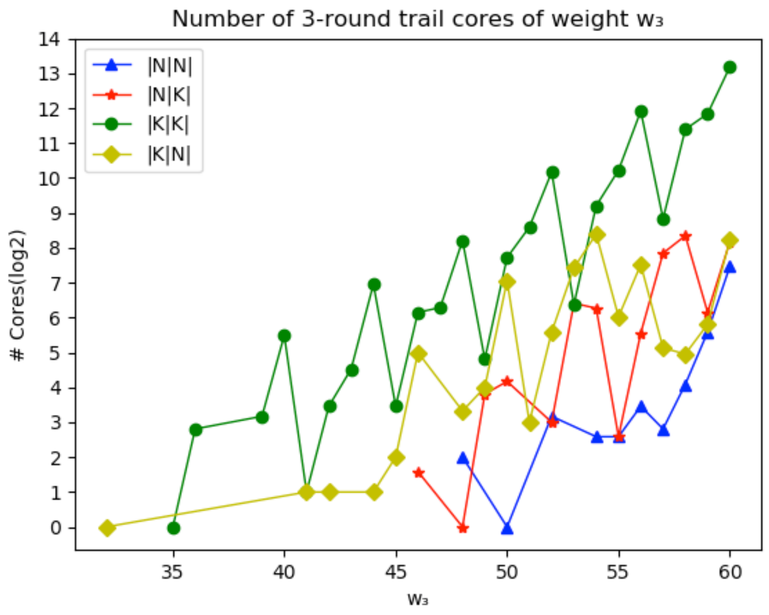


Figure 7: The propagation weight distribution of 3-round trail cores.

7.2 Bounds for 4/5/6-round trails

With all 3-round trail cores of threshold propagation weight $T_3 = 53$ at hand, we can update the lower bound on propagation weight of 4/5/6-round trails. Using the same idea as in [MDV17], lower bound for 4/5/6-round trails can be updated to 56, 58 and 108 respectively compared to the previous 48, 50 and 92.

8 Conclusion

Based on whether α_1 and α_2 are in-Kernel, 3-round trail cores are classified into four categories, i.e., $|K|K|$, $|K|N|$, $|N|K|$ and $|N|N|$ trail cores. In this work, we study the propagation properties of different kinds of 3-round trail cores and deduce dedicated exhaustive search algorithm for each of them. For $|K|K|$ trail cores, candidate in-Kernel

α_1 are constructed and filtered based on β_1 structures. For trail cores with at least one out-Kernel α_1, idealized differentials of subspaces are constructed to effectively prune the search space.

In conclusion, 3-round trail cores with threshold weight 53 are obtained. With exhaustively searched 3-round trail cores, lower bound on propagation weight of 4/5/6-round trails can be improved to 56/58/108 accordingly.

Acknowledgments

The first two authors are supported by the National Key Research and Development Program of China under Grand 2017YFB0802704, and the National Natural Science Foundation of China under Grand 61972249. The last author is supported by the National Research Foundation, Prime Minister's Office, Singapore, under its Strategic Capability Research Centres Funding Initiative, Nanyang Technological University under grant M4082123, and Singapore's Ministry of Education under grants M4012049, M4012153, and M4020466. We especially thank Prof. Guo Jian of Nanyang Technological University who offers us advice and guidance for this work. We thank Gilles Van Assche for helping us improve this paper. We also thank all the fellows and peers who give us advice, support and encouragement.

References

[AM09] Jean-Philippe Aumasson and Willi Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. Comment on the NIST Hash Competition, avaiable via https://131002.net/data/papers/AM09.pdf, 2009.

[BDH$^+$] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. The Keccak Crunchy Crypto Collision and Pre-image Contest.

[BDL$^+$19] Wenquan Bi, Xiaoyang Dong, Zheng Li, Rui Zong, and Xiaoyun Wang. MILP-aided cube-attack-like cryptanalysis on Keccak Keyed modes. *Des. Codes Cryptography*, 87(6):1271–1296, 2019.

[BDP$^+$16a] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Ketje v2. Candidate of CAESAR Competition, September 2016.

[BDP$^+$16b] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Keyak v2. Candidate of CAESAR Competition, September 2016.

[BDPV11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak Reference. https://keccak.team/files/Keccak-reference-3.0.pdf, January 2011. Version 3.0.

[DDS12] Itai Dinur, Orr Dunkelman, and Adi Shamir. New Attacks on Keccak-224 and Keccak-256. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 442–461, Washington, DC, USA, March 19–21, 2012. Springer, Heidelberg, Germany.

[DDS14a] Itai Dinur, Orr Dunkelman, and Adi Shamir. Collision Attacks on Up to 5 Rounds of SHA-3 Using Generalized Internal Differentials. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 219–240, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany.

[DDS14b] Itai Dinur, Orr Dunkelman, and Adi Shamir. Improved Practical Attacks on Round-Reduced Keccak. *Journal of Cryptology*, 27(2):183–209, April 2014.

[DGPW12] Alexandre Duc, Jian Guo, Thomas Peyrin, and Lei Wei. Unaligned Rebound Attack: Application to Keccak. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 402–421, Washington, DC, USA, March 19–21, 2012. Springer, Heidelberg, Germany.

[DHVV18] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. The design of Xoodoo and Xoofff. *IACR Transactions on Symmetric Cryptology*, 2018(4):1–38, 2018.

[DLWQ17] Xiaoyang Dong, Zheng Li, Xiaoyun Wang, and Ling Qin. Cube-like Attack on Round-Reduced Initialization of Ketje Sr. *IACR Transactions on Symmetric Cryptology*, 2017(1):259–280, 2017.

[DMP$^+$15] Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Cube Attacks and Cube-Attack-Like Cryptanalysis on the Round-Reduced Keccak Sponge Function. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 733–761, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[DV12] Joan Daemen and Gilles Van Assche. Differential Propagation Analysis of Keccak. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 422–441, Washington, DC, USA, March 19–21, 2012. Springer, Heidelberg, Germany.

[GLL$^+$19] Jian Guo, Guohong Liao, Guozhen Liu, Meicheng Liu, Kexin Qiao, and Ling Song. Practical Collision Attacks against Round-Reduced SHA-3. *Journal of Cryptology*, 2019. To appear.

[GLS16] Jian Guo, Meicheng Liu, and Ling Song. Linear Structures: Applications to Cryptanalysis of Round-Reduced Keccak. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 249–274, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.

[HWX$^+$17] Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional Cube Attack on Reduced-Round Keccak Sponge Function. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 259–288, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[LBDW17] Zheng Li, Wenquan Bi, Xiaoyang Dong, and Xiaoyun Wang. Improved Conditional Cube Attacks on Keccak Keyed Modes with MILP Method. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 99–127, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

[LS19] Ting Li and Yao Sun. Preimage Attacks on Round-Reduced Keccak-224/256 via an Allocating Approach. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 556–584, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[LSLW17] Ting Li, Yao Sun, Maodong Liao, and Dingkang Wang. Preimage Attacks on the Round-reduced Keccak with Cross-linear Structures. *IACR Transactions on Symmetric Cryptology*, 2017(4):39–57, 2017.

[MDV17] Silvia Mella, Joan Daemen, and Gilles Van Assche. New techniques for trail bounds and application to differential trails in Keccak. *IACR Transactions on Symmetric Cryptology*, 2017(1):329–357, 2017.

[QSLG17] Kexin Qiao, Ling Song, Meicheng Liu, and Jian Guo. New Collision Attacks on Round-Reduced Keccak. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 216–243, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[SGSL18] Ling Song, Jian Guo, Danping Shi, and San Ling. New MILP Modeling: Improved Conditional Cube Attacks on Keccak-Based Constructions. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 65–95, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

[SLG17] Ling Song, Guohong Liao, and Jian Guo. Non-full Sbox Linearization: Applications to Collision Attacks on Round-Reduced Keccak. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 428–451, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[The15] 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.

[The16] The U.S. National Institute of Standards and Technology. SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash. NIST Special Publication 800-185, 21st December 2016.

[WY05] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

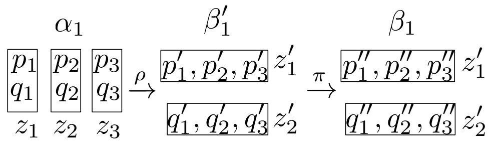
[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[WYY05b] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

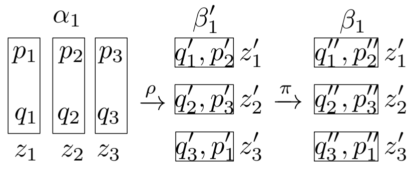
A Analytic Model In $|K|K|$ Trail Search

The candidate β_1 structures of 3, 4, and 5 orbitals are listed in this section. The β_1 structures of 6 and even 7 orbitals can be deduced in the same way. As there are too many structures, we omit them.

The n orbitals in α_1 are denoted as $p_1, q_1, \cdots\cdots, p_n, q_n$ with coordinates $(x_{p_i}, y_{p_i}, z_{p_i})$ and $(x_{q_i}, y_{q_i}, z_{q_i})$ respectively. Note it is possible that two orbitals are in the same slice z in a, i.e., there may exist z_i and z_j so that $z_i = z_j$ $(i, j \in \{1, 2, \cdots, n\})$. The $2n$ bits are transformed to bits $p'_1, q'_1, \cdots\cdots, p'_n, q'_n$ at β'_1 through ρ, and to bits $p''_1, q''_1, \cdots\cdots, p''_n, q''_n$ at β_1 through π.

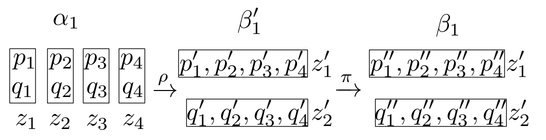

α_1
β'_1
β_1
p_1 q_1 z_1
p_2 q_2 z_2
p_3 q_3 z_3
ρ
p'_1, p'_2, p'_3 z'_1
q'_1, q'_2, q'_3 z'_2
π
p''_1, p''_2, p''_3 z'_1
q''_1, q''_2, q''_3 z'_2

(a) A candidate β_1 structure $\{3, 3\}$.

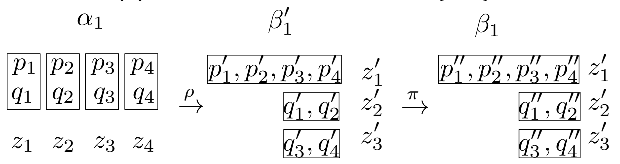


(b) A candidate β_1 structure $\{2, 2, 2\}$.

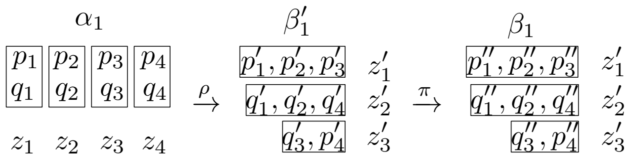
Figure 8: Candidate β_1 structures of 3 orbitals.


α_1
β_1'
β_1
p_1 q_1 p_2 q_2 p_3 q_3 p_4 q_4
z_1 z_2 z_3 z_4
ρ
p_1', p_2', p_3', p_4' z_1'
q_1', q_2', q_3', q_4' z_2'
π
$p_1'', p_2'', p_3'', p_4''$ z_1'
$q_1'', q_2'', q_3'', q_4''$ z_2'

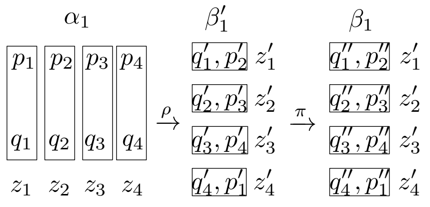
(a) A candidate β_1 structure $\{4, 4\}$.



(b) A candidate β_1 structure $\{4, 2, 2\}$.

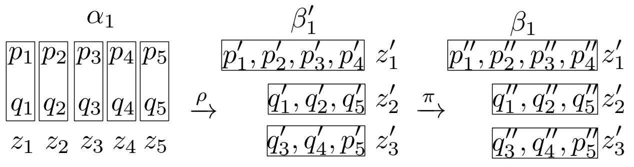

α_1
β_1'
β_1
p_1 q_1 p_2 q_2 p_3 q_3 p_4 q_4
z_1 z_2 z_3 z_4
ρ
p_1', p_2', p_3' z_1'
q_1', q_2', q_4' z_2'
q_3', p_4' z_3'
π
p_1'', p_2'', p_3'' z_1'
q_1'', q_2'', q_4'' z_2'
q_3'', p_4'' z_3'

(c) A candidate β_1 structure $\{3, 3, 2\}$.



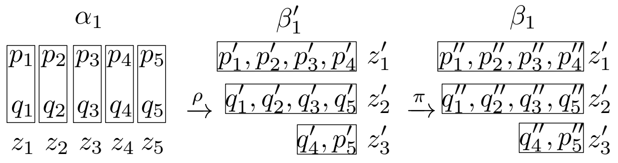
(d) A candidate β_1 structure $\{2, 2, 2, 2\}$.

Figure 9: candidate β_1 structures of 4 orbitals



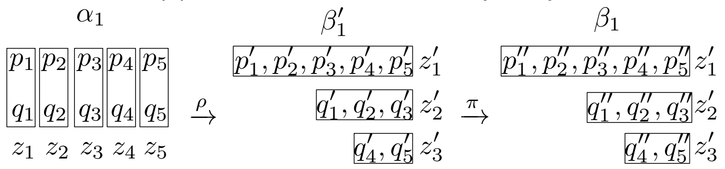
α_1
β_1'
β_1
p_1 p_2 p_3 p_4 p_5
q_1 q_2 q_3 q_4 q_5
z_1 z_2 z_3 z_4 z_5
ρ
π
p_1', p_2', p_3', p_4' z_1'
q_1', q_2', q_5' z_2'
q_3', q_4', p_5' z_3'
$p_1'', p_2'', p_3'', p_4''$ z_1'
q_1'', q_2'', q_5'' z_2'
q_3'', q_4'', p_5'' z_3'

(a) A candidate β_1 structure $\{4, 3, 3\}$.



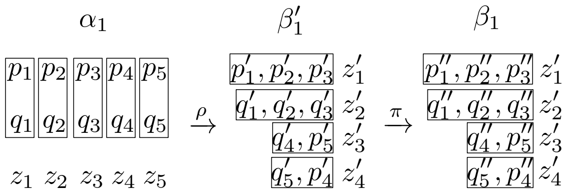
α_1
β_1'
β_1
p_1 p_2 p_3 p_4 p_5
q_1 q_2 q_3 q_4 q_5
z_1 z_2 z_3 z_4 z_5
ρ
π
p_1', p_2', p_3', p_4' z_1'
q_1', q_2', q_3', q_5' z_2'
q_4', p_5' z_3'
$p_1'', p_2'', p_3'', p_4''$ z_1'
$q_1'', q_2'', q_3'', q_5''$ z_2'
q_4'', p_5'' z_3'

(b) A candidate β_1 structure $\{4, 4, 2\}$.



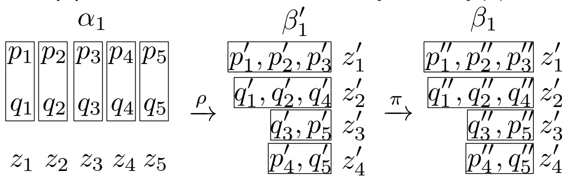
α_1
β_1'
β_1
p_1 p_2 p_3 p_4 p_5
q_1 q_2 q_3 q_4 q_5
z_1 z_2 z_3 z_4 z_5
ρ
π
$p_1', p_2', p_3', p_4', p_5'$ z_1'
q_1', q_2', q_3' z_2'
q_4', q_5' z_3'
$p_1'', p_2'', p_3'', p_4'', p_5''$ z_1'
q_1'', q_2'', q_3'' z_2'
q_4'', q_5'' z_3'

(c) A candidate β_1 structure $\{5, 3, 2\}$.



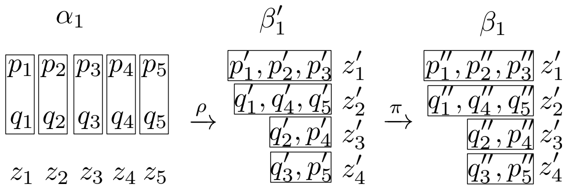
α_1
β_1'
β_1
p_1 p_2 p_3 p_4 p_5
q_1 q_2 q_3 q_4 q_5
z_1 z_2 z_3 z_4 z_5
ρ
π
p_1', p_2', p_3' z_1'
q_1', q_2', q_3' z_2'
q_4', p_5' z_3'
q_5', p_4' z_4'
p_1'', p_2'', p_3'' z_1'
q_1'', q_2'', q_3'' z_2'
q_4'', p_5'' z_3'
q_5'', p_4'' z_4'

(d) A candidate β_1 structure $\{3, 3, 2, 2\}$(a).



α_1
β_1'
β_1
p_1 p_2 p_3 p_4 p_5
q_1 q_2 q_3 q_4 q_5
z_1 z_2 z_3 z_4 z_5
ρ
π
p_1', p_2', p_3' z_1'
q_1', q_2', q_4' z_2'
q_3', p_5' z_3'
p_4', q_5' z_4'
p_1'', p_2'', p_3'' z_1'
q_1'', q_2'', q_4'' z_2'
q_3'', p_5'' z_3'
p_4'', q_5'' z_4'

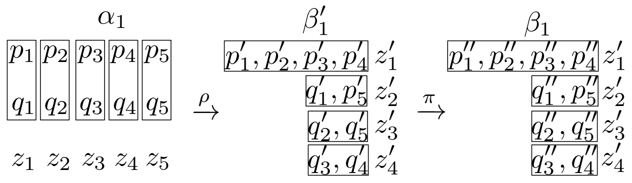
(e) A candidate β_1 structure $\{3, 3, 2, 2\}$(b).



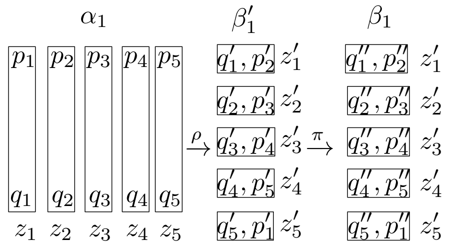
α_1
β_1'
β_1
p_1 p_2 p_3 p_4 p_5
q_1 q_2 q_3 q_4 q_5
z_1 z_2 z_3 z_4 z_5
ρ
π
p_1', p_2', p_3' z_1'
q_1', q_4', q_5' z_2'
q_2', p_4' z_3'
q_3', p_5' z_4'
p_1'', p_2'', p_3'' z_1'
q_1'', q_4'', q_5'' z_2'
q_2'', p_4'' z_3'
q_3'', p_5'' z_4'

(f) A candidate β_1 structure $\{3, 3, 2, 2\}$(c).

Figure 10: Candidate β_1 structure of 5 orbitals.



(a) A candidate β_1 structure $\{4, 2, 2, 2\}$.



(b) A candidate β_1 structure $\{2, 2, 2, 2, 2\}$.

Figure 11: Candidate β_1 structure of 5 orbitals.