Cryptanalysis of SHA-3

Stefan Kölbl

Cryptanalysis of SHA-3

Master's Thesis

at

Graz University of Technology

submitted by

Stefan Kölbl

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology
A-8010 Graz, Austria

22 Apr 2013

© Copyright 2013 by Stefan Kölbl

Assessor: Dipl.-Ing. Dr.techn. Florian Mendel
Advisor: Dipl.-Ing. Dr.techn. Martin Schläffer

Kryptoanalyse von SHA-3

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Stefan Kölbl

Institute for Applied Information Processing and Communications (IAIK),
Technische Universität Graz
A-8010 Graz

22. April 2013

© Copyright 2012, Stefan Kölbl

Diese Arbeit ist in englischer Sprache verfasst.

Assessor: Dipl.-Ing. Dr.techn. Florian Mendel
Advisor: Dipl.-Ing. Dr.techn. Martin Schläffer

Abstract

Cryptographic hash functions are a fundamental part of modern cryptography and play an important role in many practical applications. In recent years new techniques have been developed in this field and attacks for popular designs like MD5 and SHA-1 were published. As a consequence, NIST announced a public competition in 2007 to find a new hash standard, the SHA-3 competition. This competition ended in October 2012 and Keccak was selected as the winner.

In this thesis, the security of the hash function Keccak is evaluated. An overview of the current state of the security analysis is given and attacks on round-reduced variants of Keccak are presented using techniques from differential cryptanalysis. Furthermore, the applicability of algebraic attacks to find preimages is evaluated.

A tool assisted method, which was previously used for the analysis of SHA-2, is applied on the Keccak hash function and allows to find practical collisions for up to 4 rounds. The attack is of practical complexity and takes only minutes on recent hardware. In addition, a technique is shown to find new differential characteristics, for larger output sizes of Keccak, by combining multiple characteristics.

Keywords: hash function, Keccak, cryptanalysis, SHA-3, collision resistance, algebraic attacks, differential characteristics

Kurzfassung

Kryptographische Hashfunktionen sind ein wesentlicher Teil der modernen Kryptographie und haben eine bedeutende Rolle in vielen praktischen Anwendungen. In den letzen Jahren wurden neuen Techniken zur Analyse entwickelt und Attacken auf bekannte Hashfunktionen wie MD5 und SHA-1 publiziert. Infolgedessen kündigte NIST einen öffentlichen Wettbewerb an, um einen neuen Standard für Hashfunktionen zu finden, den SHA-3 Wettbewerb. Dieser Wettwerb endete im Oktober 2012 und Keccak wurde als Gewinner ausgewählt.

In dieser Arbeit wird die Sicherheit der Hashfunktion Keccak untersucht. Es wird ein Überblick über existierende Attacken auf Keccak gegeben und es werden Attacken auf runden-reduzierte Varianten von Keccak, basierend auf Differenzieller Kryptoanalyse, präsentiert. Weiters wird die Anwendbarkeit von algebraischen Attacken untersucht um Urbilder zu finden.

Eine automatisierte Methode, welche zuvor für die Analyse von SHA-2 verwendet wurde, wird auf Keccak angewendet und erlaubt es Kollisionen für bis zu 4 Runden zu finden. Die Attacke hat eine praktische Komplexität und benötigt nur wenige Minuten auf einem aktuellen Computer. Zusätzlich wird eine Methode präsentiert, um neue differentielle Charakteristiken für längere Ausgabegrößen von Keccak zu finden, in dem man mehrere Charakteristiken miteinander kombiniert.

Schlüsselwörter: Hashfunktion, Keccak, Kryptoanalyse, SHA-3, Kollision, algebraischer Angriff, Differenzielle Kryptoanalyse

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Contents

List of Figures

List of Tables

Acknowledgements

I am indebted to my colleagues at the IAIK (Institute for Applied Information Processing and Communications) for providing such a pleasent working enviroment, particularly the Krypto research group who have provided invaluable help and feedback during the course of my work.

I especially wish to thank Florian Mendel and Martin Schläffer, for their immediate attention to my questions and endless hours of discussions. I am also grateful to Vincent Rijmen, who made it possible for me to go a semester abroad to work on this thesis at the KU Leuven, Belgium.

Last but not least, without the support of my family and my girlfriend, this thesis would not have been possible.

Stefan Kölbl

Graz, Austria, April 2013

1
Introduction

Begin at the beginning and go on till you come to the end; then stop.

– Lewis Carroll, Alice in Wonderland

This thesis is about the cryptographic hash function Keccak. Cryptographic hash functions are a fundamental part of modern cryptography and are used in many practical applications, for instance verification of message integrity, message authentication or secure storage of passwords. A hash function computes a short identifier for a message, which is representatively used in cryptographic protocols, to provide integrity or authentication.

A cryptographic hash function takes an input of arbitrary finite length and produces a fixed sized output. Usually, the input domain is larger than the output domain, therefore these functions are many-to-one. As a result, the existence of two different messages having the same output is unavoidable. In consequence, for a hash function to be secure it should be computationally infeasible to find these collisions.

The most commonly used hash functions at the moment are SHA-1, SHA-256 and SHA-512 certified by NIST. They are part of several standards and based on the design principles of MD4 and MD5. In the last few years cryptanalysis made a huge leap forward and weaknesses have been found for these functions. Practical collisions have been shown for MD4 [1], MD5 [2] and SHA-0 [3]. Although the computational effort to construct collisions for SHA-1 is still impracticable, the security bound is much lower than expected [4]. Attacks on reduced rounds are possible and practical example have been shown [5]. For this reason there is a strong interest in designing new secure hash functions.

This thesis deals with the analysis of the hash function Keccak, which was selected by NIST as the winner of the SHA-3 competition.

1.1 SHA-3 Competition

The SHA-3 competition was a public competition held by NIST (National Institute of Standards and Technology), with the purpose of finding a new cryptographic hash algorithm. It was announced on November 2nd, 2007 with the goal to find a new standard by the end of 2012. There were 64 initial submissions by October 31th, 2008 and 51 were selected to advance to the first round. This round lasted till July 24th, 2009 and 14 candidates have been selected to advance to the second round. After briefly a year for the public review NIST selected five candidates for the final round: Blake [6], Grøstl [7], JH [8], Keccak [9] and Skein [10]. Out of these five finalists, NIST selected Keccak to become the new SHA-3 standard on October 2, 2012.

1.2 Outline

The thesis is structured as follows. In Chapter 2, the fundamental properties and design principles of hash functions are presented followed by generic attacks. Chapter 3 describes the Keccak hash function and its building blocks in detail. In Chapter 4 an overview of the current state of research on Keccak is given.

The main part of this thesis is the analysis of the Keccak hash function and can be found in Chapter 5. The first part of this chapter shows how algebraic attacks can be applied on Keccak and the results are evaluated. The second part deals with differential cryptanalysis. A tool-assisted approach is presented to automatically find complex differential characteristics for Keccak. The third part of the analysis presents a method, based on combining known high probability characteristic, to find new characteristics for larger output sizes. In Chapter 6, the conclusion can be found which summarises the results and discusses direction for future work.

2

Cryptographic Hash Functions

It turns out that an eerie type of chaos can lurk just behind a facade of order - and yet, deep inside the chaos lurks an even eerier type of order.

– Douglas R. Hofstadter

This chapter gives a brief introduction to cryptographic hash functions, their applications and properties. In addition some generic attacks are outlined to get a bound for the security provided by these functions. A more thorough introduction can be found in [11].

A hash function is an efficient deterministic algorithm, which maps an input of arbitrary length to a fixed size output called hash-value, digest or fingerprint (see Figure 2.1).

$$h : \{0,1\}^* \rightarrow \{0,1\}^n \tag{2.1}$$

A hash function associates a hash-value with every input which can be used as an identifier for this message. Consequently, no two messages should have the same output, a so-called collision. As the input domain is much larger than the output range, collisions are unavoidable. Thus, while these collisions are unavoidable it should be infeasible to find them efficiently.

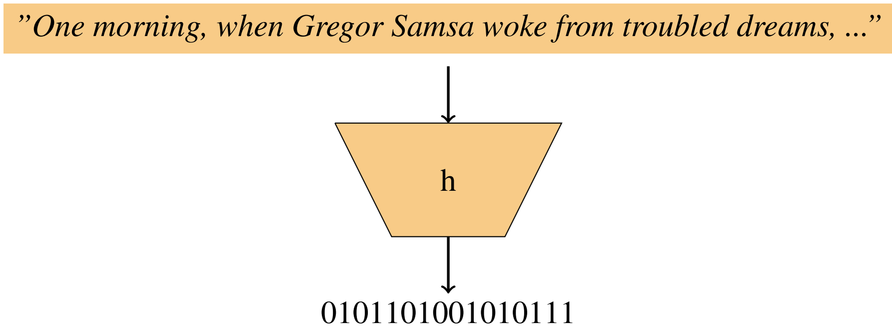


Figure 2.1: A hash function is a function h mapping an input of arbitrary length to an output of fixed length n.

2.1 Applications

Cryptographic hash functions are one of the most versatile cryptographic primitives and are a fundamental part of modern cryptography. A typical application for them is to provide message integrity. If a single bit is changed in a message it will influence the computation and result in a different output, allowing to detect any modifications.

In signature schemes, like the DSA (digital signature algorithm), hash functions are used as a short unique identifier for a message [12]. This scheme signs the hash, as a representative for the message, which speeds up the computation and also provides additional security compared to a raw RSA signature.

A second important application are MACs (message authentication code). A MAC is a keyed hash function that provides both integrity and authenticity of a message. For this a secret, shared by two parties Alice and Bob, is involved in the computation of the hash and allows the receiver to check if the message originates from the desired sender. HMAC (hash-based message authentication code) is a widespread algorithm used in standards like TLS [13] and IPSec [14].

Another common application is password protection. Passwords are usually not stored as plaintext but only the digest is stored. The password entered by the user is hashed and compared with the stored digest. This allows the original password to be kept secret due to the one-wayness of hash functions. For a secure hash function it should be infeasible to derive a password from the stored hash-value.

A further application is for confirmation of knowledge or commitment schemes. If someone wants to prove that he has some information without revealing it, the hash of this information can be made public. Once this information is public the commitment can be verified by computing this hash.

Cryptographic hash functions are also used for pseudo random number generation, key derivation and play an important role in micropayment systems like MicroMint [15] or Bitcoin [16].

Hash functions should be fast compute in general but this is not necessarily true for all applications. Password hashing and key derivation schemes like bcrypt [17], scrypt [18] and PBKDF2 [19] are designed to be computationally expensive to make brute force attacks less efficient.

Different applications might also have different security requirements. For instance if an attacker constructs two documents with the same hash-value then also the signature of this two documents will be the same. The attacker can deceive an user by signing one of the documents and gets a valid signature on the second document.

2.2 Security

As hash functions play such an important role in cryptography they have to fulfil various requirements to be considered secure. For discussing security the following three properties are used:

- Preimage resistance
- Second preimage resistance
- Collision resistance

An attack on a hash function typically tries to break one of these properties. For a secure hash function it is assumed that, if an attacker is computationally bound then it is infeasible to break any of these properties.

Furthermore, for an ideal hash function it would be desirable to behave like a random oracle. A random oracle outputs for every input a random value from the output domain. If an input is used a second time the same random value is chosen. This concept is important for security proofs in protocols and cryptographic primitives where one can prove that the system is secure if the hash function behaves like a random oracle.

No real hash function can implement a random oracle. Hence, the best that can be achieved is that there exists no efficient algorithm that can distinguish the output of a hash function from the output of a random oracle.

2.2.1 Preimage Resistance

A hash function is preimage resistant if it is hard to invert.

Definition 1. Preimage Resistance: For a given output y it should be computationally infeasible to find an input x' such that $y = f(x')$.

A hash function with n-bit output is preimage resistant if no algorithm exists that finds a preimage with a complexity of less than $\mathcal{O}(2^n)$.

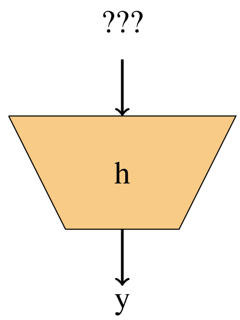


Figure 2.2: Preimage Resistance: The attacker needs to find a valid input which results in the given output y.

2.2.2 Second Preimage Resistance

Definition 2. Second Preimage Resistance: For given $x, y = h(x)$ it should be computationally infeasible to find $x' \neq x$ such that $h(x') = y$.

A hash function with n-bit output is second preimage resistant if no algorithm exists that finds a preimage with a complexity of less than $\mathcal{O}(2^n)$.

This property gives the attacker additional information on the input for the fixed output y. This could improve an attack, for instance by knowing the message length or the exact input to a block in an iterative scheme.

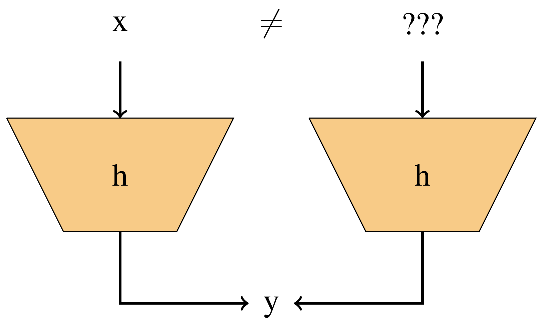


Figure 2.3: **Second Preimage Resistance:** Given an input/output pair the attacker needs to find a second input which results in the same output y.

2.2.3 Collision Resistance

Definition 3. Collision Resistance: It should be computationally infeasible to find two distinct inputs x, x' such that $h(x) = h(x')$.

A hash function with n-bit output is collision resistant if no algorithm exists that finds a preimage with a complexity of less than $\mathcal{O}(2^{n/2})$.

This problem might look similar to second preimage resistance but the attacker can choose both x and x' in this case and the output y is also not fixed, which enables the use of birthday attacks. As the input domain is much larger than the output domain collisions are unavoidable (pigeonhole principle), so the best one can achieve is that it is computationally infeasible to find a collision.

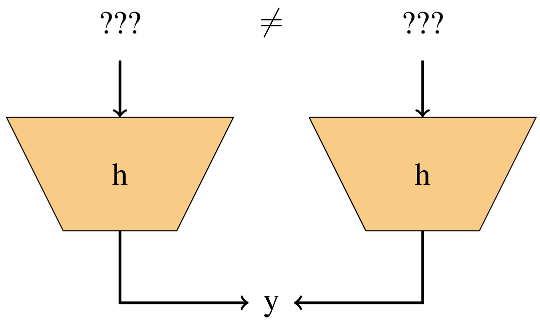


Figure 2.4: **Collision Resistance:** The attacker needs to find two different messages with the same output y.

2.3 Design

Most hash functions follow an iterative design similar to Figure 2.5. The input m is split into evenly sized blocks $M_1, M_2, \dots M_n$ and a compression function f is used

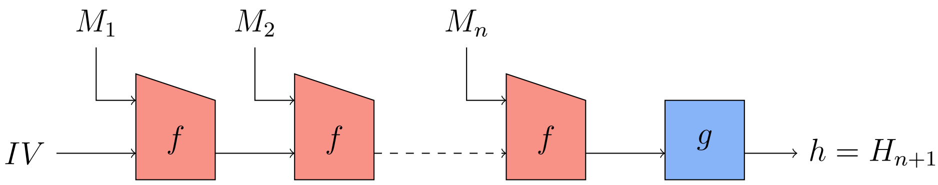


Figure 2.5: An iterative construction for a hash function, where IV is a fixed initial value.

to process each block iteratively. If the input m is not a multiple of the block size the message is padded accordingly. Examples for this padding can be found in the upcoming constructions.

2.3.1 Merkle-Damgård construction

The Merkle-Damgård construction is a method to build a collision-resistant hash function from a collision-resistant compression function [20]. It uses the same iterative approach and adds the length of the message at the end of the padding which is often referred to as Merkle-Damgård strengthening (MD-strengthening).

Algorithm 1 Merkle-Damgård Construction

Precondition: Message: m
Output: Hash: h
Apply padding to m and split in evenly sized blocks $M_1, M_2, \dots, M_n$ and
$H_0 = 0^n$
$H_i = f(H_{i-1}, M_i)$ for $1 \leq i \leq n$
$H_{n+1} = g(H_n)$

For designs like MD4 no output transformation is used and g is the identity function.

Padding

The input m is padded by appending a 1-bit followed by the minimum number of 0 bits to result in a multiple of the block-size. This is followed by an additional block which encodes the binary representation of the length of m. This is often referred to as Merkle-Damgård strengthening.

Proof of Collision-resistance

This construction gives a provable collision-resistant hash function. However, other flaws still exist which are discussed in Section 2.4.3.

Proof. Assume that the hash function h is not collision-resistant and the attacker can find a colliding message pair (M, M') such that $h(M) = h(M')$. Consider the following two cases:

- The length of the two messages is not equal. If the attacker has found a collision then the output of the last compression function call must be equal. The last message block contains the message length, hence M_n and M'_n are different but this implies that a collision for the compression function exists as $f(M_n, H_n) = f(M'_n, H'_n)$.
- The length of the two messages is equal. In this case at least one compression function call must lead to a collision $f(M_i, H_i) = f(M_j, H_j)$ for the output to be equal.

This proof assumes that no additional output transformation is applied else one has to consider collisions in the output transformation too. □

2.3.2 Sponge construction

The sponge construction is a mode of operation building a function which takes arbitrary sized input and generates arbitrary sized output. It is based on a fixed-sized permutation f, a padding rule and takes two parameters: the rate r and the capacity c. The sponge construction is iterative and operates on an internal state S of size $b = r + c$.

First, split the input m into blocks $M_0, M_1, \ldots, M_n$ of size r by using the padding rule. Set the initial state to $S = (0 \ldots 0)$ and process the input through the following two phases

- **Absorb:** XOR the ith message block to the first r bits of the state and update the state with the f-permutation. Repeat this step for all message blocks.
- **Squeeze:** Append the first r bits of the state to h and update the state. Repeat this step to generate more output bits.

An outline of this procedure can be seen in Figure 2.6. When using a random permutation the sponge construction is as secure as a random oracle apart from inner collisions [21].

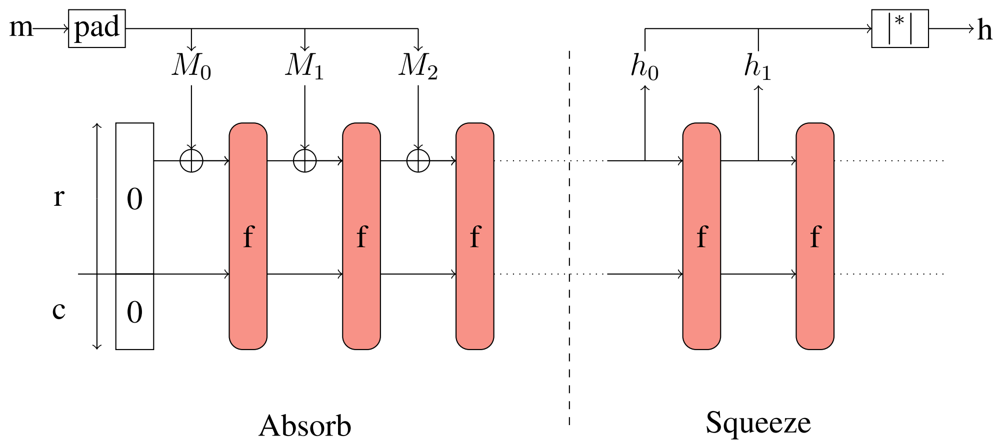


Figure 2.6: The sponge construction takes input of arbitrary length and computes an output of arbitrary length. It uses a fix-sized invertible permutation f and the input is processed iteratively.

Padding

The input m is padded to be a multiple of the block-size. This is done by appending a 1-bit followed by the minimum number of 0 bits and a 1 bit to result in a multiple of the block-size. This padding is called multi-rate padding.

$$\text{pad}(m) = (m||10^*1) \tag{2.2}$$

2.4 Generic Attacks

In this section, general attacks on hash functions are outlined. These kind of attacks can be applied to any hash function disregarding the underlying structure. The hash function acts like a black box and the only relevant parameter is the length n of the hash value. It is assumed that the output of the hash function is uniformly distributed. If this is not the case generic attacks can be more efficient.

2.4.1 Brute-Force Attack

The simplest approach for an attacker to find a preimage would be to test different messages and check if he gets the desired output y. If the hash function has n-bit output, then the probability, given a random input x, $h(x) = y$ is equal to 2^{-n}. Hence, after about $\mathcal{O}(2^n)$ trials a correct input will be found.

The same method allows to find a second preimage with the only difference is to discard x if it equals the given input. A generic attack to find a (second) preimage

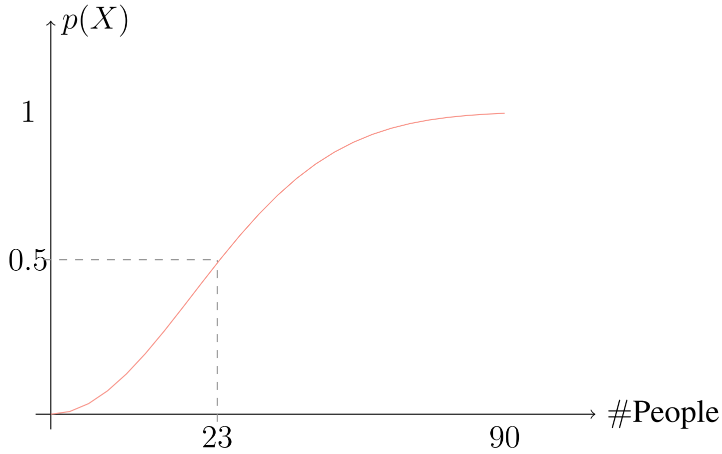


Figure 2.7: The probability that two persons share the same birthday is $>$ 0.5, for a group of 23 people.

has a complexity of $\mathcal{O}(2^n)$.

2.4.2 Birthday Attack

The birthday paradox states that in a group of 23 people, the probability is > 0.5 that two persons share the same birthday. The probability for this event grows quickly to 1 (as can be seen in Figure 2.7).

This fact appears when searching collisions. Consider a hash function with n-bit output, then the probability that two messages collide is 2^{-n}. It follows that the probability that no collision occurs after N trials is:

$$p'(N) = 1 \cdot (1 - \frac{1}{2^n}) \cdot (1 - \frac{2}{2^n}) \dots (1 - \frac{N-1}{2^n}) \approx e^{-\frac{N^2}{2^{n+1}}} \tag{2.3}$$

For details on this approximation see [22]. The probability for a collision after N trials is then given by the converse probability:

$$p(N) = 1 - p'(N) \approx 1 - e^{-\frac{N^2}{2^{n+1}}} \tag{2.4}$$

Consequently, the expected numbers of trials is $\sqrt{\ln(2)2} \cdot 2^{n/2}$ before a collision occurs. A generic attack to find a collision has a complexity of $\mathcal{O}(2^{n/2})$.

The simplest version of this attack is to store a list of hash outputs and subsequently compute new outputs [23]. If an output is already in the list then a collision was found (see Algorithm 2).

The high memory requirements make this approach infeasible in practice even for a relative small output size, but memoryless variations can be applied [24] for

Algorithm 2 Birthday Attack - Yuval

Precondition: List L

while do
  select random message m
  if $h(m) \in L$ **then**
    found collision
  else
    add $(h(m), m)$ to L

instance Floyd's cycle–finding algorithm.

From this attacks it follows that, if a hash function should have k-bit collision resistance then the output must be at least of size $n = 2k$. The digest size, of the currently most common used hash algorithms, is listed in Table 2.1.

Table 2.1: Digest sizes for different hash algorithms.

Algorithm	Output size	Year released
MD5	128-bit	1992
SHA-1	160-bit	1995
RIPEMD-160	160-bit	1996
Whirlpool	512-bit	2000
SHA-2	224-, 256-, 384- and 512-bit	2001
Keccak (SHA-3)	224-, 256-, 384- and 512-bit	2012

2.4.3 Attacks on the Merkle-Damgård construction

The Merkle-Damgård construction is provable collision-resistant but has other undesirable properties. The length extension attacks allows an attacker to compute $H(\text{pad}(m)||X)$ without knowing m (see Figure 2.8). This can be a problem, for instance if a MAC is constructed by computing $H(\textit{key}||m)$. In this case it would allow an attacker to forge valid tags for messages of the structure $H(\text{pad}(\textit{key}||m)||X)$. A random oracle would not have such a property [25].

Kelsey and Schneier showed that finding a second preimage for hash functions with Merkle-Damgård strengthening is easier for large messages [26]. Using their results, a second preimage for SHA-1 for a message of size 2^{60} can be found with a complexity of 2^{106} compared to the costs of 2^{160} for the generic attack.

Computing multi-collisions for hash functions based on the Merkle-Damgård construction can be done efficiently. Multi-collisions are t-tuples of messages which all hash to the same output. Joux presented an approach to construct 2^t-collisions at t times the costs of finding a collision for two messages [27].

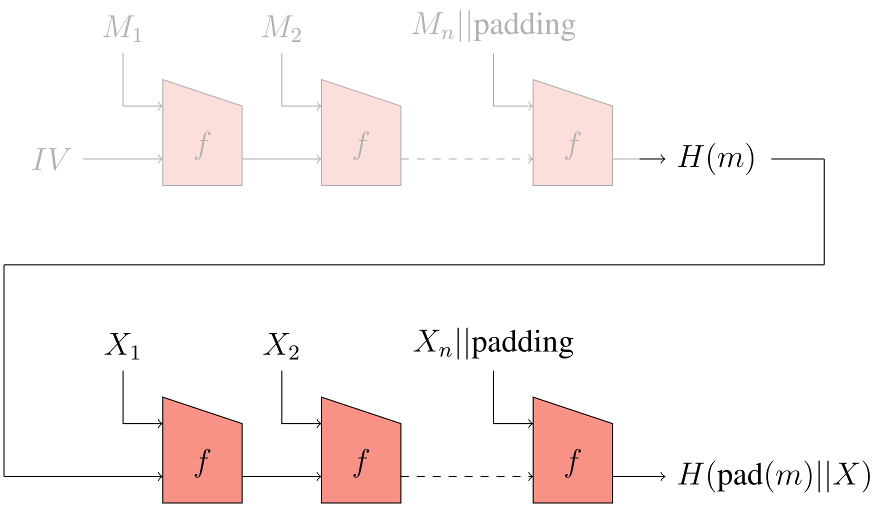


Figure 2.8: The length extension property allows to compute $H(\text{pad}(m)||X)$, using $H(m)$ as input to the compression function f. An attacker needs no further information on the structure of m.

3

Keccak

This chapter is about the Keccak hash function designed by Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. Keccak was submitted to the SHA-3 competition and selected by NIST as the winner on October 2nd, 2012.

Keccak is a sponge based construction using the fix sized permutation Keccak-f. The internal structure of this permutation is very important to understand the following analysis, thus a detailed description of the building blocks is given in this chapter.

3.1 Description of Keccak

Keccak is a family of hash functions based on the sponge construction where the state can have a size $b \in \{25, 50, 100, 200, 400, 800, 1600\}$. It uses the permutation Keccak-f and the padding scheme defined in Section 2.3.2. A specific instance of Keccak is notated as Keccak$[r, c, n_r]$ where r is the rate, c the capacity and n_r the number of rounds.

The permutation f used in Keccak operates on a three-dimensional state with elements in $\mathbb{F}_2$ (see Figure 3.1). The dimensions for this state are $5 \times 5 \times w$ with $w \in \{1, 2, 4, 8, 16, 32, 64\}$. This allows to represent each lane as a w-bit word. A three-dimensional array is used, $S[x][y][z]$, to describe the state. Some additional terms are defined to ease the description of the state:

- A **plane** is a set with constant y-coordinate ($S[*][y][*]$) of size $5w$.
- A **slice** is a set with constant z-coordinate ($S[*][*][z]$) of size 25.
- A **sheet** is a set with constant y-coordinate ($S[x][*][*]$) of size $5w$.
- A **row** is a set with constant y- and z-coordinate ($S[*][y][z]$) of size 5.
- A **column** is a set with constant x- and z-coordinate ($S[x][*][z]$) of size 5.
- A **lane** is a set with constant x- and y-coordinate ($S[x][y][*]$) of size w.

A visualisation of this terms can be found in Appendix A.1.

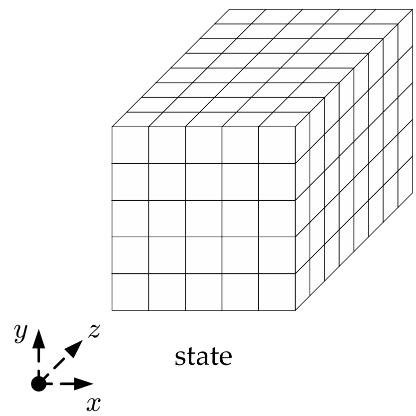


Figure 3.1: Outline of the internal state of Keccak with a width of 8[1].

The hash h for a message m is computed in the following way for Keccak$[r, c, n_r]$:

1. Initialise the state $S[x][y][z] = 0$ for $x = 0 \ldots 4$, $y = 0 \ldots 4$ and $z = 0 \ldots w$.
2. Compute the padded message $M = m||10^{*}1$ such that M is a multiple of r.
3. Absorb the next r-bit message block by computing $S[x][y] = S[x][y] \oplus M_i$ and update the state by computing $S = f(S)$.
4. Squeeze until the requested number of bits is reached.

[1]Image from `http://keccak.noekeon.org/`

3.1.1 Keccak-f

Keccak uses the iterative permutation Keccak-f operating on $\mathbb{F}_2^w$, with w being the word size. The permutation consists of multiple rounds in which five functions are used in sequence $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$. The number of rounds n_r depends on the word size of the lane:

$$n_r = 12 + 2\log_2(w) \tag{3.1}$$

Apart from ι, this functions are the same for each round.

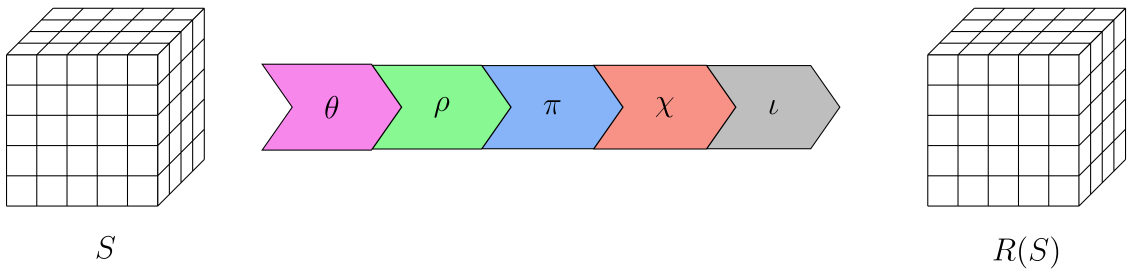


Figure 3.2: One round of the Keccak-f permutation.

Description of θ

The θ function is linear and provides diffusion over the whole state. The step adds to every bit of the state $S[x][y][z]$ the bitwise sum of the neighbouring columns $S[x-1][*][z]$ and $S[x+1][*][z-1]$.

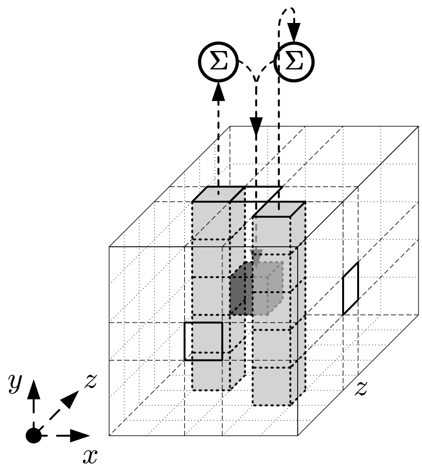


Figure 3.3: θ step of Keccak-f.

This procedure can also be described with the following equation:

$$\theta : S[x][y][z] \leftarrow S[x][y][z] + \sum_{n=0}^{4} S[x-1][n][z] + \sum_{n=0}^{4} S[x+1][y][z] \tag{3.2}$$

For computing the inverse of this step see [9].

Description of ρ

This function rotates the bits in every lane by a constant value. This is done to speed up dispersion between the slices. The constants are given in Table 3.1.

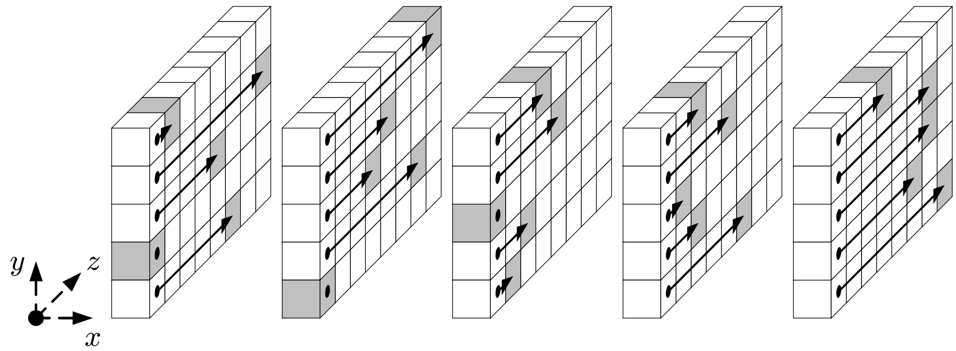


Figure 3.4: ρ step of Keccak-f.

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

Table 3.1: Rotation constants for ρ.

The inverse of ρ can be computed by shifting with the same constants in the opposite direction.

Description of π

This function transposes the lanes using the following function:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.3}$$

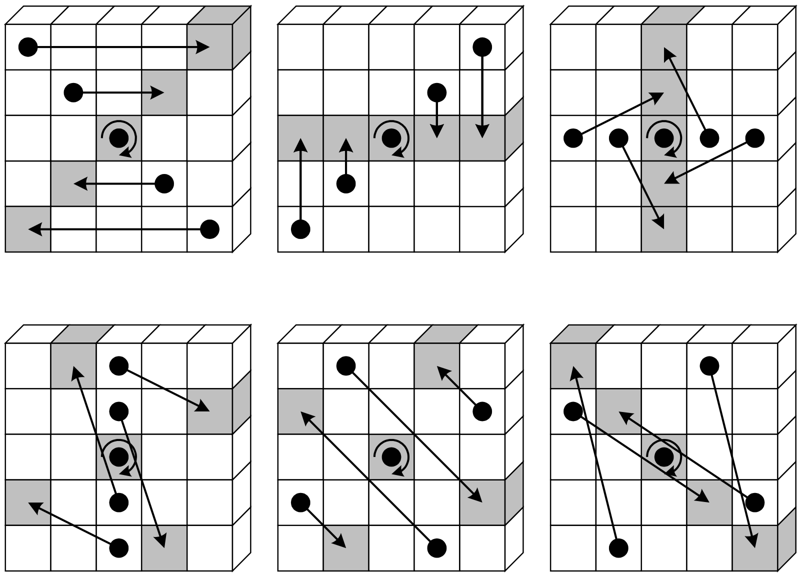


Figure 3.5: π step of Keccak-f.

The inverse of π is given by:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.4}$$

Description of χ

This step is the only non-linear step in Keccak and operates on each row of 5 bits. It can be seen as applying in parallel a 5-bit s-box to all rows:

$$\chi : S[x][y][z] \leftarrow S[x][y][z] \oplus ((\neg S[x+1][y][z]) \wedge S[x+2][y][z]) \tag{3.5}$$

The algebraic degree of χ is two and it is invertible. The inverse of this function has an algebraic degree of three.

Description of ι

This steps adds a round dependent constant to the state. The constants are different to avoid attacks exploiting symmetry like slide attacks. For a list of the constants see [9].

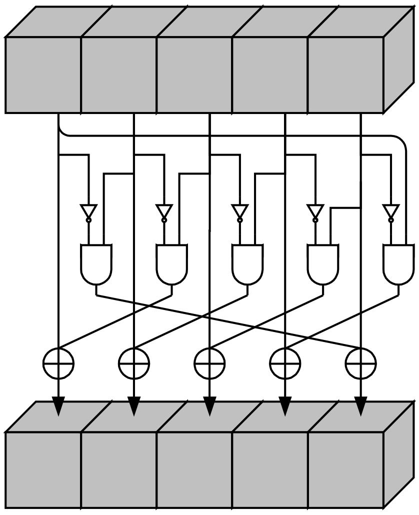


Figure 3.6: χ step of Keccak-f.

3.2 Keccak Challenges

The Keccak challenges are a set of challenges proposing different parameters to encourage the cryptanalysis of Keccak. The capacity is fixed to 160-bit which results in a security level of 2^{80} against birthday attacks. Table 3.2 contains the parameters for the challenges followed by the parameters recommended for SHA-3 with different output sizes. In the following analysis both variants will be used as attack targets.

Table 3.2: Parameters for Keccak divided up in Keccak challenges and recommended values for SHA-3.

Name	Word size	Rate	Capacity	Output size
Keccak$[40, 160]$	8	40	160	160
Keccak$[240, 160]$	16	240	160	160
Keccak$[640, 160]$	32	640	160	160
Keccak$[1440, 160]$	64	1440	160	160
Keccak$[1152, 448]$	64	1152	448	224
Keccak$[1088, 512]$	64	1088	512	256
Keccak$[832, 768]$	64	832	768	384
Keccak$[576, 1024]$	64	576	1024	512

4

Existing Analysis of Keccak

This chapter gives an overview of the current state of analysis on Keccak. The first section will discuss structural attacks, while the second is about differential attacks which are also a significant part of the analysis presented in this thesis.

4.1 Structural Attacks

Aumasson and Khovratovich published the first external analysis on Keccak [28]. They applied automated cryptanalytic tools, using the triangulation algorithm [29] and cube testers, to detect structures in reduced round versions of Keccak. The application of their tools was limited due to the good diffusion properties of the inverse of θ.

Aumasson and Meier presented a zero-sum distinguisher for Keccak-f for up to 9 rounds with a practical complexity and for up to 16 rounds [28] with a theoretical complexity. Boura and Canteaut extended this distinguisher to 20 rounds [30].

Morawiecki and Srebrny presented a SAT-based analysis to find preimages for reduced Keccak variants [31]. The idea is to formulate the problem of finding a preimage as a SAT problem. The first step is to generate the CNF which is then processed with a SAT solver to find the preimage. The main advantage of this attack is that highly optimised SAT solvers exist to solve this hard problem. The results of the SAT-based analysis suggest that Keccak is very resistant to this kind of attacks. The attack only worked on 3-round Keccak$[1024, 576]$ with 40 unknown message bits. Using this approach, they found collisions up to 2 rounds for the Keccak challenges for Keccak$[240, 160]$, Keccak$[640, 160]$ and Keccak$[1440, 160]$.

4.2 Differential Attacks

Differential cryptanalysis is an important tool in the analysis of cryptographic primitives and plays a key role in the results presented in this thesis. For a detailed explanation of differential cryptanalysis see Section 5.2.

Naya-Plasencia, Röck and Meier presented various attacks on Keccak based on differential cryptanalysis [32]. They use an efficient method to find low weight differential paths using column-parity kernels which are also an important part of the analysis in this thesis. The details of this method can be found in Section 5.3. In their work they proposed the following attacks: a preimage attack on 2 rounds, a collision attack on 2 rounds, a near-collision on 3 rounds and a distinguisher for 4 rounds. The results can be applied for Keccak$[1152, 448]$ and Keccak$[1088, 512]$ with a complexity of $\approx 2^{33}$.

Duc et al. presented new differential paths and used them for a rebound attack on the internal permutation of Keccak [33]. The method to find this new paths is also based the column-parity property. The rebound attack was first proposed by Mendel et al. in [34] and applied to AES-based hash functions like Grøstl and Whirlpool.

For the rebound attack, a permutation P is split into three parts $P = E_f \circ E_i \circ E_b$. The attack proceeds now in two phases:

- The **inbound phase** covers the middle part E_i. This part typically covers the most expensive part in a characteristic and computes solutions by propagating differences forward/backward through the linear layers and match them at a single s-box layer.
- The **outbound phase** propagates each solution, from the inbound phase, in both directions E_f and E_b.

The rebound attack is convenient to apply on AES-based permutations, as one can use truncated differentials. This is not the case for Keccak due to the bit-oriented design which makes the application more difficult. Duc et al. proposed a practical distinguisher with complexity 2^{32} for 6 rounds and with a complexity of $2^{491.47}$ for 8 rounds using the Keccak permutation with a width of 1600 bits.

The best practical attack published is a 4-round collision by Dinur, Dunkelman and Shamir. In [35] they present 4-round collisions for Keccak$[1152, 448]$ and Keccak$[1088, 512]$ and near-collisions for 5 rounds with the same parameters.

Their approach is to use a high probability characteristic and connect it to the starting point over 2 rounds. They presented the target difference algorithm to solve this problem of connecting to the starting point. This algorithm exploits that the algebraic degree of the non-linear layer is only 2 and makes use of the degree of freedom in the message input.

The high probability characteristic is a 2-round column-parity kernel. These characteristics are examined in detail in Section 5.3.

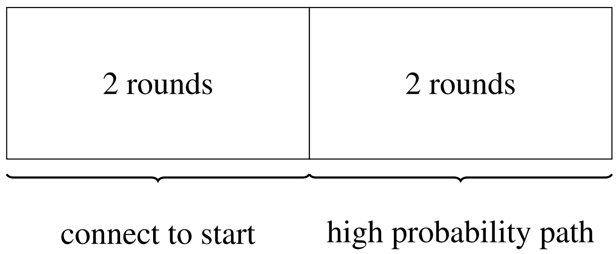


Figure 4.1: Outline of the attack by Dinur, Dunkelman and Shamir.

In [36] they presented the first attacks on Keccak$[832, 768]$ and Keccak$[576, 1024]$. The attacks are based on internal differential cryptanalysis. A practical attack on 3 rounds and an attack on 5-round Keccak$[1088, 512]$ with a complexity of 2^{115} are shown. A summary of the attacks by Dinur, Dunkelman and Shamir can be found in Table 4.1.

Table 4.1: Attacks on different Keccak versions by Dinur, Dunkelman and Shamir.

	Keccak-224	Keccak-256	Keccak-384	Keccak-512
Collision	4	4 , 5 (2^{115})	3, 4 (2^{147})	3

5 Analysis

This chapter presents the results of our analysis on the Keccak hash function. First, the use of algebraic attacks to find preimages for reduced round versions of Keccak is evaluated. This section shows how to derive a system of non-linear equations, for which finding a solution is equivalent to finding a preimage for Keccak.

The second part of this chapter deals with differential cryptanalysis and is the main part of this thesis. A tool-assisted method based on the concept of generalized conditions is used to find both a differential characteristic and the corresponding message pair. This method allows to find collisions for up to 4 rounds of Keccak with a practical complexity.

The third part is about finding new high probability differential characteristics for more than 2 rounds or larger output sizes of Keccak. Based on the column-parity property of Keccak, new characteristics are constructed and combined to find new collision attacks on Keccak.

5.1 Algebraic Attack

An algebraic attack is a method of cryptanalysis based on expressing the cryptographic primitive as a system of equations, fixing known variables and solving this system. This is done to find a secret key in an encryption system or in the case of hash functions to find a preimage or collision. A system of linear equations can be efficiently solved, therefore cryptographic primitives are designed to be highly non-linear. The sheer size of the system and the non-linearity make this a hard problem to solve.

This section starts with an introduction on the concept of Gröbner bases and the required definitions. A short description of Buchberger's algorithm to find Gröbner bases is given and it is shown how it can be used to solve systems of non-linear equations. The subsequent section shows how the problem of finding preimages for Keccak can be solved with Gröbner bases.

The following notation and preliminaries are based on the work in [37] and [38], where also a more thorough discourse of the mathematical background can be found.

5.1.1 Preliminaries

Notations

Some notation which is used in the following sections:

- $\mathbb{F}_p$ is the finite field of order p with p being prime.
- $\mathbb{F}_{p^n}$ is the finite extension field of degree n over $\mathbb{F}_p$.
- $\mathbb{P}$ is a polynomial ring $\mathbb{F}[x_1, \ldots, x_n]$ in the variables $x_1, \ldots x_n$.
- lcm(a, b) is the least common multiplier of a and b.

Polynomials and Ideals

Definition 4. A monomial in $x_1, \ldots x_n$ is a product of the form

$$x_1^{\alpha_1} \cdot x_2^{\alpha_2} \ldots x_n^{\alpha_n} \tag{5.1}$$

Definition 5. A polynomial f in $x_1, \ldots x_n$ is a finite linear combination of monomials of the form

$$f = \sum_i a_\alpha x^\alpha, \;\; a_\alpha \in k \tag{5.2}$$

where

- a_α are the coefficient of the monomial x^α
- $a_\alpha x^\alpha$ is a term of f
- $\deg(f)$ is called the total degree of f and is the maximum $|\alpha|$ such that $a_\alpha \neq 0$

Definition 6. A subset $I \subset \mathbb{F}[x_1, \ldots, x_n]$ is an **ideal** if it satisfies:

- $0 \in I$
- If $f, g \in I$, then $f + g \in I$
- If $f \in I$ and $h \in \mathbb{F}[x_1, \ldots, x_n]$, then $hf \in I$

Monomial Ordering

For univariate polynomials it is easy to determine the total degree of a given polynomial by just determining the largest monomial. For multivariate polynomials this is not the case. For instance, it is not clear whether $x^4y^3z^2$ should be greater or smaller than $x^2y^7z^4$. There exist different possibilities to define the ordering. Therefore a monomial ordering is needed to compare them.

Definition 7. A **monomial ordering** on $\mathbb{F}[x_1, \ldots, x_n]$ is a relation $>$ on $\mathbb{Z}^n_{\geq 0}$ or equivalently, any relation on the set of monomials $x^i, i \in \mathbb{Z}^n_{\geq 0}$ satisfying:

- $>$ is a total (or linear) ordering on $\mathbb{Z}^n_{\geq 0}$
- If $\alpha > \beta$ and $\gamma \in \mathbb{Z}^n_{\geq 0}$ then $\alpha + \gamma > \beta + \gamma$
- $>$ is a well-ordering on $\mathbb{Z}^n_{\geq 0}$. This means that every non-empty subset of $\mathbb{Z}^n_{\geq 0}$ has a smallest element under $>$

There are many different monomial ordering. The two most common used monomial orderings are the "lexicographical" and the "degree reverse lexicographical" ordering which is used throughout this thesis.

Definition 8. **Degree reverse lexicographical ordering:** Let $\alpha = (\alpha_1 \ldots \alpha_n)$ and $\beta = (\beta_1 \ldots \beta_n)$ than $\alpha > \beta$ if

- $|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i$ or
- $|\alpha| = |\beta|$ and the rightmost nonzero entry of $\alpha - \beta$ is negative

Example 1. To illustrate how the degree reverse lexicographical ordering works

- $x^2y^7z^4 > x^4y^3z^2$ since $|(2,7,4)| = 13 > |(4,3,2)| = 9$
- $x^2y^7z^4 > x^3y^4z^6$ since $|(2,7,4)| = 13 = |(3,4,6)| = 13$ and $(2,7,4) - (3,4,6) = (-1,3,-2)$

The monomial ordering can now be applied to polynomials by reordering the terms by size with respect to $>$.

Definition 9. Let $f = \sum_\alpha a_\alpha x^\alpha$ be a nonzero polynomial in $\mathbb{F}[x_1, \ldots, x_n]$.

- The **multidegree** of f is

$$\text{multideg}(f) = \max(\alpha \in \mathbb{Z}^n_{\geq 0} : a_\alpha \neq 0) \tag{5.3}$$

  (the maximum is taken with respect to $>$)

- The **leading term** of f is

$$\text{LT}(f) = a_{\text{multideg}(f)} x^{\text{multideg}(f)} \tag{5.4}$$

Example 2.

$$\text{LT}(-3x^4y^1 + 2x^3y^2 + 4y^2 + x) = -3x^4y^1 \tag{5.5}$$

5.1.2 Gröbner Basis

The theory of Gröbner basis was developed by Buchberger in 1965 [39]. It can be seen as a generalisation of:

- the Euclidean algorithm for computing univariate greatest common divisors.
- Gaussian elimination for linear systems of equations.

Definition 10. Fix a monomial order. A finite subset $G = g_1, \ldots, g_n$ of an ideal I is called a **Gröbner basis** if

$$\langle \text{LT}(g_1), \text{LT}(g_2), \ldots, \text{LT}(g_n) \rangle = \langle \text{LT}(I) \rangle \tag{5.6}$$

5.1.3 Algorithms to find Gröbner Bases

Finding a Gröbner basis for an ideal is a hard problem and various algorithms exist to find them. The example given here is Buchberger's algorithm (see Algorithm 3) which takes as input a finite set of polynomials and computes the Gröbner basis with respect to a given monomial order.

Definition 11. Let $f, g \in \mathbb{F}[x_1, \ldots, x_n]$ be nonzero polynomials. The **S-polynomial** of f and g is the combination

$$S(f,g) = \frac{\mathrm{lcm}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LT}(f)} \cdot f - \frac{\mathrm{lcm}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LT}(g)} \cdot g \tag{5.7}$$

The algorithm will always terminate but the worst case running time is double exponential in the number of variables [40].

Algorithm 3 Buchberger's Algorithm

Precondition: $F = \langle f_1, \ldots, f_n \rangle$
Output: Gröbner Basis $G = (g_1, \ldots, g_n)$

```
G ← F
repeat
    G' ← G
    for each pair p, q, p ≠ q in G' do
        S ← Remainder of S(p, q)/G'
        if S ≠ 0 then
            G ← G ∪ S
until G = G'
```

Other popular algorithms are:

- The Faugère F4 [41] and F5 [42], which are both based on the principles of Buchberger's algorithm.
- The slimgb algorithm, which is also used in the Sage computer algebra system [43].

Example 3. Consider the following set of polynomials in $\mathbb{F}_2[x, y]$

$$f_0 = x^4 + x^2 + x + 1 \tag{5.8}$$

$$f_1 = x^4 + x^3 + y^2 + x \tag{5.9}$$

$$f_2 = x^3 + xy \tag{5.10}$$

The Gröbner basis for this example is obtained with Sage (see Listing 5.1). First the polynomial ring R and an ideal I are defined. Using the built-in algorithms the

Gröbner basis can be computed. The resulting Gröbner basis gives the solution $x = 1, y = 1$ to the non-linear system of equations.

```
sage: R.<x,y> = PolynomialRing(GF(2),'degrevlex')
sage: f0 = x^4 + x^2 + x + 1
sage: f1 = x^4 + x^3 + y^2 + x
sage: f2 = x^3 + x*y
sage: I = Ideal(a,b,c)
sage: I.groebner_basis()
sage:          [x + 1, y + 1]
```

Listing 5.1: Computing the Gröbner basis of a simple ideal

5.1.4 Algebraic Attack

An algebraic attack is a method for cryptanalysis for cryptographic primitives. The idea for this attacks is to express the problem of finding a secret key (or a preimage), as a system of equations. Solving a system of linear equations can be done in polynomial runtime but for a non-linear system the problem is computationally hard. Therefore, cryptographic primitives are designed to achieve a high degree of non-linearity. This can be done using various techniques, for instance s-boxes.

Non-linear systems of equations can be solved by using Gröbner basis based algorithms. In general algorithms for finding Gröbner bases have at least exponential running time, which would not be feasible for systems of this size. However, for hash functions the equation systems are very structured and this might allow to compute the Gröbner basis in practice.

5.1.5 Attacking Keccak

The algebraic attack on Keccak can be split into three steps (see Figure5.1):

1. Express Keccak as a system of non-linear equations.
2. Fix the value of all variables which are predefined.
3. Use algorithms to compute the Gröbner basis.

Equation system

The first question which arises is how to describe Keccak algebraically. There are different options to represent the internal state of Keccak. Due to the bitwise structure of the steps a representation over $\mathbb{F}_2$, by mapping every bit to variable in $\mathbb{F}_2$,

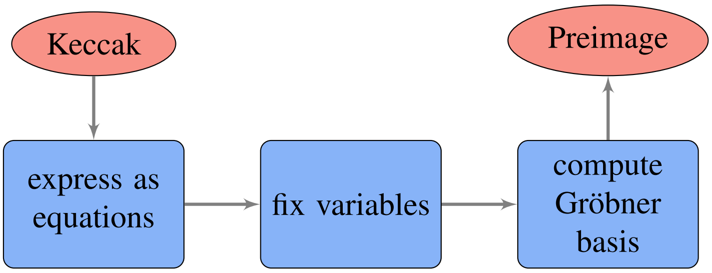


Figure 5.1: Outline of the algebraic attack for Keccak.

seems a good choice. This allows to derive a clean definition of the equations from the steps. Another possible choice would be to map every lane to an element of $\mathbb{F}_2^n$ where n equals the lane-size or to map every row to $\mathbb{F}_2^5$ but doing so would only be beneficial for some specific steps of Keccak and unfavourable for the others. The following notation is used to name the variables:

- The position of single bit in a state X is notated as

$$X^b_{y,z} \tag{5.11}$$

  where y is the position in the column, z is the position in the row and b the position on the lane.

- The states X are named in the following way: A is the initial state after the message has been xored to the state. BL is the state after applying $\pi \circ \rho \circ \theta$. BNL after applying $\iota \circ \chi$. CL is the next state after applying the linear steps followed by CNL and so on.

Getting the equations from the steps is done by first separating the linear and non-linear part:

- The linear steps ρ and π only move the position of the related bits and do not change any values. Therefore this steps can be represented by changing the corresponding variables in the related equations. In the θ step 11 bits are involved to compute a single output bit. For every bit of the state a linear equation with 12 variables is added. For example one equation might look like this:

$$BL^0_{0,0}+A^0_{0,0}+A^{15}_{0,1}+A^{15}_{1,1}+A^{15}_{2,1}+A^{15}_{31}+A^{15}_{4,1}+A^0_{0,4}+A^0_{1,4}+A^0_{2,4}+A^0_{3,4}+A^0_{4,4}=0$$

- The non-linear step χ uses 3 bits to compute one output bit. For every bit of the state a non-linear equations with 4 variables is added:

$$BNL^0_{0,0} + BL^0_{0,0} + (BL^0_{0,1} + 1) \cdot BL^0_{0,2} = 0 \tag{5.12}$$

The number of equations and variables depends on the state size of Keccak. For the initial state $b = r + c$ variables are added. The initial state is all 0 and xored to the message. Therefore, it is sufficient to add the result of this computation to the equation system.

Depending on the capacity, c of this variables are fixed and c equations have to be added. For each following round we need b variables and equations for the output of the linear step and b variables and equations for the output of the non-linear step. For the hash value only the first n bits contributing to the output are relevant and the rest of the state is truncated. This allows to drop the r corresponding equations because they do not contribute anything to the hash-value.

- Number of variables for: Keccak$[c, r, n_r] = b + 2n_r b$
- Number of equations for: Keccak$[c, r, n_r] = c + 2n_r b - r$

where n_r denotes the number of rounds.

Fixing variables

For the algebraic analysis Keccak$[r = 240, c = 160]$ is used as it is part of the Keccak challenges (see Section 3.2). Therefore 160 bits in the input state A are fixed to 0 and the output is truncated to 160 bits (see Figure 5.2). The variables at the output are fixed to the given hash for the preimage attack.

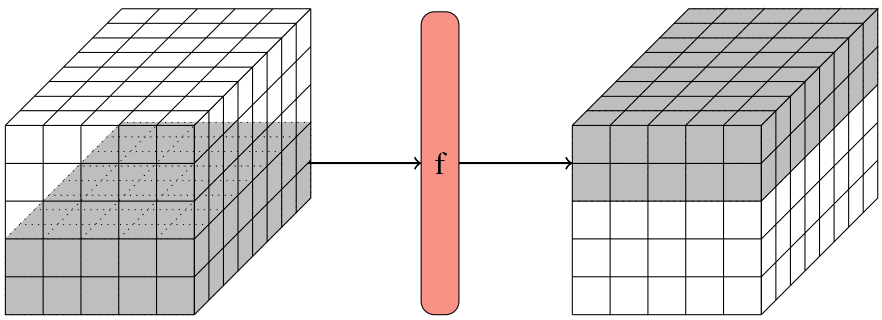


Figure 5.2: Variables corresponding to grey shaded bits are fixed.

Optimisation for last round

For a preimage some of the bits are fixed at the output. An important observation is that, if a full row at the output of χ is fixed then also the input row is known (as for Keccak$[r = 240, c = 160]$ in Figure 5.2). This allows to ignore χ in the last round when searching for a preimage. If the row is not completely fixed then some of the free variables can be set to arbitrary values to achieve this.

5.1.6 Results

For computing the Gröbner basis of the Keccak equation system, Sage with the PolyBori library is used[44][45]. The Keccak challenges for Keccak$[r = 240, c = 160]$ were chosen as a target for finding the preimage.

Table 5.1: Preimage for Keccak$[r = 240, c = 160]$ challenges.

#Rounds	m	$H(m)$
1	`f03c0243e2f090042cfe`	`d9d6d3c84d1ac1d75f96`

Finding the Gröbner basis for 1 round only takes a few seconds. The problem for 1 round seems particular easy as one can ignore χ which makes the problem linear. For computing the Gröbner basis it did not make any difference whether this optimisation was used or not. For 2 rounds no solution could be found in a reasonable amount of time. The higher number of variables and increase in degree of non-linearity makes the problem more difficult for a higher number of rounds.

Possible approaches to improve this attack might be to use a different representation of Keccak, find further optimisations or adapt the Gröbner bases algorithms for this specific problem.

5.2 Differential Cryptanalysis

Differential cryptanalysis was first published by Biham and Shamir to analyse the block cipher DES [46]. The attack scenario is a chosen plaintext attack, which means the attacker can choose arbitrary messages and gets the encryption of it. Differential cryptanalysis observes how the difference between a pair of inputs affects the resulting output difference. While it was originally devised to analyse block ciphers the technique is also used for stream ciphers and hash functions.

Resistance to differential cryptanalysis is an important design criteria. Designers of cryptographic primitives have to argue or ideally proof that their algorithm is secure against differential cryptanalysis.

Differential cryptanalysis gives a natural approach to find collisions for hash functions. A message pair (M, M') with difference $\Delta in = M \oplus M' \neq 0$ which results in an output difference $h(M) \oplus h(M') = 0$ equals a collision as can be seen in Figure 5.3.

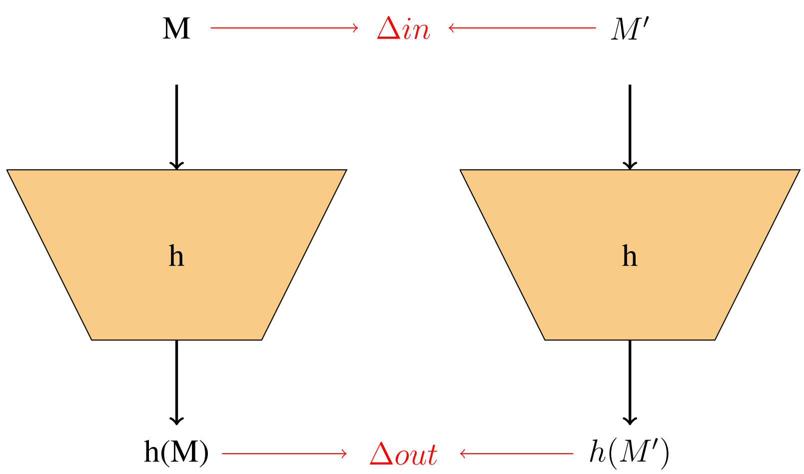


Figure 5.3: The relation between the input and output difference of two messages M and M' is used for the analysis.

5.2.1 Preliminaries

First some frequently used terminology is defined.

Definition 12. The **XOR difference** of two n-bit vectors a and a' is defined by

$$\Delta a = \Delta(a, a') = a \oplus a' \tag{5.13}$$

A cryptographic primitive is typically composed of multiple rounds. Therefore it is of interest how differences behave with respect to these functions.

Definition 13. A **differential** for a round-function f is denoted by

$$\Delta in \xrightarrow{f} \Delta out \tag{5.14}$$

Definition 14. A **differential characteristic** is a sequence of differentials of the following form

$$\Delta a_0 \xrightarrow{f_0} \Delta a_1 \xrightarrow{f_1} \Delta a_2 \ldots \xrightarrow{f_n} \Delta a_n \tag{5.15}$$

Linear Functions

Definition 15. The difference propagation for a linear function L, with respect to the difference operator, is deterministic. Given a pair of values M, M' and the difference $\Delta M = M \oplus M'$ the following equations holds

$$L(M) \oplus L(M') = L(M \oplus M') = L(\Delta M) \tag{5.16}$$

When the input difference of a linear function is known the output difference is also determined.

Non-Linear Functions

For non-linear functions the transition from a given input difference Δin to a given output difference Δout is probabilistic. A difference distribution table (DDT) is used for the analysis of non-linear functions. The DDT enumerates the number of solutions for $(\Delta in, \Delta out)$

$$\exists a \mid \text{ SBOX}(a) \oplus \text{SBOX}(a \oplus \Delta in) = \Delta out \tag{5.17}$$

The DDT shows which input/output pairs $(\Delta in, \Delta out)$ are possible and how often they occur. Entries with zero occurrences are called impossible differentials. For computing the DDT of a non-linear function Algorithm 4 is used which enumerates all the valid pairs.

The following example illustrates this important property by using the s-box used in Keccaks χ function.

Example 4. Given the s-box in the non-linear function χ and the corresponding DDT (see Appendix A.1). If for instance the difference at the input $\Delta in = 0x08$

Algorithm 4 Constructing a DDT for a k-bit function f

Precondition: DDT$[k][k]$
  for $a = 0 \dots k-1$ **do**
    for $b = 0 \dots k-1$ **do**
      DDT$[a \oplus b][f(a) \oplus f(b)]$++

then the set of possible output differences is $\Delta out = \{0x08, 0x09, 0x18, 0x19\}$ where each one is equally likely with a probability of 0.25. Depending on the values of the message pair the output difference will be one of these four choices as can be seen from Table 5.2.

Table 5.2: Message pairs and their corresponding input and output differences..

Input (x, y)	Δin	Output (x, y)	Δout
(0x10, 0x18)	*0x08*	*(0x12, 0x1a)*	*0x08*
(0x11, 0x19)	*0x08*	*(0x15, 0x1d)*	*0x08*
(0x12, 0x1a)	*0x08*	*(0x18, 0x10)*	*0x08*
(0x13, 0x1b)	*0x08*	*(0x1b, 0x13)*	*0x08*
(0x18, 0x10)	*0x08*	*(0x1a, 0x12)*	*0x08*
(0x19, 0x11)	*0x08*	*(0x1d, 0x15)*	*0x08*
(0x1a, 0x12)	*0x08*	*(0x10, 0x18)*	*0x08*
(0x1b, 0x13)	*0x08*	*(0x13, 0x1b)*	*0x08*
(0x00, 0x08)	*0x08*	*(0x00, 0x09)*	*0x09*
$\vdots$	$\vdots$	$\vdots$	$\vdots$

5.2.2 Differential Properties of Keccak

This section gives an overview of the differential behaviour of the Keccak step functions. It is essential to understand how these functions manipulate differences, because the following attacks make use of their properties. The most important features for each function are given here and a more detailed discussion can be found in the original Keccak specification [9].

Differential Properties of ρ

This step translates the bits in every lane by a constant value. Therefore differences are also rotated by the same constant.

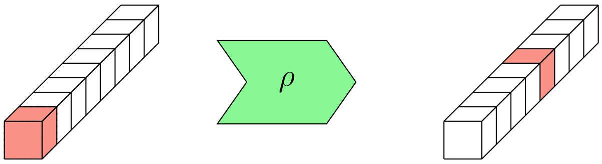


Figure 5.4: ρ shifts differences on the lanes.

Differential Properties of π

This step transposes the lanes, hence the differences in the lanes are also transposed to a new position.

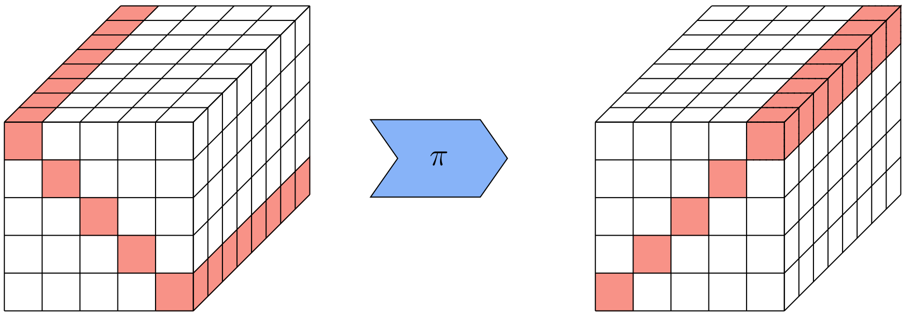


Figure 5.5: π transposes the lanes.

Differential Properties of θ

This step adds the bitwise sum of two columns to a bit. A single bit difference at the input of θ always affects two columns, hence 10 bits are changed. If there is an even number of differences in all columns (the parity is 0 for all columns) then this is called a column parity kernel[9]. This property is important for the analysis presented in Section 5.3 and will be examined in detail.

Differential Properties of ι

This step adds the same constant to both messages therefore it has no influence on the differences. For differential cryptanalysis this function is of no further interest and is omitted.

Differential Properties of χ

This step is composed of 5-bit s-boxes and is the only non-linear step in Keccak. Using Algorithm 4 the DDT for the 5-bit s-box is computed which can be found in

Appendix A.1. From the DDT it follows that the maximum differential probability (MDP) is 2^{-2}.

5.2.3 Searching for Complex Differential Paths

De Cannière and Rechberger presented a method to search automatically for differential characteristics for SHA-1 [47]. This method is a generalization of the approach by Wang et al. [4] and allows all possible conditions on the values of pairs of bits for the analysis. The notation for generalized conditions can be found in Table 5.3.

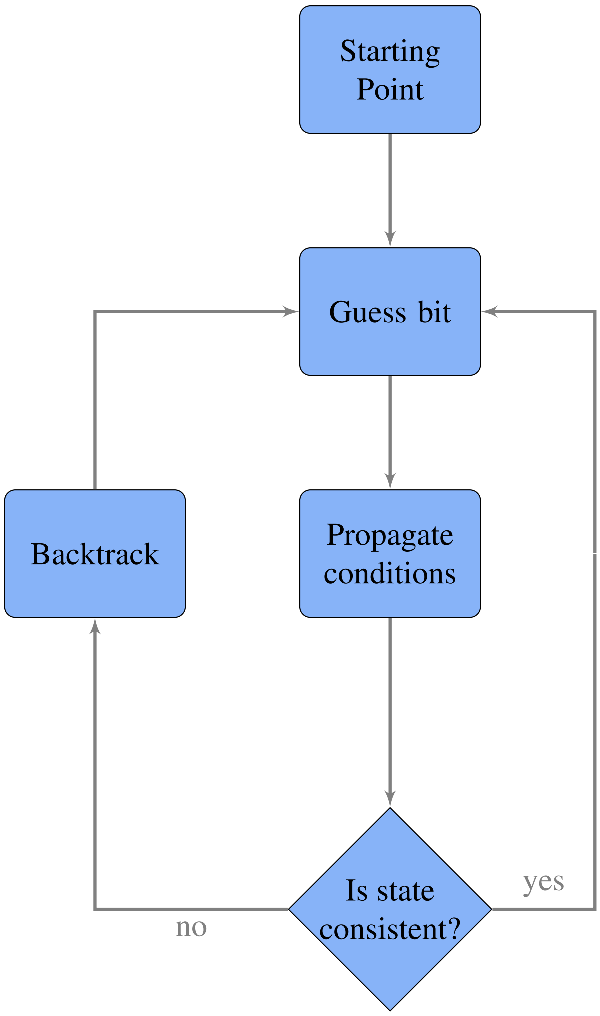


Figure 5.6: An outline showing the basic steps of the search strategy used to find differential characteristics.

The characteristics are constructed iteratively by adding more conditions on the state, propagating this conditions and check for consistency (see Figure 5.6). The idea for this search strategies originates from SAT solvers.

Table 5.3: This table shows the 16 possible combination of pairs of bits and the corresponding symbol used for notation.

(X_i, X'_i)	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
?	✓	✓	✓	✓
-	✓			✓
x		✓	✓	
0	✓			
u		✓		
n			✓	
1				✓
#				
3	✓	✓		
5	✓		✓	
7	✓	✓	✓	
A		✓		✓
B	✓	✓		✓
C			✓	✓
D	✓		✓	✓
E		✓	✓	✓

The cryptography research group at the IAIK developed a tool for cryptanalysis of hash function which is based on the concept of generalized conditions. It implements the functionality to propagate bit-conditions, check for consistency and backtracking and was used to analyse several hash functions including SHA-2[48][49][50].

Finding differential paths for Keccak

Constructing differential characteristics manually seems to be hard for Keccak due to the size of the state and the properties of the step functions. As part of this thesis the previously mentioned tool was extended to support the Keccak hash function and different search strategies have been evaluated to possibly find collisions for reduced round versions.

The state of Keccak is large compared to SHA-2. A single 32-bit word is updated in every of the 64 rounds. For Keccak with 64-bit lanesize the state for one of the 24 rounds is already 1600 bits large. This does not necessarily imply that also the search space grows by a proportional factor because conditions might propagate faster or contradictions might be detected earlier in the search process. Another concern is that there is no message input between the rounds of Keccak which could be used to create local collisions. These collisions could be used to cancel out differences at an intermediate output to reduce the overall complexity of a characteristic.

Table 5.4: Starting point for a 2-round collision search. Some bits of the message are fixed due to padding and the specification of the sponge construction. The only other restriction is that the output bits for the hash contain no difference.

Name	State				
	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
A[0]	????????????????	????????????????	????????????????	????????????????	11??????????????
	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
B[1]	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
A[1]	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
B[2]	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
A[2]	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????
	????????????????	????????????????	????????????????	????????????????	????????????????

In the following sections the search strategy for Keccak will be explained in detail. For the purpose of demonstration a lanesize of 16 bits is used but the same strategy can also be used for 32-bit and 64-bit lanesize.

Starting Point

The starting point for a collision search for Keccak$[240, 160]$ can be found in Table 5.4. Each lane is mapped to a 16-bit word and the words are ordered in a 5×5 matrix. The message block is padded and the output bits contributing to the hash value are set to "–" which equals no difference.

The naming convention (see Figure 5.7) for the states in Table 5.4 is the following:

- A is the input to the linear layer.
- B is the input to the non-linear layer.

Search Strategy

The search strategy for Keccak is based on the approach used for SHA-2 in [49]. The idea is to combine the search for a differential characteristic and a corresponding message pair in the search process. This two processes can further be split up in three parts:

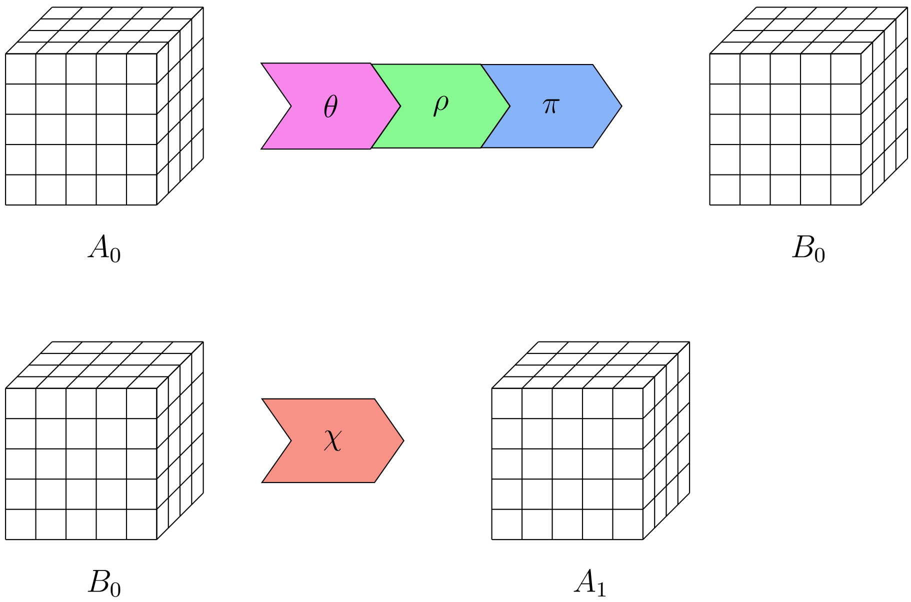


Figure 5.7: Notation used for the states.

- **Decision:** Choose a bit position for which a new condition is set.
- **Deduction:** Propagate conditions and check if the state is consistent.
- **Backtracking:** If the state is inconsistent then revert the previous choice for this bit and set a different condition. If all choices fail then it is necessary to jump back to a previous state to resolve this conflict.

The algorithm processes the state from the starting point and restricts bit conditions successively until only “1”, “0”, “n” and “u” conditions remain and the message

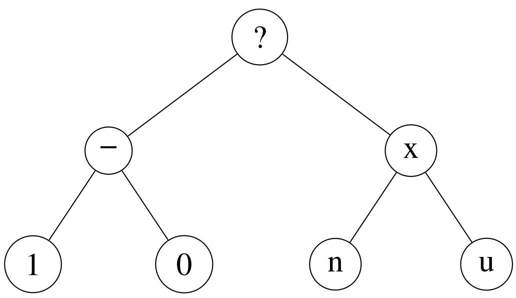


Figure 5.8: This tree shows how the search refines the conditions from free pairs of bits (’?’) to pairs of bits with a difference (’x’) and pairs of bits that are equal (’-’).

pair is fully determined. This can be seen from Table 5.3 as there is only one option for these conditions.

Finding the differential characteristic

The starting point in Table 5.4 contains no differences. Therefore, a random bit is set to "x" at the beginning of the search. This is essential for the following algorithm as only a trivial solution, where both messages are the same would be found elsewise.

The decision step for finding a differential characteristic chooses a random free bit ("?" condition) and tries to set it to equal ("-" condition) or a random difference ("x" condition) and set it to "n" or "u". If a contradiction is found after propagating then the other possible choice for this bit is used.

The reason behind choosing "-" first for a "?" condition is to have less differences in the resulting path which generally leads to a higher probability. Algorithm 5 outlines this process in detail and the result of this computation can be found in Table 5.5.

Algorithm 5 Keccak Search - Characteristic

Precondition: Starting point S

```
Choose a random '?' in S and set to 'x'
repeat
    U ← the set of all '?' and 'x'
    B ← a random bit in U
    if B == '?' then                                   Decision
        B ← '–'
    else if B == 'x' then
        B ← randomly 'n' or 'u'
    Propagate conditions for the new state
    Check the new state for contradictions             Deduction
    if found contradiction then
        if B == '–' then
            B ← 'x'
        else if B == 'n' then
            B ← 'u'
        else if B == 'u' then                          Backtracking
            B ← 'n'
        if contradiction still exists then
            Jump back until bit is resolved
until U empty
```

Finding a message pair

The process of finding a message pair is very similar. First a random "-" condition is chosen and set to either "1" or "0" and the conditions are propagated. If a contradic-

Table 5.5: A differential characteristic found with the iterative approach for 2 rounds of Keccak[240, 160].

Name	State				
	-----u-n-n-un-un	u----uu--------n	n---n--n-----u-u	u----nu-nn----n-	n-n-uun---n-uu--
	--u--u----n----n	n--u-uu---------	uu-----u-------n	-nu-uuuu--------	u-u----n-u-u--n-
A[0]	---n-u---n-n--u-	-----u--n--u-u--	---u---uu--n----	--u---nuu-------	11n---u----n-n--
	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	-----------u--nn	--n----u----nu--	--u-un--u---uu--	nun--------nu-u-	n-u-n--------u--
	--u-u----n--u---	-----n-----n----	u----------u--n-	--u-----u--un---	---uu---------n-
A[1]	-----------u--uu	-u---n--u--n---n	u--n--nu-n-----u	------n-n-n---n-	-u-u-n-u-u-u----
	n-nn-----unn---u	-----u-----u----	-u-n--n-----u-uu	nuu--------nu-u-	-uun--u----u--un
	-nu----u------n-	--u---n---un-unn	u--n--uu-n-----u	uu--u----n--nuuu	---nu---------u-
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
A[2]	----nnn--unuun-u	nnun-n--u---nnnn	nnun-n-uunn---nu	u--nnn-u--n--uu-	n--n-u--uununn-n
	u-nn------n-----	un------u-nu--n-	-un--u-nu-n--nn-	-nn--u--u---nn-n	-un--u----unn-n-
	--u--nn-nn-n--nu	-n---u-u--n-----	u-uu-n-nn-uun-n-	u---nu-----n--n-	nnuuu-nun-n-unuu

tion occurs the other possible choice is selected or if both choices fail the algorithm needs to jump back to a previous state to resolve the conflict. The whole process is outlined in Algorithm 6 and the corresponding result can be found in Table 5.6.

Algorithm 6 Keccak Search - Message

Precondition: State containing only '-', 'n' and 'u'

```
repeat
    U ← the set of all '-'
    B ← a random bit in U                      Decision
    B ← randomly '1' or '0'
    Propagate conditions for the new state
    Check the new state for contradictions     Deduction
    if found contradiction then
        if B == '1' then
            B ← '0'
        else if B == '0' then
            B ← '1'                            Backtracking
        if contradiction still exists then
            Jump back until bit is resolved
until U not empty
```

5.2.4 Collisions for 4-round Keccak

The previous approach failed to find any differential characteristics and corresponding message pairs for more than 2 rounds. It is possible to extend the attack to 4 rounds by using the approach by Dinur, Dunkelman and Shamir presented in [35]. The idea is to use a 2-round path with a high probability and combine it with a 2-round path which connects with the starting point (see Figure 5.9).

Table 5.6: A 2 round collision for Keccak$[240, 160]$.

Name	State				
	11011u0n1n1un1un	u1100uu11100100n	n111n10n11100u1u	u0111nu0nn0001n1	n1n0uun101n0uu01
	11u00u1011n0110n	n11u0uu111000001	uu01111u1000110n	0nu1uuuu00100011	u0u0100n1u0u11n0
A[0]	101n0u001n1n10u0	10101u01n10u1u10	101u001uu10n0111	01u111nuu0000011	11n110u0111n1n10
	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	11011011000u10nn	10n1110u1000nu00	11u0un11u111uu11	nun00011011nu1u0	n0u1n10111000u01
	11u1u1111n11u100	11001n01100n0110	u0000011111u11n1	01u10001u01un101	100uu001100000n1
A[1]	10000010101u00uu	1u100n00u10n010n	u01n00nu0n10011u	100001n0n1n001n1	0u1u0n1u0u0u0001
	n0nn01000unn001u	00010u11100u1110	0u0n00n00110u0uu	nuu00010000nu1u0	0uun10u1111u11un
	1nu0110u001111n1	10u101n000un1unn	u11n00uu0n10111u	uu10u0101n01nuuu	101nu110101000u0
	0100000101101011	1100000010000111	1000111001001010	1001001110000111	1111001110011111
	1111000101001101	1001110001001001	1011000010011101	1111100101010111	1011010100001000
A[2]	1100nnn00unuun1u	nnun0n11u011nnnn	nnun1n0uunn001nu	u01nnn1u01n00uu1	n10n0u11uununn1n
	u0nn110010n01001	un110000u1nu11n0	1un11u0nu1n11nn1	0nn10u01u000nn0n	0un10u0111unn0n0
	00u00nn0nn0n10nu	1n100u1u01n10111	u1uu1n0nn0uun1n1	u000nu00010n00n1	nnuuu1nun0n0unuu

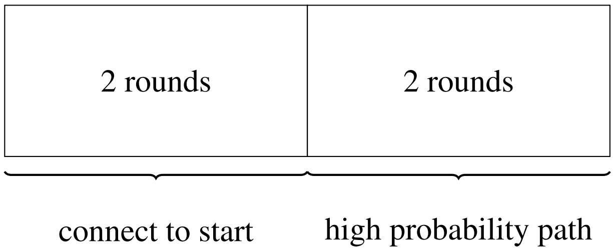


Figure 5.9: Outline of the 4-round attack.

The high probability paths are constructed using the column-parity property which is discussed in Section 5.3. The connection with the starting point is done with the iterative approach presented in the previous section. This approach only works if there are enough free bits in the message. Hence, collisions could only be constructed for Keccak with 64-bit lanesize.

In Table 5.7 the differential characteristic for Keccak$[1088, 512]$ which is used to find a 4-round collision is shown. The corresponding message pair can be found in Appendix A.4. The search for this specific message pair took 78 seconds on a standard PC[1].

[1] 1,86 GHz (SL9400) Intel Core 2 Duo, 4GB Ram, SSD

Table 5.7: A 4-round differential characteristic for Keccak with 64-bit lane-size. All words are converted to hexadecimal values and every non zero bit is a difference at this position. The dense structure in the first two rounds can be seen followed by a sparse column parity kernel over two rounds.

Name	State				
	b3-78891a9372f5-	8751b674255e59c1	b6558bf983a14d1-	4397b59dec18fec2	aeae97e4baa63e94
	2585a6a9-c7-bcf5	944c32a58-8fb985	f5acfd62-c2e-b66	cb661a9bacd18a9c	4547f7a74fd2d938
A[0]	-8dd1ad242369c6d	ca7faf6-6413b61e	8ed168---2716e45	2c2a951b7c9597ce	4-c6ab73d1adf3d1
	d-b59d454d-cbf-c	-4fced51b821cb63	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	26978af134cb835e	af224c4d78366789	c4dae35e2656f26b	357c4789af3-6af1	78d3526bc6a74c4d
	26978af134cb835e	af224c4d78366789	c4dae35e2656f26b	357c4789af3-6af1	78d3526bc6a74c4d
A[1]	26978af134cb835e	af224c4d78366789	c4dae35e2676f26b	357c4789af3-6af1	78d3526bc4a74c4d
	26978af134cb835e	af224c4d78366789	c4dae35e265ef26b	357c4789af3-4af1	78d3526bc6a74c4d
	26978af134cb835e	af226c4d78366789	c4dae35e2656f26b	35fc4789af3-6af1	78d3526bc6a74c4d
	----------------	----------------	---------------1	-------4--------	----------------
	----------------	----------------	----------------	----------------	----------------
A[2]	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	-------4--------	----8-----------
	----------------	----------------	---------------1	----------------	----8-----------
	----------------	----------------	----------------	--8-------------	2---------------
	4---------------	----------------	----------------	----------------	2---------------
A[3]	----------------	----------------	----------------	--8-------------	----------------
	----------------	----------------	----------------	----------------	----------------
	4---------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	-----------a----	-----------2----	----------------	-----------8----	-----------8----
A[4]	-----------1----	----------------	----------------	----------------	-----------1----
	-------4-1------	-------4--------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------

5.3 Combining Kernel Paths

This section presents a method to find collisions for Keccak by combining differential paths. The idea is to use multiple paths and combine them such that differences cancel out and lead to a collision. This might allow to extend the attack on more rounds, if such a combination exists. In this case the differential paths should have a high probability in order to keep the overall complexity of the attack low.

The next section presents an algorithm to find all the high probability paths for two rounds in detail followed by two techniques to find combinations leading to collisions.

5.3.1 Kernel

It is important to first argue on how to construct a differential path for Keccak, such that the resulting probability is high. The linear steps are all deterministic and only the input to χ contributes to the resulting probability of the differential path. Therefore a low Hamming weight in this inputs will result in a high probability for the differential path. The function responsible for most of the diffusion in Keccak is θ. However, θ has properties which can be used to mitigate this effect (see Section 5.2.2).

As defined by the authors of Keccak a state is in the column parity kernel, if the parity of all columns is 0. In this case θ becomes the identity function. This can be utilised to create high probability differential paths, since π and ρ provide no diffusion and χ provides only slow diffusion. There is a restriction to this by the interaction of the linear functions π, ρ and θ. This functions guarantee that no low weight kernel over three consecutive rounds exists [9].

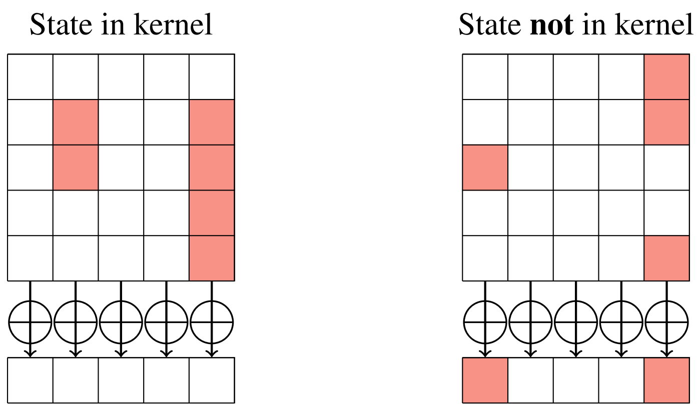


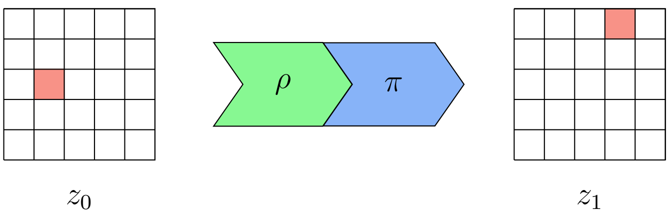
Figure 5.10: States containing differences with the corresponding column parity.

The second property of Keccak which is applied to keep the number of active bits low is that a single bit difference at the input of χ leads to a single bit difference at the output of χ with probability of 2^{-2}. This can be seen from the DDT which can be found in Appendix A.1.

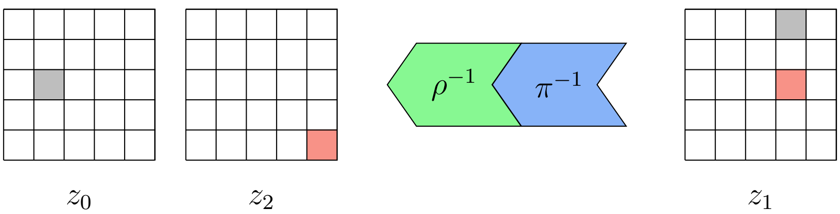
5.3.2 Finding Kernels

There is a only a limited number of kernel paths up to a given Hamming weight. The following procedure constructs all these paths [32]:

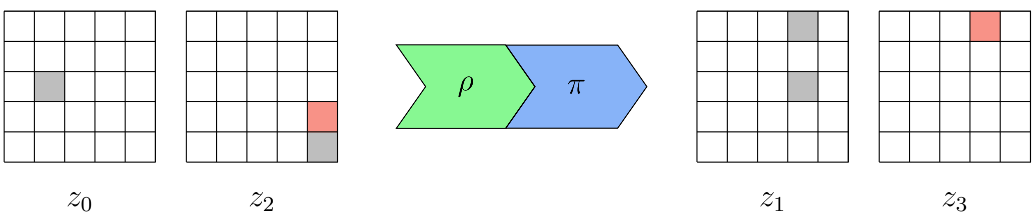
1. First select an arbitrary bit as a starting point in state S_0 and compute $\pi(\rho(S_0)) = S_1$. These functions move this bit to a different position. Note that the position on the z-coordinate is also different which is denoted as z_o, z_1, z_2 and z_3.



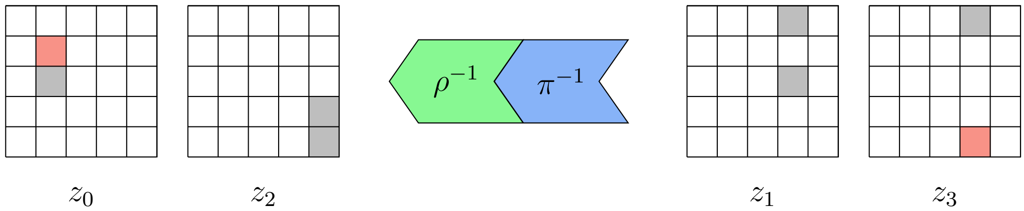
2. Next add a second difference in the same column to S_1 and compute $\rho^{-1}(\pi^{-1}(S_1))$.



3. Add another difference in the same column in S_0 and compute $S_1 = \pi(\rho(S_0))$.



4. Put a difference in the same column and compute $S_0 = \rho^{-1}(\pi^{-1}(S_1))$.



5. Check if the difference added at last is moved to z_0. If so, a column parity kernel for 2-rounds has been found with a Hamming weight of 8. Elsewise the procedure can be continued to find kernels with a higher Hamming weight.

For a kernel containing n differences (per round) the complexity of this procedure is given by

$$\mathcal{O}(25 \cdot 4^{n-1}) \tag{5.18}$$

Table 5.8 lists the results for Keccak$[240, 160]$ and Keccak$[1344, 256]$. For Keccak$[1344, 256]$ all kernels have at least a Hamming weight of 12 while for Keccak$[240, 160]$ there are 64 kernels with a Hamming weight of 8. An example for a kernel can be found in Table 5.9.

Table 5.8: Results of the kernel search.

Keccak$[r, c]$	#Kernels	#Collision	#1-bit χ input over two rounds
Keccak$[240, 160]$	672	16	608
Keccak$[1344, 256]$	512	64	448

5.3.3 Combining Kernels

Kernel paths have a high probability but are limited to two rounds. The idea is to create a differential path by combining multiple kernel paths such that a collision occurs for more than two rounds.

For an attack to be feasible the probability of the resulting differential path must be greater than $2^{-n/2}$ where n is the size of the hash-value. The kernels found in the previous section utilise the property that χ is the identity function for 1-bit difference input with probability 2^{-2}. To get a low number of active bits this property is preferable kept over the 2 rounds. This results in a probability of 2^{-24} for a 2-round kernel. Table 5.8 highlights the number of kernels for which this property holds.

Table 5.9: A 2-round kernel for Keccak$[160, 240]$ leading to a collision. The bits contributing to the hash value are marked gray. The probability that $A[1] = \chi(B[0])$ is 2^{-12} (see Section 5.2.2)..

Name	State				
	----------------	----------------	-------x--------	-----x----------	----------------
	----------------	----------------	----------------	----------------	----------------
A[0]	---------------x	----------------	-------x--------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	---------------x	----------------	----------------	-----x----------	----------------
	----------------	------------1---	------------x---	------------0---	----------------
	---------x------	---------0--1---	------------x---	------------0---	---------1------
B[0]	-------------0--	----------------	----------------	-------------1--	-------------x--
	-------------0--	----------------	----------------	-------------1--	-------------x--
	---------x------	---------0------	----------------	----------------	---------1------
	----------------	------------1---	------------x---	----------------	----------------
	---------x------	------------1---	------------x---	----------------	---------1------
A[1]	----------------	----------------	----------------	-------------1--	-------------x--
	----------------	----------------	----------------	-------------1--	-------------x--
	---------x------	----------------	----------------	----------------	---------1------
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
B[1]	----------------	------x---------	----------------	-----x----------	-------x--------
	----------------	-----x----------	----------------	----------------	----------------
	--------------x-	----------------	------x---------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
A[2]	------?---------	-----?x---------	-----?-?--------	-----x-?--------	------?x--------
	-----?----------	-----x----------	----------------	----------------	-----?----------
	------?-------x-	------?---------	------x---------	--------------?-	--------------?-

The question which arises now is how this paths behave if we propagate the conditions an additional round forward. This can be done with the same tool used to find the differential paths in Section 5.2. The results of such a propagation of a 2-round kernel can be seen in Table 5.10.

Condition for collision

The first n bits of $A[3]$ are the hash output, where n is the length of the hash. For a collision these bits are not allowed to have any differences. An important observation is that $A[3]$ contains no differences in the first n bits if and only if $B[2]$

Table 5.10: Propagation for 3 rounds of the path in Table 5.9. The bits contributing to the hash value are marked gray. $A[2] = \chi(B[1])$ holds with a probability of 2^{-12} (see Section 5.2.2).

Name	State				
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
A[2]	------1---------	------x---------	-----1----------	-----x-1--------	-------x--------
	-----1----------	-----x----------	----------------	----------------	----------------
	--------------x-	------1---------	------x---------	----------------	--------------1-
	----xx-x--------	--x------x------	---------xxx----	----------------	-------x-------x
	----------------	-x-------x------	-xx-x-----------	-x--------------	-------xx-------
B[2]	----x--------x--	x-------------xx	------------x---	-----x-------x--	--xx-x------x---
	--x-------x-----	xx-x------------	----x------xx---	-----xxx--------	----------------
	------xxx-------	----------------	x-----x-------x-	-----------xx-x-	---x--------x---
	--?-xx-x-???----	--x------???----	-------?-xxx---?	----??-?-------?	--?-??-?-?-----x
	-??-?----?------	-??-?----x------	-?x-x--??-------	-x-----??-------	-?-----xx?------
A[3]	?---x-------?x??	x----?------??xx	--??-?------??--	--????------??--	?-xx?x------x???
	??x??-----x??---	xx-x????---??---	----x???---xx---	--?--xxx--?-----	????------?-----
	?-----?xx-----?-	?-----?----??-?-	x--?--x----??-?-	---?--???--x?-x-	---x--???---x---

contains no difference in the first n bits. This fact is due the structure of χ.

The first n bits of $B[2]$ are extracted for all kernels to obtain the hash output vectors h_i. The target for combining the kernels is to find a combination of h_i such that all x conditions cancel out and the resulting state has no differences in the hash output.

Proposition 1. The kernel paths behave linear (with a certain probability), hence it can be derived that a combination leading to a collision will give a colliding input too. Given the hash output vector $h_0 = f(m_0)$, $h_1 = f(m_1)$ and assuming this combination gives a solution $h_0 \oplus h_1 = 0$. Due $f(x)$ being a linear function this implies $f(m_0 + m_1) = 0$. Therefore the input $m_0 + m_1$ will lead to a collision.

There are some restrictions and requirements on the kernels and the resulting differential path obtained by combining them. First the quantity of kernels which can be used simultaneously is limited. A single two round kernel has a probability of 2^{-24}, therefore combining more than five kernels would already result in an attack worse than brute-force for 256-bit hash output.

Apart from the problem of handling the number of kernels and combinations the resulting differential path might be inconsistent. Consequently every possible colliding path obtained has to be validated.

Consistency Checks

First of all it is important that the kernels used do not share differences at the same bit-positions in the first two rounds or the resulting differential path will not lead to a collision. If the kernels share a single difference the parity of the state changes and θ does not act as the identity function. If the differences are all distinct then the new differential path can be obtained by adding up the kernel paths.

After this process the resulting differential path can still be invalid due to the properties of χ. The kernel paths are all fixed to a 1-bit difference input to χ, but after combining them this property might be violated. For every 5-bit s-box it is necessary to check if $DDT[\Delta in][\Delta out] \neq 0$ (see Table 5.11). The resulting path might have a lower probability in this case.

Table 5.11: Checking for valid input/output pair for χ. The input/output to a single 5-bit s-box is marked blue.

Name	State				

B[1]	`----------------`
	`----------------`
	`------10--------`
	`-----1----------`
	`--------------x-`
		`----------------`
		`----------------`
		`------x---------`
		`-----x----------`
		`------1-------0-`
			`----------------`
			`----------------`
			`-----10---------`
			`-----0----------`
			`------x---------`
				`----------------`
				`----------------`
				`-----x-1--------`
				`----------------`
				`------0---------`
					`----------------`
					`----------------`
					`-----0-x--------`
					`----------------`
					`--------------1-`

A[2]	`----------------`
	`----------------`
	`------1---------`
	`-----1----------`
	`--------------x-`
		`----------------`
		`----------------`
		`------x---------`
		`-----x----------`
		`------1---------`
			`----------------`
			`----------------`
			`-----1----------`
			`----------------`
			`------x---------`
				`----------------`
				`----------------`
				`-----x-1--------`
				`----------------`
				`----------------`
					`----------------`
					`----------------`
					`-------x--------`
					`----------------`
					`--------------1-`

Linear Algebra for Combing Kernels

The first method to find these combinations of kernels is based on the problem of finding the null space[2] of a matrix. The null space of a $m \times n$ matrix A is defined by

$$\texttt{Null}(A) = \{x \in \mathbb{F}^n : Ax = 0\} \tag{5.19}$$

This set can be efficiently computed using Gaussian elimination. In fact by constructing an appropriate matrix from the hash output vectors the null space gives **all** the existing solutions for combining kernels to a collision. The matrix is defined by

$$A = \begin{pmatrix} h_0^T & h_1^T & \cdots & h_j^T \end{pmatrix} \tag{5.20}$$

where h_i are the hash output vectors mapped to $\mathbb{F}_2$. All 'x' conditions are mapped to 1, whereas all '-' conditions are mapped to 0. For example given the following five vectors

$$h_0 = (\texttt{x--x}) \tag{5.21}$$
$$h_1 = (\texttt{-x--}) \tag{5.22}$$
$$h_2 = (\texttt{-x-x}) \tag{5.23}$$
$$h_3 = (\texttt{---x}) \tag{5.24}$$
$$h_4 = (\texttt{x---}) \tag{5.25}$$

[2] Also referred to as the kernel of a matrix, but the term null space is favourable in this case to avoid any misconceptions.

The corresponding matrix obtained by using definition 5.20 is

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{5.26}$$

and computing the null space gives

$$\mathtt{Null}(A) = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \tag{5.27}$$

This matrix represents a basis for all the solutions to $Ax = 0$. The rows in the matrix give the solutions $h_0 \oplus h_3 \oplus h_4 = 0$, $h_1 \oplus h_2 \oplus h_3 = 0$ and by linear combination of them a third solution $h_0 \oplus h_1 \oplus h_2 \oplus h_4 = 0$ is found.

The same procedure is now applied to the hash output vectors obtained from the kernel paths for both variants of Keccak.

- For Keccak$[1344, 256]$ the number of kernels is 448. Therefore A is of dimensions 256×448. The null space of A is of dimensions 192×448 and provides all solutions, putting aside the consistency checks. There are a total of $2^{192} - 1$ solutions because all linear combinations are a solution too.

  The null space of A contains 38 rows with a Hamming weight of 3 which are good candidates for a solution. For each row the corresponding kernels are combined and the consistency checks are applied. For this combinations the difference all overlap, hence no solution exists for combining these kernels.

- For Keccak$[240, 160]$ the matrix A is of size 160×608 and the dimensions of $\mathtt{null}(A)$ are 448×608. The null space has 41 rows with Hamming weight of 3 and 22 with Hamming weight of 4. For all this rows the kernel combinations are constructed and the consistency checks are applied. Again no solution exists without differences overlapping in the first two rounds.

For both Keccak variants only trivial solutions are found which would result in an input state with zero differences. Consequently no colliding input can be constructed with the previous methods. This confirms that no sparse characteristic constructed from multiple kernel paths will lead to a collision for 3 rounds of Keccak.

It might still be possible to find additional solutions by using the linear combinations of $\mathtt{Null}(A)$. The problem here is to find combinations with low Hamming

Table 5.12: Below are the input difference for 3 different kernels. The combination of them leads to a trivial collision as every difference occurs exactly two times

Name	State				
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	---------------x	------------x---
A[0]	----------------	x---------------	----------------	---------------x	----------------
	----------------	----------------	----------------	----------------	----------------
	----------------	x---------------	----------------	----------------	------------x---
	----------------	----------------	----------------	----------------	----------------
	------------x---	----------------	----------------	---------------x	----------------
A[0]	----------------	x---------------	----------------	---------------x	----------------
	------------x---	x---------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	------------x---	----------------	----------------	----------------	------------x---
A[0]	----------------	----------------	----------------	----------------	----------------
	------------x---	x---------------	----------------	----------------	----------------
	----------------	x---------------	----------------	----------------	------------x---

weight which is known to be a hard problem. A possible approach might be to construct a linear code from $\texttt{Null}(A)$. This allows to search for code words with a low Hamming weight for instance with the probabilistic algorithm by Canteaut and Chabaud [51].

General Algorithm

This algorithm checks for every combination of k vectors if the combination leads to a collision. The requirement to obtain a vector with zero differences are that for all bit positions:

1. the number of 'x' conditions is even
2. there is at least one '?' condition

The main advantage of this algorithm is that it can be applied if the states contains free bits (bits with '?' conditions). This can be useful for instance if the input to χ in the second round is not fixed. In this case there will be undetermined bits in the resulting hash vector. This bits will flip to '-' or 'x' with a specific probability depending on the actual message pair. As a result this might lead to additional solutions.

Listing 5.2: An algorithm to find a combination of kernels leading to a collision.

```
def solve_combinations(hashoutput_vectors, k, wordsize):
   C = Combinations(hashoutput_vectors, k)
   result = []
   it = iter(C)
   while True:
```

```
        try:
            combination = it.next()
            isSolution = true
            for lane_index in range(0, wordsize):
                cond_diff = 0
                cond_free = 0
                for row_index in combination:
                    if(row_index[lane_index] == "x"):
                        cond_diff += 1
                    if(row_index[lane_index] == "?"):
                        cond_free += 1
                if((cond_diff % 2) != 0):
                      if(cond_free == 0):
                          isSolution = false
            if(isSolution):
                result.append(combination)
        except StopIteration:
            print "All combinations tested"
            return result
```

The main drawback with this kind of algorithm is that it needs to loop over all possible combinations which makes it infeasible to check combinations for larger values of k. Fortunately these combinations are not of interest because they lead to denser paths with lower probability. The number of ways to pick k kernels out of n possibilities is given by the binomial coefficient

$$C(n,k) = \binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{5.28}$$

The total number of combinations that need to be tested for Keccak$[160, 240]$ and Keccak$[256, 1344]$ can be found in Table 5.14.

For Keccak$[1344, 256]$ this procedure finds 64 solutions for combinations of 4 kernels. For Keccak$[240, 160]$ this procedure finds 288 solutions of 3 kernels. However, this solutions are all trivial because all of the input difference cancel out.

Table 5.13: An example for combining three hash output vectors.

Vector	Conditions		
h_0	`x----x----x`	`x---------x`	`?-----?----`
h_1	`--x--x----x`	`?----x----?`	`?-----x----`
h_2	`----------x`	`x----x----?`	`?----------`
Result	`x-x-------x`	`-----------`	`-----------`

Table 5.14: Number of combinations tested.

Keccak[r, c]	#Kernels	k	$C(n,k)$
Keccak$[240, 160]$	608	3	$\approx 2^{25.2}$
Keccak$[1344, 256]$	448	4	$\approx 2^{30.6}$

Table 5.15: A 2-round kernel with 64-bit lanesize leading to a 384-bit collision.

Name	State				
	----------------	--4-------------	----------------	----------------	--------1-------
	----------------	----------------	----------------	----------------	----------------
A[0]	---------------1	--4-------------	----------2-----	----------------	--------1-------
	----------------	----------------	----------------	----------------	----------------
	---------------1	----------------	----------2-----	----------------	----------------
	----------------	----------------	---------------1	----------------	----------------
	----------------	----------------	---------------8	----------------	-----------4----
A[1]	--8-------------	----------------	----------------	----------------	-----------4----
	--8-------------	----------------	---------------1	----------------	----------------
	----------------	----------------	---------------8	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	----------------	------4---------	-4--------------	----------------	---------------1
A[2]	----------------	-------------2--	----------------	----------------	----------------
	----------------	----------------	----------------	------------8---	----------------
	4---------------	----------------	-2--------------	-------1--------	----------------

5.3.4 Kernels for larger output sizes

The method of combining kernels can be used to find new kernels which lead to collisions for a higher number of bits. The method presented in the previous section finds new paths which could enable attacks on Keccak variants with a higher output size.

- 384-bit: Combinations of 2 kernels (128 solutions)
- 448-bit: Combinations of 4 kernels (64 solutions)
- 470-bit: Combinations of 4/12/16 kernels (33/5/4 solutions)
- 500-bit: Combinations of 4/24/28 kernels (3/2/7 solutions)
- 502-bit: Combinations of 4/28 kernels (1/9 solutions)
- 512-bit: Combinations of 256 Kernels (1 solution, impossible differential)

The following paths all have a lanesize of 64 bits, therefore the conditions are encoded as hexadecimal values to reduce the size of the tables. In these tables differences are given as XOR values.

The paths found with this method are good candidates to use in the 4 round attack presented in Section 5.2. The capacity has to be increased in relation to the

Table 5.16: A 2-round kernel with 64-bit lanesize leading to 502-bit collision. The XOR-differences are given as hex values.

Name	State				
	----------------	1-------------2-	----------------	-----4----------	-------4--------
	--------------4-	----------------	------1--8------	-----4----------	----------------
A[0]	--------------4-	1-------------2-	------1--8------	----------------	-------4--------
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	------------8-4-	----------------	----------------
	--------------4-	----------------	-------------2--	----------------	----------------
A[1]	2-------------4-	-----4-2--------	----------------	----------------	----------------
	2---------------	-----4----------	------------8-4-	----------------	----------------
	----------------	-------2--------	-------------2--	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	----------------	----------------	-------------2-1	----------8-----	--------------4-
A[2]	----------------	------------8---	----------------	----------------	----------------
	----------------	-----4----------	--1--8----------	--------4-2-----	----------------
	------------2-1-	----------------	----------------	------4---------	-------8--------

Table 5.17: A 4-round characteristic leading to a collision for 384 bits with the capacity being reduced to 512 bits.

Name	State				
	caf81dbdb8a3a36-	9415151caf6e728-	74da685b3459b13f	-5d4352d2aa3a9db	1a9-2fbc859d1273
	631816cbb789ba6f	aae789a561f1a351	ffb26761e-d8cfd8	9a9e15bf4122488-	a5991dfc33b3afa7
A[0]	e8a4f3eaf78d85c6	39541c87edcd5fb7	179838c55f368ea9	ca1871ca71fde7aa	3475a962932863f1
	29c4ca71149df311	3b33411a3f8685b4	----------------	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	ea6b826bc4d7a9e3	82f13af135e2789a	f44c5c4d789aeabc	2b5eb35e26bc4b11	ef89b789af131-d6
	ea6b826bc4d7a9e3	-2f13af135e27c9a	f44d5c4d789aeab8	2b5eb35e26bc4b11	ef89b789af131-d6
A[1]	ea6b826bc4d7a9e3	-2f13af135e2789a	f44c5c4d789aeab8	3b5eb35e26bc4b11	ef89f789af131-d6
	ea6b826bc4d7a9e3	-2f13af135e2789a	f44c5c4d789aeab8	2b5eb35e26bc4b11	ef89b789af131-d6
	ea6b826bc4d7ade3	-2f13af135e2789a	f44c5c4d789aeab8	2b5eb35e26bc4b11	ef89b789af135-d6
	----------------	--4-------------	----------------	----------------	--------1-------
	----------------	----------------	----------------	----------------	----------------
A[2]	---------------1	--4-------------	----------2-----	----------------	--------1-------
	----------------	----------------	----------------	----------------	----------------
	---------------1	----------------	----------2-----	----------------	----------------
	----------------	----------------	---------------1	----------------	----------------
	----------------	----------------	---------------8	----------------	-----------4----
A[3]	--8-------------	----------------	----------------	----------------	-----------4----
	--8-------------	----------------	---------------1	----------------	----------------
	----------------	----------------	---------------8	----------------	----------------
	----------------	----------------	----------------	----------------	----------------
	----------------	------4---------	-4-------------1	----------------	---------------1
A[4]	-------------2--	-------------2--	----------------	----------------	----------------
	----------------	------------8---	------------8---	------------8---	----------------
	42--------------	-2-----1--------	-2--------------	4------1--------	----------------

outputsize to get the desired security level when using the sponge construction. This leads to less free message input which can be used to connect the high probability 2-round paths with the starting point.

No solution was found to connect the paths in Table 5.15 and Table 5.16 to the starting point. If the capacity is reduced to 512 bits a solution can be found for the 384-bit path. The complete 4-round characteristic can be found in Table 5.17 and the message pair in the Appendix A.5. Finding this message pair took 294 seconds.

The path in Table 5.16 can be used to find two round collisions for a capacity up to 672 and output size 502.

6 Conclusion

In the first chapters of this thesis, cryptographic hash functions and the generic attacks applicable to them have been discussed. The hash function Keccak was presented in detail and an overview of the current state of attacks was given. The use of algebraic attacks and how they can be applied on Keccak has been evaluated. The main part of this thesis focused on differential cryptanalysis and the attack strategy was presented in detail.

A new method to find 4-round collisions for Keccak has been presented by using a similar approach to the attack by Dinur et al. By connecting a 2-round column-parity kernel path to the starting point an attack on 4 rounds is possible. A tool-assisted method based on generalized conditions was used to find these connections to the starting point and the corresponding message pair. The method is practical and takes only a few minutes on recent hardware. Examples for this colliding message pairs have been shown.

Furthermore, this thesis presented a technique to find new differential characteristics by combing 2-round kernel paths. The analysis of the combination of these paths shows that there is no combination of low weight kernels leading to a collision for 3 rounds. However, new differential characteristics for larger output sizes can be found, which might allow new attacks on these variants of Keccak.

Future work will include optimisations for the tool-assisted method to improve the search process. Detecting contradictions earlier would reduce the time spend in dead branches of the search tree, which could improve the running time. A further improvement could be to use multiple message blocks. The attacks in this thesis are using only a single message block. Additional message blocks would give more freedom at the input and might enable new attacks at the cost of an increased search space.

The differential characteristics found by combining kernel paths could be used in the 4-round attack scenario for larger versions of Keccak. With the current implementation no solution was found in a reasonable amount of time as there is significant less free message input. The previously mentioned optimisations might enable such an attack.

A
Appendix

A.1 Notation for Keccak State

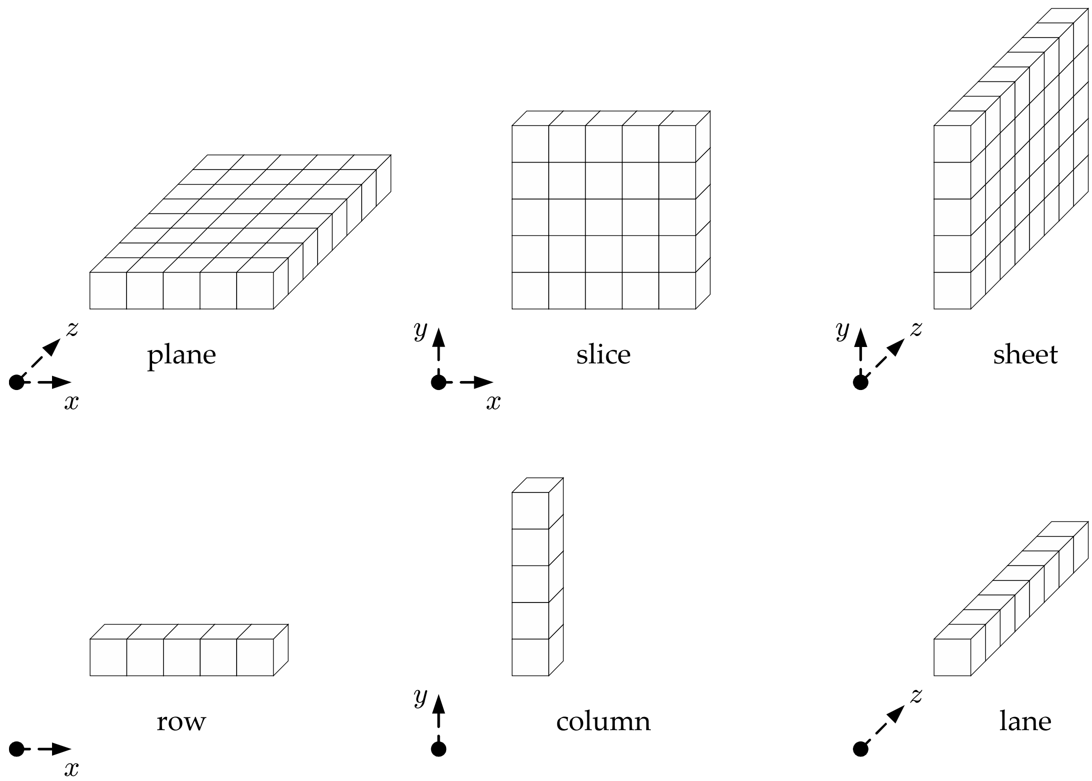


Figure A.1: Additional terms used for specific parts of the state[1].

[1]Image from `http://keccak.noekeon.org/`

A.2 Differential Distribution Table

		Δout																															
		00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
	00	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	01	0	8	0	8	0	8	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	02	0	0	8	0	0	0	8	0	0	0	8	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	03	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	04	0	0	0	0	8	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	8	0	0	0
	05	0	4	0	4	0	0	0	0	0	0	0	0	0	4	0	4	0	4	0	4	0	0	0	0	0	0	0	0	0	4	0	4
	06	0	0	4	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0	4	0
	07	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2
	08	0	0	0	0	0	0	0	0	8	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	8	0	0	0	0	0	0
	09	0	0	0	0	0	0	0	0	4	0	0	4	4	0	0	4	0	0	0	0	0	0	0	0	4	0	0	4	4	0	0	4
	0A	0	0	4	4	0	0	4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	0	0	4	4
	0B	0	4	4	0	0	4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	0	0	4	4	0
	0C	0	0	0	0	4	4	0	0	0	0	0	0	4	4	0	0	0	0	0	0	4	4	0	0	0	0	0	0	4	4	0	0
	0D	0	0	0	0	4	0	0	4	4	0	0	4	0	0	0	0	0	0	0	0	4	0	0	4	4	0	0	4	0	0	0	0
	0E	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2
Δin	0F	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2	0
	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	8	8	8	0	0	0	0	0	0	0	0	0	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4	4	4	4	4	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	0	0	0	0	4	4	4	4	0	0	0	0	4	4
	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	14	0	0	0	0	4	0	4	0	0	0	0	0	4	0	4	0	0	0	0	0	0	4	0	4	0	0	0	0	0	4	0	4
	15	0	4	0	4	0	0	0	0	0	0	0	0	0	4	0	4	4	0	4	0	0	0	0	0	0	0	0	0	4	0	4	0
	16	0	0	4	0	4	0	0	0	0	0	4	0	4	0	0	0	0	0	0	4	0	4	0	0	0	0	0	4	0	4	0	0
	17	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0
	18	0	0	0	0	0	0	0	0	4	4	4	4	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4	0	0	0	0
	19	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2
	1A	0	0	0	0	0	0	0	0	4	4	0	0	0	0	4	4	4	4	0	0	0	0	4	4	0	0	0	0	0	0	0	0
	1B	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	0	0	0	0	0	0	0	0
	1C	0	0	0	0	2	2	2	2	0	0	0	0	2	2	2	2	0	0	0	0	2	2	2	2	0	0	0	0	2	2	2	2
	1D	0	0	0	0	2	2	2	2	2	2	2	2	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	0	0	0	0
	1E	0	0	2	2	2	2	0	0	0	0	2	2	2	2	0	0	0	0	2	2	2	2	0	0	0	0	2	2	2	2	0	0
	1F	0	2	2	0	2	0	0	2	2	0	0	2	0	2	2	0	2	0	0	2	0	2	2	0	0	2	2	0	2	0	0	2

Table A.1: Differential distribution table for Keccak χ step. MSB order is used.

A.3 Colliding message pairs

Table A.2: A colliding message pair for 2 rounds of Keccak$[240, 160]$.

M_1:

```
D9 ED 61 C9 FD E2 3C C7 E3 61
C2 ED E1 C1 1E 8D 50 23 09 8E
B0 F8 A9 CA A2 57 5E 03 F8 FE
```

M_2:

```
DC B6 E7 C8 74 E7 BA 05 4D 4D
E6 CC 77 C1 DF 8C 3F 23 A8 DC
A4 AA AD 5E B3 C7 7D 83 DA EA
```

$H(M_1) = H(M_2)$:

```
78 42 97 D0 4A 8E 07 13 1F 62
4D D1 49 BC 9D 90 77 D9 28 95
```

Table A.3: A colliding message pair for 4 rounds of Keccak$[1440, 160]$.

M_1:

```
6D 3F 8F 98 19 8E CB 1A 61 BB 16 48 9D 91 4C 5B 43
CB 8F AC BD 61 5B 41 B1 11 C3 7F F2 6B E8 54 4A 87
7A 81 EE BA 17 B1 80 5D 72 0C AF 57 9D 30 86 A5 BE
9B ED DF 96 5E 78 72 AE 83 0B FE 2C 30 EA 6B 60 CF
E8 1E F0 C4 1F E8 14 05 37 57 B4 A5 76 53 66 62 7D
AF 54 E6 CA 96 55 2A 1A 77 E3 C4 99 25 D3 85 57 1C
DA 67 5D 4E D5 A0 C3 B3 33 48 41 6D 6E 47 C7 44 EC
70 33 88 64 9D 70 16 84 4F BE 3B C2 C7 61 35 C4 47
9B 5D C7 24 DD 80 0B 83 F3 48 33 43 66 4D 76 3C 08
A7 B7 59 AB 70 29 2F B2 08 02 11 62 C5 83 42 22 14
94 76 01 81 ED 89 8D 9A 38 F8
```

M_2:

```
C0 67 25 47 EA 89 71 7B 7B 56 9A 21 CF 97 BA 6F F8
94 09 D7 88 D9 9D 3B 2F 19 B4 FE 08 67 C3 79 F7 08
10 64 B8 98 66 65 36 B0 97 43 96 89 27 29 D8 53 B1
16 2C F7 51 70 55 97 22 21 37 E9 65 0D 39 AE E0 71
40 53 A4 01 6E A1 F2 FD 4D 4C 0B 3B 0C 2E B7 3E 0B
5D 2C 9A C3 62 1F 33 14 7D A7 50 9F 30 71 E0 0F 0C
A7 9B FA 96 88 66 1A 8E A5 21 B1 83 1B 66 55 DE C9
E1 5E 55 4B 69 47 80 88 30 1E 1B 21 E5 AB B7 00 25
AE 84 5A CE F1 D0 14 0B 06 AC 50 CD FC 11 68 FF B8
94 CA 90 CE 02 36 B1 A9 F6 FF 72 C3 A5 08 76 D9 21
AA 78 60 30 FB 49 E6 0B 2E C3
```

$H(M_1) = H(M_2)$:

```
69 32 CB 92 D9 02 1C CD 5C C2
C0 4F 4F 17 B8 EA 1C 67 EF 10
```

Table A.4: A colliding message pair for 4 rounds of Keccak$[1088, 512]$.

M_1:

```
DC 22 00 57 69 5E 12 71 EC 8B 83 F0 95 A3 AF 80 34
70 01 6C 1C B3 31 9C 6E F5 F6 32 D7 30 96 8D 79 E8
2C 07 71 DB 42 34 E0 53 90 76 72 78 32 65 26 A8 14
94 76 7E 15 B0 D5 C4 AA 56 AB 57 1D E2 54 9B 4C 5E
E4 12 DD D0 55 8A 86 DC 42 B5 53 48 FA 45 17 5A 4B
A5 A6 97 D2 23 40 D9 D0 DA 30 7F AB 91 06 87 B0 C4
88 11 7B E6 9F E7 A3 4D 84 99 DA 15 5D 12 72 2F CB
22 B0 8D AF E9 CB 4D 31 71 DA 38 BF A4 ED 2D 34 05
```

M_2:

```
6F 25 88 C6 C0 69 3D 21 6B DA 35 84 B0 FD F6 41 82
25 8A 95 9F 12 7C 8C 2D 62 43 AF 3B 28 68 4F D7 46
BB E3 CB 7D 7C A0 C5 D6 36 DF 7E 08 8E 90 B2 E4 26
31 F6 F1 AC 35 20 68 57 34 A7 79 16 84 9F FD 56 C5
48 C3 57 4C 10 CD 71 7B 0D 67 8A 70 F2 98 0D 88 09
93 3A FA 18 5C EF B9 B4 C9 86 61 25 40 6E 87 B2 B5
E6 54 57 CC 0A FC DF D8 13 57 9A D3 F6 61 A3 82 38
F3 60 38 32 AC 86 41 8E 7D DE C4 52 F5 55 0C FF 66
```

$H(M_1) = H(M_2)$:

```
AA 57 74 2C 5F DE CC 5F D7 67 9C F7 4E 8F 7D 96
B3 B1 C2 8D 17 46 0E DB 5E 40 FF 28 64 FC 13 78
```

Table A.5: A colliding message pair for 4 rounds of Keccak$[1088, 512]$ with 384-bit output.

M_1:

```
FB B9 31 67 E2 64 1B 3F 09 99 19 73 C6 80 4F 2E AC
5F 41 A5 36 65 96 11 B5 08 79 4A 5A 07 33 FC 2D F8
87 31 3C 9D 28 BF DC 72 94 A8 21 9F 54 8B 26 3D 2D
06 9E 13 10 72 12 25 E5 4F 2C 25 9D 1B F1 EE 17 1C
50 EC 29 3A EA 72 91 F3 2E 84 62 D6 98 73 6C 46 BA
C3 D6 B3 AB C4 1A 68 D0 F7 CD 7B EE A5 38 05 A3 D7
B7 2C FC E2 31 E0 1C 97 A2 F5 9B E1 93 41 A3 DD 5F
4A A2 03 BD 41 C3 C6 A7 BF CD F1 B2 3C C2 A4 A4 75
```

M_2:

```
31 41 2C DA 5A C7 B8 5F 9D 8C 0C 6F 69 EE 3D AE D8
85 29 FE 02 3C 27 2E B0 DC 4C 67 70 A4 9A 27 37 68
A8 8D B9 00 3A CC BF 6A 82 63 96 16 EE E4 8C DA A4
A3 FF E2 B3 23 ED 97 82 2E CC FD 52 C3 6B 70 02 A3
11 CE 61 BA 4F EB 8C 0F 1D 37 CD 71 70 D7 9F AC 4D
4E 53 75 92 90 06 EF 3D 3A 92 CC F9 3D 00 C0 FC E1
39 85 36 FA 40 2A 6D 6A 45 5F AF 94 3A 23 30 F5 3C
BB 8B C7 77 30 D7 5B 54 AE F6 C2 F3 26 FD 22 21 C1
```

$H(M_1) = H(M_2)$:

```
B2 AD 99 8F CC 49 41 07 22 EE 61 24 71 35 03 59 0C
F6 6A 32 81 0C 70 7D F7 F3 50 86 07 BF 69 4F 8A F0
94 2C E4 D4 8A 80 BF D4 F4 75 76 23 1F FD
```

Bibliography

[1] Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of LNCS., Springer (2005) 1–18 (Cited on page 2.)

[2] Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of LNCS., Springer (2005) 19–35 (Cited on page 2.)

[3] Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. In Shoup, V., ed.: CRYPTO. Volume 3621 of LNCS., Springer (2005) 1–16 (Cited on page 2.)

[4] Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. [52] 17–36 (Cited on pages 2 and 36.)

[5] De Cannière, C., Mendel, F., Rechberger, C.: Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In Adams, C.M., Miri, A., Wiener, M.J., eds.: Selected Areas in Cryptography. Volume 4876 of LNCS., Springer (2007) 56–73 (Cited on page 2.)

[6] Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 proposal BLAKE. Submission to NIST (Round 3) (2010) (Cited on page 2.)

[7] Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Submission to NIST (Round 3) (2011) (Cited on page 2.)

[8] Wu, H.: The Hash Function JH. Submission to NIST (round 3) (2011) (Cited on page 2.)

[9] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak SHA-3 submission. Submission to NIST (Round 3) (2011) (Cited on pages 2, 17, 18, 34, 35 and 44.)

[10] Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family. Submission to NIST (Round 3) (2010) (Cited on page 2.)

[11] Menezes, A., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996) (Cited on page 3.)

[12] Standards, F.I.P.: Digital signature standard. (1994) (Cited on page 4.)

[13] Dierks, T.: The transport layer security (TLS) protocol version 1.2. (2008) (Cited on page 4.)

[14] Doraswamy, N., Harkins, D.: IPSec: the new security standard for the Internet, intranets, and virtual private networks. Prentice Hall (2003) (Cited on page 4.)

[15] Rivest, R.L., Shamir, A.: PayWord and MicroMint: Two Simple Micropayment Schemes. In: Proceedings of the International Workshop on Security Protocols, London, UK, UK, Springer-Verlag (1997) 69–87 (Cited on page 5.)

[16] Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. (2009) (Cited on page 5.)

[17] Provos, N., Mazieres, D.: A future-adaptable password scheme. In: Proceedings of the Annual USENIX Technical Conference. (1999) (Cited on page 5.)

[18] Percival, C.: Stronger key derivation via sequential memory-hard functions. BSDCan 2009 (2009) (Cited on page 5.)

[19] Kaliski, B.: RFC 2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0. Technical report, IETF (2000) (Cited on page 5.)

[20] Damgård, I.: A Design Principle for Hash Functions. In: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology. CRYPTO '89, London, UK, UK, Springer-Verlag (1990) 416–427 (Cited on page 8.)

[21] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT hash workshop. Volume 2007., Citeseer (2007) (Cited on page 9.)

[22] Flajolet, P., Odlyzko, A.M.: Random mapping statistics. In: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology. EUROCRYPT '89, New York, NY, USA, Springer-Verlag New York, Inc. (1990) 329–354 (Cited on page 11.)

[23] Yuval, G.: How to Swindle Rabin. Cryptologia **3** (1979) 187–189 (Cited on page 11.)

[24] Quisquater, J.J., Delescaille, J.P.: How Easy is Collision Search. New Results and Applications to DES. [53] 408–413 (Cited on page 11.)

[25] Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: how to construct a hash function. In: Proceedings of the 25th annual international conference on Advances in Cryptology. CRYPTO'05, Berlin, Heidelberg, Springer-Verlag (2005) 430–448 (Cited on page 12.)

[26] Kelsey, J., Schneier, B.: Second Preimages on n-bit Hash Functions for Much Less than 2^n Work. IACR Cryptology ePrint Archive **2004** (2004) 304 (Cited on page 12.)

[27] Joux, A.: Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. [54] 306–316 (Cited on page 12.)

[28] Aumasson, J.P., Meier, W.: Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. Rmp session of Cryptographic Hardware and Embedded Systems-CHES (2009) (Cited on page 20.)

[29] Khovratovich, D., Biryukov, A., Nikolic, I.: Speeding up Collision Search for Byte-Oriented Hash Functions. [55] 164–181 (Cited on page 20.)

[30] Boura, C., Canteaut, A.: Zero-sum distinguishers for iterated permutations and application to keccak-*f* and hamsi-256. [56] 1–17 (Cited on page 20.)

[31] Morawiecki, P., Srebrny, M.: A SAT-based preimage analysis of reduced Keccak hash functions. Cryptology ePrint Archive, Report 2010/285 (2010) `http://eprint.iacr.org/`. (Cited on page 20.)

[32] Naya-Plasencia, M., Röck, A., Meier, W.: Practical Analysis of Reduced-Round Keccak. [57] 236–254 (Cited on pages 21 and 45.)

[33] Duc, A., Guo, J., Peyrin, T., Wei, L.: Unaligned Rebound Attack: Application to Keccak. In Canteaut, A., ed.: Fast Software Encryption. Volume 7549 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 402–421 (Cited on page 21.)

[34] Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. [58] 260–276 (Cited on page 21.)

[35] Dinur, I., Dunkelman, O., Shamir, A.: New Attacks on Keccak-224 and Keccak-256. [59] 442–461 (Cited on pages 22 and 41.)

[36] Dinur, I., Dunkelman, O., Shamir, A.: Self-Differential Cryptanalysis of Up to 5 Rounds of SHA-3. IACR Cryptology ePrint Archive **2012** (2012) 672 (Cited on page 22.)

[37] Albrecht, M.: Algorithmic Algebraic Techniques and their Application to Block Cipher Cryptanalysis. PhD thesis, Royal Holloway, University of London (2010) (Cited on page 24.)

[38] Cox, D.A., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007) (Cited on page 24.)

[39] Buchberger, B.: Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. J. Symb. Comput. **41**(3-4) (2006) 475–511 (Cited on page 26.)

[40] Dubé, T.W.: The structure of polynomial ideals and Gröbner bases. SIAM Journal on Computing **19**(4) (1990) 750–773 (Cited on page 27.)

[41] Faugere, J.C.: A new efficient algorithm for computing Gröbner bases (F4). Journal of pure and applied algebra **139**(1) (1999) 61–88 (Cited on page 27.)

[42] Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In: Proceedings of the 2002 international symposium on Symbolic and algebraic computation, ACM (2002) 75–83 (Cited on page 27.)

[43] Brickenstein, M.: Slimgb: Gröbner bases with slim polynomials. Revista Matemática Complutense **23**(2) (2010) 453–466 (Cited on page 27.)

[44] Stein, W., et al.: Sage Mathematics Software (Version 5.6). The Sage Development Team. (YYYY) `http://www.sagemath.org`. (Cited on page 31.)

[45] Brickenstein, M., Dreyer, A.: PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. Journal of Symbolic Computation **44**(9) (2009) 1326 – 1345 Effective Methods in Algebraic Geometry. (Cited on page 31.)

[46] Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology **4**(1) (1991) 3–72 (Cited on page 32.)

[47] Cannière, C.D., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. [60] 1–20 (Cited on page 36.)

[48] Mendel, F., Nad, T., Schläffer, M.: Finding Collisions for Round-Reduced SM3. [61] 174–188 (Cited on page 37.)

[49] Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. [62] 288–307 (Cited on pages 37 and 38.)

[50] Mendel, F., Nad, T., Scherz, S., Schläffer, M.: Differential Attacks on Reduced RIPEMD-160. [63] 23–38 (Cited on page 37.)

[51] Canteaut, A., Chabaud, F.: A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. Information Theory, IEEE Transactions on **44**(1) (1998) 367–378 (Cited on page 51.)

[52] Shoup, V., ed.: Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) (Cited on page 64.)

[53] Brassard, G., ed.: Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. In Brassard, G., ed.: CRYPTO. Volume 435 of Lecture Notes in Computer Science., Springer (1990) (Cited on page 66.)

[54] Franklin, M.K., ed.: Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings. In Franklin, M.K., ed.: CRYPTO. Volume 3152 of Lecture Notes in Computer Science., Springer (2004) (Cited on page 66.)

[55] Fischlin, M., ed.: Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings. In Fischlin, M., ed.: CT-RSA. Volume 5473 of Lecture Notes in Computer Science., Springer (2009) (Cited on page 66.)

[56] Biryukov, A., Gong, G., Stinson, D.R., eds.: Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August

12-13, 2010, Revised Selected Papers. In Biryukov, A., Gong, G., Stinson, D.R., eds.: Selected Areas in Cryptography. Volume 6544 of Lecture Notes in Computer Science., Springer (2011) (Cited on page 66.)

[57] Bernstein, D.J., Chatterjee, S., eds.: Progress in Cryptology - INDOCRYPT 2011 - 12th International Conference on Cryptology in India, Chennai, India, December 11-14, 2011. Proceedings. In Bernstein, D.J., Chatterjee, S., eds.: INDOCRYPT. Volume 7107 of Lecture Notes in Computer Science., Springer (2011) (Cited on page 66.)

[58] Dunkelman, O., ed.: Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers. In Dunkelman, O., ed.: FSE. Volume 5665 of Lecture Notes in Computer Science., Springer (2009) (Cited on page 66.)

[59] Canteaut, A., ed.: Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. In Canteaut, A., ed.: FSE. Volume 7549 of Lecture Notes in Computer Science., Springer (2012) (Cited on page 67.)

[60] Lai, X., Chen, K., eds.: Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings. In Lai, X., Chen, K., eds.: ASIACRYPT. Volume 4284 of Lecture Notes in Computer Science., Springer (2006) (Cited on page 68.)

[61] Dawson, E., ed.: Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings. In Dawson, E., ed.: CT-RSA. Volume 7779 of Lecture Notes in Computer Science., Springer (2013) (Cited on page 68.)

[62] Lee, D.H., Wang, X., eds.: Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings. In Lee, D.H., Wang, X., eds.: ASIACRYPT. Volume 7073 of Lecture Notes in Computer Science., Springer (2011) (Cited on page 68.)

[63] Gollmann, D., Freiling, F.C., eds.: Information Security - 15th International Conference, ISC 2012, Passau, Germany, September 19-21, 2012. Proceedings. In Gollmann, D., Freiling, F.C., eds.: ISC. Volume 7483 of Lecture Notes in Computer Science., Springer (2012) (Cited on page 68.)