The Keccak reference

Guido Bertoni[1]
Joan Daemen[1]
Michaël Peeters[2]
Gilles Van Assche[1]

http://keccak.noekeon.org/

Version **3.0**
January 14, 2011

[1]STMicroelectronics
[2]NXP Semiconductors

Contents

1 Keccak specifications **7**
1.1 Conventions and notation . . . 7
1.1.1 Bitstrings . . . 7
1.1.2 Padding rules . . . 7
1.2 The Keccak-f permutations . . . 7
1.3 The sponge construction . . . 8
1.4 The Keccak sponge functions . . . 9
1.5 Security claim for the Keccak sponge functions . . . 9
1.6 Parts of the state . . . 10

2 The Keccak-f permutations **13**
2.1 Translation invariance . . . 13
2.2 The Matryoshka structure . . . 14
2.3 The step mappings of Keccak-f . . . 14
2.3.1 Properties of χ . . . 15
2.3.2 Properties of θ . . . 17
2.3.3 Properties of π . . . 19
2.3.4 Properties of ρ . . . 21
2.3.5 Properties of ι . . . 22
2.3.6 The order of steps within a round . . . 23
2.4 Differential and linear cryptanalysis . . . 23
2.4.1 A formalism for describing trails adapted to Keccak-f . . . 23
2.4.2 The Matryoshka consequence . . . 24
2.4.3 The column parity kernel . . . 25
2.4.4 One and two-round trails . . . 25
2.4.5 Three-round trails: kernel vortices . . . 26
2.4.6 Beyond three-round trails: choice of π . . . 27
2.4.7 Truncated trails and differentials . . . 29
2.4.8 Other group operations . . . 29
2.4.9 Differential and linear cryptanalysis variants . . . 29
2.5 Solving constrained-input constrained-output (CICO) problems . . . 30
2.6 Strength in keyed mode . . . 30
2.7 Symmetry weaknesses . . . 31

3 Trail propagation in Keccak-f **33**
3.1 Relations between different kinds of weight . . . 33
3.2 Propagation properties related to the linear step θ . . . 35
3.3 Exhaustive trail search . . . 36
3.3.1 Upper bound for the weight of two-round trails to scan . . . 36

3.3.2 Constructing two-round trails . . . 37
3.3.3 Extending trails . . . 39
3.3.4 Linear and differential trail bounds for $w \leq 8$ . . . 40
3.4 Tame trails . . . 41
3.4.1 Construction of tame trails . . . 41
3.4.2 Bounds for three-round tame trails . . . 42
3.4.3 Bounds for four-round tame trails . . . 44

4 Analysis of KECCAK-f 45
4.1 Algebraic normal form . . . 45
4.1.1 Statistical tests . . . 45
4.1.2 Symmetric trails . . . 46
4.1.3 Slide attacks . . . 48
4.2 Solving CICO problems algebraically . . . 48
4.2.1 The goal . . . 48
4.2.2 The supporting software . . . 48
4.2.3 The experiments . . . 49
4.2.4 Third-party analysis . . . 51
4.3 Properties of KECCAK-f[25] . . . 51
4.3.1 Algebraic normal statistics . . . 51
4.3.2 Differential probability distributions . . . 52
4.3.3 Correlation distributions . . . 53
4.3.4 Cycle distributions . . . 57
4.4 Distinguishers exploiting low algebraic degree . . . 59
4.4.1 Zero-sum distinguishers . . . 60
4.4.2 Pre-image attacks . . . 62

5 Design rationale summary 63
5.1 Choosing the sponge construction . . . 63
5.2 Choosing an iterated permutation . . . 64
5.3 Designing the KECCAK-f permutations . . . 64
5.4 Strength estimation . . . 65

Introduction

In this document we specify, analyze and motivate the design of the cryptographic primitive Keccak. Based on the sponge construction, Keccak inherits many of its features. We gathered all our analysis on sponge functions in a separate document titled *Cryptographic sponge functions* [8]. Reading it is a requisite for understanding the usability and security properties of Keccak and the security requirements for Keccak-f, the permutation used in Keccak.

Other documents and files are of interest to the readers of this Keccak reference.

- This document comes with a set of files containing results of tests and experiments, available from `http://keccak.noekeon.org/`.
- The implementation aspects are covered in a separate document Keccak *implementation overview*, which treats software and hardware techniques and results, with or without protection against side-channel attacks [10].
- Also, KeccakTools, an open-source software aimed at helping analyze Keccak [9].

This document is organized as follows. Chapter 1 contains the formal specifications of Keccak. The subsequent three chapters are dedicated to the Keccak-f permutations:

- Chapter 2 explains the properties of the building blocks of Keccak-f and motivates the choices made in its design.
- Chapter 3 is dedicated to trail propagation in Keccak-f.
- Chapter 4 covers all other analysis of Keccak-f.

Finally, Chapter 5 summarizes the design choices behind Keccak and contains our estimation of the safety margin of Keccak.

Acknowledgments

We wish to thank (in no particular order) Charles Bouillaguet and Pierre-Alain Fouque for discussing their results later published in [13] with us, Dmitry Khovratovich for discussing with us the results published in [29] and for his analysis in [1], Jean-Philippe Aumasson for his analysis in [1] and [2], Joel Lathrop for his analysis in [34] and Willi Meier for his analysis in [2], Anne Canteaut and Christina Boura for their analysis in [15, 14, 16], Christophe De Cannière for his analysis in [16], Paweł Morawiecki and Marian Srebrny for their analysis in [36], Dan Bernstein for his analysis in [3], Ming Duan and Xuejia Lai for their analysis in [26], Yves Moulart, Bernard Kasser and all our colleagues at STMicroelectronics and NXP Semiconductors for creating the working environment in which we could work on this. Finally we would like to thank *Agentschap voor Innovatie door Wetenschap en Technologie* (IWT) for funding two of the authors (Joan Daemen and Gilles Van Assche).

Chapter 1

Keccak specifications

Keccak (pronounced [kɛtʃak]) is a family of sponge functions [8] that use as a building block a permutation from a set of 7 permutations. In this chapter, we introduce our conventions and notation, specify the 7 permutations underlying Keccak and the Keccak sponge functions. We also give conventions for naming parts of the Keccak state.

1.1 Conventions and notation

We denote the absolute value of a real number x is denoted by $|x|$.

1.1.1 Bitstrings

We denote the length in bits of a bitstring M by $|M|$. A bitstring M can be considered as a sequence of blocks of some fixed length x, where the last block may be shorter. The number of blocks of M is denoted by $|M|_x$. The blocks of M are denoted by M_i and the index ranges from 0 to $|M|_x - 1$.

We denote the set of all bitstrings including the empty string by $\mathbb{Z}_2^*$ and excluding the empty string by $\mathbb{Z}_2^+$. The set of infinite-length bitstrings is denoted by $\mathbb{Z}_2^\infty$.

1.1.2 Padding rules

For the padding rule we use the following notation: the padding of a message M to a sequence of x-bit blocks is denoted by $M||\text{pad}[x](|M|)$. This notation highlights that we only consider padding rules that append a bitstring that is fully determined by the bitlength of M and the block length x. We may omit $[x]$, $(|M|)$ or both if their value is clear from the context.

Keccak makes use of the *multi-rate* padding.

Definition 1. Multi-rate padding, *denoted by* pad10*1, *appends a single bit 1 followed by the minimum number of bits 0 followed by a single bit 1 such that the length of the result is a multiple of the block length.*

Multi-rate padding appends at least 2 bits and at most the number of bits in a block plus one.

1.2 The Keccak-f permutations

There are 7 Keccak-f permutations, indicated by Keccak-$f[b]$, where $b = 25 \times 2^\ell$ and ℓ ranges from 0 to 6. Keccak-$f[b]$ is a permutation over $\mathbb{Z}_2^b$, where the bits of s are numbered

from 0 to $b-1$. We call b the width of the permutation.

The permutation KECCAK-$f[b]$ is described as a sequence of operations on a state a that is a three-dimensional array of elements of GF(2), namely $a[5][5][w]$, with $w = 2^\ell$. The expression $a[x][y][z]$ with $x, y \in \mathbb{Z}_5$ and $z \in \mathbb{Z}_w$, denotes the bit in position (x, y, z). It follows that indexing starts from zero. The mapping between the bits of s and those of a is $s[w(5y+x)+z] = a[x][y][z]$. Expressions in the x and y coordinates should be taken modulo 5 and expressions in the z coordinate modulo w. We may sometimes omit the $[z]$ index, both the $[y][z]$ indices or all three indices, implying that the statement is valid for all values of the omitted indices.

KECCAK-$f[b]$ is an iterated permutation, consisting of a sequence of n_r rounds R, indexed with i_r from 0 to $n_\mathrm{r}-1$. A round consists of five steps:

$$\mathrm{R} = \iota \circ \chi \circ \pi \circ \rho \circ \theta, \text{ with}$$

$$\begin{aligned}
\theta: \quad & a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1], \\
\rho: \quad & a[x][y][z] \leftarrow a[x][y][z-(t+1)(t+2)/2], \\
& \qquad \text{with } t \text{ satisfying } 0 \le t < 24 \text{ and } \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \text{ in } \mathrm{GF}(5)^{2\times 2}, \\
& \qquad \text{or } t = -1 \text{ if } x = y = 0, \\
\pi: \quad & a[x][y] \leftarrow a[x'][y'], \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}, \\
\chi: \quad & a[x] \leftarrow a[x] + (a[x+1]+1)a[x+2], \\
\iota: \quad & a \leftarrow a + \mathrm{RC}[i_\mathrm{r}].
\end{aligned}$$

The additions and multiplications between the terms are in GF(2). With the exception of the value of the round constants $\mathrm{RC}[i_\mathrm{r}]$, these rounds are identical. The round constants are given by (with the first index denoting the round number)

$$\mathrm{RC}[i_\mathrm{r}][0][0][2^j-1] = \mathrm{rc}[j+7i_\mathrm{r}] \text{ for all } 0 \le j \le \ell,$$

and all other values of $\mathrm{RC}[i_\mathrm{r}][x][y][z]$ are zero. The values $\mathrm{rc}[t] \in \mathrm{GF}(2)$ are defined as the output of a binary linear feedback shift register (LFSR):

$$\mathrm{rc}[t] = \left(x^t \bmod x^8 + x^6 + x^5 + x^4 + 1\right) \bmod x \text{ in } \mathrm{GF}(2)[x].$$

The number of rounds n_r is determined by the width of the permutation, namely,

$$n_\mathrm{r} = 12 + 2\ell.$$

1.3 The sponge construction

The sponge construction [8] builds a function SPONGE$[f, \mathrm{pad}, r]$ with variable-length input and arbitrary output length using a fixed-length permutation (or transformation) f, a padding rule "pad" and a parameter *bitrate* r. The permutation f operates on a fixed number of bits, the *width* b. The value $c = b - r$ is called the *capacity*.

For the padding rule we use the following notation: the padding of a message M to a sequence of x-bit blocks is denoted by $M||\mathrm{pad}[x](|M|)$, where $|M|$ is the length of M in bits.

Initially, the state has value 0^b, called the *root state*. The root state has a fixed value and shall never be considered as an input. This is crucial for the security of the sponge construction.

Algorithm 1 The sponge construction SPONGE$[f, \text{pad}, r]$

Require: $r < b$

Interface: $Z = \text{sponge}(M, \ell)$ with $M \in \mathbb{Z}_2^*$, integer $\ell > 0$ and $Z \in \mathbb{Z}_2^\ell$
$P = M||\text{pad}[r](|M|)$
$s = 0^b$
for $i = 0$ to $|P|_r - 1$ **do**
  $s = s \oplus (P_i||0^{b-r})$
  $s = f(s)$
end for
$Z = \lfloor s \rfloor_r$
while $|Z|_r r < \ell$ **do**
  $s = f(s)$
  $Z = Z||\lfloor s \rfloor_r$
end while
return $\lfloor Z \rfloor_\ell$

1.4 The KECCAK sponge functions

We define the sponge function denoted by KECCAK$[r, c]$ by applying the sponge construction as specified in Algorithm 1 with KECCAK-$f[r + c]$, multi-rate padding and the bitrate r.

$$\text{KECCAK}[r, c] \triangleq \text{SPONGE}[\text{KECCAK-}f[r + c], \text{pad}10^*1, r].$$

This specifies KECCAK$[r, c]$ for any combination of $r > 0$ and c such that $r + c$ is a width supported by the KECCAK-f permutations.

The default value for r is $1600 - c$ and the default value for c is 576:

$$\text{KECCAK}[c] \triangleq \text{KECCAK}[r = 1600 - c, c],$$
$$\text{KECCAK}[] \triangleq \text{KECCAK}[c = 576].$$

1.5 Security claim for the KECCAK sponge functions

For each of the supported parameter values, we make a *flat sponge claim* [8, Section "The flat sponge claim"].

Claim 1. *The expected success probability of any attack against* KECCAK$[r, c]$ *with a workload equivalent to N calls to* KECCAK-$f[r + c]$ *or its inverse shall be smaller than or equal to that for a random oracle plus*

$$1 - \exp\left(-N(N+1)2^{-(c+1)}\right).$$

We exclude here weaknesses *due to the mere fact that* KECCAK-$f[r + c]$ *can be described compactly and can be efficiently executed, e.g., the so-called random oracle implementation impossibility [8, Section "The impossibility of implementing a random oracle"].*

Note that the claimed capacity is equal to the capacity used by the sponge construction.

1.6 Parts of the state

In this subsection, we define names of parts of the KECCAK-f state, as illustrated in Figure 1.1. This naming convention may help use a common terminology when analyzing or describing properties of KECCAK-f.
The one-dimensional parts are:

- A *row* is a set of 5 bits with constant y and z coordinates.
- A *column* is a set of 5 bits with constant x and z coordinates.
- A *lane* is a set of w bits with constant x and y coordinates.

The two-dimensional parts are:

- A *sheet* is a set of $5w$ bits with constant x coordinate.
- A *plane* is a set of $5w$ bits with constant y coordinate.
- A *slice* is a set of 25 bits with constant z coordinate.

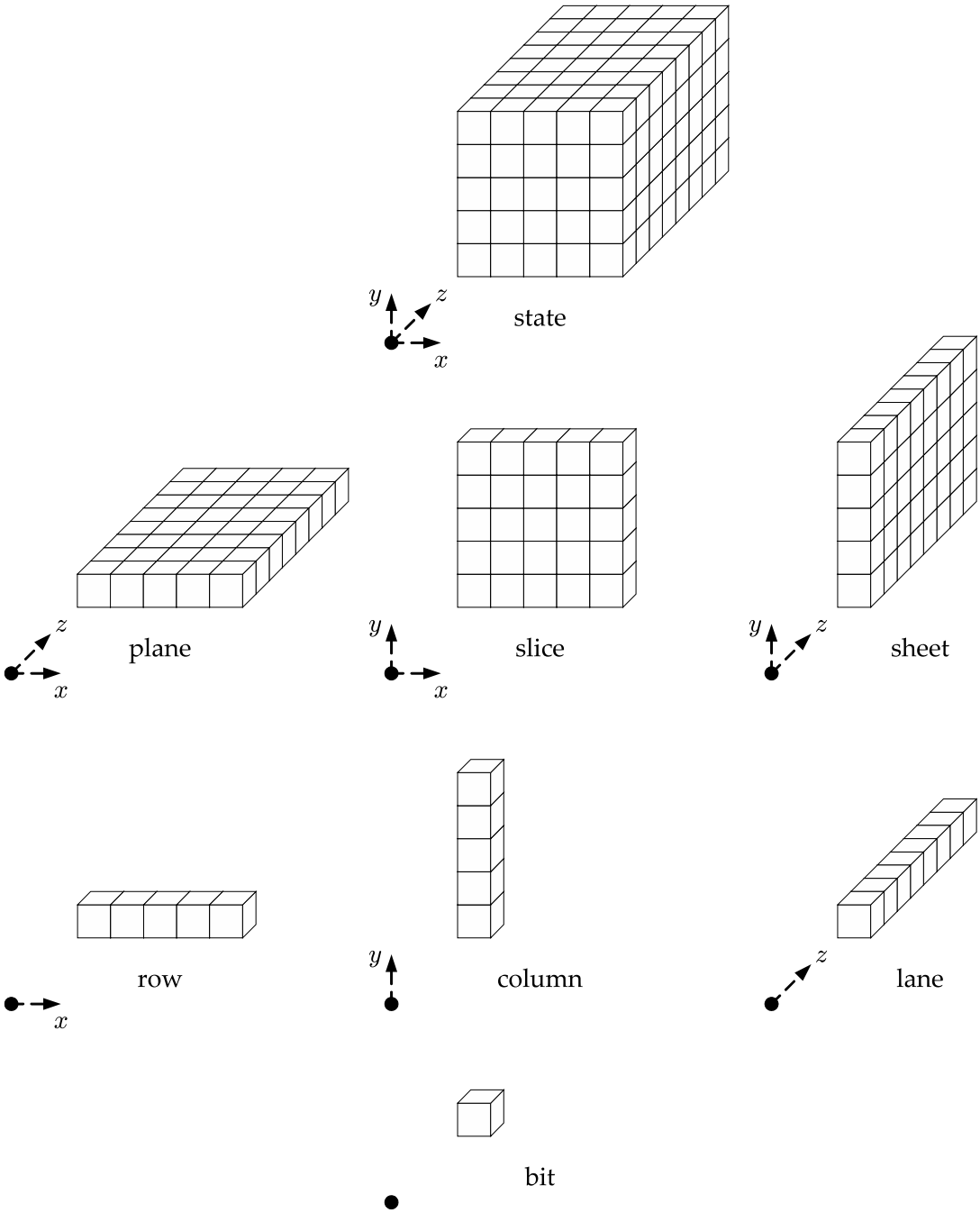

Figure 1.1: Naming conventions for parts of the Keccak-f state

Chapter 2

The KECCAK-f permutations

This chapter discusses the properties of the KECCAK-f permutations that are relevant for the security of KECCAK. After discussing some structural properties, we treat the different mappings that make up the round function. This is followed by a discussion of differential and linear cryptanalysis to motivate certain design choices. Subsequently, we briefly discuss the applicability of a number of cryptanalytic techniques to KECCAK-f.

2.1 Translation invariance

Let $b = \tau(a)$ with τ a mapping that translates the state by 1 bit in the direction of the z axis. For $0 < z < w$ we have $b[x][y][z] = a[x][y][z-1]$ and for $z = 0$ we have $b[x][y][0] = a[x][y][w-1]$. Translating over t bits gives $b[x][y][z] = a[x][y][(z-t) \bmod w]$. In general, a translation $\tau[t_x][t_y][t_z]$ can be characterized by a vector with three components (t_x, t_y, t_z) and this gives:

$$b[x][y][z] = a[(x - t_x) \bmod 5][(y - t_y) \bmod 5][(z - t_z) \bmod w] \ .$$

Now we can define *translation-invariance*.

Definition 2. *A mapping α is* translation-invariant *in direction (t_x, t_y, t_z) if*

$$\tau[t_x][t_y][t_z] \circ \alpha = \alpha \circ \tau[t_x][t_y][t_z] \ .$$

Let us now define the z-period of a state.

Definition 3. *The z-period of a state a is the smallest integer $d > 0$ such that:*

$$\forall x, y \in \mathbb{Z}_5 \text{ and } \forall z \in \mathbb{Z}_w : a[x][y][(z+d) \bmod w] = a[x][y][z] \ .$$

It is easy to prove the following properties:

- The z-period of a state divides w.
- A state a with z-period d can be represented by the lane size w, its z-period d, and its d first slices $a[.][.][z]$ with $z < d$. We call this the *z-reduced representation* of a.
- For a given w, the z-period defines a partition on the states.
- The number of states with z-period d is zero if d does not divide w and fully determined by d only, otherwise.
- For w values that are a power of two (the only ones allowed in KECCAK), the state space consists of the states with z-period 1, 2, 2^2 up to $2^\ell = w$.
- The number of states with z-period 1 is 2^{25}. The number of states with z-period 2^d for $d \geq 1$ is $2^{2^d 25} - 2^{2^{d-1} 25}$.
- There is a one-to-one mapping between the states a' with z-period d for any lane length w that is a multiple of d and the states a with z-period d of lane length d: $a'[.][.][z] = a[.][.][z \bmod d]$.
- If α is translation-invariant in the direction of the z axis, the z-period of $\alpha(a)$ divides the z-period of a. Moreover, the z-reduced state of $\alpha(a)$ is independent of w.
- If α is injective and translation-invariant in the direction of the z axis, α preserves the z-period.

2.2 The Matryoshka structure

With the exception of ι, all step mappings of the KECCAK-f round function are translation-invariant in the direction of the z axis. This allows the introduction of a size parameter that can easily be varied without having to re-specify the step mappings. As in several types of analysis abstraction can be made of the addition of constants, this allows the re-use of structures for small width versions as symmetric structures for large width versions. We refer to Section 2.4.2 for an example. As the allowed lane lengths are all powers of two, every smaller lane length divides a larger lane length. So, as the propagation structures for smaller width version are embedded as symmetric structure in larger width versions, we call it Matryoshka, after the well-known Russian dolls.

2.3 The step mappings of KECCAK-f

A round is composed from a sequence of dedicated mappings, each one with its particular task. The steps have a simple description leading to a specification that is compact and in which no trapdoor can be hidden.

Mapping the *lanes* of the state, i.e., the one-dimensional sub-arrays in the direction of the z axis, onto CPU words, results in simple and efficient software implementation for the step mappings. We start the discussion of each of the step mappings by pseudocode where the variables $a[x, y]$ represent the old values of lanes and $A[x, y]$ the new values. The operations on the lanes are limited to bitwise Boolean operations and rotations. In our pseudocode we denote by $\mathrm{ROT}(a, d)$ a translation of a over d bits where bit in position z is mapped to position $z + d \bmod w$. If the CPU word length equals the lane length, the latter can be implemented

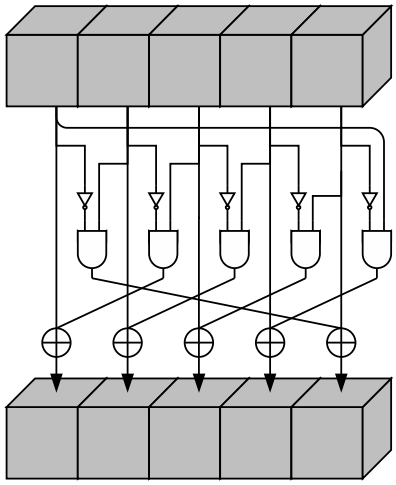

Figure 2.1: χ applied to a single row

with rotate instructions. Otherwise a number of shift and bitwise Boolean instructions must be combined or bit-interleaving can be applied [10].

In this section we discuss the difference propagation and input-output correlation properties of the different mappings. We refer to [8, Sections "Differential cryptanalysis" and "Linear cryptanalysis"] for an introduction of the terminology and concepts.

2.3.1 Properties of χ

Figure 2.1 contains a schematic representation of χ and Algorithm 2 its pseudocode.

Algorithm 2 χ

```
for y = 0 to 4 do
  for x = 0 to 4 do
    A[x, y] = a[x, y] ⊕ ((NOT a[x + 1, y]) AND a[x + 2, y])
  end for
end for
```

χ is the only nonlinear mapping in Keccak-f. Without it, the Keccak-f round function would be linear. It can be seen as the parallel application of $5w$ S-boxes operating on 5-bit rows. χ is translation-invariant in all directions and has algebraic degree two. This has consequences for its differential propagation and correlation properties. We discuss these in short in Sections 2.3.1.1 and Section 2.3.1.2 and refer to [20, Section 6.9] for an in-depth treatment of these aspects.

χ is invertible but its inverse is of a different nature than χ itself. For example, it does not have algebraic degree 2. We refer to [20, Section 6.6.2] for an algorithm for computing the inverse of χ.

χ is simply the complement of the nonlinear function called γ, that is used in RadioGatún [4], Panama [21] and several other ciphers [20]. We have chosen it for its simple nonlinear propagation properties, its simple algebraic expression and its low gate count: one XOR, one AND and one NOT operation per state bit.

2.3.1.1 Differential propagation properties

Thanks to the fact that χ has algebraic degree 2, for a given input difference a', the space of possible output differences forms a linear affine variety [19] with $2^{\mathrm{w_r}(a',b')}$ elements. Moreover, the cardinality of a differential (a', b') over χ is either zero or a power of two. The corresponding (restriction) weight $\mathrm{w_r}(a', b') = \mathrm{w_r}(a')$ is an integer that only depends on the input difference a'. A possible differential imposes $\mathrm{w_r}(a')$ linear conditions on the bits of input a.

We now provide a recipe for constructing the affine variety of output differences corresponding to an input difference, applied to a single row. Indices shall be taken modulo 5 (or in general, the length of the register). We denote by $\delta(i)$ a pattern with a single nonzero bit in position i and $\delta(i, j)$ a pattern with only non-zero bits in positions i and j.

We can characterize the linear affine variety of the possible output differences by an offset A' and a basis $\langle c_j \rangle$. The offset is $A' = \chi(a')$. We construct the basis $\langle c_j \rangle$ by adding vectors to it while running over the bit positions i:

- If $a'_i a'_{i+1} a'_{i+2} a'_{i+3} \in \{\cdot 100, \cdot 11 \cdot, 001 \cdot\}$, extend the basis with $\delta(i)$.
- If $a'_i a'_{i+1} a'_{i+2} a'_{i+3} = \cdot 101$, extend the basis with $\delta(i, i+1)$.

This algorithm is implemented in KECCAKTOOLS [9]. The (restriction) weight of a difference is equal to its Hamming weight plus the number of patterns 001. The all-1 input difference results in the affine variety of odd-parity patterns and has weight 4 (or in general the length of the register minus 1). Among the 31 non-zero differences, 5 have weight 2, 15 weight 3 and 11 weight 4.

A differential (a', b') leads to a number of conditions on the bits of the absolute value a. Let $B = A' \oplus b' = \chi(a') \oplus b'$, then we can construct the conditions on a by running over each bit position i:

- $a'_{i+1} a'_{i+2} = 10$ imposes the condition $a_{i+2} = B_i$.
- $a'_{i+1} a'_{i+2} = 11$ imposes the condition $a_{i+1} \oplus a_{i+2} = B_i$.
- $a'_{i+1} a'_{i+2} = 01$ imposes the condition $a_{i+1} = B_i$.

The generation of these conditions given a differential trail is implemented in KECCAKTOOLS [9].

2.3.1.2 Correlation properties

Thanks to the fact that χ has algebraic degree 2, for a given output mask u, the space of input mask v whose parities have a non-zero correlation with the parity determined by u form a linear affine variety. This variety has $2^{\mathrm{w_c}(v,u)}$ elements, with $\mathrm{w_c}(v, u) = \mathrm{w_c}(u)$ the (correlation) weight function, which is an even integer that only depends on the output mask u. Moreover, the magnitude of a correlation over χ is either zero or equal to $2^{-\mathrm{w_c}(u)/2}$.

We now provide a recipe for constructing the affine variety of input masks corresponding to an output mask, applied to a single row. Indices shall again be taken modulo 5 (or in general, the length of the register). We use the term *1-run of length ℓ* to denote a sequence of ℓ 1-bits preceded and followed by a 0-bit.

We characterize the linear affine variety with an offset U' and a basis $\langle c_j \rangle$ and build the offset and basis by running over the output mask. First initialize the offset to 0 and the basis to the empty set. Then for each of the 1-runs $a_s a_{s+1} \ldots a_{s+\ell-1}$ do the following:

- Add a 1 in position s of the offset U'.
- Set $i = s$, the starting position of the 1-run.
- As long as $a_i a_{i+1} = 11$ extend the basis with $\delta(i+1, i+3)$ and $\delta(i+2)$, add 2 to i and continue.
- If $a_i a_{i+1} = 10$ extend the basis with $\delta(i+1)$ and $\delta(i+2)$.

This algorithm is implemented in KeccakTools [9]. The (correlation) weight of a mask is equal to its Hamming weight plus the number of 1-runs of odd length. The all-1 output mask results in the affine variety of odd-parity patterns and has weight 4 (or in general the length of the register minus 1). Of the 31 non-zero mask, 10 have weight 2 and 21 have weight 4.

2.3.2 Properties of θ

Figure 2.2 contains a schematic representation of θ and Algorithm 3 its pseudocode.

Algorithm 3 θ

```
for x = 0 to 4 do
  C[x] = a[x, 0]
  for y = 1 to 4 do
    C[x] = C[x] ⊕ a[x, y]
  end for
end for
for x = 0 to 4 do
  D[x] = C[x − 1] ⊕ ROT(C[x + 1], 1)
  for y = 0 to 4 do
    A[x, y] = a[x, y] ⊕ D[x]
  end for
end for
```

The θ mapping is linear and aimed at diffusion and is translation-invariant in all directions. Its effect can be described as follows: it adds to each bit $a[x][y][z]$ the bitwise sum of the parities of two columns: that of $a[x-1][\cdot][z]$ and that of $a[x+1][\cdot][z-1]$. Without θ, the Keccak-f round function would not provide diffusion of any significance. The θ mapping has a branch number as low as 4 but provides a high level of diffusion on the average. We refer to Section 2.4.3 for a more detailed treatment of this.

In fact, we have chosen θ for its high average diffusion and low gate count: two XORs per bit. Thanks to the interaction with χ each bit at the input of a round potentially affects 31 bits at its output and each bit at the output of a round depends on 31 bits at its input. Note that without the translation of one of the two sheet parities this would only be 25 bits.

2.3.2.1 The inverse mapping

Computing the inverse of θ can be done by adopting a polynomial notation. The state can be represented by a polynomial in the three variables x, y and z with binary coefficients.

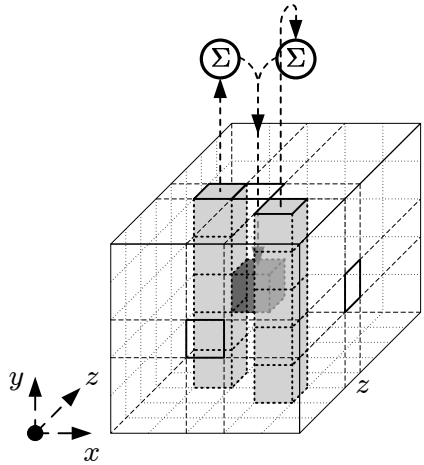

Figure 2.2: θ applied to a single bit

Here the coefficient of the monomial $x^i y^j z^k$ denotes the value of bit $a[i][j][k]$. The exponents i and j range from 0 to 4 and the exponent k ranges from 0 to $w-1$. In this representation a translation $\tau[t_x][t_y][t_z]$ corresponds with the multiplication by the monomial $x^{t_x} y^{t_y} z^{t_z}$ modulo the three polynomials $1+x^5$, $1+y^5$ and $1+z^w$. More exactly, the polynomial representing the state is an element of a polynomial quotient ring defined by the polynomial ring over GF(2)$[x,y,z]$ modulo the ideal generated by $\langle 1+x^5, 1+y^5, 1+z^w \rangle$. A translation corresponds with multiplication by $x^{t_x} y^{t_y} z^{t_z}$ in this quotient ring. The z-period of a state a is d if d is the smallest nonzero integer such that $1+z^d$ divides a. Let a' be the polynomial corresponding to the z-reduced state of a, then a can be written as

$$a = (1 + z^d + z^{2d} + \ldots + z^{w-d}) \times a' = \frac{1+z^w}{1+z^d} \times a' \ .$$

When the state is represented by a polynomial, the mapping θ can be expressed as the multiplication (in the quotient ring defined above) by the following polynomial :

$$1 + \bar{y}\left(x + x^4 z\right) \text{ with } \bar{y} = \sum_{i=0}^{4} y^i = \frac{1+y^5}{1+y} \ . \tag{2.1}$$

The inverse of θ corresponds with the multiplication by the polynomial that is the inverse of polynomial (2.1). For $w = 64$, we have computed this with the open source mathematics software SAGE [38] after doing a number of manipulations. First, we assume it is of the form $1 + \bar{y}Q$ with Q a polynomial in x and z only:

$$\left(1 + \bar{y}(x + x^4 z)\right) \times (1 + \bar{y}Q) = 1 \bmod \left\langle 1+x^5, 1+y^5, 1+z^{64} \right\rangle \ .$$

Working this out and using $\bar{y}^2 = \bar{y}$ yields

$$Q = 1 + (1 + x + x^4 z)^{-1} \bmod \left\langle 1+x^5, 1+z^{64} \right\rangle \ .$$

The inverse of $1 + x + x^4 z$ can be computed with a variant of the extended Euclidian algorithm for polynomials in multiple variables. At the time of writing this was unfortunately

not supported by SAGE. Therefore, we reduced the number of variables to one by using the change of variables $t = x^{-2}z$. We have $x = t^{192}$ and $x^4 z = t^{193}$, yielding:

$$Q = 1 + (1 + t^{192} + t^{193})^{-1} \bmod (1 + t^{320}) .$$

By performing a change in variables from t to x and z again, Q is obtained.

For $w < 64$, the inverse can simply be found by reducing Q modulo $1 + z^w$. For $w = 1$, the inverse of θ reduces to $1 + \bar{y}(x^2 + x^3)$.

For all values of $w = 2^\ell$, the Hamming weight of the polynomial of θ^{-1} is of the order $b/2$. This implies that applying θ^{-1} to a difference with a single active bit results in a difference with about half of the bits active. Similarly, a mask at the output of θ^{-1} determines a mask at its input with about half of the bits active.

2.3.2.2 Propagation of linear masks

A linear Boolean function defined by a mask u at the output of a linear function has non-zero correlation to a single linear Boolean function at its input. Given the matrix representation of the linear function, it is easy to express the relation between the input and output mask. Given $b = Ma$, we have:

$$u^{\mathrm{T}} b = u^{\mathrm{T}} M a = (M^{\mathrm{T}} u)^{\mathrm{T}} a .$$

It follow that $u^{\mathrm{T}} b$ is correlated to $v^{\mathrm{T}} a$ with $v = M^{\mathrm{T}} u$ with correlation 1. We say that a mask u at the output of a linear mapping M *propagates to* $v = M^{\mathrm{T}} u$ at its input. We denote the mapping defined by M^{T} the *transpose* of M.

As θ is linear, we have $v = \theta^{\mathrm{T}}(u)$, with u a mask at the output of θ, v a mask at its input and where θ^{T} the transpose of θ. We now determine the expression for the transpose of θ in the formalism of [5]. Let $b = \theta(a)$ and

$$\sum_{x,y,z} u[x][y][z] b[x][y][z] = \sum_{x,y,z} v[x][y][z] a[x][y][z] .$$

Filling in the value of $b[x][y][z]$ from the specification of θ in [5] and working this out yields:

$$\sum_{x,y,z} u[x][y][z] b[x][y][z] =$$

$$\sum_{x,y,z} \left(u[x][y][z] + \sum_{y'} u[x+1][y'][z] + \sum_{y'} u[x-1][y'][z+1] \right) a[x][y][z]$$

It follows that:

$$v = \theta^{\mathrm{T}}(u) \Leftrightarrow v[x][y][z] = u[x][y][z] + \sum_{y'} u[x+1][y'][z] + \sum_{y'} u[x-1][y'][z+1] \tag{2.2}$$

In polynomial notation the application of θ^{T} is a multiplication by

$$1 + \bar{y} \left(x^4 + x z^4 \right) .$$

2.3.3 Properties of π

Figure 2.3 contains a schematic representation of π and Algorithm 4 its pseudocode. Note that in an efficient program π can be implemented implicitly by addressing.

Algorithm 4 π

for $x = 0$ to 4 **do**
  for $y = 0$ to 4 **do**
    $\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$
    $A[X, Y] = a[x, y]$
  end for
end for

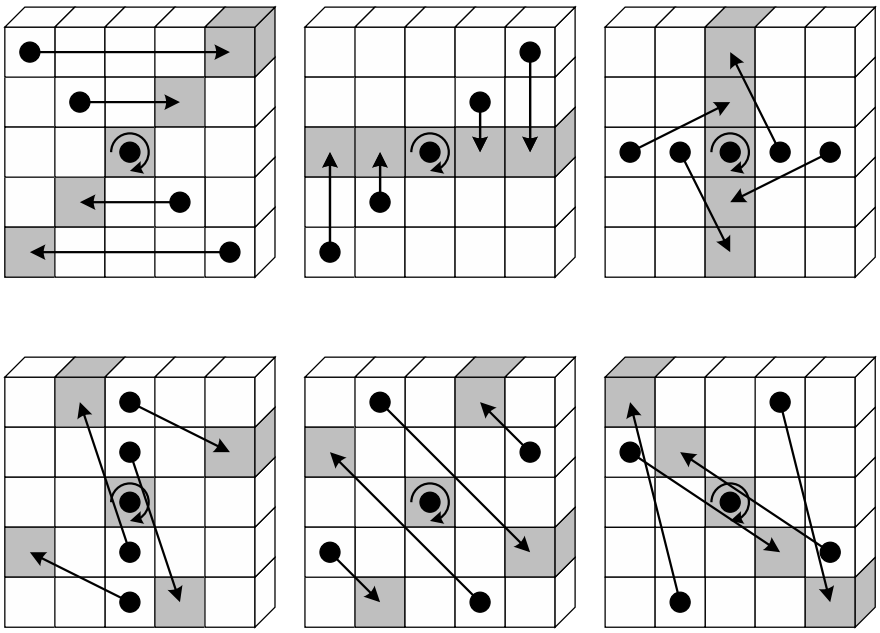

Figure 2.3: π applied to a slice. Note that $x = y = 0$ is depicted at the center of the slice.

The mapping π is a transposition of the lanes that provides dispersion aimed at long-term diffusion. Without it, Keccak-f would exhibit periodic trails of low weight. π operates in a linear way on the coordinates (x, y): the lane in position (x, y) goes to position $(x, y)M^{\mathrm{T}}$, with $M = \left(\begin{smallmatrix} 0 & 1 \\ 2 & 3 \end{smallmatrix}\right)$ a 2 by 2 matrix with elements in GF(5). It follows that the lane in the origin $(0, 0)$ does not change position. As π operates on the slices independently, it is translation-invariant in the z-direction. The inverse of π is defined by M^{-1}.

Within a slice, we can define 6 axes, where each axis defines a *direction* that partitions the 25 positions of a slice in 5 sets:

- x axis: rows or planes;
- y axis: columns or sheets;
- $y = x$ axis: rising 1-slope;
- $y = -x$ axis: falling 1-slope;
- $y = 2x$ axis: rising 2-slope;
- $y = -2x$ axis: falling 2-slope;

The x axis is just the row through the origin, the y axis is the column through the origin, etc.

There are many matrices that could be used for π. In fact, the invertible 2 by 2 matrices with elements in GF(5) with the matrix multiplication form a group with 480 elements containing elements of order 1, 2, 3, 4, 5, 6, 8, 10, 12, 20 and 24. Each of these matrices defines a permutation on the 6 axes, and equivalently, on the 6 directions. Thanks to its linearity, the 5 positions on an axis are mapped to 5 positions on an axis (not necessarily the same). Similarly, the 5 positions that are on a line parallel to an axis, are mapped to 5 positions on a line parallel to an axis.

For π we have chosen a matrix that defines a permutation of the axes where they are in a single cycle of length 6 for reasons explained in Section 2.4.6. Implementing π in hardware requires no gates but results in wiring.

As π is a linear function, a mask u at the output propagates to the mask v at the input with $v = \pi^{\mathrm{T}}(u)$ (see Section 2.3.2.2). Moreover, we have $\pi^{\mathrm{T}} = \pi^{-1}$, yielding $u = \pi(v)$. This follows directly from the fact that π is a bit transposition and that subsequently its matrix is orthogonal: $M^{\mathrm{T}}M = I$.

2.3.4 Properties of ρ

Figure 2.4 contains a schematic representation of ρ, while Table 2.1 lists its translation offsets. Algorithm 5 gives pseudocode for ρ.

Algorithm 5 ρ

$A[0, 0] = a[0, 0]$
$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
for $t = 0$ to 23 **do**
  $A[x, y] = \mathrm{ROT}(a[x, y], (t + 1)(t + 2)/2)$
  $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$
end for

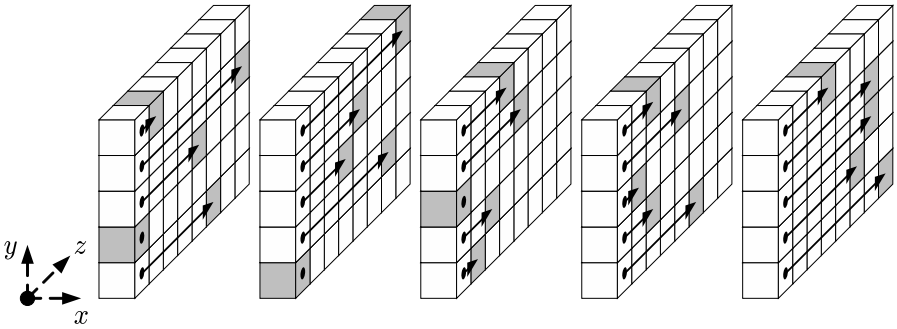

Figure 2.4: ρ applied to the lanes. Note that $x = y = 0$ is depicted at the center of the slices.

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	153	231	3	10	171
$y=1$	55	276	36	300	6
$y=0$	28	91	0	1	190
$y=4$	120	78	210	66	253
$y=3$	21	136	105	45	15

Table 2.1: The offsets of ρ

The mapping ρ consists of translations within the lanes aimed at providing inter-slice dispersion. Without it, diffusion between the slices would be very slow. It is translation-invariant in the z-direction. The inverse of ρ is the set of lane translations where the constants are the same but the direction is reversed.

The 25 translation constants are the values defined by $i(i+1)/2$ modulo the lane length. It can be proven that for any ℓ, the sequence $i(i+1)/2 \bmod 2^\ell$ has period $2^{\ell+1}$ and that any sub-sequence with $n2^\ell \leq i < (n+1)2^\ell$ runs through all values of $\mathbb{Z}_{2^\ell}$. From this it follows that for lane lengths 64 and 32, all translation constants are different. For lane length 16, 9 translation constants occur twice and 7 once. For lane lengths 8, 4 and 2, all translation constants occur equally often except the translation constant 0, that occurs one time more often. For the mapping of the (one-dimensional) sequence of translation constants to the lanes arranged in two dimensions x and y we make use of the matrix of π. This groups the lanes in a cycle of length 24 on the one hand and the origin on the other. The non-zero translation constants are allocated to the lanes in the cycle, starting from $(1, 0)$.

ρ is very similar to the transpositions used in RadioGatún[4], Panama [21] and StepRightUp [20]. In hardware its computational cost corresponds to wiring.

As ρ is a linear function, a mask u at the output propagates to the mask v at the input with $v = \rho^{\mathrm{T}}(u)$ (see Section 2.3.2.2). Moreover, we have $\rho^{\mathrm{T}} = \rho^{-1}$, yielding $u = \rho(v)$. This follows directly from the fact that ρ is a bit transposition and that subsequently its matrix is orthogonal: $M^{\mathrm{T}}M = I$.

2.3.5 Properties of ι

The mapping ι consists of the addition of round constants and is aimed at disrupting symmetry. Without it, the round function would be translation-invariant in the z direction and all rounds would be equal making Keccak-f subject to attacks exploiting symmetry such as slide attacks. The number of *active bit positions* of the round constants, i.e., the bit positions

in which the round constant can differ from 0, is $\ell + 1$. As ℓ increases, the round constants add more and more asymmetry.

The bits of the round constants are different from round to round and are taken as the output of a maximum-length LFSR. The constants are only added in a single lane of the state. Because of this, the disruption diffuses through θ and χ to all lanes of the state after a single round.

In hardware, the computational cost of ι is a few XORs and some circuitry for the generating LFSR. In software, it is a single bitwise XOR instruction.

2.3.6 The order of steps within a round

The reason why the round function starts with θ is due to the usage of Keccak-f in the sponge construction. It provides a mixing between the inner and outer parts of the state. Typically, the inner part is the part that is unknown to, or not under the control of the adversary. The order of the other step mappings is arbitrary.

2.4 Differential and linear cryptanalysis

In this section we discuss the differential and linear cryptanalysis aspects that have determined our choice of step mappings. For a more in-depth discussion on the propagation of differential and linear trails, we refer to Chapter 3.

2.4.1 A formalism for describing trails adapted to Keccak-f

The propagation of differential and linear trails in Keccak-f is very similar. Therefore we introduce a formalism for the description of trails that is to a large extent common for both types of trails. Differential trails describe the propagation of differences through the rounds of Keccak-f and linear trails the propagation of masks. We will address both with the term *patterns*.

As explained in Section 2.3.1, for a given difference a at the input of χ, the set of possible output differences is a linear affine variety. For a given mask a at the *output* of χ, the set of input masks with non-zero correlation to the given output mask is also a linear affine variety. Hence, to make the pattern propagation similar, for differential trails we consider the propagation from input to output and for linear trails we consider the propagation from output to input.

A difference at the input of χ is denoted by a_i and we call it a pattern *before* χ (in round i). A difference at the output of χ is denoted by b_i and we call it the pattern *after* χ. Similarly, a mask at the output of χ is denoted by a_i and we call it a pattern *before* χ. A mask at the input of χ is denoted by b_i and we call it the pattern *after* χ. In both cases we denote the linear affine variety of possible patterns after χ compatible with a_i by $\mathcal{B}(a_i)$.

Thanks to the fact that χ is the only nonlinear step in the round, a difference b_i after χ fully determines the difference a_{i+1} before χ of the following round: we have $a_i = \pi(\rho(\theta(b_i)))$. We denote the linear part of the round by λ, so:

$$\lambda = \pi \circ \rho \circ \theta \,.$$

Similarly, a mask b_i after χ fully determines the mask a_{i+1} before the χ of the following round. Now we have $a_i = \theta^{\mathrm{T}}(\rho^{\mathrm{T}}(\pi^{\mathrm{T}}(b_i))) = \theta^{\mathrm{T}}(\rho^{-1}(\pi^{-1}(b_i)))$. Here again, we denote this linear transformation by λ, so in this case we have:

$$\lambda = \theta^{\mathrm{T}} \circ \rho^{-1} \circ \pi^{-1} \,.$$

Note that the way $\mathcal{B}(a_i)$ is formed depends on whether we consider differential or linear trails. Moreover, the meaning of λ depends on whether we consider differential or linear trails.

Consider now the set obtained by applying λ to all elements of $\mathcal{B}(a_i)$. Thanks to the linearity of λ this is again a linear affine variety and we denote it by $\mathcal{A}(a_i)$.

We now define a ℓ-round *trail* Q by a sequence of state patterns a_i with $0 \leq i \leq \ell$. Every a_i denotes a state pattern before χ and a_i must be *compatible with* a_{i-1}, i.e., $a_i \in \mathcal{A}(a_{i-1})$. We use b_i to denote the patterns after χ, i.e., $a_{i+1} = \lambda(b_i)$. So we have:

$$a_0 \xrightarrow{\chi} b_0 \xrightarrow{\lambda} a_1 \xrightarrow{\chi} b_1 \xrightarrow{\lambda} a_2 \xrightarrow{\chi} b_2 \xrightarrow{\lambda} \ldots a_\ell. \tag{2.3}$$

The restriction weight of a differential trail Q is the number of conditions it imposes on the absolute values on the members of a right pair. It is given by

$$\mathrm{w_r}(Q) = \sum_{0 \leq i < \ell} \mathrm{w_r}(a_i) \ .$$

Note that the restriction weight of the last difference a_ℓ does not contribute to that of the trail. Hence the weight of any ℓ-round trail is fully determined by its ℓ first differences. For weight values well below the width of the permutation, a good approximation for the DP of a trail is given by $\mathrm{DP}(Q) \approx 2^{-\mathrm{w_r}(Q)}$. If $\mathrm{w_r}(Q)$ is near the width b, this approximation is no longer valid due to the fact that the cardinality of a trail is an integer. While the mapping ι has no role in the existence of differential trails, it does in general impact their DP. For trails with weight above the width, it can make the difference between having cardinality zero or non-zero.

The correlation weight of a linear trail over an iterative mapping determines its contribution to a correlation between output and input defined by the masks a_0 and a_ℓ. The correlation weight of a trail is given by

$$\mathrm{w_c}(Q) = \sum_{0 \leq i < \ell} \mathrm{w_c}(a_i) \ .$$

Here also the correlation weight of a_ℓ does not contribute and hence the weight of any ℓ-round trail is fully determined by its ℓ first masks. The magnitude of the correlation contribution of a trail is given by $2^{-\mathrm{w_c}(Q)}$. The sign is the product of the correlations over the χ and ι steps in the trail. The sign of the correlation contribution of a linear trail hence depends on the round constants.

In our analysis we focus on the weights of trails. As the weight of a ℓ-round trail is determined by its first ℓ patterns, in the following we will ignore the last pattern and describe ℓ-round trail with only ℓ patterns a_i, namely a_0 to $a_{\ell-1}$.

2.4.2 The Matryoshka consequence

The existence of trails (both differential and linear) and their weight is independent of ι. The fact that all other step mappings of the round function are translation-invariant in the direction of the z axis, makes that a trail Q implies $w - 1$ other trails: those obtained by translating the patterns of Q over any non-zero offset in the z direction. If all patterns in a trail have a z-period below or equal to d, this implies only $d - 1$ other trails.

Moreover, a trail for a given width b implies a trail for all larger widths b'. The patterns are just defined by their z-reduced representations and the weight must be multiplied by b'/b. Note that this is not true for the cardinality of differential trails and the sign of the correlation contribution of linear trails, as these do depend on the round constants.

2.4.3 The column parity kernel

The mapping θ is there to provide diffusion. As said, it can be expressed as follows: add to each bit $a[x][y][z]$ the bitwise sum of the parities of two columns: that of $a[x-1][\cdot][z]$ and that of $a[x+1][\cdot][z-1]$. From this we can see that for states in which all columns have even parity, θ is the identity. We call this set of states the *column parity kernel* or *CP-kernel* for short.

The size of the CP-kernel is 2^{20w} as there are in total $2^b = 2^{25w}$ states and there are 2^{5w} independent parity conditions. The kernel contains states with Hamming weight values as low as 2: those with two active bits in a single column. Due to these states, θ only has a branch number (expressed in Hamming weight) of 4.

The low branch number is a consequence of the fact that only the column parities propagate. One could consider changing θ to improve the worst-case diffusion, but this would significantly increase the computational cost of θ as well. Instead, we have chosen to address the CP-kernel issue by carefully choosing the mapping π.

We can compute from a $25w$-bit state its $5w$-bit *column parity pattern*. These patterns partition the state space in 2^{5w} subsets, called the *parity classes*, with each 2^{20w} elements. We can now consider the branch number restricted to the states in a given parity class. As said, the minimum branch number that can occur is 4 for the CP-kernel, the parity class with the all-zero column parity pattern. Over all other parity classes, the branch number is at least 12.

Note that for states where *all* columns have odd parity, θ adds 0 to every bit and also acts as the identity. However, the Hamming weight of states in the corresponding parity class is at least $5w$ resulting in a branch number of $10w$.

2.4.4 One and two-round trails

Now we will have a look at minimum weights for trails with one and two rounds. The minimum weight for a one-round differential trail (a_0) is obtained by taking a difference a_0 with a single active bit and has weight 2. For a linear trail this is obtained by a mask a_0 with a single active bit or two neighboring active bits in the same row, and the weight is also 2. This is independent of the width of Keccak-f.

For the minimum weight of two-round trails we use the following property of χ: if a difference before χ restricted to a row has a single active bit, the same difference is a possible difference after χ. Hence for difference with zero or one active bits per row, χ can behave as the identity. Similarly, for masks with zero or one active bits per row, χ can behave as the identity. We call such trails in which the patterns at the input and output of χ are the same, *χ-zero trails*. Note that all patterns in a χ-zero trail are fully determined by the first pattern a_0.

For all widths, the two-round trails with minimum weight are χ-zero trails. For a differential trail, we choose for a_0 a difference with two active bits that are in the same column. After χ the difference has not changed and as it is in the CP-kernel, it goes unchanged through θ as well. The mappings π and ρ move the two active bits to different columns, but in no case to the same row. This results in a value of a_1 with two active bits in different rows. As the weight of both a_0 and a_1 is 4, the resulting trail has weight 8. For linear trails, the two active bits in a_0 must be chosen such that after ρ and π they are in the same column. with a similar reasoning it follows that the minimum trail weight is also 8. Note that the low weight of these trails is due to the fact that the difference at the input of θ in round 0 is in the CP-kernel.

2.4.5 Three-round trails: kernel vortices

From here on, we concentrate on differential trails as the explanation is very similar for linear trails. We can construct a three-round χ-zero trail where both differences a_0 and a_1 are in the CP-kernel. As in a χ-zero trail χ behaves as the identity and a_0 is in the CP-kernel, we have $a_1 = \pi(\rho(a_0))$. Hence, we can transfer the conditions that a_0 is in the kernel to conditions on a_1, or vice versa.

We will now look for patterns a_0 where both a_0 and $\pi(\rho(a_0))$ are in the CP-kernel. a_0 cannot be a pattern with only two active bits in one column since $\pi \circ \rho$ maps these bits to two different columns in a_1.

The minimum number of active bits in a_0 is four, where both a_0 and a_1 have two active columns with two active bits each. We will denote these four active bits as *points* 0, 1, 2 and 3. Without loss of generality, we assume these points are grouped two by two in columns in a_0: $\{0,1\}$ in one column and $\{2,3\}$ in another one. In a_1 we assume they are grouped in columns as $\{1,2\}$ and $\{3,0\}$.

The mapping π maps sheets (containing the columns) to falling 2-slopes and maps planes to sheets. Hence the points $\{0,1\}$ and $\{2,3\}$ are in falling 2-slopes in a_1 and the points $\{1,2\}$ and $\{3,0\}$ are in planes in a_0. This implies that projected on the (x,y) plane, the four points of a_0 form a rectangle with horizontal and vertical sides. Similarly, in a_1 they form a parallelogram with vertical sides and sides that are falling 2-slopes.

The (x,y) coordinates of the four points in a_0 are completely determined by those of the two opposite corner points (x_0,y_0) and (x_2,y_2). The four points have coordinates: (x_0,y_0), (x_0,y_2), (x_2,y_2) and (x_2,y_0). The number of possible choices is $\binom{2}{5}^2 = 100$. Now let us have a look at their z coordinates. Points 0 and 1 should be in the same column and points 2 and 3 too. Hence $z_1 = z_0$ and $z_3 = z_2$. Moreover, ρ shall map points 1 and 2 to the same slice and bits 3 and 0 too. This results in the following conditions for their z-coordinates:

$$\begin{array}{rcll} z_0 + r[x_0][y_2] & = & z_2 + r[x_2][y_2] & \bmod w\,, \\ z_2 + r[x_2][y_0] & = & z_0 + r[x_0][y_0] & \bmod w\,, \end{array} \tag{2.4}$$

with $r[x][y]$ denoting the translation offset of ρ in position (x,y). They can be converted to the following two conditions:

$$\begin{array}{rl} z_2 = z_0 + r[x_0][y_2] - r[x_2][y_2] & \bmod w\,, \\ z_2 = z_0 + r[x_0][y_0] - r[x_2][y_0] & \bmod w\,. \end{array}$$

In any case z_0 can be freely chosen, and this determines z_2. Subtracting these two equations eliminates z_0 and z_2 and results in:

$$r[x_0][y_0] - r[x_0][y_2] + r[x_2][y_2] - r[x_2][y_0] = 0 \bmod w\,. \tag{2.5}$$

If this equation is not satisfied, the equations (2.4) have no solution.

Consider now $w = 1$. In that case Equation (2.5) is always satisfied. However, in order to be χ-zero, the points must be in different rows, and hence in different planes, both in a_0 and a_1, and this is not possible for a rectangle.

If $\ell \geq 1$, Equation (2.5) has a priori a probability of $2^{-\ell}$ of being satisfied. Hence, we can expect about $2^{-\ell}100$ rectangles to define a state a_0 with both a_0 and $\pi(\rho(a_0))$ in the CP-kernel. So it is not inconceivable that such patterns exists for $w = 64$. This would result in a 3-round trail with weight of 8 per round and hence a total weight of 24. However, for our choice of π and ρ, there are no such trails for $w > 16$.

Note that here also the Matryoshka principle plays. First, the z-coordinate of one of the points can be freely chosen and determines all others. So, given a rectangle that has a solution

for Equation (2.5), there are 2^ℓ patterns a_0, one for each choice of z_0. Second, if Equation (2.5) is not satisfied for ℓ but it is for some $\ell' < \ell$, it implies a pattern a_0 with $2^{\ell-\ell'}4$ points rather than 4 for which both a_0 and $\pi(\rho(a_0))$ are in the kernel.

These patterns can be generalized by extending the number of active bits: a pattern a_0 with both a_0 and $\pi(\rho(a_0))$ in the kernel can be constructed by arranging $2e$ points in a cycle in the (x, y) plane and giving the appropriate z-coordinates. In such a cycle each combination of points $\{2i, 2i+1\}$ are in the same sheet and each combination of points $\{2i+1, 2i+2\}$ are in the same plane. We call such a cycle of $2e$ (x, y) positions a *kernel vortex* V.

For the z coordinates, the conditions that the points $\{2i, 2i+1\}$ are in the same column in a_0 and the points $\{2i+1, 2i+2\}$ are in the same column in a_1 results in $2e$ conditions. Similar to the rectangle case, these conditions only have a solution if the ρ rotation constants in the lanes of the cycle satisfy a condition. For a given kernel vortex V, we define its depth $d(V)$ as:

$$d(V) = \sum_{i=0}^{2e-1} (-1)^i r[\text{point } i]. \tag{2.6}$$

Now, the vortex results in a valid pattern a_0 if $d(V) \bmod w = 0$. We call the largest power of 2 dividing $d(V)$ the *character* of the vortex $c(V)$. If $d(V) = 0$, we say its character is $c(V) = \infty$. Summarizing, a vortex V defines a valid pattern a_0 with $2e$ active bits for lane length $w \leq c(V)$. For constructing low-weight 3-round trails, it suffices to find vortices with small e and large character: given a vortex V it results in a 3-round trail with weight $12e$ for all values of $2^\ell \leq c(V)$ and with weight $12e2^\ell/c(V)$ for all values of $2^\ell > c(V)$ (using symmetric trails of period $c(V)$).

As the length of vortices grows, so does their number. There are 600 vortices of length 6, 8400 of length 8 and 104040 of length 10. The character $c(V)$ over these vortices has an exponential distribution: about half of them has character 1, $1/4$ have character 2, $1/8$ have character 4 and so on. It follows that as their length $2e$ grows, there are more and more vortices that result in valid pattern a_0 with $2e$ active bits, even for lane length 64.

Moreover, one can construct patterns a_0 containing two or more vortices, provided that they do not result in a row with two active bits in either a_0 or a_1. The character of such a combination is just the minimum of the characters of its component vortices. Clearly, due the large number of kernel vortices, it is likely that there are three-round trails with low weight for any choice of ρ and π. For our choice of π and ρ, the vortex that leads to the 3-round trail with the smallest weight for Keccak-f is one of length 6 and character 64. It results in a 3-round trail with weight 36.

2.4.6 Beyond three-round trails: choice of π

We will now try to extend this to four-round trails: we try to find patterns a_0 such that a_0, a_1 and a_2 are in the CP-kernel.

A vortex of length 4, i.e., with $e = 2$ cannot do the job with our choice of π: a rectangle in a_0 with sheets and planes as sides results in a parallelogram in a_1 with falling 2-slopes and columns as sides and in a parallelogram in a_2 with rising 2-slopes and falling 2-slopes as sides. Hence the four points in a_2 cannot group in columns 2 by 2 and therefore it cannot be in the kernel.

Consider now a vortex of length 6. We choose the points such that the grouping in columns is $\{0,1\}, \{2,3\}, \{4,5\}$ in a_0, it is $\{1,2\}, \{3,4\}, \{5,0\}$ in a_1 and $\{1,4\}, \{2,5\}, \{3,0\}$ in a_2. The grouping in a_1 simply implies that $\{1,2\}, \{3,4\}, \{5,0\}$ are grouped in planes in a_0. Actually, the first two groupings are similar to the three-round trail case: they determine a

character $c(V)$ and fix the z coordinates of all points but one. We will now study the implications of the grouping in a_2 on the (x, y) coordinates. Grouping in columns (sheets) in a_2 implies grouping in planes in a_1 and subsequently grouping in rising 1-slopes in a_0.

For the z-coordinates this results in 3 additional conditions: points 1 and 4, points 2 and 5 and points 3 and 0 must have the same z-coordinate in a_2. Similar to Equation (2.4) these conditions are equalities modulo 2^ℓ. For each of the equations, the a priori probability that it is satisfied for a given value of 2^ℓ is $2^{-\ell}$. With each of these equations we can again associate a character: the largest value w that is a power of two for which the equation is satisfied. The 4-round character (i.e. leading to a_0, a_1 and a_2 all three in the kernel) of the vortex in this context is now the minimum of the 3-round character (i.e. leading to both a_0 and a_1 in the kernel) of the vortex and the characters of the three additional equations. The probability that the 4-round character is larger than 2^ℓ is approximately $2^{-4(\ell+1)}$. It turns out that for our choice of π and ρ, 8 of the 50 candidate vortices have 4-round character 2 and the others have all 4-round character 1.

The conditions on the (x, y) coordinates imply that only vortices are suited that have an even number of active points in each sheet, each plane and each rising 1-slope. This limits the number of suitable vortices of length 6 to 50, of length 8 to 300, of length 10 to 4180 and of length 12 to 53750. To illustrate this, let us now study the number of activity patterns in the (x, y) coordinates of a_0 assuming there is only a single active bit in each lane. In total there are $2^{25} - 1$ nonzero patterns. If we impose the pattern to be in the CP-kernel, the parity of each sheet must be even, resulting in 5 independent linear equations. Hence there are $2^{20} - 1$ patterns in the kernel. Additionally requiring a_1 to be in the kernel imposes that the number of points in each plane of a_0 must be even. This adds 5 parity conditions. However, one is redundant with the ones due to a_0 as the total parity of the activity pattern over the state is even. Hence there are $2^{16} - 1$ such patterns. Additionally requiring a_2 to be in the kernel imposes that the number of points in each rising 1-slope of a_0 must be even. This adds again 5 new parity condition, with one of them redundant and reduces the number of possible patterns to $2^{12} - 1$. Since π runs through all directions, adding more rounds results in $2^8 - 1$, and $2^4 - 1$ and finally 0 patterns. It follows that the range of possible activity patterns shrinks exponentially as the number of rounds grows.

This is the main reason for choosing a π that runs through all axes in a single cycle. Consider a π that would map sheets to rising 1-slopes and rising 1-slopes back to sheets. For such a π there would be $2^{16} - 1$ activity patterns with a_0, a_1 and a_2 in the kernel. Moreover, this number would not decrease for more rounds and periodic χ-zero trails of low weight might appear.

When trying vortices with length above 6, the conditions on the z coordinates can be more involved. If in a particular sheet of a_2 the number of active points is 2, the condition is the same as for the case described above: their z coordinates should match. However, if there are 4, 6 or any even number of active points, there are several ways for them to be grouped in different columns. In general a character can be computed per sheet and the character of the complete structure is the minimum of all these characters. The character for a given sheet can be computed in a recursive way. The probability that an active sheet has character 1 is 1/2. For larger characters, the probability decreases faster with growing number of active bits in the character.

We have done tests for vortex lengths up to 14 and for constructions making use of two vortices totaling to about 1 million valid a_0 patterns. The vast majority have character 1, less than 13000 have character 2, 103 have character 4 and one has character 8. This last one is based on vortex of length 8 and it results in a 4-round trail with weight 512 in Keccak-f[1600].

2.4.7 Truncated trails and differentials

Truncated trails deal with the propagation of activity patterns rather than differences [30]. A partition of the state in sub-blocks is defined where the activity patterns describe whether a sub-block has no active bits (passive or 0) or has at least one active bit (active or 1). The structure of the state in Keccak-f suggests several bundlings. In a first order, one may take rows, columns, lanes, sheets, planes or slices as sub-blocks. We have gone through an exercise of attempting this but got stuck very soon for each of the choices. The problem is that for every choice, at least one of the step mappings completely tears apart the sub-blocks. We have also considered hybrid state definitions, such as the combination of row activities with column parities. However, in the cases that could be interesting, i.e., states with low weight (with respect to the truncation considered), this soon lead to the full specification of the difference.

In [29] truncated cryptanalysis was applied to RadioGatún [4], where the truncation was defined by a linear subspaces of the word vectors. In the attack it made sense as part of the RadioGatún state (the *belt*) is updated in a linear way. In Keccak-f the round function is uniformly non-linear and we do not believe that this approach can work.

2.4.8 Other group operations

We have considered differential and linear cryptanalysis while assuming the bitwise addition as the group operation. One may equivalently consider differential and linear properties with respect to a wide range of other group operations that can be defined on the state. However, for any other choice than the bitwise addition, θ becomes a nonlinear function and for most choices also ι, π and ρ become nonlinear. We do not expect this to lead to better results.

2.4.9 Differential and linear cryptanalysis variants

There are many attacks that use elements from differential cryptanalysis and/or linear cryptanalysis. Most are applied to block ciphers to extract the key. We have considered a number of techniques:

- Higher-order differentials [30]: the algebraic order of the Keccak-f round function is only two and therefore constructing higher-order differentials seems like an interesting route to structural distinguishers. We believed and still believe that due to the high average diffusion it is very difficult to construct higher-order differentials of practical significance for Keccak-f. However, the low degree of Keccak-f is indeed the cause for the distinguishers covering the highest number of rounds of Keccak-f to date. We discuss these in Section 4.4.

- Impossible differentials [41]: we expect the Keccak-f permutations to behave as random permutations. If so, the cardinality of differentials has a Poisson distribution with $\lambda = 1/2$ [25] and hence about 60 % of the differentials in Keccak-f will have cardinality 0, and so are impossible. However, given a differential (a, b), it is a priori hard to predict whether it is impossible. Settings in which one could exploit impossible differentials are keyed modes, where part of the input is fixed and unknown. In this case one would need truncated impossible differentials. If the number of rounds is sufficient to avoid low-weight differential trails, we believe this can pose no problem.

- Differential-linear attacks [33]: in these attacks one concatenates a differential over a number of rounds and a correlation over a number of subsequent rounds. We think that for reduced-round versions of Keccak-f differential-linear distinguishers are a candidate for the most powerful structural distinguisher. The required number of pairs

is of the order $\mathrm{DP}^{-2}\mathrm{LP}^{-2}$ with DP the differential probability of the distinguisher's differential and LP the square of the distinguisher's correlation. If we assume the differentials are dominated by a single low-weight differential trail, we have $\mathrm{DP} \approx 2^{-\mathrm{w_r}(Q_d)}$. Additionally, if we assume the correlation is dominated by a single low-weight linear trail, we have $\mathrm{DP} \approx 2^{-\mathrm{w_c}(Q_l)}$. This gives for the number of required pairs: $2^{2(\mathrm{w_r}(Q_d)+\mathrm{w_c}(Q_l))}$. The number of required pairs to exploit a trail in a simple differential or linear attack is of the order $2^{\mathrm{w_c}(Q)}$. Hence, over a number of rounds, the differential-linear distinguisher is more powerful than a simple differential or linear distinguisher if $\mathrm{w_r}(Q_d) + \mathrm{w_c}(Q_l) < \mathrm{w_c}(Q)/2$. Where Q is a trail over all rounds, Q_d a trail of the first n rounds and Q_l a trail over the remaining rounds. As we expect in the KECCAK-f variants with large width and a low number of rounds, the minimum trail weight tends to grow exponentially, and the chaining of two half-length trails is favored over a single full-length trail.

- (Amplified) Boomerang [39, 28] and rectangle attacks [11]: These attacks chain (sets of) differentials over a small number of rounds to construct distinguishers over a larger number of rounds. These are also likely candidates for good structural distinguishers, for the same reason as differential-linear ones.
- Integral cryptanalysis (Square attacks) [22]: this type of cryptanalysis lends itself very well to ciphers that treat the state in blocks. It was applied to bit-oriented ciphers in [42]. Based on the findings of that paper we estimate that it will only work on reduced-round versions of KECCAK-f with three to four rounds.

In this section we have limited ourselves to the construction of structural distinguishers. We have not discussed how these distinguishers can be used to attack the sponge function making use of the permutation. In [8, Section "Some structural distinguishers"], we discuss the applicability of structural distinguishers when attacking sponge functions.

2.5 Solving constrained-input constrained-output (CICO) problems

In [8, Section "The constrained-input constrained output problem"] we introduce CICO problems for permutations and show that the resistance of a sponge function against many attacks is determined by the hardness of solving these problems for their underlying permutation.

There are several approaches to solving a CICO problem for KECCAK-f. The most straightforward way is to use the KECCAK-f specification to construct a set of algebraic equations in a number of unknowns that represents the CICO problem and try to solve it. Thanks to the simple algebraic structure of KECCAK-f, constructing the algebraic equations is straightforward. As a matter of fact, this is supported by KECCAKTOOLS [9]. A single instance of KECCAK-f results in $(n_\mathrm{r} - 1)b$ intermediate variables and about as many equations. Each equation has algebraic degree 2 and involves about 31 variables. Solving these sets of equations is however not an easy task. This is even the case for the toy version KECCAK-f[25] with lane length $w = 1$. We refer to Section 4.2 for the results of some experiments for solving CICO-solving algebraically.

2.6 Strength in keyed mode

In keyed modes we must consider attack scenario's such as explained in [8, Section "Keyed modes"]. Here we see two main approaches to cryptanalysis. The first one is the exploita-

tion of structural distinguishers and the second one is an algebraic approach, similar to the one presented in Section 2.5. A possible third approach is the intelligent combination of exploiting a structural distinguisher and algebraic techniques. In our opinion, the strength in keyed modes depends on the absence of good structural distinguishers and the difficulty of algebraically solving sets of equations.

2.7 Symmetry weaknesses

Symmetry in the state could lead to properties similar to the complementation property of DES [35]. Symmetry between the rounds could lead to slide attacks. We believe that the asymmetry introduced by ι is sufficient to remove all exploitable symmetry from Keccak-f. We refer to Section 4.1.2 for some experimentally obtained evidence of this.

Chapter 3

Trail propagation in Keccak-f

As explained in [8, Section "Some structural distinguishers"], the existence of differential or linear trails in Keccak-f with a weight below the width of Keccak-f may result in a structural distinguisher of Keccak-f. In this chapter we report on our investigations related to lower bounds for weights of such trails.

In Section 3.1 we define different types of weight and in Section 3.2 we discuss the relevant properties of θ. In Section 3.3 we discuss the techniques used to come up with the lower bounds on trail weights and report on the results obtained. Finally, in Section 3.4 we report on experiments that allows us to get an idea what to expect for higher width values.

3.1 Relations between different kinds of weight

In this section, we recall or define the various kinds of weight we use in our treatment of trail propagation. We speak of patterns that may have the shape of a state, slice, plane, sheet, row, column, lane or even a single bit. Some types of weight are only defined for certain shapes and we will indicate it when this is the case.

We call a bit equal to 1 in a pattern an *active* bit and a bit equal to zero a *passive* bit.

Definition 4. *The* Hamming weight *of a pattern a is the number of active bits in the pattern and is denoted by $||a||$.*

For a generic definition of the restriction weight and the correlation weight we refer to [8, Sections "Differential cryptanalysis" and "Linear cryptanalysis"]. These two weights relate to the properties of χ, which are detailed in Section 2.3.1. Since χ operates on each row independently, the restriction and correlation weights of a pattern can be computed row per row and the results are summed. The weights for all row patterns are listed explicitly in Tables 3.1 and 3.2.

Definition 5. *The (propagation)* weight *of a pattern a, denoted by $\mathrm{w}(a)$, is a generic term for either the restriction weight or the correlation weight of a pattern before χ. Since χ operates on rows, the pattern a must consist of full rows implying that the weight is only defined for states, slices, planes and rows.*

Note that the size of the linear affine varieties $\mathcal{B}(a_i)$ and $\mathcal{A}(a_i)$ (see Section 2.3.1) is determined by the propagation weight of a_i:

$$|\mathcal{B}(a_i)| = |\mathcal{A}(a_i)| = 2^{\mathrm{w}(a_i)} .$$

Difference a	$\mathrm{w_r}(a)$	$\mathrm{w_r^{rev}}(a)$	$\|\|a\|\|$	$\|\|a\|\|_{\mathrm{row}}$
00000	0	0	0	0
10000	2	2	1	1
11000	3	2	2	1
10100	3	2	2	1
11100	4	2	3	1
11010	3	3	3	1
11110	4	3	4	1
11111	4	3	5	1

Table 3.1: Weights of all row differences (up to cyclic shifts)

Mask a	$\mathrm{w_c}(a)$	$\mathrm{w_c^{rev}}(a)$	$\|\|a\|\|$	$\|\|a\|\|_{\mathrm{row}}$
00000	0	0	0	0
10000	2	2	1	1
11000	2	2	2	1
10100	4	2	2	1
11100	4	2	3	1
11010	4	2	3	1
11110	4	2	4	1
11111	4	4	5	1

Table 3.2: Weights of all row masks (up to cyclic shifts)

Definition 6. *For a pattern b after χ, we define the* minimum reverse weight $\mathrm{w^{rev}}(b)$ *as the minimum weight over all compatible a. Namely,*

$$\mathrm{w^{rev}}(b) = \min_{a\,:\,b\in\mathcal{B}(a)} \mathrm{w}(a).$$

This weight applies to states, slices, planes and rows.

Given an ℓ-round trail $Q = (a_1 \dots a_\ell)$, it is easy to find the $\ell+1$-round trail $Q' = (a'_0, a'_1 \dots a'_\ell)$ with minimum weight such that $a_i = a'_i$ for $1 \le i \le \ell$. In this case, $\mathrm{w}(Q') = \mathrm{w}(Q) + \mathrm{w^{rev}}(\lambda^{-1}(a_1))$.

It may also be useful to express the *number of active rows* in a given pattern.

Definition 7. *For a pattern a before (or after) χ, the number of active rows, denoted by $\|a\|_{\mathrm{row}}$ is simply the number of rows whose value is non-zero. This weight applies to states, slices, planes and rows.*

The different kinds of weight for all row patterns are given in Tables 3.1 and 3.2. We now give some relations between the various kinds of weights. The following bounds relate the Hamming weight to the weight:

$$\hat{\mathrm{w}}(||a||) \triangleq ||a|| - \left\lfloor \frac{||a||}{5} \right\rfloor + [1 \text{ if } ||a|| = 1 \pmod 5] \le \mathrm{w}(a) \le 2||a||,$$

$$\left\lceil \frac{\mathrm{w}(a)}{2} \right\rceil \le ||a|| \le \left\lfloor \frac{5\mathrm{w}(a)}{4} \right\rfloor.$$

The following bounds relate the number of active rows to the weight:

$$2\|a\|_{\text{row}} \leq \mathrm{w}(a) \leq 4\|a\|_{\text{row}},$$
$$\left\lceil \frac{\mathrm{w}(a)}{4} \right\rceil \leq \|a\|_{\text{row}} \leq \left\lfloor \frac{\mathrm{w}(a)}{2} \right\rfloor.$$

Given the Hamming weight, the minimum reverse restriction weight can be lower bounded as follows:

$$\mathrm{w}_{\mathrm{r}}^{\mathrm{rev}}(a) \geq \hat{\mathrm{w}}_{\mathrm{r}}^{\mathrm{rev}}(||a||) \triangleq 3\left\lfloor \frac{||a||}{5} \right\rfloor + \begin{cases} 0 & \text{if } ||a|| = 0 \pmod 5, \\ 1 & \text{if } ||a|| = 1 \pmod 5, \\ 2 & \text{if } ||a|| = 2 \pmod 5, \\ 2 & \text{if } ||a|| = 3 \pmod 5, \\ 3 & \text{if } ||a|| = 4 \pmod 5. \end{cases}$$

Given the Hamming weight, the minimum reverse correlation weight can be lower bounded as follows:

$$\mathrm{w}_{\mathrm{c}}^{\mathrm{rev}}(a) \geq \hat{\mathrm{w}}_{\mathrm{c}}^{\mathrm{rev}}(||a||) \triangleq 2\left\lceil \frac{||a||}{4} \right\rceil.$$

Other relations on the minimum reverse weight follow:

$$\mathrm{w}^{\mathrm{rev}}(b) \leq 2||b||,$$
$$\left\lceil \frac{\mathrm{w}^{\mathrm{rev}}(b)}{2} \right\rceil \leq ||b|| \leq \left\lfloor \frac{5\mathrm{w}^{\mathrm{rev}}(b)}{4} \right\rfloor,$$
$$2\|b\|_{\text{row}} \leq \mathrm{w}^{\mathrm{rev}}(b) \leq 4\|(\|_{\text{row}} b),$$
$$\left\lceil \frac{\mathrm{w}^{\mathrm{rev}}(b)}{4} \right\rceil \leq \|b\|_{\text{row}} \leq \left\lfloor \frac{\mathrm{w}^{\mathrm{rev}}(b)}{2} \right\rfloor.$$

And finally,

$$\mathrm{w}^{\mathrm{rev}}(b) \leq \mathrm{w}(b).$$

Definition 8. *A weight function $f(a)$ is said to be* monotonous *if setting a passive bit of a pattern a to active does not decrease $f(a)$. More formally, let the partial ordering $a \leq a'$ be defined as $a[x][y][z] = 1 \Rightarrow a'[x][y][z] = 1$. Then, f is a monotonous weight if $\forall a, a' : a \leq a'$, we have $f(a) \leq f(a')$.*

All weights defined in this section are monotonous. This follows directly from Tables 3.1 and 3.2.

3.2 Propagation properties related to the linear step θ

Definition 9. *The* column parity *(or* parity *for short) $P(a)$ of a pattern a is a pattern $P(a) = p[x][z]$ defined as the parity of the columns of a, namely $p[x][z] = \sum_y a[x][y][z]$. The parity is defined for states, slices, sheets and columns. The parity of a state has the shape of a plane, the parity of a slice has the shape of a row, the parity of a sheet has the shape of a lane and the parity of a column is a bit.*

Equivalently, the parity is the result of applying the operator $\bar{y}$ (see Section 2.3.2), and the value C in Algorithm 3 is the parity of a state. A column is *even* (resp. *odd*) if its parity is 0 (resp. 1). When the parity of a pattern is zero (i.e., all its columns are even), we say it is in the *CP-kernel*, as in Section 2.4.3. Note that the column parity defines a partition on the set of possible states.

Definition 10. *The θ-effect of a state a before θ is a pattern $E(a)[x][z]$ defined as the result of applying the operator $\bar{y}(x + x^4 z)$ to the state, or equivalently by applying the operator $(x + x^4 z)$ to its parity, i.e., $E(a)[x][z] = P(a)[x-1][z] + P(a)[x+1][z-1]$.*

In difference propagation, the θ-effect is also the value of D in Algorithm 3. In mask propagation, the θ-effect is defined by θ^{T} rather than θ itself and the expression becomes $E(a)[x][z] = P(a)[x+1][z] + P(a)[x-1][z+1]$ (see Section 2.3.2.2). Note that the θ-effect always has an even Hamming weight. A column of coordinates (x, z) is *affected* iff $E(a)[x][z] = 1$; otherwise, it is *unaffected*.

The θ-effect entirely determines the effect of applying θ to a state a. For a fixed θ-effect $e[x][z]$, θ just adds a constant pattern $e[x][y][z]$ of bits to the state, constant in each column, namely $e[x][y][z] = e[x][z]$ for all y. Since the θ-effect has even Hamming weight, it means that the number of affected columns is even.

Definition 11. *The θ-gap is defined as the Hamming weight of the θ-effect divided by two.*

Hence, if the θ-gap of a state at the input of θ is g, the number of affected columns is $2g$ and applying θ to it results in $10g$ bits being flipped.

When a state is in the CP-kernel, the θ-gap is zero. However, the θ-gap is also zero when the parity is all-one, i.e., when all columns have odd parity before θ.

We have defined the θ-gap using the θ-effect, but it can also be defined using the parity itself. For this, we need to represent the parity $p[x][z]$ differently. We map the (x, z) coordinates to a single coordinate t as specified in Section 2.3.2.1 (i.e., t goes to $(x, z) = (-2t, t)$) and denote the result by $p[t]$.

In this representation, a *run* is defined as a sequence R of consecutive t-coordinates, $R = \{s, s+1, \ldots, s+n-1\}$, such that $p[s-1] = 0$, $p[t] = 1 \forall t \in R$ and $p[s+n] = 0$. The following lemma links the number of runs to the θ-gap.

Lemma 1. *The parity p has θ-gap g iff $p[t]$ has g distinct runs.*

3.3 Exhaustive trail search

In Section 3.3.2 we will show an efficient method for generating all two-round trails up to a given propagation weight T_2. We will then show in section Section 3.3.3 how these trails can be extended to more rounds. In general, we want to generate all ℓ-round trails up to some propagation weight T_ℓ. We start with Section 3.3.1 deriving the minimum value of T_2 given ℓ and T_ℓ. Finally, we report on our bounds obtained in Section 3.3.4

3.3.1 Upper bound for the weight of two-round trails to scan

The idea is have bounds on ℓ-round trails by starting from all two-round trails of weight up to T_2 and extending them both forwards and backwards. More precisely, each two-round trail is extended n rounds backwards and $\ell - 2 - n$ rounds forwards, for each value of $n \in \{0, \ldots, \ell - 2\}$. This way, we cover all trails that have a two-round subtrail with weight up to T_2.

Lemma 2. *To list all ℓ-round trails of weight not higher than T_ℓ exhaustively, it is necessary to start from all 2-round trails with weight up to (and including) T_2, with $T_2 = \left\lfloor \frac{2T_\ell}{\ell} \right\rfloor$ if ℓ is even, or $T_2 = \left\lfloor \frac{2(T_\ell - 2)}{\ell - 1} \right\rfloor$ if ℓ is odd.*

Proof. For a sequence of weights $W = w_1, w_2, \ldots, w_\ell$, let $\delta(W) = \min_{i=1\ldots\ell-1}(w_i + w_{i+1})$. We want to make sure that $\delta(W) \leq T_2$ for all W such that $\sum_i w_i \leq T_\ell$.

If ℓ is even, $T_\ell \geq \sum_{i=1}^{\ell} w_i = \sum_{j=1}^{\ell/2}(w_{2j-1} + w_{2j}) \geq \frac{\ell}{2}\delta(W)$ and thus $\delta(W) \leq \frac{2T_\ell}{\ell}$. Setting $T_2 = \left\lfloor \frac{2T_\ell}{\ell} \right\rfloor$ satisfies the condition.

If ℓ is odd, we can always assume that $w_\ell \geq 2$ (as in any non-trivial trail) and thus we can consider the same problem with sequences of $\ell - 1$ weights and $T_{\ell-1} = T_\ell - 2$. □

So we have all the necessary ingredients to make exhaustive search of trails up to a given weight, within the limits of a reasonable computation time, and we can use that to find trails with minimum weight.

3.3.2 Constructing two-round trails

In this section we describe an efficient method for constructing all two-round trails (a_0, a_1) with $\mathrm{w}(a_0) + \mathrm{w}(a_1) \leq T_2$ for a given value T_2. For any such trail, we know that there exists a trail (a_0', a_1) with $\mathrm{w}(a_0') = \mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(a_1)) \leq \mathrm{w}(a_0)$. The quantity $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(a)) + \mathrm{w}(a) \leq T_2$ imposes a lower bound on the weight of a 2-round trail trail that has a as its state before χ of the second round and we give it the following name.

Definition 12. *The* propagation branch number *of a state a before χ is denoted by $B_{\mathrm{p}}(a)$ and given by:*

$$B_{\mathrm{p}}(a) = \mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(a)) + \mathrm{w}(a) \ .$$

Hence, rather than explicitly constructing all two-round trails, we can generate the set of states a such that $B_{\mathrm{p}}(a) \leq T_2$. We call this set $\alpha(T_2)$. It contains the second members a_1 of all two-round trails Q with $\mathrm{w}(Q) \leq T_2$. Each state in this set $\alpha(T_2)$ then serves as a starting point for trail extension. For the resulting trails, we know that the subtrail (a_{i-1}, a_i) with $a_i = a$ has a weight of at least $B_{\mathrm{p}}(a)$.

In generating $\alpha(T_2)$ we use a strategy that exploits the monotonicity of the propagation weight and the properties of θ in terms of the Hamming weight of its input/output. In Section 3.1 we listed equations providing lower bounds on the weight and reverse minimum weight of a state as a function of its Hamming weight. Hence, $||a||$ and $||\lambda^{-1}(a)||$ gives a lower bound on $B_{\mathrm{p}}(a)$.

Definition 13. *The* Hamming branch number *of a state a before χ is denoted by $B_{\mathrm{h}}(a)$ and given by:*

$$B_{\mathrm{h}}(a) = ||\lambda^{-1}(a)|| + ||a|| \ .$$

We can now just generate all states up to some given propagation branch number by generating all states up to a sufficiently high Hamming branch number value.

Now let a' be the state before θ corresponding with a. In difference propagation, we have $a' = \lambda^{-1}(a)$ and in mask propagation this is $a' = \theta^{\mathrm{T}^{-1}}(a)$. It follows that in difference propagation, we have $||a'|| = ||\lambda^{-1}(a)||$ and $||\theta(a')|| = ||a||$ and hence $B_{\mathrm{h}}(a) = ||a'|| + ||\theta(a')||$. In mask propagation we can similarly derive $B_{\mathrm{h}}(a) = ||a'|| + ||\theta^{\mathrm{T}}(a')||$. Hence we can instead move our reference from a to a' and just compute the value of a from a'. With a slight abuse of notation we will write $B_{\mathrm{h}}(a')$ and $B_{\mathrm{p}}(a')$ to denote $B_{\mathrm{h}}(a)$ and $B_{\mathrm{p}}(a)$.

Consider now the set of states a' with a given parity p, i.e., $P(a') = p$. As the θ-effect e is fully determined by the parity, all these states have the same parity effect. It follows that θ is reduced to the addition of a constant value, facilitating the computation of $B_{\mathrm{h}}(a')$ and the deduction of lower bounds on $B_{\mathrm{p}}(a')$.

Similar to the Hamming branch number of a state, we can define the Hamming branch number of a parity.

Definition 14. *The Hamming branch number of a parity p before θ is defined as the minimum Hamming branch number over all states with the given parity:*

$$B_{\mathrm{h}}(p) = \min_{a' : P(a')=p} B_{\mathrm{h}}(a') \ .$$

In a state a' we can use its parity p to partition its columns $a'[x][z]$ in four kinds: odd ($P(a')[x][z] = 1$) and even ($P(a')[x][z] = 0$), combined with affected ($E(a')[x][z] = 1$) and unaffected ($E(a')[x][z] = 0$). We can use this to easily compute $B_{\mathrm{h}}(p)$ for any parity p.

- An unaffected odd column has at least one active bit before θ and is preserved after it. Hence, it contributes at least 2 to $B_{\mathrm{h}}(a')$. As this minimal case can be constructed, the contribution to $B_{\mathrm{h}}(p)$ is strictly equal to 2.
- An affected (odd or even) column having n active bits before θ has $5 - n$ bits afterwards, hence contributes exactly 5 to $B_{\mathrm{h}}(a')$ and to $B_{\mathrm{h}}(p)$.
- An unaffected even column can have zero active bits, hence does not contribute to $B_{\mathrm{h}}(p)$.

Hence, it turns out that

$$B_{\mathrm{h}}(p) = 5||E(p)|| + 2||p \cdot (\bar{0} + E(p))|| = 10g + 2u_{\mathrm{o}},$$

with g the θ-gap of p, $\cdot$ the componentwise product, $\bar{0}$ the all-1 state and u_{o} the number of unaffected odd columns.

We can now generate all states a' up to some given propagation branch number and with given parity p in two phases.

- In a first phase we generate all states a' with $B_{\mathrm{h}}(a') = B_{\mathrm{h}}(p)$. We call those states *branch-parity-minimal*.
- In a second phase, we can generate states a' that are not branch-parity-minimal by taking branch-parity-minimal states and adding pairs of active bits in columns such that the parity is unchanged.

The generation of branch-parity-minimal states is done as follows:

- For each unaffected odd column, put a single active bit. There are 5 possibilities: one for each positions y.
- For each affected even column, put an even number of active bits. There are 2^4 possibilities.
- For each affected odd column, put an odd number of active bits. There are in 2^4 possibilities.

The number of branch-parity-minimal states a' with given parity p is thus $2^{8g}5^{|p(1+e)|}$.

From the monotonicity of the weights it follows that in the set of states a' with given parity p, the subset of branch-parity-minimal states contain the states that minimize $B_{\mathrm{p}}(a')$. Adding a pair of active bits in a single column of a' leaves its parity intact and thanks to the monotonicity cannot decrease $B_{\mathrm{p}}(a')$. From this, we devise the following strategy to generate all states a' with given parity p with $B_{\mathrm{p}}(a') \leq T_2$.

For each branch-parity-minimal state a' with $B_{\mathrm{p}}(a') \leq T_2$ do the following:

- Output a'.
- Iteratively construct states a' by adding pairs of active bits in each column, as long as $B_{\mathrm{p}}(a') \leq T_2$. To avoid duplicates the active bits shall have y coordinates with larger values than any active bits in the columns.

To generate all states a' for which $B_{\mathrm{p}}(a') \leq T_2$ we must do this for all parities with a small enough θ-gap. Actually, we can compute a lower bound on $B_{\mathrm{p}}(a')$ given only $P(a')$. We then have to consider only those parities for which this bound is lower than or equal to T_2. We compute it in the following way. First, we consider the Hamming branch number $B_{\mathrm{h}}(p)$ and assume that $|\lambda^{-1}(a)| = B_{\mathrm{h}}(p) - n$ and $|a| = n$ for some value n. Then, we use the bounds found in Section 3.1 and minimize over n. Note that we have checked that the minimum is always at $n = 1$, hence:

$$\begin{aligned} B_{\mathrm{p}}(a') &\geq \min_{n \in \{1 \ldots B_{\mathrm{h}}(p)-1\}} \hat{\mathrm{w}}^{\mathrm{rev}}(B_{\mathrm{h}}(p) - n) + \hat{\mathrm{w}}(n) \\ &= \hat{\mathrm{w}}^{\mathrm{rev}}(B_{\mathrm{h}}(p) - 1) + 2 \\ &= \hat{\mathrm{w}}^{\mathrm{rev}}(10g + 2u_{\mathrm{o}} - 1) + 2. \end{aligned}$$

Hence the θ-gap of the parity p imposes a lower bound to the propagation branch number of a state.

Then, it is possible to determine the maximum θ-gap $g_{\max}$ above which the lower bound is above T_2. If we further relate $g_{\max}$ to the number of runs in the parity, as in Lemma 1, we can generate all possible parities we need to consider by generating those with up to $g_{\max}$ runs.

Notice that both χ and λ are invariant by translation along z. It is thus necessary to keep only a single member of the states (or parities) in $\alpha(T_2)$ that are equal modulo the translations along z.

3.3.3 Extending trails

We can now use the elements in $\alpha(T_2)$ to recursively generate longer and longer trails up to some given length and some given weight. We can extend the trails in two directions: forward and backward. Given a trail Q, extending the trail forward (resp. backward) means

constructing trails Q' of which Q is a prefix (resp. suffix). It can also be in both directions, i.e., adding a number of steps as a prefix and another number of steps as a suffix of Q.

In the forward direction, the general idea is the following. Given the last state $a_{\ell-1}$ of the trail Q, we characterize the affine space $\mathcal{A}(a_{\ell-1})$ as an offset and a basis, i.e.,

$$\mathcal{A}(a_\ell) = s + \langle t_1, t_2, \ldots, t_{\mathrm{w}(a_\ell)} \rangle \ .$$

We can then loop through the affine space, produce the state values a_ℓ and check the weight $\mathrm{w}(a_\ell)$. If the weight is low enough for the extended trail to be interesting in the search, we can append a_ℓ to Q and recursively continue the search from there if necessary.

In the backward direction, we cannot use an affine space representation of a_{-1} as a function of a_0. As the weight is determined by the to-be-found state a_{-1}, we can list, for each active row of b_0, the possible input rows and their corresponding weight in increasing order. The weight of a_{-1} is the sum of the weights of each individual row, and we can take advantage of this to choose input rows such that the weight stays below or equal to a given threshold. We set each active row to the input row with lowest weight, the total weight $\mathrm{w}(a_{-1})$ being equal to $w^{\mathrm{rev}}(b_0)$ for the output state b_0. Then, we can generate all other input states a_{-1} by looping through the input rows, locally knowing up to which weight we can go.

3.3.4 Linear and differential trail bounds for $w \leq 8$

We have investigated the different instances of KECCAK-$f[b]$ starting from the smallest widths $b = 25w$, resulting in the lower bounds for trail weights listed in Table 3.3. Thanks to the Matryoshka structure, a lower bound on trails for KECCAK-$f[b]$ implies a lower bound on symmetric trails for all larger widths. More specifically, a differential or linear trail for KECCAK-$f[25w]$ with weight W corresponds with a w-symmetric trail for KECCAK-$f[25w']$ with weight $W' = W\frac{w'}{w}$. For instance, in Table 3.3 the column DC, $w = 8$ expresses a lower bound of 46 on the weight of 4-round trails in KECCAK-$f[200]$. This also expresses a lower bound for 4-round symmetric trails in KECCAK-f versions with larger width: 92 for 2-symmetric trails in KECCAK-$f[400]$, 184 for 4-symmetric trails in KECCAK-$f[800]$ and 268 for 8-symmetric trails in KECCAK-$f[1600]$.

For $w = 1$, five rounds are sufficient to have no trails with weight below 25, the width of the permutation. For $w = 2$, six rounds are sufficient to have no differential trails with weight below the width. It can be observed that as the number of rounds grows, the difference between the bounds for width 25 and those for width 50 grows. We expect this effect to be similar for larger widths.

For $w = 4$, the search was complete up to weight 36 and 38 for 4 rounds, for differential and linear trails respectively:

- For 4 rounds, the differential trail with minimum weight has weight 30. For the small number of trails found up to weight 36, we checked that these trails cannot be chained together. Hence, this guarantees that a 8-round differential trail has at least weight $36 + 37 = 73$. For 5 and 6 rounds, the best trails we have found so far have weight 54 and 85, respectively (but these do not provide bounds). For the 16 rounds of KECCAK-$f[100]$, we can guarantee that there are no differential trails of weight below $2 \times 73 = 146$.

- For 4 rounds, the linear trail with minimum weight has weight 38. For 5 and 6 rounds, the best trails we have found so far have weight 66 and 94, respectively (but these do not provide bounds). For the 16 rounds of KECCAK-$f[100]$, we can guarantee that there are no linear trails of weight below $4 \times 38 = 152$.

Number of rounds	DC $w=1$	DC $w=2$	DC $w=4$	DC $w=8$	LC $w=1$	LC $w=2$	LC $w=4$	LC $w=8$
2	8	8	8	8	8	8	8	8
3	16	18	19	20	16	16	20	20
4	23	29	30	46	24	30	38	46
5	30	42	≤ 54		30	40	≤ 66	
6	37	54	≤ 85		38	52	≤ 94	

Table 3.3: Minimum weight of w-symmetric trails

For $w = 8$, the search was complete up to weight 49 and 48 for 4 rounds, for differential and linear trails respectively:

- For 4 rounds, the differential trail with minimum weight has weight 46. For the small number of trails found up to weight 49, we checked that these trails cannot be chained together. Hence, this guarantees that a 8-round differential trail has at least weight $49 + 50 = 99$. For the 18 rounds of Keccak-$f[200]$, we can guarantee that there are no differential trails of weight below $2 \times 99 + 8 = 206$.

- For 4 rounds, the linear trail with minimum weight has weight 46. For the small number of trails found up to weight 48, we checked that these trails cannot be chained together. Hence, this guarantees that a 8-round differential trail has at least weight $48 + 50 = 98$. For the 18 rounds of Keccak-$f[200]$, we can guarantee that there are no linear trails of weight below $8 \times 98 + 8 = 204$.

3.4 Tame trails

In this section we report on our investigations related to the search for 3-round and 4-round differential trails with low weight and for high width. In this context, we consider differential trails for which the intermediate states b_i are in the CP-kernel and call such trails *tame*. Linear trails are expected to behave in qualitatively the same way.

This is a generalization of the kernel vortices introduced in Section 2.4.5 where only patterns are considered for which both χ and θ behave as the identity. In kernel vortices only intermediate patterns at the input of χ where considered with a single active bit per row. In the trails considered in this section, this restriction is no longer present.

3.4.1 Construction of tame trails

Let us start with 3-round differential trails. The weight of such a trail is defined by three states: (a_0, a_1, a_2). This trail is tame if $b_0 = \lambda^{-1}(a_1)$ and $b_1 = \lambda^{-1}(a_2)$ are both in the CP-kernel. We have written a program that generates all values of a_1 of a given Hamming weight, such that b_0 is in the CP-kernel and there is at least one three-round differential trail (a_0, a_1, a_2) that is tame, i.e., with b_1 in the CP-kernel. The latter condition imposes that a_1 must be such that the intersection of $\mathcal{A}(a_1)$ and the CP-kernel is not empty. As χ operates on rows and the CP-kernel is determined by individual columns, this can be treated slice by slice. In other words, every slice of a_1 must be such that its linear affine variety of possible output patterns contains patterns in the CP-kernel.

We call such a slice *tame* and a state with only tame slices also tame. For slice patterns with few active bits it can be easily verified whether it is tame. A slice with no active bits is tame, a slice with a single active bit can never be tame and a slice with two active bits is tame iff the active bits are in the same column. As the number of active bits grows, the proportion of slice patterns that are not tame decreases exponentially. As there are only 2^{25} different slice patterns, the tameness check can be precomputed and implemented by a simple table-lookup. Generating all states a_1 of a given (small) Hamming weight, that are tame and for which b_0 is in the CP-kernel can be done efficiently.

We can now construct all valid states a_1 as the combination of a number of *kernel chains*. A kernel chain is set of active bits determined by a sequence of bit positions c_i in a_1. A kernel chain forms a set of active bits with the following properties:

- When moved to position b_0 the kernel chain it is in the CP-kernel.
- In position a_1 every slice contains exactly two bits of the kernel chain and is tame, except the slice containing the initial bit c_0 and the slice containing the final bit c_{2n+1}, that each just contain a single slice and are not tame.

Actually, a kernel chain is a generalization of a kernel vortex. It follows that in b_0, the bits c_{2i} and c_{2i+1} are in the same column and in a_1, the bits c_{2i+1} and c_{2i+2} are in the same column. This implies that the total number of kernel chains of a given length $2n$ starting from a given position is only 4^{2n-1}. Clearly, any combination of kernel chains is in the CP-kernel in b_0. Likewise, all slices in a_1 that contain only two kernel chain bits (excluding the initial and final bits) are tame. Now we must arrange the initial and final kernel chain bits such that the slides in a_1 that contain them are tame. The first possibility is that the bits c_0 and c_{2n-1} are in the same column in a_1: this kernel chain forms a kernel vortex. The second possibility is to group the initial and final bits of kernel chains in *tame knots*. We call a slice in a_1 with more than 2 active bits a *knot*. We construct states a_1 by combining kernel chains such that their initial and final bits are grouped in a set of knots. If all knots are tame, the state is tame. For a given Hamming weight x, valid states may exist with 0 up to $\lfloor x/3 \rfloor$ knots.

In our program we first fix the number of knots and their slice positions and then efficiently search for all valid states. Table 3.4 lists the number of valid states a_1 (modulo translation over the z-axis) for all Keccak-f widths and up to a Hamming weight of 14. The question marks mark the limitations of our search algorithm: we have not yet been able to compute those values due to time constraints. It can be seen that for a given Hamming weight, overall the number of valid states decreases with increasing width. For a given width, the number of valid states increases with increasing Hamming weight.

3.4.2 Bounds for three-round tame trails

Starting from the valid patterns a_1 we have for each one searched for the 3-round tame trail with the smallest (restriction) weight. This was done in the following way. Given a_1 we compute the minimum weight of a_0 by taking $\mathrm{w}^{\mathrm{rev}}(\lambda^{-1}(a_1))$. In the forward direction, we iterate over all states a_2 for which b_1 is in the CP-kernel. The results are given in Table 3.5. Cases containing a dash indicate that there are no tame 3-round trails for the given width and Hamming weight. Entries of the form "$\leq n$" indicate that not all the patterns were investigated. It can be seen that increasing the width results only in a limited growth of the (restriction) weight. Clearly, the limited diffusion due to the existence of the CP-kernel results in low-weight trails.

Width	**Hamming weight**					
	4	6	8	10	12	14
25	825	12100	95600	465690	1456725	?
50	150	13835	905135	22392676	?	?
100	48	2712	137078	6953033	?	?
200	10	481	24037	1143550	56824109	?
400	4	83	4006	164806	7290847	?
800	0	28	918	30771	1154855	44788752
1600	0	10	304	8231	259567	8399589

Table 3.4: The number of valid difference patterns a_1 per Keccak-f width and Hamming weight

Width	**Hamming weight**					
	4	6	8	10	12	14
25	18	20	22	≤ 22	?	?
50	18	22	25	28	?	?
100	19	24	29	≤ 36	?	?
200	20	29	33	38	?	?
400	24	30	35	40	47	?
800	-	35	41	47	53	58
1600	-	35	41	48	56	62

Table 3.5: Minimum differential trail weight values of tame 3-round trails per Keccak-f width and Hamming weight of a_1.

Width	**Hamming weight**					
	4	6	8	10	12	14
25	25	28	29	$\leq$ 31	?	?
50	30	31	34	38	?	?
100	30	36	41	$\leq$ 48	?	?
200	-	56	56	61	?	?
400	-	-	-	-	90	?
800	-	-	-	-	-	-
1600	-	-	-	-	-	-

Table 3.6: Minimum differential trail weight values of tame 4-round trails per Keccak-*f* width and Hamming weight of pattern after first χ

3.4.3 Bounds for four-round tame trails

We have conducted the a similar search for the 4-round tame trails with the smallest (restriction) weight. This was done in the same way as for 3-round trails with this difference: rather than stopping at a_2, we iterate over all states $a_3 \in \mathcal{A}(a_2)$ in the CP-kernel and compute its restriction weight. To speed up the search we apply pruning if the restriction weight of (a_0, a_1, a_2) is such that a_3 cannot result in a trail with a lower restriction weight than one found for the given category. The results are given in Table 3.6. While for small widths the increase in width does not substantially increase the restriction weights, for higher widths it can be observed that the minimum Hamming weight of a_1 for which there exist tame 4-round differential trails increases dramatically. For width 400 we have to go up to a Hamming weight of 12 to find a tame 4-round trail and for 800 and 1600 no tame 4-round trails exist for a_1 patterns with Hamming weight below 16. As for the minimum restriction weights of the 4-round tame trails found, the increase in 30 to 56 from width 100 to 200 and to 90 for width 400 suggests that the CP-kernel plays a much smaller role than for 3-round trails.

Chapter 4

Analysis of Keccak-f

In this chapter we report on analysis and experiments performed on reduced-round versions of Keccak-f, either by the designers or third parties. In Section 4.1 we describe our experiments based on the algebraic normal form representations. In Section 4.2 we report on the outcome of attempts to solve CICO problem instances. In Section 4.3 we describe statistical properties of reduced-round versions of Keccak-f[25]. Finally, in Section 4.4 we discuss distinguishers that exploit the low algebraic degree of the round function and its inverse.

4.1 Algebraic normal form

In this section, we explain how the algebraic normal form can be used to evaluate the pseudo-randomness of Keccak-f in different aspects.

4.1.1 Statistical tests

There are several ways to describe Keccak-f algebraically. One could compute the algebraic normal form (ANF) [8, Section "Algebraic expressions"] with elements in GF(2), GF(2^5), GF(2^{25}) or GF(2^w), but given the bit-oriented structure and matching θ, ρ, π, ι and χ as operations in GF(2), the ANF in GF(2) seems like a natural way to represent the Keccak-f permutation. For instance, one could take the rows as variables in GF(2^5). This way, the χ operation applies independently per variable. However, the other operations will have a complex expression.

We performed several statistical tests based on the ANF of Keccak-$f[b, n_r = n]$, from $b = 25$ to $b = 1600$ and their inverses in GF(2). The number of rounds n is also varied from 1 to its nominal value, although in practice we can limit ourselves to a reasonable number of rounds after which no significant statistical deviation can be found.

In general, the test consists in varying 25 bits of input and counting the number of monomials of degree d of all b output bits. The statistical test is performed per degree independently. The number of monomials of degree d should be present in a ratio of about one half. The test fails when the observed number of monomials is more than two standard deviations away from the theoretical average. We look for the highest degree that passes the test.

The different tests determine which input bits are varied and/or which variant of the Keccak-f permutation is used.

- **Bits in slice (BIS)** In this test, the 25 bits of the slice $z = 0$ are varied. For $b > 25$, the remaining $b - 25$ bits are set to zero.

Rounds	Maximum degree to pass test		Monomials exist up to degree	
	f	f^{-1}	f	f^{-1}
1	(none)	(none)	2	3
2	(none)	3	3	9
3	(none)	10	5	17
4	5	25	9	25
5	16	25	17	25
6-24	25	25	25	25

Table 4.1: The BIS ANF statistical test on KECCAK-$f[1600]$ and its inverse

- **Bits in lane (BIL)** In this test, the first 25 bits of lane $x = y = 0$ are varied. This test applies only to $b \geq 800$. The remaining $b - 25$ bits are set to zero.
- **Bits in two lanes, kernel** In this test, the first 25 bits of lane $x = y = 0$ and of lane $(x, y) = (0, 1)$ are varied simultaneously. The remaining $b - 50$ bits are set to zero. The idea behind this test is that θ will behave like the identity in the first round since the input state is always in the column-parity kernel (see 2.4.3).
- **BIS, symmetry in lanes** In this test, the 25 bits of the slice $z = 0$ are varied. The $b - 25$ other bits are set as $a[x][y][z] = a[x][y][0]$ so that all lanes contain either all zeroes or all ones. The output bits $a[x][y][0]$ are xored into $a[x][y][z]$ for all $z > 0$. Only the $b - 25$ output bits with $z > 0$ are considered for the test—this test applies only when $b > 25$. The purpose of this test is discussed in Section 4.1.2.
- **BIS, slide** This test is like the first BIS test, except that the function tested is different. Instead of testing it against KECCAK-$f[b, n_r = n]$ itself, we test it against the function slide$[b, n]$ defined in Equation (4.1). The purpose of this test is discussed in Section 4.1.3.

The results for KECCAK-$f[1600]$ are summarized in Tables 4.1 and 4.2. All results of the tests can be found in the file `ANF-Keccak-f.ods`. It is interesting to observe the fast increase of degree of the monomials that are densely present in the algebraic description of the KECCAK-f permutations. Note that the round function has only degree two and thus no monomial of degree higher than 2^i can appear after after i rounds. The degree of the inverse of the round function is three and thus no monomial of degree higher than 3^i can appear after after i inverse rounds.

Taking the worst case among all these tests, the KECCAK-f permutations and their inverses pass the test for the maximum degree tested here (i.e., $\max(25, b - 1)$) after 6 to 8 rounds, depending on the width.

4.1.2 Symmetric trails

The design of KECCAK-f has a high level of symmetry. Due to this, the weight of symmetric trails may no longer be relevant for the security. (See Section 2.4.2 for more details.) We investigate how an attacker may be able to exploit the symmetry in his advantage.

The weight and existence of trails (both differential and linear) is independent of ι. The fact that all other step mappings of the round function are translation-invariant in the direction of the z axis, makes that a trail Q implies $w - 1$ other trails: those obtained by translating

Rounds	**Maximum degree to pass test**		**Monomials exist up to degree**	
	f	f^{-1}	f	f^{-1}
1	(none)	(none)	2	1
2	(none)	3	4	3
3	(none)	8	8	9
4	8	25	15	25
5	25	25	25	25
6	24	25	25	25
7-24	25	25	25	25

Table 4.2: The BIL ANF statistical test on Keccak-$f[1600]$ and its inverse

the patterns of Q over any non-zero offset in the z direction. If all patterns in a trail have a z-period below or equal to d, this implies only $d-1$ other trails.

Moreover, a trail for a given width b implies a trail for all larger widths b'. The patterns are just defined by their z-reduced representations and the weight must be multiplied by b'/b. Note that this is not true for the cardinality of differential trails and the sign of the correlation contribution of linear trails, as these do depend on the round constants.

To find a pair in a differential trail of weight W requires the attacker to fulfill W conditions on the absolute values when following that trail. In the case of a b'-symmetric case, the conditions are repeated b'/b times on translated sets of bits. The question is to determine whether the symmetry induced by this duplication can be exploited by the attacker, even with the asymmetry introduced by ι.

In the absence of ι, Keccak-$f[b]$ presented with a symmetric input behaves as b parallel identical instances of Keccak-$f[25]$. In such a modified permutation, a symmetric trail with weight eb would only impose e conditions (on the symmetric absolute values) rather than eb.

To determine how much asymmetry ι introduces on the absolute values, we express the Keccak-$f[b]$ permutation on a different set of variables and compute the ANF on it. The change of variables is defined as:

$$a'[x][y][0] = a[x][y][0],$$
$$a'[x][y][z] = a[x][y][z] \oplus a[x][y][0], z > 0.$$

In the absence of asymmetry (i.e., without ι), $a'[x][y][z]$, $z > 0$ remains zero if the input of the permutation is 1-symmetric, i.e., if $a[x][y][z]$ depends only on x and y.

The attacker can try to introduce a symmetric difference and to keep it symmetric through the rounds. In these new variables, it is equivalent to keeping $a'[x][y][z] = 0$ for $z > 0$. By analyzing the ANF in these new variables, it gives the degree of the equations to solve to keep the symmetry in the state at a given round by adjusting the input.

The results of these tests can be found in the file `ANF-Keccak-f.ods`. The maximum degree to pass test is indicated in the *BIS, sym lane* columns. To impose dense non-linear equations to the attacker, Keccak-$f[50]$ to Keccak-$f[400]$ need at least 3 rounds, while Keccak-$f[800]$ and Keccak-$f[1600]$ need at least 4 rounds. A dense number of monomials with maximum degree tested here (i.e., $\max(25, b-1)$) is reached after 6 rounds for all widths. Similar conclusions apply to the inverse of Keccak-f: The maximum degree tested is reached after 6 rounds for the inverse of Keccak-$f[50]$, 5 rounds for the inverse of Keccak-$f[100]$ and of Keccak-$f[200]$ and 4 rounds for the other inverses. According to this test, the attacker should have a very difficult time to keep the differences symmetric after such a number of rounds.

4.1.3 Slide attacks

Slide attacks [12, 27] are attacks that exploit symmetry in a primitive that consists of the iteration of a number of identical rounds. We investigate how much asymmetry must be applied for these attacks to be non-effective.

In the absence of ι, all rounds are identical. In such a case, this allows for distinguishing properties of KECCAK-f: the distribution of the cycle lengths will be significantly different from that of a typical randomly-chosen permutation.

To evaluate the amount of inter-round asymmetry brought by ι, we performed a specific test based on ANF. We compute the ANF of the function slide$[b, n]$, which is obtained by XORing together the output of the first n rounds of KECCAK-$f[b]$ and the output of n rounds starting from the second one. Alternatively, it is defined as

$$\text{slide}[b, n] = (\text{round}_{n-1} \circ \cdots \circ \text{round}_0) \oplus (\text{round}_n \circ \cdots \circ \text{round}_1), \tag{4.1}$$

where round_i is the round permutation number i. In the absence of ι, the slide$[b, n]$ function would constantly return zeroes. With ι, it returns the difference between two sets of n rounds slid by one round.

The results of these tests can be found in the file `ANF-Keccak-f.ods`. The maximum degree to pass test is indicated in the *BIS, slide* columns. The degree increases more slowly than for other tests such as *bits in slice*. However, the maximum degree tested here (i.e., $\max(25, b - 1)$) is reached after 8 rounds for KECCAK-$f[25]$ and after 6 rounds for all other widths. For the inverse of KECCAK-$f[25]$, the maximum degree tested is reached after 7 rounds and, for the other inverses, after 4 rounds.

4.2 Solving CICO problems algebraically

4.2.1 The goal

As explained in [8, Section "Conducting primary attacks using structural distinguishers"] the security of a sponge function relies critically on the infeasibility of solving non-trivial CICO problems for its underlying permutation. The resistance of KECCAK-f against solving non-trivial CICO problems is hence critical for the security of KECCAK. We have developed software to generate the sets of equations in a form that they can be fed to mathematics software such as MAGMA or SAGE, and subsequently using this software to solve instances of the CICO problem for different sets of parameters.

4.2.2 The supporting software

KECCAKTOOLS [9] supports the generation of round equations of all supported width values, compatible with SAGE or other computer algebra tools. It can generate the equations for all operations within a round, θ, ρ, π, χ and ι, or a sequence of them. The functions that generate the equations are based on the same code that actually evaluates the KECCAK-f permutations. Using C++ template classes, the evaluation of the operations is made symbolically, hence producing equations. This way of working reduces the chances of making mistakes, as it avoids to duplicate the description of KECCAK-f in C++.

The format of the generated equations has been chosen to match the syntax of SAGE [38]. Also, the bits inside the state are named in such a way that their alphabetical order matches the bit numbering defined at the level of the sponge construction. This feature comes in handy when defining a concrete problem to solve, e.g., finding a pre-image or a collision. One needs to separate bits that are known and fixed from those that are the unknowns of the

problem instance. This separation is defined at the sponge level and can be done alphabetically on the bit names.

We have installed a SAGE server version 1.4 [38] and automated the tests using Python scripts [18] interpreted by the SAGE server.

4.2.3 The experiments

In our experiments we have used SAGE to solve a wide range of CICO problems applied to the Keccak-f permutations. In all problems a subset of the input bits and output bits are fixed to randomly generated values. More particularly, the range of CICO problems we investigated can be characterized with the following parameters:

- Number of rounds n_r: the number of rounds of Keccak-f
- Width b: the width of Keccak-f
- Rate r: the number of unknown input bits
- Output length n: the number of known output bits

There is a variable for every bit at the input of each round and for the output bits, totalling to $b(n_r + 1)$ variables. There is a round equation for each output bit of every round, expressing the output bit as a Boolean expression of the input bits. Additionally there is an equation for each input or output bit that is fixed, simply expressing the equality of the corresponding variable with a binary constant. Hence in total the number of equations is $bn_r + (b - r) + n = b(n_r + 1) + n - r$.

We use SAGE to solve the CICO problems using Ideals and Gröbner bases [19]. We provide a short intuitive explanation here and refer to [19] for thorough treatment of Ideals and Gröbner bases. To solve a CICO problem we do the following:

- Define a ring R of Boolean polynomials, providing the $b(n_r + 1)$ variable names and specifying the so-called term ordering [19].
- Define an ideal over the ring, providing the $b(n_r + 1) + n - r$ equations as generator polynomials.
- Compute a Gröbner basis for this ideal.

The generated Gröbner basis consists of a sequence of polynomials (equivalent to equations) that allows to easily generate all solutions of the set of equations generating the ideal. If there is no solution, it simply consists of the polynomial 1 (implying the equation $1 = 0$ that has no solution). If there is one solution, it simply consists of a sequence of equations of type $x + 1$ (implying $x = 1$) or x (implying $x = 0$). The order of the variable names and term ordering provided in the definition of the ring R have an impact on the Gröbner basis. The sequence of polynomials in the Gröbner basis satisfy an ordering that is determined by the order of variable names and the term ordering. This is not just the way the basis is presented but has an impact on the way it is generated and hence may impact its computational and memory complexity.

For executing these functions, SAGE calls an underlying library dedicated to Boolean polynomials and monomials. This library is called PolyBoRi [17] and is part of the SAGE distribution. As opposed to the generic polynomial ideal oriented functions in SAGE, it heavily exploits the fact that Boolean polynomials can be modelled in a very simple way, with both coefficients and degree per variable in $\{0, 1\}$. The ring of Boolean polynomials

is not a polynomial ring, but rather the quotient ring of the polynomial ring over the field with two elements modulo the field equations $x^2 = x$ for each variable x. Therefore, the usual polynomial data structures seem not to be appropriate for fast Gröbner basis computations. The PolyBoRi authors state that they introduces a specialised data structure for Boolean polynomials, capable of handling these polynomials more efficiently with respect to memory consumption and also computational speed.

If the number of equation is smaller than the number of variables, we expect there to be multiple solutions. If the number of equations is larger than the number of variables, chances are that there is no solution at all. A priori, the expected number of solutions of a CICO problem is 2^{r-n}. In our experiments we have focused on the case $r = n$ where we expect to have one solution on the average and across experiments we expect the number of solutions to have a Poisson distribution with $\lambda = 1$. This is an interesting case as it corresponds with the CICO problems encountered when searching a (second) pre-image.

We investigated the case $n = r$ for values of r up to 12, values of n_r up to 8 and widths from 25 to 400. We used lexicographical order with the output variables declared first and the input variables declared last, as from preliminary experiments this turned out to be the most efficient choice. For each set of parameters, a number of CICO problems was solved with at least 5 problems resulting in a solution and 5 problems resulting in the absence of a solution. We arranged the output of SAGE for analysis in the file `Keccak-CICO-results.ods`.

Analysing this data, we can make the following observations:

- For small values of r PolyBoRi is efficient in computing the Gröbner basis, and this for all values of n_r and width b.

- For certain parameter choices, solving CICO problems that have no solution takes significantly less time than CICO problems that have a solution. The difference is especially large for large widths and small rate values. When the rate increases, the computation times for the two cases (solution and no solution) converge.

- Doubling the width, keeping all other parameters constant also roughly doubles the computation time. Hence the computation time appears to grows roughly linearly in the width.

- Increasing the number of rounds from n_r to $n_\mathrm{r} + 1$, keeping all other parameters constant, results in an increase roughly independent from the value of n_r. Hence the computation time appears to grow linearly in the number of rounds.

- The effect of increasing bitrate by 1 from r to $r + 1$, keeping all other parameters constant, appears to increase the computation time by a factor that weakly increases with r and the number of rounds. At $r = 12$ its value is around 3. If we may extrapolate this behaviour, solving a CICO problem quickly becomes infeasible with this method as r grows.

Hence for this class of CICO problems the case of a small rate and small output length can be relatively easily solved. Although surprising at first sight, this poses no threat to the security of Keccak-f as such a CICO problem can be efficiently solved by exhaustive search for any permutation. It suffices to try all 2^r possible values of the r unknown input bits, apply the permutation and verify whether the generated output has the correct value in the known bit positions.

Rounds	Maximum degree to pass test		Monomials exist up to degree	
	f	f^{-1}	f	f^{-1}
1	(none)	(none)	2	3
2	1	2	4	9
3	6	10	8	17
4	14	18	16	21
5	22	23	22	23
6	23	23	24	24
7-12	24	24	24	24

Table 4.3: The ANF statistical test on Keccak-f[25] and its inverse

4.2.4 Third-party analysis

Aumasson and Khovratovich report in [1] on attempts of solving instances of the CICO problem for different widths of Keccak-f using a so-called triangulation tool. Solutions were found for Keccak-f[1600] reduced to three rounds. For more rounds the fast backwards diffusion in θ apparently prevented solving the CICO problems.

Morawiecki and Srebrny report in [36] on attempts of solving instances of the CICO problem for different widths of Keccak-f by first expressing the round equations in conjunctive normal form (CNF) and then run SAT solvers on the resulting set of equations. The SAT solver performed better than exhaustive search for some CICO problem instances of Keccak-f[1600], Keccak-f[200] and Keccak-f[50] reduced to three rounds. For more rounds the SAT solvers were less efficient than exhaustive search.

4.3 Properties of Keccak-f[25]

The Keccak-f permutations should have no propagation properties significantly different from that of a random permutation. For the smallest Keccak-f version, Keccak-f[25], it is possible to experimentally verify certain properties.

First, we report on the algebraic normal form investigations, applied to Keccak-f[25]. Second, we have reconstructed significant parts of the distribution of differential probabilities and input-output correlation of Keccak-f[25] and its reduced-round versions. Third, we have determined the cycle structure of Keccak-f[25] and all its reduced-round versions.

As a reference for the distributions, we have generated a pseudorandom permutation operating on 25 bits using a simple algorithm from [32] taking input from a pseudorandom bit generator based on a cipher that is remote from Keccak and its inventors: RC6 [37]. We denote this permutation by the term Perm-R.

4.3.1 Algebraic normal statistics

The results of the ANF analysis of Keccak-f[25] are displayed in Table 4.3. Starting from 7 rounds, all monomials up to order 24 exist and appear with a fraction close to one half. Since Keccak-f[25] is a permutation, the monomial of order 25 does not appear.

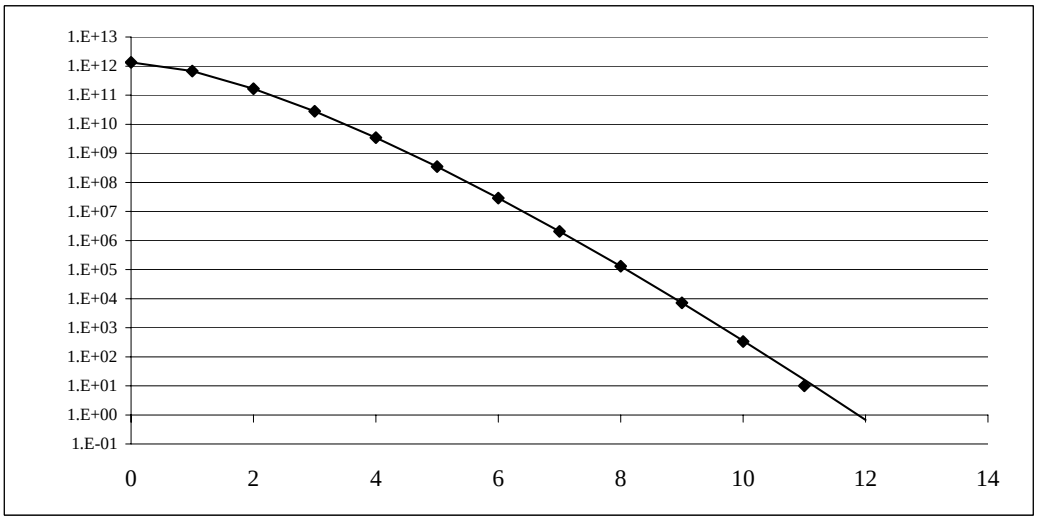

Figure 4.1: Cardinality histogram of sampling of Perm-R

4.3.2 Differential probability distributions

We have investigated the distribution of the cardinality of differentials for KECCAK-f[25], several reduced-round versions of KECCAK-f[25] and of Perm-R. For these permutations, we have computed the cardinalities of 2^{41} differentials of type (a', b') where a' ranges over 2^{16} different non-zero input patterns and b' over all 2^{25} patterns. For Perm-R we just tested the first (when considered as an integer) 2^{16} non-zero input patterns. For the KECCAK-f[25] variants we tested as input patterns the first 2^{16} non-zero entries in the lookup table of Perm-R.

In a random permutation the cardinality of differentials has a Poisson distribution with $\lambda = 1/2$. This is studied and described among others in [25]. Moreover, [25] also determines the distribution of the maximum cardinality of a large number of differentials over a random permutation. According to [25, Section 5.2], the expected value of the maximum cardinality over the 2^{41} samples is 12 and the expected value of the maximum cardinality over all $2^{50} - 1$ non-trivial differentials (a', b') is 14.

We provide in a sequence of diagrams the histograms obtained from these samplings, indicating the envelope of the theoretical Poisson distribution for a random permutation as a continuous line and the results of the measurements as diamond-shaped dots. We have adopted a logarithmic scale in the y axis to make the deviations stand out as much as possible.

Figure 4.1 shows that Perm-R exhibits a distribution that follows quite closely the theoretically predicted one. The maximum observed cardinality is 11.

Figure 4.2 shows the distribution for the two-round version of KECCAK-f[25]: the distribution deviates significantly from the theoretical Poisson distribution. Note that here also the x axis has a logarithmic scale. The largest cardinality encountered is 32768. It turns out that the pairs of this differential are all in a single trail with weight 9. The number of pairs is equal to the number of pairs predicted by the weight: $2^{24-9} = 2^{15}$. Note that there are 2-round trails with weight 8 (see Table 3.3) but apparently no such trail was encountered in our sampling.

Figure 4.3 shows the distribution for the three-round version of KECCAK-f[25]. The deviation from the theoretical Poisson distribution is smaller. The largest cardinality encountered is now 146. The pairs of this differential are all in a single trail with weight 17. The number

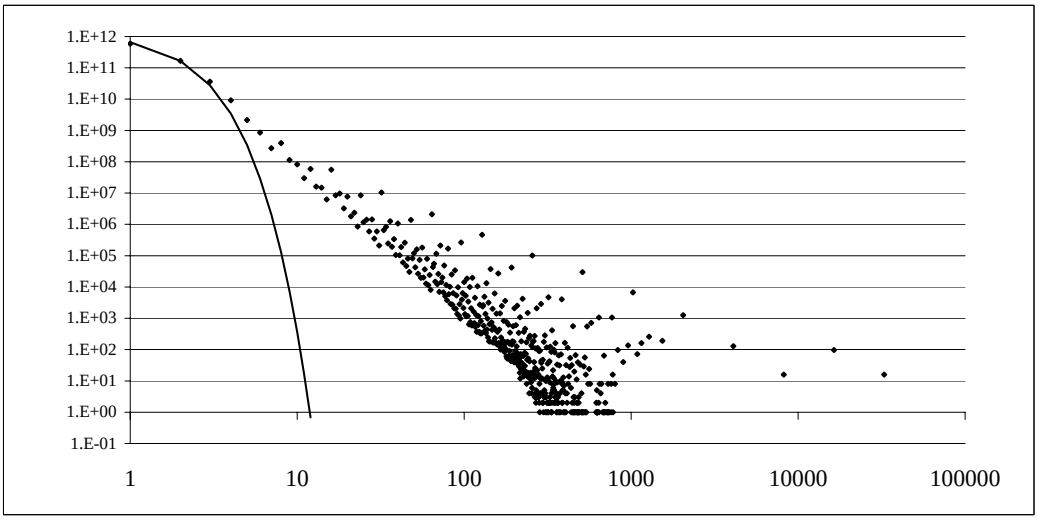

Figure 4.2: Cardinality histogram of sampling of 2-round version of Keccak-f[25]

of pairs is slightly higher than the number of pairs predicted by the weight: $2^{24-17} = 2^7$. The 3-round trails with weight 16 (see Table 3.3) were not encountered in our sampling.

Figure 4.4 shows the distribution for the four-round version of Keccak-f[25]. The sampling does no longer allow to distinguish the distribution from that of a random permutation. The largest cardinality encountered is now 12. The pairs of this differential are in 12 different trails with weight ranging from 56 to 64. For the 4-round trails with weight 23 (see Table 3.3) it is not clear whether they were encountered in our sampling: the expected number of pairs is only 2 and this may have gone unnoticed.

Finally, Figure 4.5 shows the distribution for the 12-round version of Keccak-f[25]. As expected, the distribution is typical of a random permutation. The maximum cardinality observed is 12.

4.3.3 Correlation distributions

We have investigated the distribution of the correlations for Keccak-f[25], several reduced-round versions of Keccak-f[25] and Perm-R. For these permutations, we have computed the correlations of 2^{39} couples (v, u) where u ranges over 2^{14} different non-zero output masks and v over all 2^{25} patterns. For Perm-R we just tested the first (when considered as an integer) 2^{14} non-zero output masks. For the Keccak-f[25] variants we tested as output masks the first 2^{14} non-zero entries in the lookup table of Perm-R.

In a random permutation with width b the input-output correlations have a discrete distribution enveloped by a normal distribution with $\sigma^2 = 2^{-b}$. This is studied and described in [25]. Moreover, [25] also determines the distribution of the maximum correlation magnitude of a large number of couples (v, u) over a random permutation. According to [25, Section 5.4], the expected value of the maximum correlation magnitude over the 2^{39} samples is 0.00123 and the expected value of the maximum correlation magnitude over all $2^{50} - 1$ non-trivial correlations (v, u) is 0.0017.

We provide in a sequence of diagrams the histograms obtained from these samplings, indicating the envelope of the theoretical normal distribution for a random permutation as a continuous line and the results of the measurements as diamond-shaped dots. We have

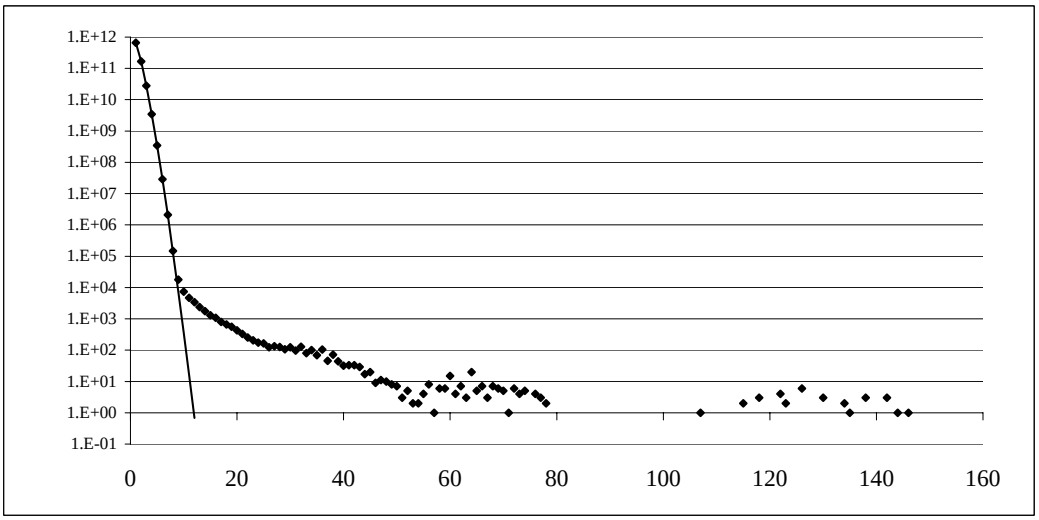

Figure 4.3: Cardinality histogram of sampling of 3-round version of Keccak-f[25]

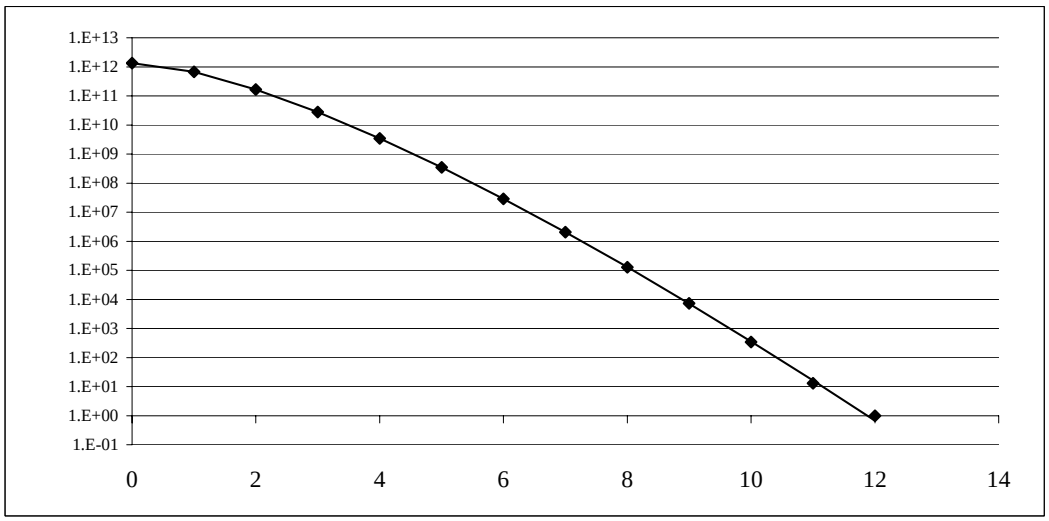

Figure 4.4: Cardinality histogram of sampling of 4-round version of Keccak-f[25]

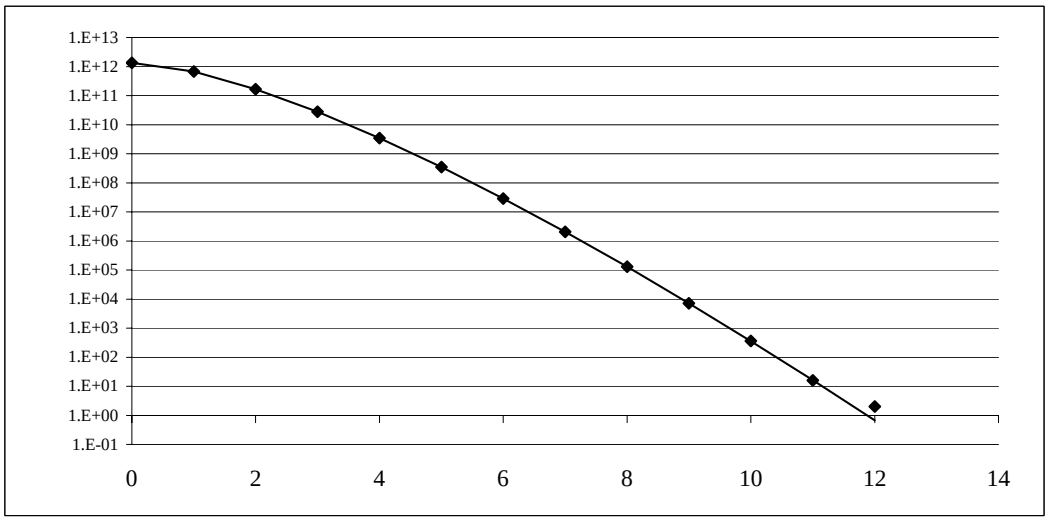

Figure 4.5: Cardinality histogram of sampling of Keccak-f[25]

adopted a logarithmic scale in the y axis to make the deviations stand out as much as possible.

Figure 4.6 shows that Perm-R exhibits a distribution that follows quite closely the normal envelope. At its tails the experimental distribution exhibits its discrete nature. Because it is a permutation, the correlation can only be non-zero in values that are a multiple of 2^{2-b}. For a given correlation value c that is a multiple of 2^{2-b}, the a priori distribution of the corresponding value in the histogram is a Poisson distribution with λ given by the value of the normal envelope in that point. The largest correlation magnitude observed is 0.001226, quite close to the theoretically predicted value.

Figure 4.7 shows the distribution for the two-round version of Keccak-f[25]: the distribution deviates significantly from the theoretical normal envelope. Additionally, it is zero for all values that are not a multiple of 2^{-15} (rather than 2^{-23}). This is due to the fact that the Boolean component functions of Keccak-f[25] have only reached degree 4 after two rounds, rather than full degree 24. The largest correlation magnitude encountered is 0.03125 (outside the scale of the figure). This is the correlation magnitude 2^{-5} one would obtain by a single linear trail with weight 10. By measuring the correlation of the same pair of masks for variants of the two-round version of Keccak-f[25] where different constant vectors are XORed in between the two rounds, it turns out that the correlation value is either 2^5 or -2^{-5}. This implies that the correlation is the result of a single trail. The 2-round linear trails with weight 8 (see Table 3.3) were apparently not encountered in our sampling.

Figure 4.8 shows the distribution for the three-round version of Keccak-f[25]: the deviation from the theoretical normal envelope becomes smaller. This distribution is zero for all values that are not a multiple of 2^{-18} due to the fact that the Boolean component functions of Keccak-f[25] have only reached degree 8 after three rounds. The largest correlation magnitude encountered is 0.003479. This is a correlation magnitude that cannot be obtained by a single linear trail. 3-round linear trails with weight 16 would give correlation magnitude $2^{-8} \approx 0.0039$. It is quite possible that the observed correlation value is the sum of the (signed) correlation contributions of some trails, including one with weight 16 and some with higher weight. By measuring the correlation of this pair of masks in variants of the three-round version of Keccak-f[25] where different constant vectors are XORed in between the rounds, we obtain 491 different values. This implies that this correlation has contributions from at

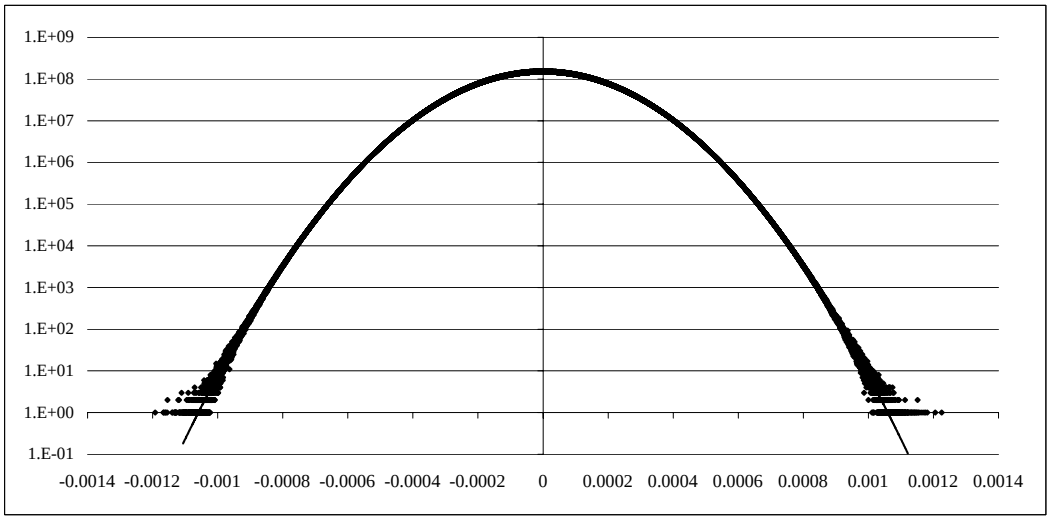

Figure 4.6: Correlation histogram of sampling of Perm-R

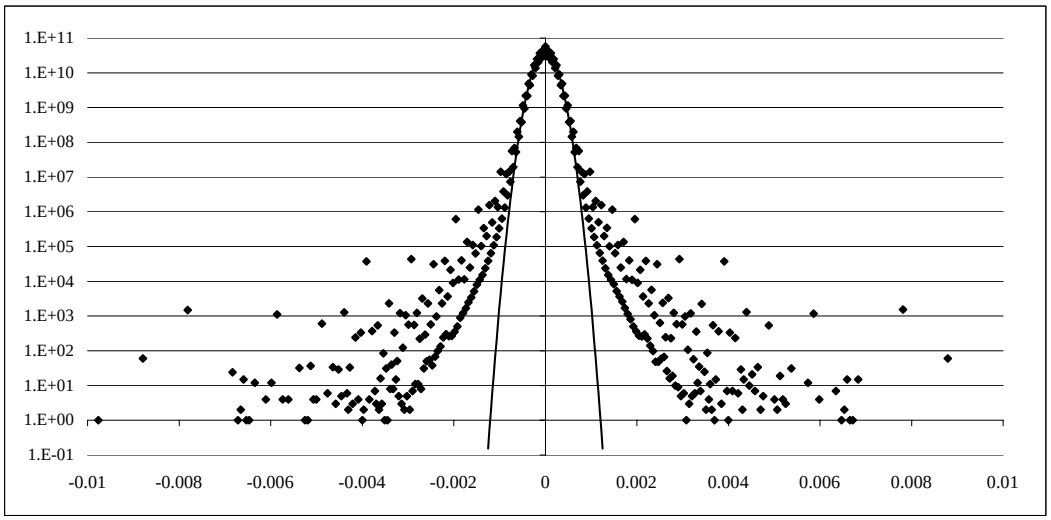

Figure 4.7: Correlation histogram of sampling of 2-round version of Keccak-*f* [25]

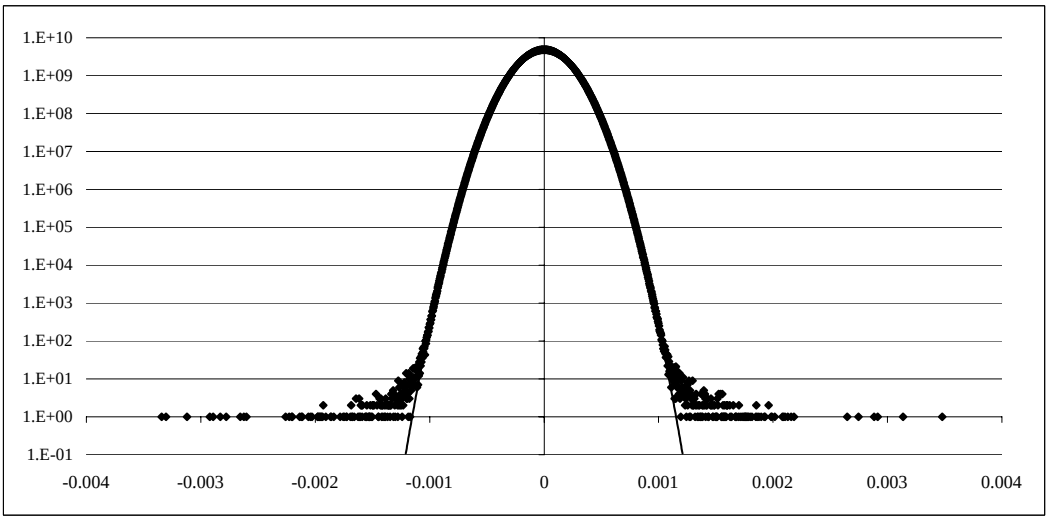

Figure 4.8: Correlation histogram of sampling of 3-round version of Keccak-f[25]

least 9 trails.

Figure 4.9 shows the distribution for the four-round version of Keccak-f[25]. The shape of the distribution and the maximum values do no longer allow to distinguish the distribution from that of a random permutation. The largest correlation magnitude encountered is 0.001196. However, this distribution differs from that of a random permutation because it is zero for all values that are not a multiple of 2^{-20} due to the fact that the Boolean component functions of Keccak-f[25] have only reached degree 16 after four rounds. By measuring the correlation of this pair of masks in variants of the four-round version of Keccak-f[25] where different constant vectors are XORed in between the rounds, we obtain many different values implying that this correlation is the result of a large amount of trails. Moreover, the value of the correlation exhibits a normal distribution.

After 5 rounds the distribution is zero for values that are not a multiple of 2^{-22} and only after 6 rounds this becomes 2^{-23}.

Finally, Figure 4.10 shows the distribution for the 12-round version of Keccak-f[25]. As expected, the distribution is typical of a random permutation. The maximum correlation magnitude observed is 0.001226.

4.3.4 Cycle distributions

We have determined the cycle structure of Keccak-f[25] and all its reduced-round versions. Table 4.4 lists all cycles for Keccak-f[25] and Table 4.5 the number of cycles for all reduced-round versions. For a random permutation, the expected value of the number of cycles is $\ln(2^{25}) = 25 \ln 2 \approx 17.3$. The average of Table 4.5 is 16.3.

It can be observed that Keccak-f[25] and all its reduced-round versions have an even number of cycles. For a permutation operating on a domain with an even number of elements, an even number of cycles implies that it is an even permutation [40], hence they are all even permutations. Actually, it is easy to demonstrate that all members of the Keccak-f family are even permutations. We do however not think this property can be exploited in an attack or to build a usable distinguisher.

The members of the Keccak-f family are even permutations because the composition of

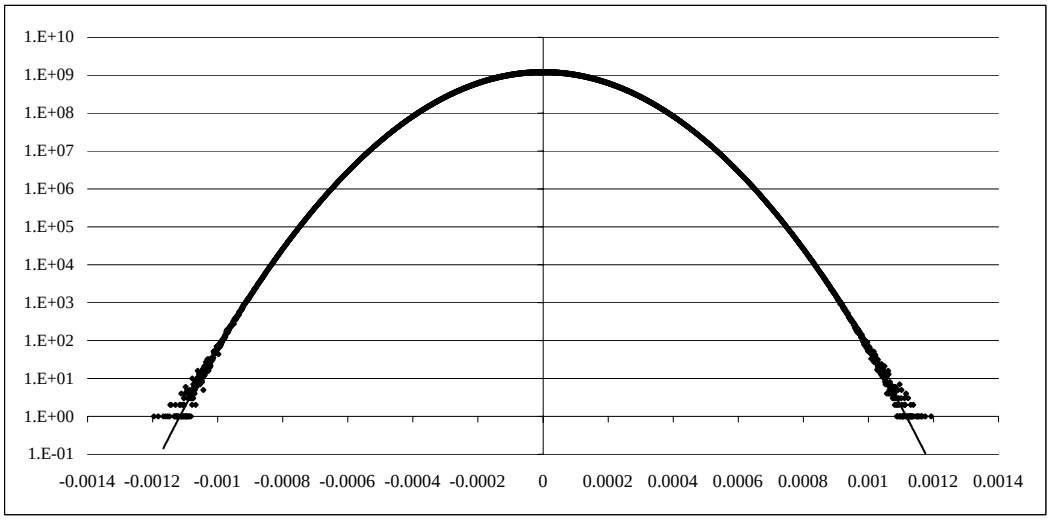

Figure 4.9: Correlation histogram of sampling of 4-round version of Keccak-*f* [25]

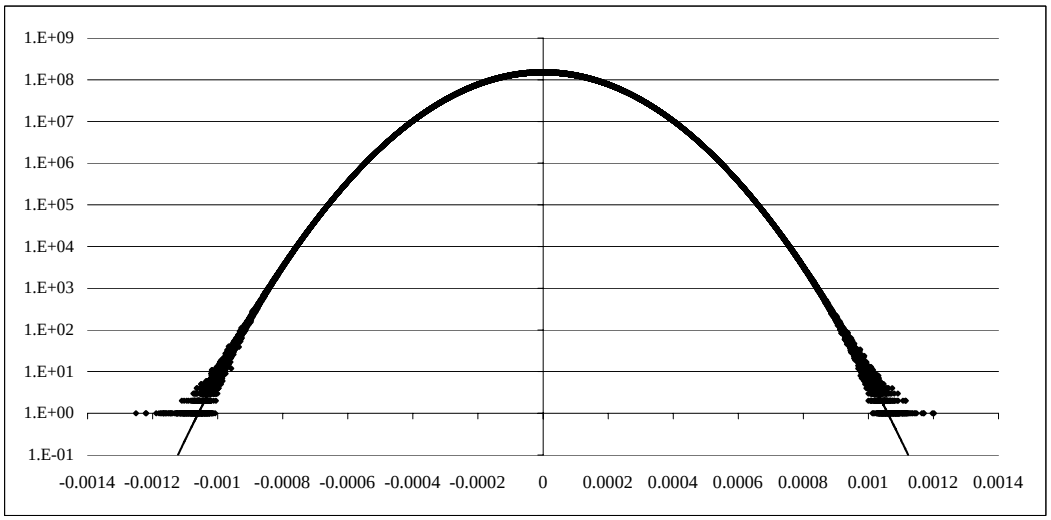

Figure 4.10: Correlation histogram of sampling of Keccak-*f* [25]

18447749	147821	168	12
13104259	40365	27	3
1811878	2134	14	2

Table 4.4: Cycle lengths in Keccak-f[25]

rounds	cycles	rounds	cycles	rounds	cycles
1	14	5	18	9	14
2	12	6	20	10	20
3	16	7	18	11	18
4	16	8	18	12	12

Table 4.5: Number of cycles in reduced-round versions of Keccak-f[25]

two even permutation is an even permutation and that all step mappings of Keccak-f are even permutations. We cite here a number of arguments we found in [40, Lemma 2]:

- The mappings θ, π and ρ are linear. In fact all invertible linear transformations over $\mathbb{Z}_2^b$ with $b > 2$ are even permutations. This follows from the fact that each invertible binary matrix can be obtained from the identity matrix by elementary row transformations (binary addition of one row to another row), and that these elementary row transformations (considered as linear mappings) are permutations with 2^{b-1} fixed points and 2^{b-2} cycles of length 2.

- The mapping ι consists of the addition of a constant. Addition of a constant in $\mathbb{Z}_2^b$ is the identity mapping if the constant is all-zero and has 2^{b-1} cycles of length 2 if the constant is not all-zero.

- The mapping χ is an even permutation because it can be represented as a composition of $5w$ permutations that carry out the χ mapping for one row and leave the other $5w - 1$ rows fixed. The cycle representation of each such permutation contains a number of cycles that is a multiple of 2^{25w-5} and hence even.

4.4 Distinguishers exploiting low algebraic degree

The Keccak-f round function and its inverse have low algebraic degrees, 2 and 3 respectively. This has been exploited in third-party cryptanalysis to construct a number of distinguishers.

Aumasson and Khovratovich report in [1] on observations showing that Keccak-f[1600] reduced to 3 and 4 rounds does not have an ideal algebraic structure and conjectures that this can be observed up to reduced-round versions of ten rounds, for which the algebraic degree is at most 1024 and hence not maximal.

In [34] Lathrop tested the resistance of Keccak against cube attacks by executing efficient cube attack against the 224-bit output-length version of Keccak configured as a MAC and calling a version of Keccak-f[1600] reduced to 4 rounds or less. Based on his analysis, the author suggests that a cube attack against such a configuration would only be practical against reduced-round versions up to 7 rounds.

4.4.1 Zero-sum distinguishers

In [2] Aumasson and Meier introduced very simple and elegant distinguishers against reduced-round versions of KECCAK-f[1600]. They are based on the simple observation that the degree of n KECCAK-f rounds is at most 2^n and that the degree of n inverse rounds is at most 3^n. We cite [2]:

Suppose one fixes $1600 - 513 = 1087$ *bits of the initial state to some arbitrary value, and consider the* 2^{513} *states obtained by varying the 513 bits left. Our main observation is that applying the 9-round* KECCAK-f *to each of those states and xoring the* 2^{513} *1600-bit final states obtained yields the zero state. This is because, for each of the 1600 Boolean components, the value obtained is the order-513 derivative of a degree-512 polynomial, which by definition is null.*

If these states are chosen in some intermediate round and one computes the corresponding inputs and outputs, this is also the case for those inputs as long as the polynomials expressing the inputs in terms of the intermediate state bits have small enough degree. Hence one has a systematic way to construct a set of inputs that XOR to zero and for which the corresponding outputs XOR to zero, which is qualitatively different from the generic method [7]. Additionally, if the positions of the free bits in the intermediate round are chosen carefully, one may even reduce the degree of the forward and/or backward polynomials. Using these simple elements, Aumasson and Meier constructs distinguishers for up to 16 rounds of KECCAK-f[1600] [2]. This last one makes use of 6 backward rounds and 10 forward rounds.

Given an upper bound of N free bits one can construct a zero-sum distinguisher with $\log_2 N$ forward rounds and $\log_3 N$ backward rounds. While in a distinguisher using only the forward direction maximum degree cannot be reached before 11 rounds, the start-in-the-middle technique allows going up to 16 rounds.

This was further refined by Boura and Canteaut in two papers [15, 14] extending the distinguishers to 18 rounds for KECCAK-f[1600] in [15] and up to 20 rounds in [14]. The new ideas are the following. First, the Walsh spectrum of χ allows one to bound the degree of the inverse rounds more tightly. In particular, while 7 inverse rounds are expected to be of maximal degree (as $3^7 > 1599$), they show that it cannot be higher than 1369. Second, by aligning the bits that are varied and those that are fixed on row boundaries, χ works independently and bijectively row per row. This can be generalized to cover several rounds.

Boura and Canteaut were joined by De Cannière in [16] and they extended the distinguisher to the full 24 rounds of KECCAK-f[1600]. The novel idea that allowed this is the discovery of a saturation effect imposing an upper bound on the degree of iterated permutations. They showed this effect for round functions in which the nonlinear step consists of the parallel application of S-boxes with a very elegant and simple argument. The effect is strongest for small S-boxes but can also be observed for large S-boxes. While initially the algebraic degree D *increases* exponentially by a factor d (with d the algebraic degree of the round function) per round, above some degree it only *converges* exponentially to b.

Finally, Duan and Lai further considered the evolution of the degree of the inverse permutation [26]. They noticed that the product of two output bits of χ^{-1} yields degree-3 polynomials, and this leads to better bounds on the degree of the inverse rounds. In Table 4.6, we list the bounds on the algebraic degrees of reduced-round versions of KECCAK-f[1600] and its inverse, based on this property and our own analysis.

Triggered by the initial publication on zero-sums by Aumasson and Meier in [2], we decided to increase the number of rounds in KECCAK-f from $12 + \ell$ to $12 + 2\ell$. The motivation behind this was the following. The applicability of the zero-sum distinguishers is limited by the maximum number of free bits N, namely the width. Doubling the width allows to add a forward round in the distinguisher and possibly a backward round. So when doubling the width, roughly two additional rounds are required to provide resistance against zero-zum

# **Rounds**	**Degree** f	1600− **Degree** f	**Degree** f^{-1}	1600− **Degree** f^{-1}
1	2		3	
2	4		9	
3	8		27	
4	16		81	
5	32		243	
6	64		729	871
7	128		1164	432
8	256		1382	218
9	512		1491	109
10	1024	576	1545	55
11	1408	192	1572	28
12	1536	64	1586	14
13	1578	22	1593	7
14	1592	8	1596	4
15	1597	3	1598	2
16	1599	1	1599	1

Table 4.6: Bounds on the algebraic degrees of reduced-round versions of KECCAK-f[1600] and its inverse.

distinguishers. This may be the case for other distinguishers too and so we felt it would be more appropriate to increase the number of rounds by 2 when doubling the width.

The zero-sum partitions distinguish the 24 rounds of KECCAK-f[1600] from a randomly-chosen permutation, although without implying a distinguisher on KECCAK itself [16, 26]. Not increasing the number of rounds, strictly speaking, contradicts the hermetic sponge strategy. Still, we decided to stick the number of rounds specified by $12 + 2\ell$ due to the fact that these distinguishers can in no way be used to attack KECCAK for many reasons, of which the following two are the most important (see also [8, Section "The usability of structural distinguishers"]).

First, the advantage of a zero-sum distinguisher (or partition) is 0 for any number of queries N to f or f^{-1} with N smaller than the size of the partition. This means that using a permutation f that has zero-sum distinguishers has no impact on the $\mathcal{RO}$ differentiating advantage for N below the size of the partition. In the flat sponge claim, no resistance is claimed for attacks requiring a workload of more than $2^{c/2}$ queries to f and the maximum value for the capacity is $c = b - 1$. So distinguishers that have zero advantage for N below $2^{b/2}$ do not compromise the flat sponge claim. The zero-sum distinguishers for the full 24 rounds of KECCAK-f[1600] have zero advantage below something like 2^{1575} queries. Only structural distinguishers on f that have non-zero advantage below 2^{800} queries can possibly qualify as a threat for the security of a sponge function that uses it.

Second, exploiting zero-sum distinguishers would require the adversary to apply inputs to the sponge function such that the input to f has the values of her choice, for many different inputs. In the sponge construction, an attacker can not choose the value of bits in the inner part of the input to f directly, but only *influence* their value in an indirect way by injecting bits in the previous iteration of f.

Further discussions about the applicability of the zero-sum distinguishers can be found in [7].

4.4.2 Pre-image attacks

The first exploitation of the low algebraic degree of the KECCAK-f round function in an actual attack on KECCAK was published by Bernstein in [3]. It is a pre-image attack that, making abstraction of the cost of memory access, would be able to break the claimed security level for reduced-round versions of KECCAK-f[1600] up to 8 rounds. When counting the cost of memory access, these attacks are much less efficient than generic pre-image attacks.

The attack exploits the fact that KECCAK-f reduced to 6, 7 and 8 rounds has an algebraic degree of only 64, 128 and 256 respectively. This allows evaluating a subset of the output bits for large sets of inputs more efficiently than computing KECCAK-f[1600] for the same sets of inputs. This can be used as a filter for second pre-image candidates, reducing the number of candidates for which KECCAK-f[1600] must be computed to only a fraction.

We give here short summary of what the attack can do.

Consider a hash function with an m-bit digest and consider messages of a certain fixed length L. Let there be:

- Subset A (with $|A| = n$) of the message bit positions and
- Subset B (with $|B| = m$) of the digest bit positions, such that for any fixed value of the message bits not in A, the bits in B have algebraic degree $d < n$ as functions of bits in A.

If this degree d is sufficiently small, the computation effort of finding (second) pre-images can be reduced.

This attack can be applied to a hash function (possibly round-reduced) where a subset B of the digest bits have a low algebraic degree in a subset A of the message bits of the last block. The bits in B are only separated from the bits in A by a single compression function (or permutation) call.

The downside of the attack is that this workload reduction comes at the cost of memory. For it to be a reduction, one considers the computational effort not including the cost of memory access. The memory required for the attack is $m2^n$ bits. This may be reduced somewhat by some clever programming techniques at the cost of increasing the computational effort.

For KECCAK with KECCAK-f[1600] reduced to 6, 7 or 8 rounds, the length of the message L plays no role. The active bits must all be in the last block, so n is limited by the bitrate (minus the padding). One can tune m and n resulting in different memory-time trade-offs. The ones reported in [3] are:

- 6 rounds: 2^{176} bits of memory give a workload reduction by a factor 50 ($\approx$ 6 bits)
- 7 rounds: 2^{320} bits of memory give a workload reduction by a factor 37 ($\approx$ 5 bits)
- 8 rounds: 2^{508} bits of memory give a workload reduction by a factor 1.4 (half a bit)

Chapter 5

Design rationale summary

The purpose of this chapter is to list the design choices and to briefly motivate them, although further analysis is provided in the subsequent chapters.

5.1 Choosing the sponge construction

The Keccak hash function makes use of the sponge construction, following the definition of [8]. This results in the following property:

Provability It has a proven upper bound for the success probability, and hence also a lower bound for the expected workload, of generic attacks. We refer to [8, Chapter "Security proofs"] for a more in-depth discussion.

The design philosophy underlying Keccak is the *hermetic sponge strategy*. This consists of using the sponge construction for having provable security against all generic attacks and calling a permutation (or transformation) that should not have structural properties with the exception of a compact description [8, Section "The philosophy"].

Additionally, the sponge construction has the following advantages over constructions that make use of a compression function:

Simplicity Compared to the other constructions for which upper bounds have been proven for the success of generic attacks, the sponge construction is very simple, and it also provides a bound that can be expressed in a simple way.

Variable-length output It can generate outputs of any length and hence a single function can be used for different output lengths.

Flexibility Security level can be incremented at the cost of speed by trading in bitrate for capacity, using the same permutation (or transformation).

Functionality Thanks to its long outputs and proven security bounds with respect to generic attacks, a sponge function can be used in a straightforward way as a MAC function, stream cipher and a mask generating function. Thanks to the duplex construction, a sponge function can be used as a reseedable pseudorandom bit generator and for efficient authenticated encryption (see [8, Section "Authenticated encryption"]).

To support arbitrary bit strings as input, the sponge construction requires a padding function. We have chosen the simplest padding rule that allows to use the same permutation in combination with different bitrate values [8, Section "Optimum security of multi-rate sponge functions"] without loss of security.

5.2 Choosing an iterated permutation

The sponge construction requires an underlying function f, either a transformation or a permutation. Informally speaking, f should be such that *it does not have specific properties that can be exploited in attacks*. We have chosen a permutation, constructed as a sequence of (almost) identical rounds because of the following advantages:

Block cipher experience An iterated permutation is an iterated block cipher with a fixed key. In its design one can build on knowledge obtained from block cipher design and cryptanalysis (see Chapter 2).

Memory efficiency Often a transformation is built by taking a permutation and adding a feedforward loop. This implies that (at least part of) the input must be kept during the complete computation. This is not the case for a permutation, leading to a relatively small RAM footprint.

Compactness Iteration of a single round leads to a compact specification and potentially compact code and hardware circuits.

5.3 Designing the KECCAK-f permutations

The design criterion for the KECCAK-f permutations is to have no specific properties that can be exploited in an attack when being used in the sponge construction. It is constructed as an iterated block cipher similar to NOEKEON [23] and RIJNDAEL [24], with the key schedule replaced by some simple round constants. Here we give a rationale for its features:

Bit-oriented structure Attacks where the bits are grouped (e.g., in bytes), such as integral cryptanalysis and truncated trails or differentials, are unsuitable against the KECCAK-f structure.

Bitwise logical operations and fixed rotations Dependence on CPU word length is only due to rotations, leading to an efficient use of CPU resources on a wide range of processors. Implementation requires no large tables, removing the risk of table-lookup based cache miss attacks. They can be programmed as a fixed sequence of instructions, providing protection against timing attacks.

Symmetry This allows to have very compact code in software and a very compact co-processor circuit suitable for constrained environments [10] .

Parallelism Thanks to its symmetry and the chosen operations, the design is well-suited for ultra-fast hardware implementations and the exploitation of SIMD instructions and pipelining in CPUs.

Round degree 2 This makes the analysis with respect to differential and linear cryptanalysis easier, leads to relatively simple (albeit large) systems of algebraic equations and allows the usage of very powerful protection measures against differential power analysis (DPA) both in software and hardware that are not suited for most other nonlinear functions [6].

Matryoshka structure The analysis of small versions is relevant for larger versions (see Section 2.2).

Number of rounds $n_r = 12 + 2\ell$: The value of n_r has been chosen to have a good safety margin and still have good performance. See Section 5.4.

Attack	Number of rounds
KECCAK-f distinguisher below $2^{b/2}$	$9 + 2\ell$
KECCAK distinguisher	$7 + \ell$
Inner collision	$5 + \ell$
State recovery	$5 + \ell$

Table 5.1: Sufficient number of rounds of KECCAK-f to resist to different types of attack or distinguisher

5.4 Strength estimation

We here provide our estimates of how many rounds in KECCAK-f are sufficient to provide resistance against two types of distinguisher and two types of attack. These estimations cover all instances of KECCAK$[r, c]$.

KECCAK-f distinguisher below $2^{b/2}$ A structural distinguisher for KECCAK-$f[b]$ requiring less than $2^{b/2}$ queries, for any supported width.

KECCAK distinguisher A structural distinguisher for KECCAK$[r, c]$.

Inner collision The generation of an inner collision with success probability above $N^2 2^{-(c+1)}$, with a workload equivalent to N calls to KECCAK-$f[r + c]$.

State recovery State recovery from a keyed sponge with success probability above $NM2^{-c}$ with a workload equivalent to N calls to KECCAK-$f[r + c]$ and a sequence of queries to the keyed sponge requiring it to make in total M calls to the permutation.

We list the number of rounds that we estimate to be sufficient to resist these attacks in Table 5.1. These estimates are based on the results of our preliminary analysis that is treated in this document and the third-party analysis in [1, 34, 2, 15, 14, 36, 16, 3].

Bibliography

[1] J.-P. Aumasson and D. Khovratovich, *First analysis of Keccak*, Available online, 2009, `http://131002.net/data/papers/AK09.pdf`.

[2] J.-P. Aumasson and W. Meier, *Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi*, Available online, 2009, `http://131002.net/data/papers/AM09.pdf`.

[3] D. J. Bernstein, *Second preimages for 6 (7? (8??)) rounds of keccak?*, 2010, http://cr.yp.to/hash/keccak-20101127.txt.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, RadioGatún, *a belt-and-mill hash function*, Second Cryptographic Hash Workshop, Santa Barbara, August 2006, `http://radiogatun.noekeon.org/`.

[5] ———, Keccak *specifications, version 2*, NIST SHA-3 Submission, September 2009, `http://keccak.noekeon.org/`.

[6] ———, *Note on side-channel attacks and their countermeasures*, Comment on the NIST Hash Competition Forum, May 2009, `http://keccak.noekeon.org/NoteSideChannelAttacks.pdf`.

[7] ———, *Note on zero-sum distinguishers of* Keccak-f, Comment on the NIST Hash Competition Forum, January 2010, `http://keccak.noekeon.org/NoteZeroSum.pdf`.

[8] ———, *Cryptographic sponge functions*, January 2011, `http://sponge.noekeon.org/`.

[9] ———, KeccakTools *software*, January 2011, `http://keccak.noekeon.org/`.

[10] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, Keccak *implementation overview*, January 2011, `http://keccak.noekeon.org/`.

[11] E. Biham, O. Dunkelman, and N. Keller, *The rectangle attack - rectangling the serpent*, Advances in Cryptology – Eurocrypt 2001 (B. Pfitzmann, ed.), Lecture Notes in Computer Science, vol. 2045, Springer, 2001, pp. 340–357.

[12] A. Biryukov and D. Wagner, *Slide attacks*, in Knudsen [31], pp. 245–259.

[13] C. Bouillaguet and P.-A. Fouque, *Analysis of the collision resistance of RadioGatún using algebraic techniques*, Selected Areas in Cryptography, Lecture Notes in Computer Science, vol. 4876, Springer, 2008.

[14] C. Boura and A. Canteaut, *Zero-sum distinguishers on the Keccak-f permutation with 20 rounds (working draft)*, private communication, 2010.

[15] ———, *A zero-sum property for the Keccak-f permutation with 18 rounds*, Available online, 2010, http://www-roc.inria.fr/secret/Anne.Canteaut/Publications/zero_sum.pdf.

[16] C. Boura, A. Canteaut, and C. De Cannière, *Higher-order differential properties of Keccak and Luffa*, Fast Software Encryption 2011, 2011, to appear, draft available from Cryptology ePrint Archive, Report 2010/589.

[17] M. Brickenstein and A. Dreyer, *PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials*, Journal of Symbolic Computation **44** (2009), no. 9, 1326–1345, Effective Methods in Algebraic Geometry.

[18] The Python community, *Python Programming Language*, Python Software Foundation, 2009, http://www.python.org/.

[19] D. A. Cox, J. B. Little, and D. O'Shea, *Ideals, varieties, and algorithms*, third ed., Springer, 2007.

[20] J. Daemen, *Cipher and hash function design strategies based on linear and differential cryptanalysis, PhD thesis*, K.U.Leuven, 1995.

[21] J. Daemen and C. S. K. Clapp, *Fast hashing and stream encryption with PANAMA*, Fast Software Encryption 1998 (S. Vaudenay, ed.), LNCS, no. 1372, Springer-Verlag, 1998, pp. 60–74.

[22] J. Daemen, L. R. Knudsen, and V. Rijmen, *The block cipher Square*, Fast Software Encryption 1997 (E. Biham, ed.), Lecture Notes in Computer Science, vol. 1267, Springer, 1997, pp. 149–165.

[23] J. Daemen, M. Peeters, G. Van Assche, and V. Rijmen, *Nessie proposal: the block cipher* NOEKEON, Nessie submission, 2000, http://gro.noekeon.org/.

[24] J. Daemen and V. Rijmen, *The design of Rijndael — AES, the advanced encryption standard*, Springer-Verlag, 2002.

[25] ———, *Probability distributions of correlation and differentials in block ciphers*, Journal of Mathematical Cryptology **1** (2007), no. 3, 221–242.

[26] M. Duan and X. Lai, *Improved zero-sum distinguisher for full round keccak-f permutation*, Cryptology ePrint Archive, Report 2011/023, 2011, http://eprint.iacr.org/.

[27] M. Gorski, S. Lucks, and T. Peyrin, *Slide attacks on a class of hash functions*, Asiacrypt (J. Pieprzyk, ed.), Lecture Notes in Computer Science, vol. 5350, Springer, 2008, pp. 143–160.

[28] J. Kelsey, T. Kohno, and B. Schneier, *Amplified boomerang attacks against reduced-round mars and serpent*, Fast Software Encryption 2000 (B. Schneier, ed.), Lecture Notes in Computer Science, vol. 1978, Springer, 2000, pp. 75–93.

[29] D. Khovratovich, *Two attacks on RadioGatún*, 9th International Conference on Cryptology in India, 2008.

[30] L. R. Knudsen, *Truncated and higher order differentials*, Fast Software Encryption 1994 (B. Preneel, ed.), Lecture Notes in Computer Science, vol. 1008, Springer, 1994, pp. 196–211.

[31] L. R. Knudsen (ed.), *Fast software encryption, 6th international workshop, fse '99, rome, italy, march 24-26, 1999, proceedings*, Lecture Notes in Computer Science, vol. 1636, Springer, 1999.

[32] D. E. Knuth, *The art of computer programming, vol. 2, third edition*, Addison-Wesley Publishing Company, 1998.

[33] S. K. Langford and M. E. Hellman, *Differential-linear cryptanalysis*, Advances in Cryptology – Crypto '94 (Y. Desmedt, ed.), Lecture Notes in Computer Science, vol. 839, Springer, 1994, pp. 17–25.

[34] J. Lathrop, *Cube attacks on cryptographic hash functions, Master's thesis*, Available online, 2009, `http://www.cs.rit.edu/~jal6806/thesis/`.

[35] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1997.

[36] P. Morawiecki and M. Srebrny, *A sat-based preimage analysis of reduced KECCAK hash functions*, Cryptology ePrint Archive, Report 2010/285, 2010, `http://eprint.iacr.org/`.

[37] R. L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin, *The RC6 block cipher*, AES proposal, August 1998.

[38] W. A. Stein et al., *Sage Mathematics Software*, The Sage Development Team, 2009, `http://www.sagemath.org/`.

[39] D. Wagner, *The boomerang attack*, in Knudsen [31], pp. 156–170.

[40] R. Wernsdorf, *The round functions of Rijndael generate the alternating group*, Fast Software Encryption 2002 (J. Daemen and V. Rijmen, eds.), Lecture Notes in Computer Science, vol. 2365, Springer, 2002, pp. 143–148.

[41] Wikipedia, *Impossible differential cryptanalysis*, 2008, `http://en.wikipedia.org/wiki/Miss_in_the_middle_attack`.

[42] M. R. Z'aba, H. Raddum, M. Henricksen, and E. Dawson, *Bit-pattern based integral attack*, Fast Software Encryption 2008 (K. Nyberg, ed.), Lecture Notes in Computer Science, vol. 5086, Springer, 2008, pp. 363–381.