KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Cryptanalysis of Hash Functions

Deniz Toz

Dissertation presented in partial
fulfillment of the requirements for the
degree of Doctor in Engineering

March 2013

Cryptanalysis of Hash Functions

Deniz TOZ

Supervisory Committee:
Prof. dr. ir. Hendrik Van Brussel, chair
Prof. dr. ir. Vincent Rijmen, supervisor
Prof. dr. ir. Joos Vandewalle
Prof. dr. ir. Karl Meerbergen
Prof. dr. ir. Bart Preneel
Dr. Christian Rechberger
(Technical University of Denmark)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

March 2013

D/2013/7515/28
ISBN 978-94-6018-641-7

Acknowledgements

A PhD is a long journey that requires the involvement of many people. Here, I would like take this opportunity to express my gratitude to people who made this dream come true. Unfortunately it is not possible to list everyone here, so I will try to single out only a few names without whom this dissertation would have not been possible.

First and foremost I would like to express my deep gratitude to Prof. Vincent Rijmen for accepting me as his student, I consider it an honour to work with him. I would like to thank him for his friendly attitude, limitless support, motivation, and enthusiasm. His guidance and immense knowledge helped me during the past four years with my research and writing of this thesis. I could not have imagined a better mentor for my PhD.

I would like to express my gratitude to Prof. Karl Meerbergen, Prof. Bart Preneel, Dr. Christian Rechberger, and Prof. Joos Vandewalle for serving in my jury and reviewing this manuscript and to Prof. Hendrik Van Brussel for chairing the jury. The support of the Research Council of KU Leuven for my research is greatly appreciated.

I give many thanks to Kerem Varıcı first for encouraging me to apply for a PhD in COSIC and then for all the joint work and discussions, and finally for his patience and putting up with me 24/7. Sevgili Bücür, iyi ki varsın ve yanımdasın.

In the course of my research I had the opportunity to meet and work with several smart and brilliant researchers. I share the credit of my work with my co-authors: Andrey Bogdanov, Orr Dunkelman, Miroslav Knežević, Gregor Leander, Florian Mendel and María Naya-Plasencia.

I am indebted to my many colleagues and ex-colleagues who made COSIC a nice working environment. Especially to Elmar, Qingju and Christophe with whom I shared an office and very enjoyable moments, to Roel P., Vesselin and

Stefan for their friendship, Filipe, Nessim and Yoni for occasional chats.

Special thanks go to our secretary Péla Noë for helping me out countless times, without her I would have been lost in bureaucracy. Not to forget her positive attitude, kindness and the eggs. I also would like to thank Elsy Vermoesen and Wim Devroye for financial aspects and Saartje Verheyen for various organisations.

Many thanks to all of my friends at Leuven, in particular to the Turkish community Begül, Deniz, Eralp, Ilker, Özgül, Suna and Barış for all the nice memories and to Onur Özen for the remote support and the last minute corrections.

Last but not least, I would like to thank my parents for supporting me spiritually throughout my life.

Deniz Toz
Leuven, March 2013

Abstract

This thesis deals with the analysis and design of cryptographic hash functions that are fundamental components of many cryptographic applications such as digital signatures, authentication, key derivation, random number generation and many others. Due to this versality they are considered as the "Swiss army knives" of modern cryptology.

A hash function is a one-way mathematical function that takes a message of arbitrary length as input and produces an output of fixed (smaller) length. In recent years, several of the approved cryptographic hash functions which are generally inspired by MD4 have been successfully attacked, and serious attacks have been published against the world-wide standard SHA-1. In response, the National Institute of Standards and Technology (NIST) has opened a public competition to develop a new cryptographic hash algorithm, SHA-3, to replace the older SHA-1 and SHA-2 hash functions.

The first part of this thesis is focused on the analysis of the hash function JH, one of the finalists of this competition. We demonstrate attacks on JH showing that the algorithm is not as secure as claimed by its designer. We find a semi-free-start collision for the hash function and semi-free-start near-collisions for the compression function of reduced-round JH. Moreover, we present distinguishers for the full internal permutation.

The second part of this thesis is focused on the design of hash functions. We propose a new family of sponge-based lightweight hash function called SPONGENT. We first explain the design strategy of SPONGENT and then we present its security analysis by applying the most important state-of-the-art methods of cryptanalysis and by investigating their complexity.

Beknopte samenvatting

Het onderwerp van deze thesis is de analyse en het ontwerp van cryptografische klutsfuncties (hash functions) die essentiële componenten vormen in een groot aantal cryptografische toepassingen zoals onder andere digitale handtekeningen, authenticatie, sleutelafleiding en het genereren van willekeurige getallen. Omwille van deze veelzijdigheid, worden de klutsfuncties beschouwd als "het Zwitser zakmes" van de cryptografie.

Een klutsfunctie is een functie die een bericht van willekeurige lengte transformeert in een reeks met een vaste (kleinere) lengte. In de afgelopen jaren zijn verschillende algemeen aanvaarde cryptografische klutsfuncties, gebaseerd op MD4, met succes aangevallen en werden ernstige aanvallen gepubliceerd op de werelwijde standaard SHA-1. Als reactie hierop, organiseerde het National Institute of Standards and Technology (NIST) een publieke wedstrijd om een nieuwe cryptografische klutsfunctie, SHA-3, te ontwikkelen om de oudere SHA-1 en SHA-2 klutsfuncties te vervangen.

Het eerste deel van dit proefschrift richt zich op de analyse van de klutsfunctie JH, één van de finalisten van deze wedstrijd. We demonstreren aanvallen op JH waaruit blijkt dat het algoritme niet zo veilig is als de ontwerper beweert. We vinden botsingen met semivrije start (semi-free-start collision) voor de klutsfunctie en en bijna-botsingen met semivrije start voor de compressiefunctie van JH met een verminderd aantal ronden. Bovendien presenteren we algoritmes om de volledige interne permutatie te onderscheiden van een ideale permutatie.

Het tweede deel van dit proefschrift richt zich op het ontwerp van klutsfuncties. We stellen SPONGENT voor: een nieuwe familie van lichtgewichtklutsfuncties, gebaseerd op het sponsmodel. In de eerste plaats leggen we de ontwerpstrategie van SPONGENT uit. Daarna geven we de veiligheidsanalyse door de belangrijkste state-of-the-artmethoden van cryptanalyse toe te passen en hun complexiteit te onderzoeken.

Contents

Acknowledgements i

Abstract iii

Contents vii

List of Figures xi

List of Tables xiii

1 Introduction 1

1.1 Cryptology 1

1.1.1 The Evolution of Cryptology 2

1.1.2 Modern Cryptology 4

1.1.3 Goals of Cryptography 7

1.2 Hash Functions 7

1.2.1 State of the Art 8

1.2.2 SHA-3 Competition 8

1.2.3 Lightweight Hashing 9

1.3 About this Dissertation 10

2 Hash Functions **11**

2.1 Security Requirements 12

2.2 Iterated Hash Functions 15

2.2.1 Merkle-Damgård Construction 15

2.2.2 Tree Construction 17

2.2.3 Sponge Construction 17

2.3 Construction Methods 19

2.3.1 Hash Functions Based on Mathematical Problems . . . 19

2.3.2 Hash Functions Based on Block Ciphers 19

2.3.3 Hash Functions Based on Permutations 21

2.3.4 Hash Functions Based On Stream Ciphers 21

2.4 Attacks on Hash Functions 22

2.4.1 Generic Attacks 22

2.4.2 Dedicated Attacks 25

3 Cryptanalysis Methods **29**

3.1 Differential Cryptanalysis 30

3.1.1 Definitions 30

3.1.2 Towards Attacks 32

3.1.3 Application to Hash Functions 33

3.1.4 The Rebound Attack 34

3.1.5 Splice-and-Cut Technique and Bicliques 35

3.1.6 Boomerang Attack 36

3.1.7 Higher-Order Collisions 38

3.2 Linear Cryptanalysis 39

3.3 Algebraic Cryptanalysis 40

3.4 Cube Attack 41

4 Cryptanalysis of JH **45**

4.1 Preliminaries 46

4.2 The JH Hash Function 46

4.2.1 Properties of the Linear Transformation L 48

4.2.2 Observations on the Compression Function 49

4.3 The Start-From-The-Middle Attack on JH 49

4.3.1 Attack on 8 Rounds of JH for $d = 4$ 49

4.3.2 The Attack on 16 Rounds of JH with $d = 8$ 52

4.4 The Rebound Attack on the Compression Function of JH . . . 54

4.4.1 The Improved Rebound Attack on JH with d= 4 55

4.4.2 The Improved Rebound Attack on JH with $d = 8$ 59

4.5 Rebound Attack on JH42 63

4.5.1 Matching the Active Bytes 63

4.5.2 Matching the Passive Bytes 68

4.5.3 Outbound Phase 72

4.6 Distinguishers on JH42 73

4.6.1 Distinguishers for the Reduced Round Internal Permutation 73

4.6.2 Distinguishers for the Full Internal Permutation 73

4.6.3 Distinguishers for the Full Compression Function 76

4.7 Contribution 76

4.8 Conclusion 76

5 SPONGENT **79**

5.1 Design Considerations for Lightweight Hashing 80

5.2 The Design of SPONGENT 81

5.2.1 Permutation-based Sponge Construction 81

5.2.2 Parameters 81

5.2.3 PRESENT-type Permutation 83
5.2.4 Design Rationale 84
5.3 Security Analysis 85
5.3.1 Resistance Against Differential Cryptanalysis 85
5.3.2 Collision Attacks 90
5.3.3 Preimage Resistance 93
5.3.4 Linear Attacks 95
5.4 Contribution 96
5.5 Conclusion 97

6 Conclusion **99**
6.1 Contributions of This Thesis 99
6.2 Directions for Future Work 100

Bibliography **103**

A Appendix to Chapter 4 **121**

B Appendix to Chapter 5 **123**

Curriculum Vitae **129**

List of Figures

2.1 Hash function 11

2.2 Security requirements of hash functions 13

2.3 Collision attacks for hash functions 14

2.4 An iterated hash function 15

2.5 Binary tree construction for $n = 4$ 17

2.6 Sponge construction based on a b-bit permutation π_b with capacity c bits and rate r bits. m_i are r-bit message blocks. h_i are parts of the hash value 18

2.7 Some block cipher based compression function constructions . . 20

2.8 Schematic representation of multicollision attack 26

2.9 Finding a linking message and producing the suffix 27

3.1 A characteristic over a composed map 31

3.2 The Rebound Attack 34

3.3 Splice-and-cut technique with initial structure 35

3.4 Biclique with dimension $d = 2$ in the meet-in-the-middle attack 36

3.5 The Boomerang attack and a right quartet 37

4.1 The compression function F_d 47

4.2 Round Function of E_d for $d = 4$ 48

4.3 Inbound phase of JH for $d = 4$ (bit-slice implementation) . . . 50

4.4 Differential characteristic for 16 rounds of the JH Hash Function (bit-slice representation) . . . 53

4.5 Inbound phase of JH ($d = 4$) . . . 56

4.6 Outbound phase of JH ($d = 4$) . . . 58

4.7 Inbound and outbound phases of JH compression function (bit-slice representation) . . . 60

4.8 Differential characteristic for 32 rounds of the JH42 compression function (bit-slice representation) . . . 64

4.9 Improving the complexity of finding a solution for the differential part for rounds $4 - 16$ of JH42: Red boxes (above) denote the sets $L_{A,2}$ (filled) and $L_{B,2}$, blue boxes (below) denote the sets $L_{A,3}$ and $L_{B,3}$ (filled). . . . 69

4.10 Matching the passive bytes of JH42 (bit-slice representation). Blue and green boxes denote the passive words whose values are fixed . . . 70

4.11 Differential characteristic for the outbound phase of JH Compression Function (bit-slice representation) . . . 75

5.1 The bit permutation layer of SPONGENT-88 . . . 83

5.2 The grouping and subgrouping of S-boxes for $b = 256$. The input numbers indicate the S-box origin from the previous round and the output numbers indicate the destination S-box in the following round. . . . 88

5.3 Differential path for the rebound attack on SPONGENT-128/256/128 (S: sBoxLayer_{384}, P: pLayer_{384}) . . . 92

5.4 Meet-in-the-middle attack against sponge construction . . . 94

List of Tables

4.1 The difference distribution of the linear layer 48

4.2 Overview of inbound phase 54

4.3 Overview of inbound phases for the attack on JH-512 61

4.4 Complexity of the generic attack for near-collisions 62

4.5 Overview of inbound phases of the attack on 32 rounds of JH . 65

4.6 Comparison of complexity of the generic attack for near-collisions and our results on JH42 72

4.7 Comparison of best attack results on JH (sfs: semi-free-start) . 77

5.1 13 SPONGENT variants 82

5.2 Initial values of lCounter_b for all SPONGENT variants 84

5.3 Differential characteristics with lowest numbers of differentially active S-boxes (ASN). The probabilities are calculated assuming the independency of round computations. 86

5.4 The longest characteristics obtained for all SPONGENT variants with probability in the range of 2^{-b} 90

5.5 Bounds for rebound attack 93

5.6 Meet-in-the-middle attack results for SPONGENT 95

5.7 Results of linear trail correlation based on one bit masks for SPONGENT 97

A.1 Example for rebound attack with one inbound phase ($d = 4$) . 122

A.2 Example for rebound attack with three inbound phases ($d = 4$) 122

B.1 Sample differential path for SPONGENT-160/160/16 123

B.2 Sample differential paths for SPONGENT-160/160/80 and SPONGENT-224/224/16 124

B.3 Sample differential paths for SPONGENT-128/256/128 125

B.4 Sample differential paths for SPONGENT-256/256/128 126

B.5 Sample differential paths for SPONGENT-160/320/160 127

Abbreviations

AES	Advanced Encryption Standard
AIDA	Algebraic IV Differential Attack
ANF	Algebraic Normal Form
ARX	Addition Rotation XOR
CFTP	Chosen Target Forced Prefix
DES	Data Encryption Standard
GF	Galois Field
HMAC	Hash-based Message Authentication Code
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
MAC	Message Authentication Code
MDC	Modification Detection Code
MDP	Maximum Differential Probability
MDPC	Maximum Differential Characteristic Probability
MDS	Maximum Distance Separable
MQ	Multivariate Quadratic
NBS	National Bureau of Standards
NIST	National Institute of Standards and Technology
NSA	National Security Agency
PRNG	Pseudorandom Number Generator

RFID	Radio-Frequency IDentification
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SPN	Substitution Permutation Network
UNAF	Unsigned Non-Adjacent Form
XL	Extended Linearization
XSL	eXtended Sparse Linearization

List of Symbols

0^c	c consecutive zeros
$\cdot$	cross-product: an operation on two arrays that result in another array whose elements are obtained by combining each element in one array with every element in the second one
$\leftarrow$ and $\rightarrow$	shows the direction of the computations for the inbound phase
$\oplus$	exclusive OR operation (XOR)
ζ	the root of unity in the corresponding binary finite field
$\{0,1\}^n$	set of all binary vectors of dimension n
d	dimension of a block of bits i.e. a d-dimensional block of bits consists of 2^d words
H_i	the i-th chaining value
M_i	Message block i
$m_{i,j}$	the j^{th} word of the i^{th} round value
JH-X	the member of the JH family whose message digest is X bits
t	number of blocks in the padded message
word	4-bits

Chapter 1

Introduction

This thesis deals with the analysis and design of cryptographic hash functions which are considered as the "Swiss army knives" of modern cryptology. In this chapter, we cast a light on how cryptology evolved from an "ancient art" into a science. We begin with the very early methods and present an overview of the major developments in the field of cryptology. In the meantime we describe the basic notions and explain why we need cryptology and what the role of cryptographic hash functions in this field is.

1.1 Cryptology

Being the science of secret communication, cryptology is derived from the Greek words *kryptos* meaning hidden and *logos* meaning speech. It is divided into two major branches: *Cryptography*, which is the science of modifying and sending the message in such a way that only the intended people are able to read it; and *cryptanalysis*, which evaluates the security of the methods that are used to conceal the message and tries to break them. For thousands of years, the history has witnessed the ongoing battle between these two branches, deciding the outcomes of wars, shaping the fate of people and inspiring scientific developments.

The process of transforming the message, known as *plaintext*, by using a tool is called *encryption*. The tool, more specifically the algorithm, designed and

used for this purpose is referred to as *cipher*. The resulting message is called *ciphertext* and is unreadable to anyone without the knowledge of a secret information. Therefore, most ciphers require a *key* and the original message can be revealed by an inverse process called *decryption*.

> Cryptography is like literacy in the Dark Ages.
> Infinitely potent, for good and ill...
> yet basically an intellectual construct, an idea, which by its nature
> will resist efforts to restrict it to bureaucrats and others who deem
> only themselves worthy of such Privilege.
>
> *A Thinking Man's Creed for Crypto*
> Vin McLellan

1.1.1 The Evolution of Cryptology

Over 4000 years, cryptography has been used to conceal sensitive information. According to the historical sources, the first form of secret communication dates back to the ancient Greeks [93]. The message was written on wooden tablets which were then covered with wax. In this way, the tablets seemed as if they were blank or sometimes an innocent text was also written on them. In another incident, the head of the messenger was shaved and the message was tattooed on his scalp. When the hair grew back, the message was invisible to the eyes. This technique in which the message is only hidden without any modification is called as *steganography*, meaning "concealed writing".

Ancient Greeks were also the first civilization that used a tool for to perform encryption. The message was written on a strip of parchment which was wound around a cylinder called *scytale*. When the parchment was taken off, the resulting text was meaningless since the letters of the message were jumbled. Reading the original message was only possible by using a cylinder having the right diameter. This technique constitutes the first known example of a *transposition cipher*.

Romans, in particular Julius Caesar used a simple substitution in which each letter of the message to be encrypted was replaced by the letter three positions after it in the alphabet (more specifically 'a' is replaced by 'd', 'b' with 'e', etc.). To decrypt the message it was sufficient to know the substitution rule (*cipher alphabet*). The Caesar cipher represents the first use of a *substitution cipher* for military purposes.

Substitution ciphers that replace only one letter, cipher or symbol at a time by another letter are called *monoalphabetic*. It had been possible to break this

type of ciphers with the discovery of frequency analysis in the ninth century [1]. The analysis is based on the fact that certain letters and their combinations occur with varying frequencies. By calculating the frequency distribution of the ciphertext letters and comparing it with the one of the language under consideration, the attacker is able to guess the plaintext.

Several improvements have been proposed to avoid this attack by cryptographers. Using multiple substitutes for a letter (*homophones*), using ciphers that have two or more cipher alphabets (*polyalphabetic cipher*), or substituting letter-pairs (*digraph*) and rarely larger group of letters (*polygraph*) instead of one letter, are some of these schemes.

The best known polyalphabetic cipher is the *Vigenère Cipher*. It uses 26 different cipher alphabets[1] written in rows, forming a square called *tabula recta*. To perform both the encryption and the decryption, a different cipher alphabet is used for each letter depending on the repeating key word. The cipher was considered to be unbreakable at that time and called as *le chiffre indéchiffrable*.

Apart from ciphers mentioned above, *codes* that are composed of thousands of codewords or codenumbers used to replace syllables, words or phrases of the messages have been used to conceal messages. In these schemes, encryption and decryption is possible with the use of a codebook. According to Kahn [107], "for 450 years, from about 1400 to about 1850, a system that was half a code and half a cipher dominated cryptography".

Without doubt, the *Enigma* machine used by Germans during the World War II is the most famous cipher system of its era. The basic idea of Enigma was no different than that of a code except for its complicated structure. It consisted of three coding wheels, each having 26 contacts on either side, that could be inserted in any possible order. Whenever a key was pressed, some of the wheels rotated by one step resulting in a different wiring and allowing each letter to be encoded differently.

Note that in all these examples given above, the security of the system depends on the knowledge of the same secret information shared among the parties. The field of cryptography that studies this type of cryptographic systems is called *symmetric-key cryptography*.

[1] 26 corresponds to the number of letters in the alphabet

1.1.2 Modern Cryptology

For most cryptographers the era of modern cryptography begins with Shannon who is the father of information theory and mathematical cryptography. He not only worked on the problem of most efficiently transmitting information but also stated the basic principles for the design of cryptographic algorithms [185, 184]. In 1950's, he proposed information-theoretic definitions of security such as "theoretical secrecy", "perfect secrecy" and "practical secrecy" which are still very significant today.

Moreover he introduced the concept of *product cipher*. The main idea was to combine two or more simple operations such as modular arithmetic, substitution and permutation to obtain a more secure cipher (than either of its components). Almost all of the algorithms used today are based on this concept.

In the meanwhile, with the birth of modern computers cryptanalysts were able to search through all possible keys and break all sorts of ciphers. In response, cryptographers started to design more complex ciphers having larger key sizes to overcome the power of computers. As a result, both fields have evolved by a considerable amount in parallel.

Block Ciphers Go Civilian

Even though computers were initially only for governmental and military use, in the 1960's they became affordable and powerful enough also for the private sector. Consequently, the need of a common system for communicating with the other companies in addition to internal communication arose. Lucifer [78] was developed by IBM to fulfill this need.

Lucifer is considered as the first non-military *block cipher* where the plaintext is partitioned into blocks of fixed length and each block is encrypted using the same secret key. The ciphertext is then obtained by combining the outputs of all encryptions. Most block ciphers use an iterative round function (based on the product cipher) as the building block.

Apart from block ciphers, there are also *stream ciphers* in which the plaintext bits (words) are encrypted one at a time by using the corresponding bit (word) of the keystream which is generated by a shift register. However, we will not go into details of stream ciphers in this thesis.

With the beginning of the information age in the 1970's, the exchange of digital information became an essential part of our society. In 1973, the

National Bureau of Standards (NBS) made a call for a candidate symmetric-key encryption algorithm for the protection of sensitive but unclassified information. Unfortunately none of the proposals were found viable and a second call was issued in 1974. After the evaluation phase, the submission of IBM which was heavily influenced by Lucifer, was chosen to become the Data Encryption Standard (DES) after some modifications.

Birth of Public-key Cryptography

The adaptation of DES encouraged businesses to use cryptography for security. Although DES was strong enough for communication, it had one major drawback: the *key distribution*. In order to establish a secure connection via symmetric-key cryptography, both parties required the knowledge of the secret key before the communication starts. If the same key were shared by many users the whole system would be broken once the key was revealed. On the other hand, there was no trivial way to produce and distribute several keys. Using a telephone line was insecure, handing it over in person was impractical, finally sending the key via a courier was time consuming and it became too costly as the number of users increased.

In 1976, Diffie and Hellman [69] introduced their key exchange protocol to address these issues. The basic idea was using a *one-way function* which is easy to compute but hard to invert (in this case taking powers in modular arithmetic). Each user had a pair of keys, one private and one public, that allowed them to compute the shared key that will be used for the symmetric-key algorithm.

This work was followed in 1978 by the RSA encryption algorithm. In this scheme, the users no longer had to exchange keys before the communication starts. The sender simply used the public-key of the recipient, which is known by everybody, to encrypt a message, and only the intended recipient was able to read the message by using his private key. This relatively new field of cryptography is called *public-key cryptography* which is also known as asymmetric cryptography.

Advances in Symmetric-key Cryptography

At the end of 1980's and the beginning of 1990's new block ciphers were designed as an alternative to DES. Some examples include RC5, IDEA, FEAL, Blowfish and CAST. Meanwhile the cryptographers made a great effort in analyzing the security of DES. Differential cryptanalysis [28] and linear cryptanalysis [136]

which are the core of many other cryptanalysis techniques that are used today, were introduced in this period. The DESCHALL Project, which consisted of thousands of volunteers connected over the Internet, was the first to break DES in public in 1997. Only two years later, it was possible to perform an exhaustive key search for DES in less than a day.

Obviously the short key-length of DES was no longer sufficient for sensitive applications. Moreover, being initially designed for hardware performance, DES was not as efficient in software as the new block ciphers. In 1997, NIST (National Institute of Standards and Technology) made a public call for a new algorithm "capable of protecting sensitive government information well into the next century" [155] to replace the DES.

As a result, fifteen submissions were submitted to the competition and after two years evaluation five of them were chosen as finalists. Finally in 2001 Rijndael [57], designed by Rijmen and Daemen, was chosen as the Advanced Encryption Standard (AES). Unsurprisingly, in the past 10 years many attacks have been published against AES, yet none of them is considered a practical threat to its security. All the attacks were either against reduced round AES or they worked under some special conditions until the recent work of [34] which reduces the complexity of exhaustive search by a factor of four.

Cryptography Today

Without doubt cryptography became a must-have part of our daily lives. The Internet provided the infrastructure for electronic mails, online banking, and e-commerce, all of which requires reliability and security. Our identity cards, passports, ATM and SIM cards all have chips with embedded cryptographic modules to ease our needs. Many governmental applications can already be made online and there are ongoing studies for bringing e-voting to life in the near future. Our cellphones and smartphones are much more capable than being just a phone and they contain sensitive data which needs to be protected, not to mention our computers, laptops and the data in cloud storage services.

The success of all the systems given above depends on their ability to protect the information and they all have different constraints to be satisfied. Simply put, the need for cryptography has increased with the information age and the researchers are trying to develop efficient algorithms that best fits our needs for different platforms.

1.1.3 Goals of Cryptography

Cryptography aims to provide security by attaining (at least) the following four main goals:

- **Confidentiality:** Being the original purpose of cryptography, confidentiality ensures that the data is accessible and can be understood only by authorized people. It is usually achieved by encrypting the data using symmetric-key cryptography.
- **Entity Authentication:** This concept uniquely defines individuals / identities allowing the users to prove who they are. Login IDs and passwords, tokens, identity cards and biometrics data are typical methods used for this purpose.
- **Data Authentication:** This notion guarantees that the data is not damaged or manipulated during the transmission by unauthorised third parties. The integrity of data is usually checked by using hash functions.
- **Non-Repudiation:** It means that the users will not be able to deny that they have sent or received the data. It is mostly used in the verification of digital signatures.

Although all these notions are required in today's cryptography as described above, their importance can vary depending on their application and usage. For example, while confidentiality is more important in communication, authenticity can be more crucial in financial transactions.

Upto now we tried to cast a light on the history of cryptography by providing the milestones throughout the history. For a more comprehensive study we refer to *The Codebreakers* by Kahn [107] and *The Code Book* by Singh [187]. For a more technical study we refer to *The Handbook of Applied Cryptography* by Menezes et al. [144].

1.2 Hash Functions

A hash function is a mathematical function that takes a *message* of arbitrary length as input and produces an output of fixed (smaller) length, which is commonly called a *fingerprint* or *message digest*. They are fundamental components of many cryptographic applications such as digital signatures, password protection, message authentication, random number generation, etc. (see Chapter 2 for more detail).

1.2.1 State of the Art

The concept of cryptographic hash functions was first introduced by Diffie and Hellman [69] to make the digital signatures more efficient. Most of the early designs are based on block ciphers. Merkle [147] was to give a method for constructing one-way hash functions from random block ciphers. Later, Preneel et al. [163] studied how to construct a hash function whose the output size equals to the block length of the cipher.

In 1990 Rivest introduced MD4 [171], the first dedicated hash function, which contains a block cipher designed for hashing. In 1991, after a number of attacks, Rivest proposed a strengthened version called MD5 [170]. A similar design was used in the first version of Secure Hash Standard (SHS) by NIST in 1993. The algorithm, known as SHA-0, was replaced by its successor SHA-1 [154] after some minor modifications in 1995.

SHA-2, the last member of this family, is composed of four algorithms named after their hash sizes in bits: SHA-224, SHA-256, SHA-384, SHA-512. It has been used in many security applications and protocols along with SHA-1 as a standard in the past decade.

Other hash functions of the MD family are HAVAL [208], RIPEMD [38], and its successors RIPEMD-128 and RIPEMD-160 [73]. Whirlpool [15] is another well-known hash function in the IS0/IEC standard on dedicated hash functions [100]. It has a different design strategy than the MD family: it contains a block cipher called W which is very similar to AES (with a larger block size) as the building block.

1.2.2 SHA-3 Competition

Recent years witnessed the continuous works on analysis of hash functions which reveal that most of them are not as secure as claimed. Wang et al. presented collisions on the MD family [199, 201, 200] using an attack technique on hash functions which is based on differential cryptanalysis. This idea was further developed and used in the analysis of the widely used hash functions SHA-1 and SHA-2 [64, 62, 190]. In response, the National Institute of Standards and Technology (NIST) announced a public competition for designing a new hash function which will be chosen as the hash function standard: Secure Hash Algorithm 3 (SHA-3) [153].

Submissions of the NIST Hash Competition were due October 2008 and 64 algorithms have been submitted. A list of candidates accepted for the first round was published on December 2008 and 51 of the submissions had passed

the preliminary elimination. In less than a year many of these algorithms have been officially conceded to be broken by their submitters and withdrawn. In July 2009, NIST announced that 14 submissions had made it into the the second round.

In December 2010, five algorithms —BLAKE [13], Grøstl [82], JH [205], KECCAK [21] and Skein [79]— had been selected for the final round. In October 2012, KECCAK was announced as the winner "based on the public analysis and internal review of the candidates" [156].

1.2.3 Lightweight Hashing

As technology is embedded in everyday objects (tools, devices, clothing, homes, etc.), the need for security in RFID and sensor networks is dramatically increasing, which requires secure yet efficiently implementable cryptographic primitives including hash functions.

Until recently the existing work on lightweight functions was mainly focused on minimizing the area requirements. However in reality, there are also other design parameters such as latency, throughput or power consumption that should be taken into consideration while designing a lightweight algorithm and different parameters needs to be optimized depending on the purpose of the application.

Once this research problem was identified, the cryptographic community designed a number of tailored lightweight cryptographic algorithms to specifically address this challenge: stream ciphers like Trivium [63, 60], Grain [89, 90], and Mickey [14] as well as block ciphers like SEA [189], DESL, DESXL [127], HIGHT [95], mCrypton [130], KATAN/KTANTAN [61], and PRESENT [36] — to mention only a small selection of the lightweight designs.

Rather recently, some significant work on lightweight hash functions has been also performed: [37] describes ways of using the PRESENT block cipher in hashing modes of operation and [12] and [86] take the approach of designing a dedicated lightweight hash function based on a sponge construction [55, 22] resulting in two hash functions QUARK and PHOTON.

1.3 About this Dissertation

This dissertation consists of two parts. The first part gives a brief introduction to cryptography, cryptographic hash functions and cryptanalysis methods. The second part is composed of selected publications on cryptanalysis and design of hash functions.

The outline of the thesis is given below:

Chapter 2. We give a brief introduction to hash functions. We first describe their security requirements and design strategy. We then review the generic and dedicated attacks on hash functions.

Chapter 3. We define the cryptanalysis methods used for the analysis of symmetric-key primitives. We mainly focus on differential cryptanalysis techniques which will be used for the rest of the chapters.

Chapter 4. We apply the rebound attack to the JH Hash Function (one of the five finalists of the SHA-3 competition). We first give a brief description of the JH hash function and its properties. We first describe the main idea of our attack on small scale version of JH and then give the results on the submitted version of JH. We follow the same outline for the improved version of the rebound attack. Finally, we extend our attack and present results on the tweaked version JH42. Based on these results, we describe a distinguisher for the full internal permutation, that also applies to the full compression function.

Chapter 5. We introduce a new lightweight hash function SPONGENT. We first describe the design of SPONGENT and give its design rationale. We then presents some results of the the security analysis including proven lower bounds on the number of differentially active S-boxes, the best differential characteristics found, rebound attacks, and linear attacks.

Chapter 2

Hash Functions

A hash function is a mathematical function that takes a *message* of arbitrary length as input and produces an output of fixed (smaller) length, which is commonly called as *fingerprint* or *message digest*, see Figure 2.1. More formally, it can be defined as:

Definition 1. *A* hash function $H : D \rightarrow R$ *is a function that maps variable-length input bit strings* $M \in \{0,1\}^*$ *to fixed-length output bit strings* $H(M) \in \{0,1\}^n$ *for a positive integer* n*.*

Since $|D| > |R|$, this function is always many-to-one. For that reason, there are always at least two messages that have the same fingerprint, which is called a *collision*.

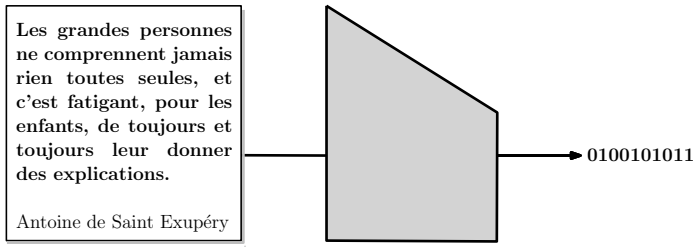


Figure 2.1: Hash function

Although it is not possible to avoid collisions, finding them easily can be avoided if each output value is seen approximately equally likely. For a cryptographic hash function H, it is expected that it should not be computationally feasible to find collisions.

Hash functions are fundamental components of many cryptographic applications. In digital signatures, the signing algorithm is applied to the hash value instead of the message increasing the performance (by processing less data) and security (by detecting forgery/tampering). Without doubt the security of the scheme depends on the security of the hash function: if the attacker is able to produce two messages with the same hash, and convince a party to sign one of them, then he will have a valid signature for the other message.

Another widespread application of cryptographic hash functions is password protection. For security purposes, instead of the users' passwords their hash values are stored in the database. When a user wants to logon to the system, the hash of his password is computed and it is compared with the correct digest in the database. The security of the system of this scheme – in case of exposure of the database – also depends on the security of the hash function. If the attacker is able to find the password from its hash value, then the scheme fails.

In message authentication, before sending the data its digest is computed and sent over a secure channel to the recipient, which enables the verification of the integrity of data sent over an insecure communication channel. HMAC [16] uses a hash function and a secret key for this purpose however if the adversary can modify the message in a way that the hash value does not change then he can perform an attack. Hash functions are also used in key derivation (deriving keys from a secret information) or random number generation as building blocks. Moreover, hash functions can be used for non-cryptographic purposes. For instance, in hash tables to locate the data quickly or in check-sums to detect accidental changes in data and identical records in a database.

In this thesis we focus only on cryptographic hash functions. As explained above, in order to achieve the desired security [123] there are certain properties that a hash function should satisfy. Any weakness in the hash function that causes the security goal to fail, results in an attack.

2.1 Security Requirements

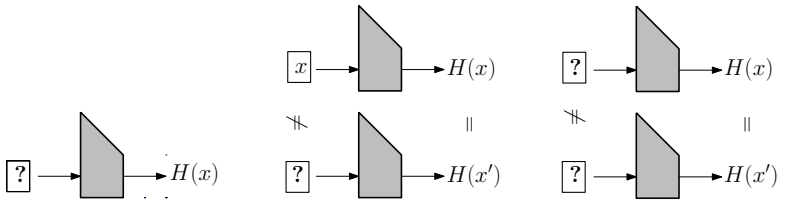
In the design of hash functions, security and performance have a great importance. Some desired properties of cryptographic hash functions that

should be satisfied and the trade-off between them are given below:

1. **Preimage Resistance:** A hash function H is preimage resistant if it is computationally infeasible to find a message that produces a given hash value. In other words, given any $y \in R$, it should be 'hard' to find a message $x \in D$ such that $H(x) = y$. A function that is preimage resistant is known as a *one-way function*.

2. **Second Preimage Resistance** A hash function H is second preimage resistant if for as any given message, it is computationally infeasible to find another message with the same message digest. Equivalently, given $x \in D$ and its hash value $H(x)$, it should be 'hard' to find $x' \in D$ such that $x \neq x$ and $H(x) = H(x')$.

3. **Collision Resistance** A hash function H is collision resistant if it is computationally infeasible to find two distinct messages which have the same hash value. That is, it should be 'hard' to find $x \neq x' \in D$ such that $H(x) = H(x')$.



(a) Preimage **(b) Second Preimage** **(c) Collision**

Figure 2.2: Security requirements of hash functions

In terms of the desired security, these properties are often related to the length of the message digest. The standard security requirements for a hash function with an n-bit output size are collision resistance of $2^{n/2}$ as well as preimage and second-preimage resistance of 2^n hash function computations (see Section 2.4.1).

These security requirements are also related with each other. It has been shown that collision resistance implies second preimage resistance, and under certain conditions collision resistance also implies preimage resistance [191]. These standard security notions have been later formalized into seven: Pre, ePre, aPre, Sec, eSec, aSec, Coll and the relations among them are given within the provable-security frame-work [175, 4].

There are also weaker versions of collision: *near-collision* in which two messages collide only in some parts of their hash values, *semi-free-start collision* where the attacker uses another initial value (IV) instead of the one specified or *free-start-collision* (also called as *pseudo-collision*) when there is a difference in the initial value. A schematic description of collision attacks for hash functions is given in Figure 2.3.

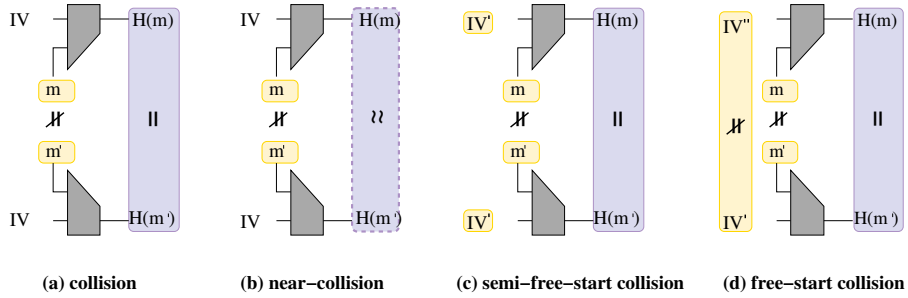


Figure 2.3: Collision attacks for hash functions

Another key concept in security definitions is the notion of *indistinguishability* [140]. Informally, it can be described as a game where the adversary's goal is to distinguish a cryptographic primitive (i.e. a compression function or a blockcipher) from an ideal one (i.e. a random function or a permutation). Here the adversary is formalised as an efficient algorithm either in the information- or complexity-theoretic setting where the former cares only the number of queries made by the adversary whereas for the latter the actual time it takes to achieve her goal is the crucial parameter. The adversary wins the game as long as she achieves her goal. How well the adversary performs is usually measured by its advantage which is defined as a positive real number between zero and one; the closer the advantage is to one, the better the adversary performs.

A related notion, which is more common in the cryptographic hash function context, is the so called *indifferentiability* [133]. While indistinguishability interacts with monolithic primitives (i.e. a random oracle), indifferentiability makes a step forward and concerns more the interaction of atomic primitives (i.e. a compression function) which makes it more suitable for the security analysis of hash functions. Indeed, most of the recent hash functions use either compression functions or internal permutations as building blocks in their design. The absence of a security analysis even when these building blocks are idealised not only impacts the security claims of the corresponding hash function but it also helps to point out the potential flaws in the design.

2.2 Iterated Hash Functions

It is not easy to design a function that can process arbitrary and huge amounts of data. The natural approach is to use some kind of iterative method that processes data in chunks. The main idea, which dates back to Rabin [167], is using a compression function with fixed size input that maps long fixed sized inputs to shorter outputs.

In this type of hash functions, every message block is processed in a similar way. The input is first padded such that the length of the input is a multiple of the block length. Then it is divided into t blocks such that $M||pad = m_1||m_2||\dots||m_t$. Finally, the hash result is computed as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(m_i, H_{i-1}) \\ H(x) &= g(H_t) \end{aligned}$$

Here, the function $f : \{0,1\}^v \to \{0,1\}^w$ is called the *compression function*, and the function $g : \{0,1\}^w \to \{0,1\}^n$ is called the *output transformation* where $v \geq w$. For most of the constructions, g is the identity function. IV denotes the *Initial Value*, and H_i is the *chaining value*. This process is illustrated in Figure 2.4.

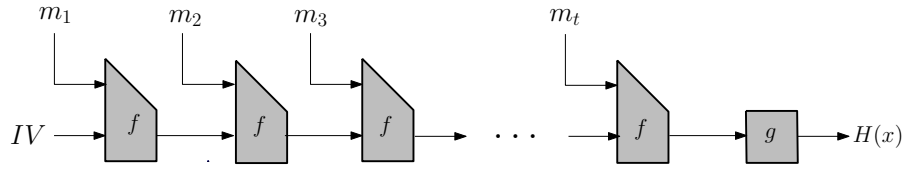


Figure 2.4: An iterated hash function

2.2.1 Merkle-Damgård Construction

It was proven by Merkle [148] and Damgård [58] independently that, if the padding contains the length of the message, known as *MD strengthening*, the resulting hash function is collision resistant if the compression function is collision resistant. However, this construction has still a number of weaknesses against length-extension attack, multicollision attack [101], second preimage

attacks [110], herding attacks [108] and it is possible to distinguish them from random oracle (see Section 2.4.2).

In order to eliminate the flaws described above, there have been many proposals for alternative hash function constructions using smaller building blocks as well as extensions to the Merkle-Damgård design. These approaches can be summarized as follows:

Wide Pipe

This approach is proposed by Lucks [133] is very similar to the Merkle-Damgård design except that the the compression function has a wider state size than the hash size and the final transformation reduces the state size to the hash size. If the composition $g \circ f$ is collision resistant, this modification to MD construction prevents some of the attacks such as multicollision, length-extension or second-preimage. Most of the new designs [18, 82, 205, 84] use this approach.

Randomized Hashing

In this approach proposed by Halevi and Krawczyk [88], the classical MD construction is used without any structural changes. They introduce two ways of randomizing the message: xoring each message block m_i with a random block r, called as *salt*, before entering to the compression function (i.e., $H_{i+1} = f(H_i||m_i \oplus r)$) or prepending a random block r to the message while still xoring the same random block r with each message block. In [81], Knudsen and Gauravaram [81] showed that a second preimage can be found for this construction by using Dean's method [66, 110] if there exists fixed points for the compression function.

HAIFA

This approach by Biham and Dunkelman [26] suggests that two more parameters should be appended to the input of the compression function f: the number of bits hashed so far and salt (i.e., $H_{i+1} = f(H_i||M_i||\#bits||salt)$. This modification increases the security against (second) preimage attacks, prevents the use of fixed points (i.e., $h = f(h, m)$) and multicollision attacks.

2.2.2 Tree Construction

This construction was proposed by Merkle [147] for the authentication of digital signatures. In this type of construction the data blocks are called *leaves* and are processed recursively to form the binary *tree* as follows:

$$\begin{aligned} H(i,i,Y) &= F(Y_i) \\ H(i,j,Y) &= F(H(i,(i+j-1)/2,Y)||H(i,(i+j+1)/2,Y)) \end{aligned}$$

where F is a one-way function, $0 \leq i,j \leq n$ and Y is a vector of data blocks $Y = Y_1, Y_2, \ldots, Y_n$. If the input to F is less than 100 bits, then one can pad it by adding zeros until it is exactly 100 bits. A binary tree (each node has two child nodes) for $n = 4$ is given in Figure 2.5.

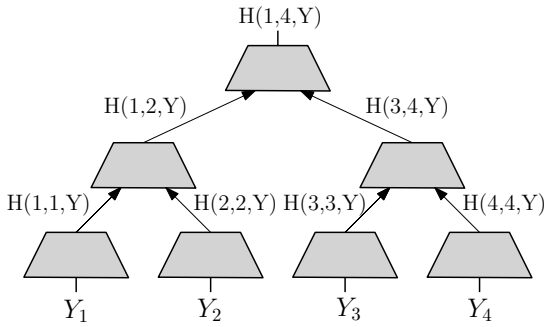


Figure 2.5: Binary tree construction for $n = 4$

In order to authenticate a leaf Y_i, one requires only the values of H from that leaf towards the *root* $H(1,n,Y)$. This can be seen as an advantage especially in peer-to-peer networks since it is efficient and only the damaged or altered blocks need to be redownloaded.

Although most tree constructions are binary, it is possible to have other configurations. For example the SHA-3 candidate MD6 [172] uses a 4-ary tree for hashing. Other examples of tree construction and can be found in [20, 74].

2.2.3 Sponge Construction

A sponge [55] is a cryptographic model defined by Bertoni et al. for iterated hash functions and stream ciphers. The sponge construction is a simple iterated

design that takes a variable-length input and can produce an output of an arbitrary length based on a permutation π_b operating on a state of a fixed number b of bits. The size of the internal state $b = r + c \geq n$ is called *width*, where r is the *rate* and c the *capacity*.

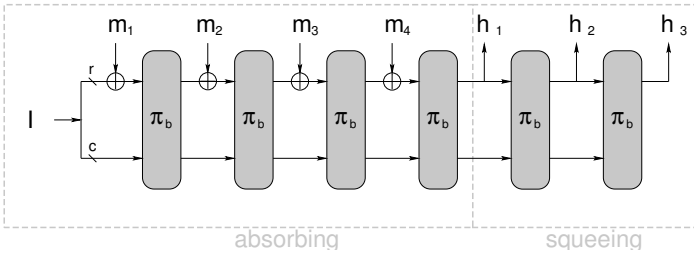


Figure 2.6: Sponge construction based on a b-bit permutation π_b with capacity c bits and rate r bits. m_i are r-bit message blocks. h_i are parts of the hash value

The sponge construction proceeds in three phases (see also Figure 2.6):

- **Initialization phase:** The message is padded to a multiple of r bits. Then it is divided into t blocks of r bits.

$$M||pad = m_1||m_2||\ldots||m_t$$

- **Absorbing phase:** the r-bit input message blocks are xored into the first r bits of the state, interleaved with applications of the permutation π_b.

$$\begin{aligned} H_0 &= IV \\ H_i &= \pi_b(H_{i-1} \oplus (0^c||m_i)) \end{aligned}$$

- **Squeezing phase:** the first r bits of the state are returned as output, interleaved with applications of the permutation π_b, until n bits are returned.

$$\begin{aligned} H_i &= \pi_b(H_{i-1}) \\ h_i &= trunc_r(H_{i+t}) \\ H(x) &= h_1||h_2||\ldots||h_{n/r}\,. \end{aligned}$$

A sponge function is called a hermetic sponge if the underlying permutation has no properties that are exploitable in attacks (called structural distinguishers).

It was shown in [12, 22, 23, 55] that for $n \geq c$ and reasonably small r, the preimage and second-preimage resistances are reduced to 2^{n-r} and $2^{c/2}$, correspondingly, while the collision resistance remains at the level of $2^{c/2}$ hash function computations.

2.3 Construction Methods

Up to now many different construction methods have been proposed to obtain a collision resistant hash function and new construction methods are also being developed.

2.3.1 Hash Functions Based on Mathematical Problems

In this method, the security of hash function is based on a known, well-studied mathematical problem which is considered to be difficult (i.e., unsolvable in polynomial time). Therefore, it is expected that breaking the hash function (for example finding collisions) is at least as hard as breaking the underlying problem. In comparison to classical hash functions, this type of construction tends to be relatively slow and inefficient to use in practice.

Classical examples of these problems and some hash functions that use them are integer factorization, discrete logarithm problem (muHASH [17]), finding modular square roots (VSH [48]), etc.. There exist also hash functions based on algebraic structures such as *Modular Arithmetic* (MASH [164]), *Fourier Transform* (SWIFFT [134]), *Knapsack* [58], *Lattice* and *Coding Theory* problems.

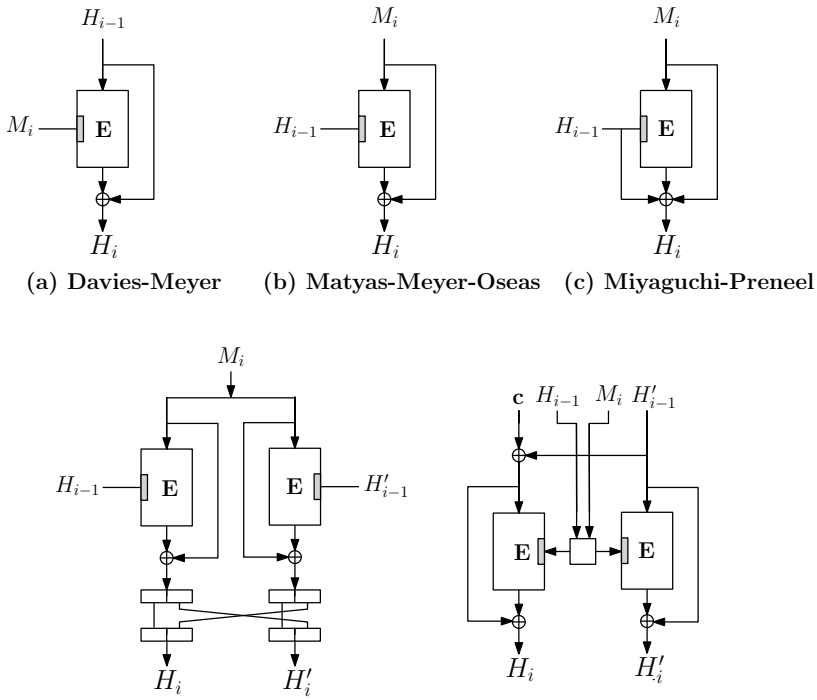
2.3.2 Hash Functions Based on Block Ciphers

The traditional approach in the design of hash functions is to use a known trusted block cipher in a special mode to construct the compression function, and hence turning it into a hash function. The security of a hash function constructed with this method is closely related to the security of the underlying block cipher. Not only since the security of block ciphers is well studied and their weaknesses are exploited, but also because they have good implementations both in hardware and software, such hash functions have been

popular. All well-known hash functions, including MD4 [171], MD5 [170], SHA-1 and SHA-2 are of this type.

Although there are many possibilities, most designers concentrate on the following constructions: *single-block length compression functions* where the hash size equals to the block length of the ciphers and *double-block length compression functions* where the hash size is twice the block length of the cipher.

Preneel et al. [165] considered all possible configurations of a single block length compression function and Black et al. [32] proved that 12 of these configurations are secure. For single-block length hashing, the three most known modes are Davies-Meyer [59], Matyas-Meyer-Oseas [139] and Miyaguchi-Preneel [151, 165] (see Figure 2.7). For double-block length hashing, MDC-2 (Manipulation Detection Code), MDC-4 [40] and Hirose [94] can be given as popular modes.



(a) Davies-Meyer **(b) Matyas-Meyer-Oseas** **(c) Miyaguchi-Preneel**

(d) MDC-2 **(e) Hirose**

Figure 2.7: Some block cipher based compression function constructions

2.3.3 Hash Functions Based on Permutations

One drawback of the hash functions based on block ciphers is that they require a strong key schedule and it needs to be executed for every encryption, which increases the computational cost. Recently, hash functions based on permutations (which can be considered as fixed-key block ciphers) have been introduced to avoid this problem. In this approach, the security of the hash function no longer depends on a secure compression function but on a trusted fixed-length permutation on a large state.

One of the first examples of this construction is Snefru [149] proposed by Merkle. Sponge functions are the most prominent construction in this category. The SHA-3 finalist Keccak [21] and light-weight hash functions such as QUARK [12], PHOTON [86] and SPONGENT [35] use the sponge construction in their design. There are also sponge-like constructions such as Grindahl [119], SHA-3 second round candidate Luffa [65] and the finalist JH [205].

There are also other constructions based on ideal permutations that are proven to be secure. In [177, 176] permutation-based hashing and the construction of $2n$-to-n-bit compression functions using three distinct permutations with finite field scalar multiplications and Davies-Meyer-like compression functions using permutations is studied. Recently, this work has been generalized for the entire family of $2n$-to-n-bit compression functions using only three permutations and of XOR-operators in [146].

2.3.4 Hash Functions Based On Stream Ciphers

There are also hash functions based on stream ciphers. Since stream ciphers are usually faster than block ciphers and efficient in hardware, they are suitable for applications where high speed is required. Yet there are only a few hash functions based on stream ciphers and it is hard to analyze them and give any mathematical or information-theoretical proof for their security.

The first such design Panama [54] was broken by Rijmen et al. [169] and then by one of its designers [53]. Later, it has been strengthened and a variant Radiogatun [19] that does not have the known weaknesses of Panama is proposed. There is also RC4-hash [45] based on the well know stream cipher RC4 which was also broken [99]. Finally, some of the SHA-3 candidates such as LUX [159], SHAMATA [9] and Shabal [42] also use this approach but none of them has been selected for the final round.

2.4 Attacks on Hash Functions

An attack on a cryptographic hash function is an algorithm that targets the hash function to find a weakness in one of the security requirements. Attacks on hash functions can be divided into two groups: A first group consists of generic attacks that are independent of the hash algorithm, whereas the second type of attacks focus on structural weaknesses and exploit the weaknesses of the algorithm.

2.4.1 Generic Attacks

A generic attack considers the hash function as a "black box" and hence is independent of the details of the algorithm. The complexity of the attack (i.e., the amount of calculations that needs to be done to mount an attack) depends only on the parameters of the hash function. Since it is not possible to prevent this type of attacks, the designers choose the parameters of a hash function in a way that the attack will be infeasible. In this section, we describe some generic attack methods in detail.

Exhaustive Search

Exhaustive search is the simplest and the most straightforward way of finding (second) preimages. For preimages, the attacker randomly chooses an input in hope of finding the given hash result. The probability of finding a match is equal to $1/|R|$ where $|R|$ is the size of the output space. This means that for an n-bit hash the success probability is 2^{-n} assuming that each output value occurs approximately equally likely.

The same attack also applies for finding second preimages. The only difference is that the adversary knows in advance another input whose hash is the given value. For long messages, it has been shown that obtaining a second preimage is easier unless the length of the message is included in the padding [163]. When the padding includes the length of the message, second primages can be still found with a complexity of $k \cdot 2^{n/2+1} + 2^{n-k+1}$ for a 2^k-message-block message.

The complexity of the attack is decreased if multiple targets are aimed for. Consider that one aims to find (second) preimages for 2^t simultaneous targets, then the probability to find a match becomes 2^{-n+t} and after 2^{n-t} trials one expects to find a (second) preimage.

Time-Memory Trade-Off

Instead of making an exhaustive search over all possible messages, it is possible to reduce the time complexity of an attack by precomputing and storing messages and the corresponding hash values in a table. On the other hand, the precomputation phase comes with an extra cost in terms of storage space that increases memory complexity.

The time-memory trade-off attack was first introduced by Hellman [92] in 1980. Using this method, it is possible to find (second) preimages. The main idea is to precompute long hash chains: The attacker chooses m different starting points and iterates it for t steps, and stores the starting and ending points of each chain in memory. To find a preimage in the online phase, the same iteration is performed until a known ending point is found. Unfortunately, it is possible that the chains with different starting points collide and merge reducing the coverage. The success probability can be increased by generating multiple tables using different reduction functions.

This technique was later improved by Rivest [68] by using distinguished points (points for which some property holds) as endpoints for the chains. This significantly reduces the number of table look-ups during the analysis, but the analysis becomes more complex due to the variable chain length.

Birthday Attack

The birthday attack is based on the birthday paradox that states in a group of randomly chosen people, the probability that at least one pair of them will have the same birthday is high. The generalized birthday problem aims to find the minimum size of the group N such that there is at least M people that have the same birthday with probability 50%.

From a cryptographic point of view, this is equivalent to finding collisions for a hash function. The attacker selects a set of r random messages, stores their hash values in a table and hopes that at least two of them will have the same hash value. The probability of a collision for an n-bit hash can be approximated as follows:

$$p \approx 1 - \exp(-\frac{r^2}{2|R|}) = 1 - \exp(-\frac{r^2}{2^{n+1}}) \,.$$

The expected number of trials until a collision is found is given by $\sqrt{\pi/2} \cdot 2^{n/2}$ [194, Appendix A]. Therefore, the time complexity of the birthday attack, omitting the constant, is $2^{n/2}$ evaluations of the hash function. The memory complexity is determined by the size of the table, which is also $2^{n/2}$.

A variant of birthday attack [207] uses two sets of distinct messages to obtain meaningful collisions. The attacker selects r_1 messages, and modifies them to obtain the resulting r_2 messages. If $r = r_1 = r_2 = 2^{n/2}$ the computational complexity of this attack becomes optimal.

Memoryless Birthday Attack

The traditional birthday attack requires storing $2^{n/2}$ messages and their hashes, however it is possible to reduce the large memory requirements significantly by using a cycle finding algorithm.

The main idea of the attack is based on the following fact: When a function f is applied iteratively (i.e., $x_{i+1} = f(x_i)$) starting from a random value x_0, the resulting sequence has to repeat since the output set is finite (in this case $\{0,1\}^n$). The graph of this sequence looks like the Greek letter ρ for a noninvertible f and is composed of a *tail* and a *cycle*. Hence, a collision for f is found at the entrance of the cycle.

The most known cycle finding algorithm is due to Floyd [120]: it uses two sequences, applying f once and twice per step respectively starting from x_0, and compares the outputs of each step. When the two outputs match, say $x_i = x_{2i}$, a collision is found but neither the input values resulting the collision nor the entrance point of the cycle is known. This problem can be solved by iterating once more starting from x_0 and x_i but applying f just once for both sequences.

The expected length of the tail and the largest cycle are both given as $\sqrt{\pi/8} \cdot 2^{n/2}$ [80] and the time complexity of this attack roughly three times more than the standard birthday attack with negligible memory requirements. This collision search was later parallelized by van in Oorschot and Wiener [194] using distinguished points.

Meet-in-the-Middle Attacks

This attack applies to hash functions for which the compression function f is easy to invert. It is a variant of the birthday attack that allows finding a preimage, second preimage or collision for intermediate chaining values instead of the hash results. The attacker chooses r_1 initial values, computes the corresponding values at some intermediate step in the forward direction and stores them in a list L. Similarly, he chooses r_2 hash values, computes them backwards and compares them with the values in L.

The probability of matching the intermediate value from two different sets is given by:

$$p \approx 1 - \exp\left(-\frac{r_1 \cdot r_2}{2^n}\right).$$

where n is the length of the hash and chaining values. The time complexity of the attack is r_1 calls to the function f_1 and r_2 calls to the function f_2 hence it attains the minimum value when $r_1 = r_2 = 2^{n/2}$ assuming that computing in both directions takes the same amount of time. The memory complexity is dominated by the size of the list L which is also $2^{n/2}$.

The memory complexity of the meet-in-the-middle attack can also be decreased by using cycles and storing only the distinguished points as in the birthday attack [166].

2.4.2 Dedicated Attacks

This class of attacks depends on some high level properties of the compression function f or the construction method. They are also known as chaining attacks.

Fixed Point Attack

A *fixed point* is a pair (m_i, h_i) such that $h_{i+1} = f(m_i, h_i) = h_i$. This means that the message value m_i does not affect the chaining value h_{i+1} and hence the hash value. Therefore, it is possible to construct second preimages by removing m_i. To be more precise, given the message $m = m_1 m_2 \dots m_t$, if the attacker finds a fixed point for the i-th iteration then $m' = m_1 m_2 \dots m_{i-1} m_{i+1} \dots m_t$ has the same hash value.

This attack is applicable to hash functions that have a compression function. Fixed points can be easily found for most of the compression functions. For example, for Davies-Meyer construction even if the underlying block cipher is totally secure, the fixed points can be computed as follows: the attacker sets $h_i = E_{m_i}^{-1}(0)$ then $h_{i+1} = E_{m_i}(h_i) \oplus h_i = h_i$ and hence (m_i, h_i) is a fixed point.

Length-Extension Attack

The aim in length-extension attack is to compute $H(m||m')$ given the hash value $H(m)$ but not the message m itself. A trivial attack for Merkle-Damgård construction if there is no output transformation can be given as follows: The last chaining value equals to the hash value $H(m)$, hence the attacker is able to

directly append message blocks and resume the computation without knowing the message m.

When MD-strengthening is used, the attacker is no longer able to directly apply the attack described above, but it is still possible to perform a length-extension attack. Provided that the attacker knows the length of message m, he can choose m' such that the padding for $m||m'$ is encoded in the last l bits of m'.

Multicollision Attack

A multicollision is a set of messages that all hash to the same value. Joux [101] showed that finding multicollisions for an iterated hash function is not much harder than finding a single collision. The main idea is concatenating a chain of internal collisions. The attacker computes the colliding pairs $\{m_i, m_i'\}$ such that $f(h_{i-1}, m_i) = f(h_{i-1}, m_i')$ for t steps. Then it is possible to construct 2^t messages $x_1||\ldots||x_t$ where $x_i \in \{m_i, m_i'\}$ that hash to the same value. A schematic representation is given in Figure 2.8.

Using Joux's method, it is possible to obtain 2^t different multicollisions for an iterated hash function with a computational complexity of $t \times 2^{n/2}$. On the other hand, for an ideal hash function with an n-bit digest, it was believed that $2^{n(t-1)/t}$ calls to f are required to find a t-fold multicollison. However, Suzuki et al. [192] showed that by that many calls, the probability of finding a t-fold multicollision is only $1/t!$ which is very small for large t, and the complexity should be increased by a factor of $(t!)^{1/t}$ to achieve a success probability of at least 50%.

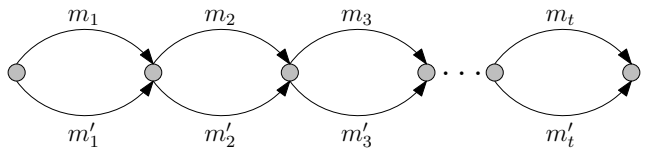


Figure 2.8: Schematic representation of multicollision attack

Herding Attack

The herding attack [108], also known as the Chosen Target Forced Prefix (CFTP) attack, allows the attacker to find a preimage for a yet unknown hash value for MD construction with a lower complexity than the preimage attack. It is composed of two phases:

In the offline phase, the attacker chooses 2^k chaining values and randomly generates message blocks to compute the new chaining values among which he searches for collisions between the pairs. The procedure is repeated iteratively with the new chaining values and the message blocks until one chaining variable, which can be used as a hash value, is left.

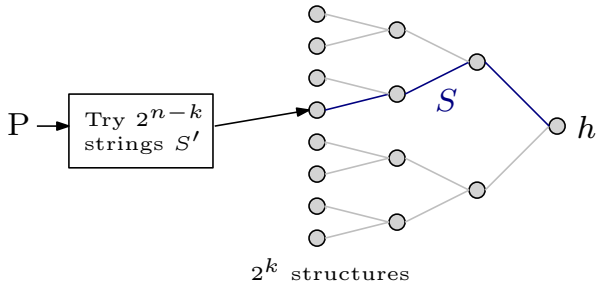


Figure 2.9: Finding a linking message and producing the suffix

As a result, the attacker finds many collisions by using brute force and forms a *diamond structure*, which is a data structure that resembles a tree. In the online phase, a prefix P is given to the attacker and he exhaustively searches for a string S' such that $P||S'$ matches with one of the intermediate states of this structure. Finally, the suffix S such that $H(P||S'||S) = h$ can be computed from the diamond structure.

The multicollisions obtained by this method are the same length but more expensive than that of Joux's. The complexity of creating a diamond structure with 2^k hash values at its widest point is at most $2^{k/2+n/2+2}$ calls to the compression function f. The complexity of this attack is improved in [33] and several extensions to this idea are presented in [2, 3].

Chapter 3

Cryptanalysis Methods

Cryptanalysis evaluates the security of the cryptographic algorithms that are used to protect the information and tries to "break" them. Depending on the purpose of the algorithm, breaking can have different meanings in symmetric-key cryptography. For example for block ciphers and stream ciphers, which are used primarily for encryption, the goals include recovering the secret key or obtaining plaintext information. Whereas for hash functions finding a colliding pair or recovering (a part of) the message from its digest is the aim.

An efficient algorithm that exploits the weakness of a cryptographic scheme or one of its underlying primitives such that the required security goals are not met is called an *attack*. The most strong scenario is the *ciphertext only attack* in which the attacker uses only the ciphertexts to perform cryptanalysis. There are also weaker scenarios such as the *chosen ciphertext (plaintext) attack* in which the attacker chooses the ciphertexts (plaintexts), alternatively relations between them, or the *known plaintext attack* in which the attacker knows a few ciphertexts and the corresponding plaintexts.

While some cryptographic schemes have formal security proofs i.e., their security can be reduced to a hard mathematical problem, for most of the time it is not the case. Therefore during the design process, the parameters of the algorithm are chosen such that the complexity of the generic attacks (see Section 2.4.1) is beyond the available computing power. Moreover the designers evaluate the resistance of their algorithms against the existing attack methodologies and try to develop new design strategies such that the attacks do not apply.

For cryptanalysts, it is common to consider the underlying primitives as well as reduced round or weakened variants of cryptographic algorithms. This usually gives an idea on the security margin of the full algorithm when it cannot be broken. Nevertheless the absence of an attack on the full scheme does not guarantee that it is secure.

In this chapter we provide a brief introduction to the cryptanalytic methods used for hash functions. We focus on differential cryptanalysis since most of the attacks presented in the following chapters are based on it.

3.1 Differential Cryptanalysis

Differential cryptanalysis was introduced by Biham and Shamir [28] and is one of the most powerful techniques used in the analysis of block ciphers, hash functions, stream ciphers, etc. It is based on examining input/output differences and the relation (probability) between them. For most of the cases this difference is an XOR difference, although it is possible to use modulo 2^n addition, arbitrary group operations [122] or UNAF (unsigned non-adjacent form) differences [196].

In the course of time, differential cryptanalysis has pioneered many other cryptanalysis methods: higher-order differential cryptanalysis [121], truncated differential cryptanalysis [117], impossible differential cryptanalysis [116, 25] and boomerang attacks [198] can be given among the examples of these methods.

3.1.1 Definitions

A *differential* [122] over a map f is denoted by $(\Delta a, \Delta b)$ where Δa is the input difference and Δb is the output difference.

Definition 2. *The* differential probability $\mathrm{DP}(\Delta a, \Delta b)$ *of a differential over a map f is the fraction of pairs with input difference Δa that have output difference Δb. For an n-bit map, this can be shown as:*

$$\mathrm{DP}(\Delta a, \Delta b) = \frac{1}{2^n} \#\{x \in \mathbb{F}_2^n | f(x \oplus \Delta a) = f(x) \oplus \Delta b\,.\}$$

For a keyed map, the differential probability $\mathrm{DP}[k](\Delta a, \Delta b)$ can be defined for each value k of the key. Then, the *expected differential probability* (EDP) is the

average of the differential probability over all keys.

$$\mathrm{EDP}(\Delta a, \Delta b) = \frac{1}{|\kappa|} \sum_{k \in \kappa} \mathrm{DP}[k](\Delta a, \Delta b)\,.$$

Here, k is assumed to be uniformly distributed and taking values in κ.

In practice, it is difficult to compute the differential probabilities for functions $f : \{0,1\}^n \to \{0,1\}^n$ when n is large. To our advantage, these functions are usually designed in a way that they can be decomposed into sub-functions. Let f consist of a sequence of sub-maps, then $f = f_m \circ f_{m-1} \circ \ldots \circ f_0$ and f is called a *composed map*.

Definition 3. *A* differential characteristic Q *over a composed map is a sequence of differences through various stages of the encryption. The sequence consists of an input difference* Δ_0*, followed by the output differences of all the steps* $(\Delta_1, \Delta_2, \cdots, \Delta_m)$*.*

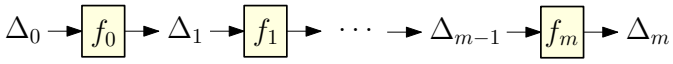


Figure 3.1: A characteristic over a composed map

A (differential) characteristic is often called a *differential path* or a *differential trail*.

A pair is called a *right pair* if it satisfies the given differential characteristic. This is equivalent to satisfying the following set of equations:

$$\begin{aligned} f_0(x \oplus \Delta_0) &= f_0(x) \oplus \Delta_1 \\ (f_1 \circ f_0)(x \oplus \Delta_0) &= (f_1 \circ f_0)(x) \oplus \Delta_2 \\ &\cdots \\ (f_m \circ f_{m-1} \circ \ldots \circ f_0)(x \oplus \Delta_0) &= (f_m \circ f_{m-1} \circ \ldots \circ f_0)(x) \oplus \Delta_m \end{aligned} \tag{3.1}$$

Consequently, the differential probability $\mathrm{DP}(Q)$ of a characteristic over a composed map f is the fraction of its right pairs. Similarly, for an n-bit map, this is formulated as:

$$\mathrm{DP}(Q) = \frac{1}{2^n} \cdot \#\{x \in \mathbb{F}_2^n | x \text{ satisfies (3.1) } .\}$$

A characteristic $(\Delta_1, \Delta_2, \cdots, \Delta_m)$ is said to be *in* differential $(\Delta a, \Delta b)$ if $\Delta a = \Delta_1$ and $\Delta b = \Delta_m$. Then, it is possible to define a relation between the differential probabilities $\mathrm{DP}(\Delta a, \Delta b)$ and $\mathrm{DP}(Q)$:

$$\mathrm{DP}(\Delta a, \Delta b) = \sum_{Q \in (\Delta a, \Delta b)} \mathrm{DP}(Q)\,.$$

It is known that, the same relation holds between $\mathrm{DP}[k](\Delta a, \Delta b)$ and $DP[k](Q)$ for Markov ciphers (i.e., iterated ciphers whose round functions are independent) [122].

3.1.2 Towards Attacks

In differential cryptanalysis the attacker tries to find the best differential characteristic to mount an attack. Since the attack complexity is determined by the inverse of the probability of a differential trail, in this context 'best' means having the maximum differential probability,

The *maximum differential probability* (MDP) is the maximum value of the differential probabilities over all pairs of non-zero input and output differences:

$$\mathrm{MDP}(f) = \max_{\Delta a \neq 0, \Delta b} \mathrm{DP}(f)(\Delta a, \Delta b)$$

For composed maps *the maximum differential characteristic probability* (MDCP) is defined as the product of the maximum differential probabilities of sub-maps f_i for all i.

$$\mathrm{MDCP}(f) = \prod_{i=1}^{m} \mathrm{MDP}(f_i)$$

Since this equation is based on the assumption of independent sub-functions, it gives an approximation of the maximum differential characteristic probability.

For SPN (Substitution Permutation Network) based constructions, substitution layer consists of several parallel S-box instances and hence S-boxes are the smallest building blocks. An S-box is called *active* if it has a non-zero input difference, otherwise it is called *passive*. For certain designs (for instance, the wide trail design strategy [56]), the concept of counting active S-boxes is central to the differential cryptanalysis and one can use the minimum number of active S-boxes to estimate the maximum differential characteristic probability as follows:

$$\mathrm{MDCP}(f) = \mathrm{MDP}(S)^{\#\text{active S-boxes}}\,.$$

Note that this relation does not take the actual values of the input and output of the active S-boxes into consideration, hence it is not exact for hash functions

and most block ciphers. Yet, counting differentially active S-boxes is still the major technique used to evaluate the security of cryptographic functions.

For ARX (Addition Rotation Xor) based constructions the analysis should be performed at bit-level and finding paths is difficult, consequently more complicated techniques are required. The first analysis of the probability distribution of the carry bits in integer addition was done by Meier and Staffelbach [188]. Later, Lipmaa et al. proposed algorithms for computing differential properties (including probability) of addition modulo 2^n [131]. Its dual, the differential probability of xor when differences are expressed using addition modulo 2^n is studied in [132] and new algorithms based on matrix multiplications are proposed.

Once a differential characteristic is constructed, the attacker chooses random input pairs and checks if the characteristic is satisfied. For hash functions, the lack of a secret key addition gives the attacker more power; there is no need for searching the correct key, the results can be checked at intermediate steps and message modification can be used. Moreover, the attack complexity can be decreased by using truncated differentials where only a part of the difference is known instead of the whole path.

3.1.3 Application to Hash Functions

The aim of the differential attack on hash functions is mostly finding collisions. For example, for hash functions based on block ciphers where there is a feed-forward of the plaintext, the output difference should be equal to the input difference in order to have a collision.

The important collision attacks on well-known hash functions are all applications of the differential attack described above. The first attack is on MD5 [67] by den Boer and Bosselaers, followed by the work of Dobbertin on MD4 [72] while the first results on SHA-0 [44] are presented by Chabaud and Joux. In 2005, Wang published results first on MD4 and RIPEMD [199], and then on MD5 [201], SHA-0 [202] and SHA-1 [200] by using message modification techniques.

Another important application of a differential attack is finding distinguishers for hash functions, their compression functions or even for the construction. By injecting a difference at the input and observing its propagation at the output (or after some number of rounds), the attacker might be able to distinguish the function from a random mapping; this paradigm works subject to the condition that the probability of having the expected output difference is higher than the probability of having the same output difference in the random function. In

other words, if the output difference follows a desired pattern with sufficiently high probability, then the attacker is able to say that the hash values are generated by the given function.

3.1.4 The Rebound Attack

The rebound attack [142], introduced by Mendel et al., is a relatively new tool for cryptanalysis of hash functions based on differential cryptanalysis. It can be considered as a combination of the *meet-in-the-middle attack* and the *inside-out* approach [198] using truncated differentials.

Let E be the the underlying block cipher or the permutation of the hash function under consideration, then in the rebound attack E is considered as a combination of three sub-ciphers $E = E_{bck} \circ E_{inb} \circ E_{fwd}$ (Figure 3.2).

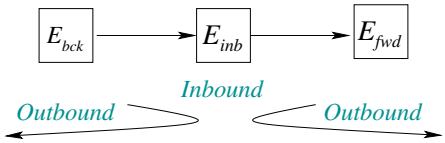


Figure 3.2: The Rebound Attack

The rebound attack is performed in two phases:

- **Inbound Phase:** The aim of the this phase is to find solutions for the middle rounds (E_{inb}) that have a low probability. The differential path is divided into two or more subcharacteristics and solutions for each of them is calculated separately. Finally, these solutions are combined by using the match-in-the-middle technique.
- **Outbound Phase:** The aim of this phase is to find solutions for the entire differential. Each solution of the inbound phase is computed in both forward (E_{fwd}) and backward (E_{bck}) directions. Then it is checked whether they satify the desired pattern.

The Inbound phase can be repeated to obtain more starting points for the outbound phase which is probabilistic. As a result, many solutions for the truncated differential can be obtained with a lower complexity.

This attack has been used first for the cryptanalysis of reduced versions of Whirlpool and Grøstl, and then extended to obtain distinguishers for the

full Whirlpool compression function [124]. Later, linearized match-in-the-middle and start-from-the-middle techniques have been introduced by Mendel et al. [141] to improve the rebound attack. Moreover, a sparse truncated differential path is recently used in the attack on LANE by Matusiewicz et al. [98] rather than an all active state in the matching part of the attack.

Then, these techniques were used to improve the results on AES-based algorithms (ECHO, Grøstl,) in the following papers: [83, 97, 143, 161, 182, 183]. The rebound attack has also been successfully applied to hash functions based on permutations (Luffa [26], Keccak [18]) and on Feistel structures.

3.1.5 Splice-and-Cut Technique and Bicliques

The Splice-and-cut technique, introduced by Aoki and Sasaki [5], is an improvement to the meet-in-the-middle attack. In this technique, the first and last steps of the target algorithm are considered as consecutive steps. Then the algorithm is divided into two parts called *chunks* in a way that each chunk includes at least one independent message word from the other chunk referred to as *neutral word*. Finally, pseudo-preimages can be found by using the meet-in-the-middle attack.

This method allows the attacker to start the meet-in-the-middle attack from any step. It was first used to attack MD4 and MD5 [5], followed by the preimage attacks [180] on HAVAL. It was later applied to SHA-0 and SHA-1 in [6]. This technique was further improved with the introduction of *initial structure* in [181] and used to attack full MD5.

The initial structure is composed of a few consecutive steps that has at least two neutral words m^1, m^2 such that the second chunk can be computed independently from m^1 and the first chunk can be computed independently from m^2. A sketch of the splice-and-cut technique is given in Figure 3.3.

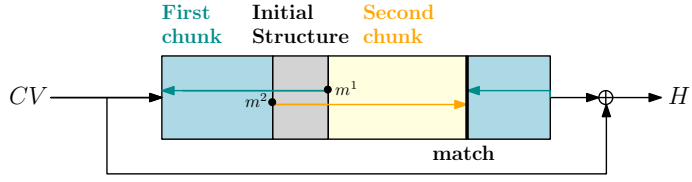


Figure 3.3: Splice-and-cut technique with initial structure

Bicliques [113] can be seen as a special case of initial structure composed of two sets, namely $\{Q_i\}$ and $\{P_j\}$, where each state in a set is related with all

states in the other set after some consecutive steps. A biclique is parametrized with the *dimension* d where 2^d is the size of the sets.

Bicliques are used to search preimages as follows: let $M = \{M[i;j]\}$ be a group of messages and let v be chosen outside of f so that the two chunks g_1 and g_2 are independent of i and j, respectively. Then the attacker checks if

$$P_j \xrightarrow[g_1]{M[*;j]} v \stackrel{?}{=} v \xleftarrow[g_2]{M[i;*]} Q_i$$

is satisfied for some i, j. The adversary computes v from Q_i by first computing CV and then deriving the output of E as $CV \oplus H$. 2^{n-2d} bicliques of dimension d are required to test 2^n preimage candidates. Upto now, this technique has been applied to Skein, the SHA-2 family and the output transformation of Grøstl [113, 111] as well as key recovery attack for block ciphers AES and IDEA [34, 112].

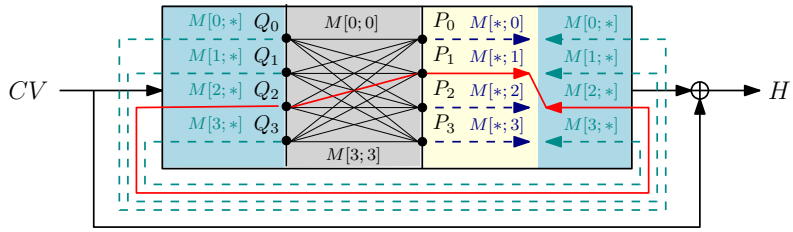


Figure 3.4: Biclique with dimension $d = 2$ in the meet-in-the-middle attack

3.1.6 Boomerang Attack

The boomerang attack [198], introduced by Wagner, aims to reduce the complexity of the differential cryptanalysis. The main idea is to use two short differential characteristics with high probabilities instead of one long characteristic with a lower probability. The attack attempts to generate a structure called *quartet* halfway through the cipher.

For this purpose, the block cipher E is treated as a cascade of two sub-ciphers E_0 and E_1 (i.e, $E = E_1 \circ E_0$). Assume that a differential characteristics $\Delta \to \Delta^*$ with probability p for E_0, and $\nabla^* \to \nabla$ with probability q for E_1 are known. The boomerang attack is based on generating right quartets (P_1, P_2, P_3, P_4)

which satisfy a set of relations:

$$\begin{aligned} P_1 \oplus P_2 &= \Delta = P_3 \oplus P_4 \\ E_0(P_1) \oplus E_0(P_2) &= \Delta^* = E_0(P_3) \oplus E_0(P_4) \\ E_0(P_1) \oplus E_0(P_3) &= \nabla^* = E_0(P_2) \oplus E_0(P_4) \\ C_1 \oplus C_3 &= \nabla = C_2 \oplus C_4 \end{aligned}$$

where $C_i = E_1(E_0(P_i))$. Hence the boomerang attack requires both chosen plaintext and chosen ciphertext and is depicted in Figure 3.5.

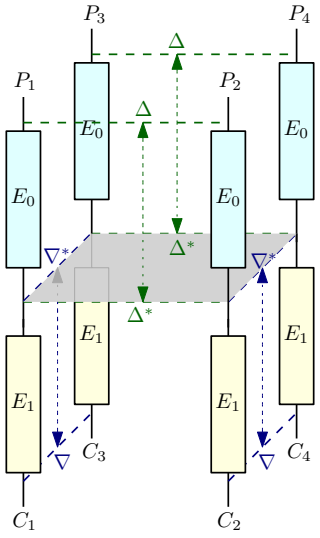


Figure 3.5: The Boomerang attack and a right quartet

The amplified boomerang attack [109] is a chosen plaintext attack in which the same differential conditions have to be satisfied. But instead of generating quartets, a set of random plaintext pairs with input difference Δ is generated. Then the aim is to find quartets satisfying the relations by using random trials. From the birthday paradox we know that quartets exist and we can determine their number.

In [27, 198], it is shown that it is possible to use all possible Δ^*'s and ∇^*'s simultaneously. However, recently in [152], it has been proved that

the probability of the attack given in [198] is inaccurate and the complexity estimates does not hold when the two differential characteristics are dependent.

The boomerang attack has been applied to internal functions and reduced round versions of many hash functions such as MD4, MD5 and HAVAL [114, 179], SHA-1 [104], BLAKE [31], and Skein [206, 10].

3.1.7 Higher-Order Collisions

Whereas the classic differential cryptanalysis studies the propagation of difference between plaintexts, higher-order differential cryptanalysis exploits the propagation of difference between differences. The concept of higher-order derivatives has been first introduced by Lai [121] and applied to differential cryptanalysis by Knudsen [117].

Definition 4 ([121])**.** *Let $(S,+)$ and $(T,+)$ be Abelian groups. For a function $f : S \to T$, the* derivative *at a point $a_1 \in S$ is defined as*

$$\Delta_a f(x) = f(x+a) - f(x)\,.$$

The i-th derivative of f at the point $(a_1, a_2, \ldots, a_i)$ is then defined as

$$\Delta_{a_1,\ldots,a_i} f(x) = \Delta_{a_i}(\Delta_{a_1,\ldots,a_{i-1}} f(x))\,.$$

In the light of this definition, the differential characteristics and differentials introduced in the previous sections actually correspond to first order derivatives. This notion is extended to higher-order differentials as follows:

Definition 5 ([117])**.** *A one-round differential of order i for a function $f : S \to T$ is an $(i+1)$-tuple $(a_1, a_2, \ldots, a_i, b)$ such that*

$$\Delta_{(a_1,\ldots,a_i)} f(x) = b\,.$$

Similar to the regular collisions, a higher-order collision is obtained when $b = 0$. A *second-order differential collision* is a special case of higher-order collisions when $i = 2$.

$$f(x) - f(x+a_2) + f(x+a_1+a_2) - f(x+a_1) = 0\,.$$

Recently, higher-order differential attacks have been applied to SHA-3 candidates Keccak [39] and Luffa [39, 203] and second-order collisions have been found for SHA-256 [30].

3.2 Linear Cryptanalysis

Linear cryptanalysis introduced by Matsui [136] is another powerful technique used in the analysis of cryptographic systems. It is based on examining the relation between linear combinations of plaintext and ciphertext bits.

Definition 6. *Let C be the encryption of P under the key K. A* linear approximation *over a map is defined as:*

$$\Gamma_P \cdot P \oplus \Gamma_C \cdot C = \Gamma_K \cdot K\,. \tag{3.2}$$

where $\cdot$ denotes the standard inner product. The vectors Γ_P, Γ_C and Γ_K represent a fixed linear combination of bits and are called linear masks.

In order to mount an attack, the adversary should find an efficient linear approximation, which means that Equation (3.2) should hold with probability $p \neq 1/2$ (i.e., not random). However, finding the best linear approximation for an arbitrary cipher is generally not an easy task.

As in differential cryptanalysis, the linear approximation of a composed map f can be constructed by combining the linear approximations of the sub-maps. Let $(\Gamma_{X_0}, \Gamma_{X_1}, \ldots, \Gamma_{X_r})$ be a linear characteristic and $X_0, X_1, ..., X_r$ denote the intermediate values where $P = X_0$ and $C = X_r$. We use the following lemma to compute its probability.

Lemma 1 (Piling-up Lemma [136])**.** *Let $\Gamma_i \cdot X_i = \Gamma_{i-1} \cdot X_{i-1}$ be independent linear approximations which are satisfied with a probability $p_i = 1/2 + \epsilon_i$, then the combined probability of the approximation $\Gamma_n \cdot X_n = \Gamma_0 \cdot X_0$ is:*

$$1/2 + 2^{n-1} \prod_{i=1}^{n} (p_i - 1/2) = 1/2 + 2^{n-1} \prod_{i=1} \epsilon_i\,.$$

The ϵ_i values are called the *biases* of the linear approximations. This lemma can be simplied by defining the *correlation* c_i as $c_i = 2\epsilon_i$. Then, we have $c = \prod_{i=1}^{n} c_i$. The square of the correlation is called the *linear probability*. Matsui showed that the number of plaintexts required for an attack is inversely proportional to the linear probability; consequently the best linear approximation is the one with the highest bias.

In linear cryptanalysis, given the encryptions of plaintext-ciphertext pairs, the adversary counts how many times Equation (3.2) is satisfied for each value of the key bits involved in the approximation (refereed as *partial key*) and computes the biases. Then the partial key value that has the closest bias to the expected will be the most probable key.

Kaliski and Robshaw proposed using multiple linear approximations at the same time [105]. Unfortunately, this method required all linear approximations to use the same subkey bits and provided only a small advantage. In [160], Nyberg showed that it is possible to multiple linear characteristics that have the same input and out masks but different (sub)key masks. This concept which can be seen as the analogue of differentials in differential cryptanalysis is called as *linear hull*.

Since its introduction linear cryptanalysis has been successfully applied to many block ciphers and stream ciphers; on the other hand it hasn't received much attention in terms of cryptanalysis of hash functions. As there is no key to recover in hash functions, linear cryptanalysis can only be used to evaluate the security of the function. A few examples include the attacks on Hamsi [129] and Cube Hash [8].

3.3 Algebraic Cryptanalysis

In algebraic cryptanalysis, a cipher is expressed as a system of non-linear equations over $GF(2)$ or $GF(2^n)$ with a large number of unknown variables and a solution for the system is searched. Although it is possible to describe every cipher in terms of multivariate equations, solving them is an NP-hard problem even for quadratic equations (known as the MQ problem [204]).

Using algebraic methods in cryptanalysis has been first formulated by Shannon in 1949: breaking a good cipher should require "as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type" [184]. In 1965, the theory of *Gröbner basis*, a set of multivariate nonlinear polynomials with certain properties for solving algebraic systems, was developed by Buchberger [43]. He also proposed an algorithm, which is a generalization of the Euclidean algorithm and the Gaussian elimination to multivariate polynomial rings, to transform every polynomial system into Gröbner basis form. More efficient algorithms to compute Gröbner bases, namely the algorithms F4 [76] and F5 [77], have been later introduced by Faugère.

In [115], Kipnis and Shamir showed that the complexity of the MQ problem is decreased when it becomes overdefined (i.e., there exists more equations than unknown variables). The main idea is linearizing the system by replacing any product of variables by a new variable and solving the resulting linear system. Their technique called *relinearization* aims to solve systems of quadratic equations in polynomial time. In [50] Courtois et al. showed that many of the equations generated by relinearization are linearly dependent hence this

technique is not as efficient as expected, and they introduced the Extended Linearization (XL) algorithm.

The basic idea of XL algorithm is multiplying each polynomial equation with all possible monomials (of some bounded degree) to generate higher degree variants and then linearizing the new system. The algorithm was later improved in [49]. The actual complexity of the XL algorithm is not known (especially for very big systems of equations) and it offers a small advantage compared to Gröbner basis techniques [7].

In [51], it was claimed that if MQ is sparse and has a regular structure,[1] then there is a more efficient method to find a solution, called eXtended Sparse Linearization (XSL) algorithm. In this algorithm each equation is multiplied by "carefully selected monomials[2]" instead of all possible polynomials. Additionally, there exists a final step (called T' method), in which new linearly independent equations are tried to be obtained without creating any new monomials.

Another approach used in algebraic cryptanalysis is the use of SAT-solvers which are based on heuristic search algorithms. The main idea is guessing some variables and examining the consequences. Whenever a contradiction is found, a new restriction of equation can be added saying that in this set of constraints one is false. For most of the cases, SAT-solvers are much faster and can break more instances than the current Gröbner bases techniques.

3.4 Cube Attack

While solving large systems of multivariate polynomial equations for cryptographic schemes has been studied extensively in algebraic cryptanalysis, it was stated in [70] that these equations are not arbitrary and unrelated. Instead they are derived from a *master polynomial* by fixing the values of some public variables such as message bits and IV bits. The basic cube attack is an algorithm for solving such equations.

In the cube attack the desired algorithm is considered as a black box that evaluates a polynomial p over $GF(2)$ of $n+m$ inputs bits $(x_1, \ldots, x_n; v_1, \ldots, v_m)$ where x_i are the secret variables (key bits) and v_j are the public variables (plaintext bits or IV bits). For simplicity, the distinction between public and private variables is ignored for the rest of the section.

[1] The algorithm is intended for only on special types of ciphers in which layers of S-Boxes are interconnected by linear layer and key addition

[2] only by products of monomials that occur in the original system

Let $p(x_1, \ldots, x_n)$ be a multivariate master polynomial over $GF(2)$ given in algebraic normal form (ANF) and t_I be the multiplication of variables x_i where i is in an index set $I \subseteq \{1, \ldots, n\}$. Then the polynomial p can be written as :

$$p(x_1, \ldots, x_n) \equiv t_I \cdot p_{S(I)} + q(x_1, \ldots, x_n)$$

The idea behind the cube attack is that the symbolic sum of all polynomials derived from the master polynomial by assigning all possible 0/1 values to the variables x_i gives the coefficient of the monomial t_I.

$$p_{S(I)} \equiv \bigoplus_I p(x_1, \ldots, x_n)\,.$$

Following the terminology of [70], $p_{S(I)}$ is called the *superpoly* of I in p, and the term t_I is called a *maxterm* of p if and only of $p_{S(I)}$ is of degree one (a linear polynomial which is not a constant). Any subset I of size k defines a k-dimensional Boolean *cube* of 2^k vectors.

The cube attack is composed of two phases:

- **Preprocessing Phase:** The attacker sets the values of all variables and uses the black box to find sufficiently many maxterms with linearly independent superpolys. In other terms, he reconstructs the ANF of the superpoly for each t_I. This phase is not key dependent and performed only once for each cryptographic scheme.
- **Online Phase:** The n secret variables are set to unknown values, and the attacker evaluates the superpoly's by setting the values of m public variables and finds a linear combination of the key bits. Finally the key is recovered from linear algebra. This phase requires at most 2^{d-1} evaluations of the derived polynomials assuming p is of degree d.

The cube attack can be seen as an extension to Algebraic IV Differential Attack (AIDA) [197] and higher order differentials [121]. The main idea in all these attacks is to generate a sufficient number of plaintexts and sum up the corresponding ciphertexts to obtain some properties that will ease the attack.

In addition to key recovery, by evaluating superpolys and performing algebraic tests on them, it is possible to to distinguish a cryptographic scheme from a random function or to detect non-randomness. This algorithm is called a *cube tester* [11]. Properties such as balance, low degree, the presence of linear variables and presence of neutral variables can be tested by cube testers.

The dynamic cube attack [71] is an extension of the basic attack which recovers the key by using distinguishers obtained from cube testers instead of solving

a system of linear equations. Whereas in static cube testers the values of all public variables that are not in a cube is set to a constant (usually zero), in the dynamic cube attack the values of some such variables are not fixed and vary depending on some public variables in the cube and private variables.

Chapter 4

Cryptanalysis of JH

JH [205] is the submission of Hongjun Wu to the NIST Hash Competition and it is one of the five finalist algorithms. It is a sponge-like iterative hash function that supports four different hash sizes (224, 256, 384 and 512-bit). The hash algorithm is very simple and efficient in both software and hardware, and is among the fastest contestants.

The compressions function of JH is a simple substitution permutation network that processes 1024-bit blocks. The round function uses 4·4 S-boxes followed by a linear transformation and a permutation. Although the permutation seems complex, its bit-slice implementation is highly structured and symmetric which allows us to analyze the algorithm easily. JH has been tweaked for the final round by increasing its number of rounds. This chapter is composed of a brief description of the JH hash function and its properties followed by our results on JH.

Our first attack [168] is based on the rebound attack [142] and it applies to both the hash function and the compression function of JH. Using the rebound technique we first find semi-free-start collision for the hash function. Then we improve the complexity of our attack by using three inbound phases instead of one and obtain semi-free-start near-collisions for the compression function of JH. We implement the rebound attack to the small scale variant of JH to describe the attack better.

Our second attack [158] applies to the tweaked version of JH and uses six inbound phases. However, due to increased number of rounds compared with

the previous attacks, finding the whole solution for the path can no longer be done easily. We first find partial solutions for the active part of the differential path by using the ideas from [157]. We then propose a novel algorithm that allows us to find solutions for the passive part without contradicting any of the already fixed values from the inbounds. As a result we obtain semi-free-start near-collisions for the compression function of JH. Using the same differential characteristic, we finally present distinguishers for 42 rounds of the internal permutation.

4.1 Preliminaries

Throughout this chapter, we will use the following notation:

word	4 bits
d	dimension of a block of bits i.e. a d-dimensional block of bits consists of 2^d words
H_i	the i-th chaining value
JH-X	the member of the family whose message digest is X bits
M_i	Message block i
$m_{i,j}$	the j^{th} word of the i^{th} round value
$\|\|$	concatenation operation
$\cdot$	cross-product: an operation on two arrays that result in another array whose elements are obtained by combining each element in one array with every element in the second one

4.2 The JH Hash Function

The hash function JH is an iterative hash function that accepts message blocks of 512 bits and produces a hash value of 224, 256, 384 and 512 bits. The message is padded to be a multiple of 512 bits. The bit '1' is appended to the end of the message, followed by $384-1+(-l \bmod 512)$ zero bits. Finally, a 128-bit block is appended which is the length of the message, l, represented in big endian form. Note that this scheme guarantees that at least 512 additional bits are padded. In each iteration, the compression function F_d, given in Figure4.1, is used to update the 2^{d+2} bits as follows:

$$H_i = F_d(H_{i-1}, M_i)\,.$$

where H_{i-1} is the previous chaining value and M_i is the current message block.

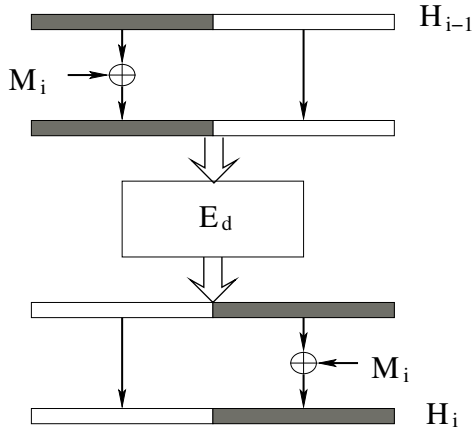


Figure 4.1: The compression function F_d

The compression function F_d is defined as follows:

$$F_d(H_{i-1}, M_i) = E_d(H_{i-1} \oplus (M_i || 0^{2^{d+1}})) \oplus (0^{2^{d+1}} || M_i)\,.$$

In the submitted version of JH to the SHA-3 competition, E_d is a permutation and is composed of an initial grouping of bits followed by $5(d-1)$ rounds, plus an additional S-box layer and a final degrouping of bits. In the tweaked version (2011) the number of rounds is increased to $6(d-1)$. The grouping operation arranges bits such that the input to each S-box has two bits from the message and two bits from the chaining value.

In each round, the input is divided into 2^d words and then each word passes through an S-Box. JH uses two 4-bit-to-4-bit S-Boxes ($S0$ and $S1$) and every round constant bit selects which S-Boxes are used. Then two consecutive words pass through the linear transformation L, which is based on a $[4, 2, 3]$ Maximum Distance Separable (MDS) code over $GF(2^4)$. Finally all words are permuted by the permutation P_d. After the degrouping operation each bit returns to its original position. The round function for $d = 4$ is shown in Figure 4.2 and $d = 8$ is the version for the SHA-3 competition. For a more detailed information we refer to the specification of JH [205].

The initial hash value H_0 is set depending on the message digest size. The first two bytes of H_1 are set to the message digest size, and the remaining bytes of H_1 are set to 0. Finally, the message digest is generated by truncating H_t where t is the number of blocks in the padded message., i.e, the last X bits of H_t are given as the message digest of JH-X for $X = 224$, 256, 384, 512.

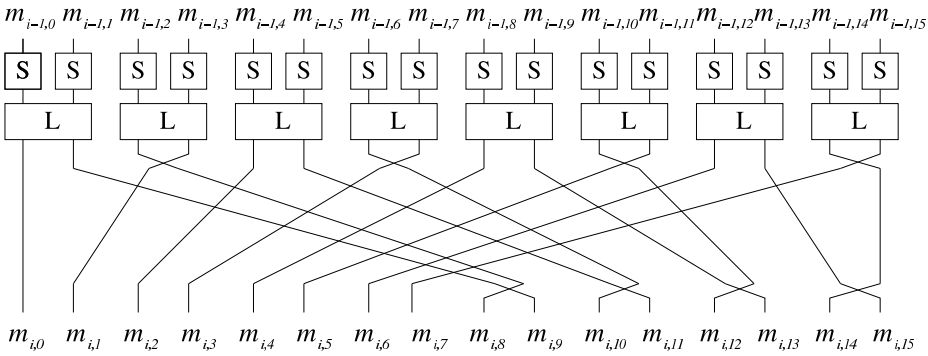


Figure 4.2: Round Function of E_d for $d = 4$

4.2.1 Properties of the Linear Transformation L

Since the linear transformation L implements a $(4, 2, 3)$ MDS matrix, any difference in one of the words of the input (output) will result in a difference in two words of the output (input). If one tries all possible 2^{16} pairs, the number of pairs satisfying each condition ($2 \rightarrow 1$ or $1 \rightarrow 2$) is 3840, which gives a probability of $3840/65536 \approx 2^{-4.09}$. Note that, if the words are arranged in a way that they will be both active this probability increases to $3840/57600 \approx 2^{-3.91}$. For the latter case, if both words remain active ($2 \rightarrow 2$), the probability is $49920/57600 \approx 2^{-0.21}$. (see also Table 4.1)

Table 4.1: The difference distribution of the linear layer

Pattern[a]	00	01	10	11
01	256	0	0	0
01	0	0	0	3840
10	0	0	0	3840
11	0	3840	3840	49920

[a](0:passive word, 1:active word)

4.2.2 Observations on the Compression Function

The grouping of bits at the beginning of the compression function ensures that the input of every first layer S-Box is xor-ed with two message bits. Similarly, the output of each S-Box is xor-ed with two message bits. Therefore, for a random non-zero 4-bit difference, the probability that this difference is related to a message is $3/15 \approx 2^{-2.32}$.

The bit-slice implementation of F_d uses $d-1$ different round function descriptions. The main difference between these round functions is the permutation function. In each round permutation, the odd bits are swapped by $2^r \bmod (d-1)$ where r is the round number. Therefore, for the same input passing through multiple rounds, the output is identical to the output of the original round function for the $\alpha \cdot (d-1)$-th round where α is any integer. Three rounds of the bit-sliced representation can be seen in Figure 4.3 between rounds 1 and 4.

4.3 The Start-From-The-Middle Attack on JH

In our work, we first apply the start-from-the-middle technique [18] with an all active state. We begin by guessing the middle values and then proceed forwards and backwards using the filtering conditions to reduce the number of active S-Boxes in each round.

In this section, we describe the steps of our attack on JH. We will first describe the attack on a smaller version of JH, i.e., $d = 4$ in detail, and then give the algorithm and analysis for $d = 8$.

4.3.1 Attack on 8 Rounds of JH for $d = 4$

Attack Procedure

The inbound phase of the attack described in this section is composed of 8 rounds, and the number of active S-Boxes in each round is:

$$1 \leftarrow 2 \leftarrow 4 \leftarrow 8 \leftarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1\,.$$

Here, the arrows represent the direction of the computations for the inbound phase. The bit-slice implementation allows us to analyze the algorithm easily. The truncated differential path is given in Figure 4.3. The attack can be summarized as follows:

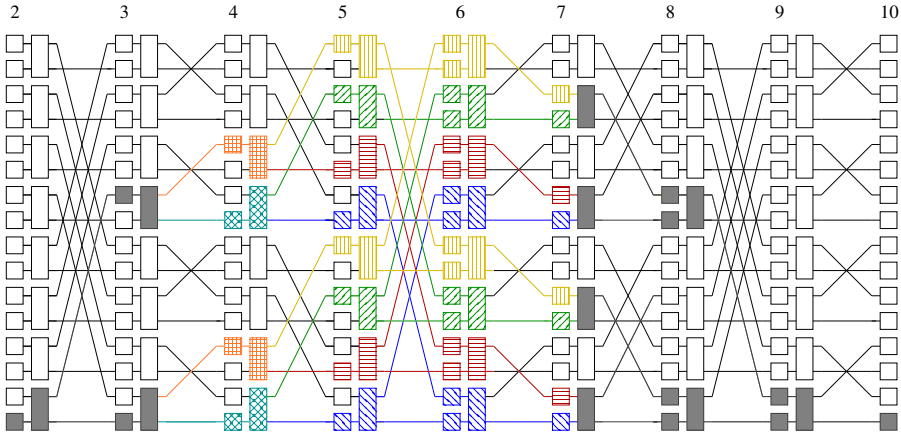


Figure 4.3: Inbound phase of JH for $d = 4$ (bit-slice implementation)

- **Step 1:** Try all possible 2^{16} values for the middle values $m_{4,j}||m_{4,j+1}$ and $m'_{4,j}||m'_{4,j+1}$ in Round 4 for each of the four sets (shown with colors and different shapes in Figure 4.3), and keep only the ones that satisfy the desired pattern ($2 \rightarrow 4 \rightarrow 2$). Therefore, the expected number of remaining pairs is $2^{16} \cdot 2^{16}/[(2^{4.09})^2 \cdot (2^{3.91})^2] = 2^{16}$ for each set.

- **Step 2:** Compute the cross-product of the sets: ▨ · ▨ and ▤ · ▥, check if the differences satisfy $2 \rightarrow 1$ when they pass the inverse L transform, L^{-1}. For the pairs that satisfy the filtering condition, store only the values in the active words and the middle values for each of the 2 sets. After this step, the number of pairs in each set is approximately $2^{16} \cdot 2^{16}/(2^{3.91})^2 = 2^{24.18}$.

- **Step 3:** Compute the cross-product of the sets: ▩ · ▦, check whether the remaining 10 filtering conditions (marked with ▬) are satisfied or not. This control can be done by calculating $L \circ S$ or $S^{-1} \circ L^{-1}$ for the active words only and does not require the use of the round function entirely, hence it is very efficient. The total number of remaining pairs that pass the inbound phase is $2^{24.18} \cdot 2^{24.18}/(2^{3.91})^{10} = 2^{9.26}$.

Note that, due to the symmetry, the actual number of remaining pairs is $2^{8.26}$ and the duplication can be avoided in the earlier steps of the algorithm, but for simplicity our description ignores this. The attack algorithm only stores the middle values for the pairs that follows the desired differential path and

the values in the n-th round can be computed by calling the round (or inverse round) function.

The active S-Boxes in the input and output to the compression function must satisfy the desired property, so out of $2^{8.26}$ pairs only $2^{8.26} \cdot (2^{-2.32})^2 = 2^{4.32}$ remain. In order to obtain a collision, the differences in both S-Boxes also need to match, which happens with a probability of 1/3. Therefore for $2^{4.32} \cdot 1/3 \simeq 2^{2.74}$ pairs we obtain a semi-free-start collision for the hash function.

Complexity of the Attack

The inbound phase is the part of the attack where most of the calculations is done. Let $U = L \circ S$ and $U^{-1} = S^{-1} \circ L^{-1}$. Then, a round function consists of 8 U-functions, similarly an inverse round functions has 8 U^{-1}-functions.

For each of the four sets in Step 1 of the algorithm, we try all possible 2^{16} pairs and apply the filtering condition, Although we have 2 U-functions in the forward direction and 2 U^{-1}-functions in the backward direction, we only check one condition if the previous one is satisfied, so the total number of calls, n_1 is:

$$n_1 = 2^{32} + 2^{32}/2^{4.09} + 2^{32}/(2^{4.09})^2 + 2^{32}/[(2^{4.09})^2 \cdot 2^{3.91}] = 2^{32.09}\,.$$

which is approximately 2^{291} round functions. However this step can be done by 2^{32} table look-ups if precomputation is used.

For each of the two sets in Step 2 of the algorithm, since L and L^{-1} are a linear transformations, it is sufficient to check whether the differences satisfy the desired property, i.e.$L^{-1}(\Delta c, \Delta d) = (\Delta a, 0)$ or $(0, \Delta b)$. The total number of calls in this step is:

$$n_2 = (2^{16})^2 \cdot (1 + 2^{-3.91}) = 2^{32.09}\,.$$

In the final step of the inbound phase, 3 of the 10 conditions to be checked are again linear transformations and the remaining 7 require the use of the U-function. The complexity of the attack is dominated by this step and the total number of operations performed is:

$$n_{3,bck} = (2^{24.18})^2 \cdot [(1 + 2^{-3.91} + (2^{-3.91})^2] \simeq 2^{48.46}$$

$$n_{3,fwd} = (2^{24.18})^2/(2^{-3.91})^3 \cdot \sum_{i=0}^{6} (2^{-3.91})^i \simeq 2^{36.72}\,.$$

[1] $2^{32.09} \cdot 1/8 = 2^{29.09}$

where *fwd* and *bck* denote the forward and backward direction respectively. In the outbound phase, for each of the $2^{8.26}$ remaining pairs, starting from the middle values, we call the round and inverse round functions the obtain the input and output values.

Results

The above algorithm has been implemented for $d = 4$ and we observed that we can obtain semi-free-start collision for 8-rounds JH-16 with the calculated complexity. An example is given in Table A.1.

4.3.2 The Attack on 16 Rounds of JH with $d = 8$

In this section, we first present an outline for the start-from-the middle attack on a reduced round version of JH for all hash sizes, and then give the calculations for the complexity analysis of the attack.

Attack Procedure:

For the compression function E_8, the attack is composed of 16 rounds and the number of active S-Boxes is:

$$1 \leftarrow 2 \leftarrow 4 \leftarrow 8 \leftarrow 16 \leftarrow 32 \leftarrow 64 \leftarrow 128 \leftarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

The algorithm is similar to the one of E_4. We again start from the middle and then propagate outwards by computing the cross-product of the sets and using the filtering conditions. However, instead of trying all 2^{16} possible pairs, we start with $2^{7.92}$ values for each middle value. The number of sets, the bit length of the middle values (size) of each set, and the number of filtering conditions followed by the number of pairs in each set are given in Table 4.2. Similarly, we only store the values in the active bytes for the outermost rounds and the middle round for each set, i.e., no other intermediate value is stored.

Complexity of the Attack

The time complexity of the attack for $d = 8$ is calculated in a manner similar to that of $d = 4$. Instead of giving all equations explicitly, we summarize the results in terms of function calls and their direction for each step in Table 4.2. The time complexity of the given attacks is $2^{190.24}$ U-function calls.

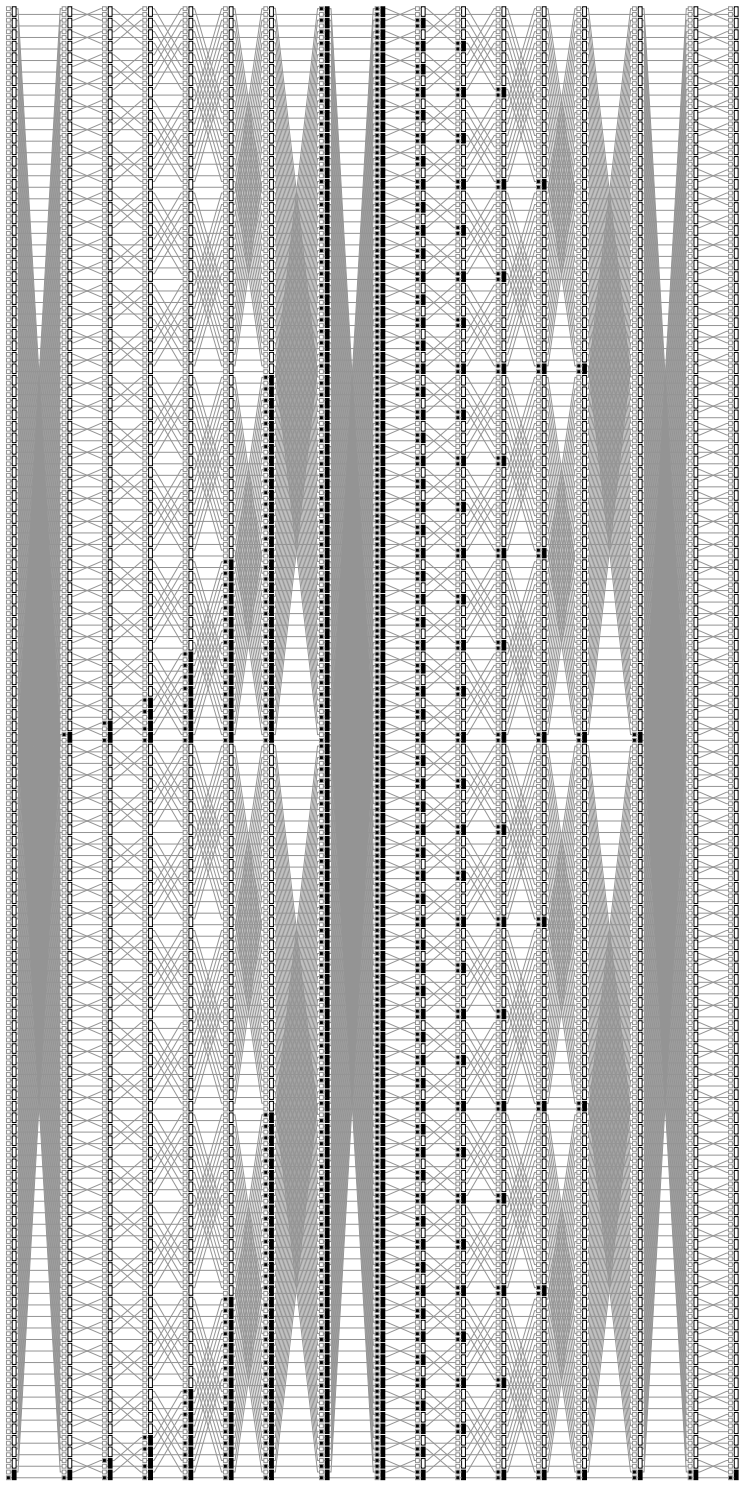


Figure 4.4: Differential characteristic for 16 rounds of the JH Hash Function (bit-slice representation)

Table 4.2: Overview of inbound phase

Step	Size (bits)	Sets	Filtering Conditions	Pairs Remain	Time Complexity	Direction
0	8	128	1	$2^{11.75}$	$2^{15.84}$	*fwd*
1	16	64	1	$2^{19.59}$	$2^{23.50}$	*bck*
2	32	32	4	$2^{23.54}$	$2^{39.18}$	*fwd*
3	64	16	4	$2^{31.44}$	$2^{47.08}$	*fwd*
4	128	8	4	$2^{47.24}$	$2^{62.88}$	*fwd*
5	256	4	8	$2^{63.20}$	$2^{94.48}$	*fwd*
6	512	2	8	$2^{95.12}$	$2^{124.40}$	*fwd*
7	1024	1	46	$2^{10.38}$	$2^{190.24}$	*fwd + bck*

Results

We may lose up to half of the remaining pairs due to symmetry. In addition, similar to the case for d=4, the active S-Boxes in the input and output to the compression function should correspond only to the message bits and then match each other in order to obtain a collision. Therefore, out of $2^{10.38}$ pairs only $2^{10.38} \cdot 1/2 \cdot (2^{-2.32})^2 \cdot 1/3 \simeq 2^{3.15}$ remain.

Suppose that we intend to attack a block M_i where $(i < N)$. Since we obtain a zero difference in the chaining value that guarantees that the outputs of the compression function will be the same provided that both messages have the same length. As mentioned earlier the same compression function is used for all hash sizes, and the message digest is generated by truncating H_t where t is the number of blocks in the padded message. Therefore, we have a semi-free-start collision for all hash sizes of JH reduced to 16 rounds from 35.5.

4.4 The Rebound Attack on the Compression Function of JH

The attack on the hash function given in Section 4.3.2 can be easily converted to an attack on 19 rounds of the compression function for the pairs that satisfy the inbound phase by using the following differential trails in the outbound phases:

$$2 \leftarrow 1 \leftarrow \text{ Inbound Phase } \rightarrow 1 \rightarrow 2 \rightarrow 4$$

The complexity of the attack remains same (i.e., $2^{190.24}$ U-function calls) and we obtain a semi-free-start near-collision for 1008 bits. In this section, we improve these results by using three inbound phases. Once again, we first describe the steps of our attack for $d = 4$ in detail, and then give the algorithm and complexity analysis for $d = 8$.

4.4.1 The Improved Rebound Attack on JH with d$= 4$

We can improve our results by using three inbound phases with partially active states rather than one all active one. This allows us to decrease the complexity of the attack. The inbound phase of the attack described in this section is composed of 8 rounds, using the following trail:

$$2 \leftarrow 4 \rightarrow 2 \leftarrow 4 \leftarrow 8 \rightarrow 4 \leftarrow 8 \rightarrow 4 \rightarrow 2\,. \tag{4.1}$$

It is perhaps interesting to make here some observations on the number of active S-boxes in the trail. Similar to the AES, the linear diffusion layer of JH imposes a lower bound on the number of active S-boxes: if $d \geq 2$, then there are at least $3^2 = 9$ active S-boxes in every sequence of 4 rounds. The conjectured bound on the number of active S-boxes over $2d + 1$ rounds [205], as well as the trail (4.1), demonstrate that the *higher dimension* of the JH diffusion layer allows for relatively long and *narrow trails*.

We decompose the inbound phase into a sequence of three smaller inbound phases, each of which are 3, 2 and 3 rounds respectively. The number of active SBoxes for each of the steps in each round is:

$$\begin{aligned} &2 \leftarrow 4 \rightarrow 2 \\ &2 \leftarrow 4 \leftarrow 8 \rightarrow 4 \\ &4 \leftarrow 8 \rightarrow\ 4 \rightarrow 2\,. \end{aligned}$$

We use the bit-sliced representation to analyze the algorithm. We first calculate the results of the first and the third inbound phases, and then match them with the second inbound phase. The truncated differential path is given in Figure 4.5. The attack can be summarized as follows:

First Inbound Phase

- Try all possible 2^8 values for each of the middle values $m_{1,j}||m_{1,j+1}$ and $m'_{1,j}||m'_{1,j+1}$ in Round 1 for each of the two sets, and keep only the ones that satisfy the desired pattern. Therefore, the expected number of remaining pairs is $2^8 \cdot 2^8/2^{4.09} = 2^{11.91}$ for each set.

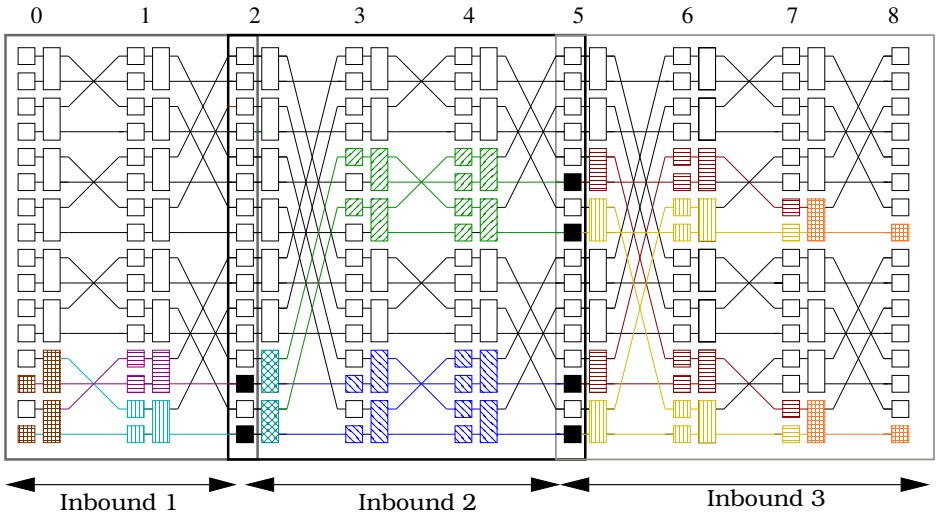


Figure 4.5: Inbound phase of JH ($d = 4$)

- Compute the cross-product of the two sets and check if the differences satisfy $2 \rightarrow 1$ when they pass L^{-1}. For the pairs that satisfy the filtering condition, store only the values in the active words and the middle values. After this step, the number of pairs is approximately $2^{11.91} \cdot 2^{11.91}/(2^{3.91})^2 = 2^{16}$.

- Check whether the remaining pairs satisfy the desired input difference, and store these values in list L_1. Therefore, the size of L_1 is $2^{16} \cdot (2^{-2.32})^2 = 2^{11.36}$

Second Inbound Phase

- Try all possible 2^8 values for each of the middle values $m_{4,j}||m_{4,j+1}$ and $m'_{4,j}||m'_{4,j+1}$ in Round 4 for each of the four sets, and keep only the ones that satisfy the desired pattern. The expected number of remaining pairs is again $2^8 \cdot 2^8/2^{4.09} = 2^{11.91}$ for each set.

- Compute the cross-product of the sets having the same pattern and check if the differences satisfy $2 \rightarrow 1$ when they pass L^{-1}. For the pairs that satisfy the filtering condition, store the values for each of the 2 sets. The expected number of pairs in each set is approximately $(2^{11.91})^2/(2^{3.91})^2 = 2^{16}$.

- Compute the cross-product of the sets, ▨ · ▨, and check if the differences satisfy the filtering condition when they pass L^{-1}. The expected number of remaining pairs that pass the second inbound phase is $(2^{16})^2/(2^{-3.91})^2 = 2^{24.18}$.

Third Inbound Phase

- Try all possible 2^8 values for each of the middle values $m_{6,j}||m_{6,j+1}$ and $m'_{6,j}||m'_{6,j+1}$ in Round 6 for each of the four sets, and keep only the ones that satisfy the desired pattern. The expected number of remaining pairs is again $2^{11.91}$ for each set.

- Compute the cross-product of the sets having the same pattern and check if the differences satisfy $2 \rightarrow 1$ when they pass the inverse L transform. For the pairs that satisfy the filtering condition, store the values for each of the 2 sets. The expected number of pairs in each set is approximately $(2^{11.91})^2/(2^{3.91})^2 = 2^{16}$.

- Compute the cross-product of the two sets, ▤ · ▥, to check the final filtering conditions in round 7. The expected number of pairs that pass the third phase is $(2^{16})^2/(2^{-3.91})^2 = 2^{24.18}$. Store these values in list L_3.

Merging Inbound Phases

The three previous inbound phases overlap in the 2 and 4 active words (denoted with black in Figure 4.5) in rounds 2 and 5 respectively. Since we have to match these active words in both values, we get a condition on $16 + 32 = 48$ bits in total. These conditions are checked as soon as we have a remaining pair for the second inbound phase, by using the lists L_1 and L_3. As a result, we expect to find $(2^{11.36} \cdot 2^{24.18} \cdot 2^{-16}) \cdot 2^{24.18} \cdot 2^{-32} \cdot (2^{-2}) \simeq 2^{9.72}$ solutions for the inbound phase. The last 1/4 factor in the calculation follows from symmetry.

Outbound Phase

For the pairs that satisfy the inbound phase, we expect to see the following differential trail in the outbound phase:

$$2 \leftarrow \text{ Inbound Phase } \rightarrow 2 \rightarrow 4 \rightarrow 2\,.$$

Therefore, for the compression function E_4, we have the 10-round differential path shown in Figure 4.6. Note that, there are two filtering conditions in the last round of the outbound phase. Thus, out of $2^{9.72}$ solutions, only $2^{9.72} \cdot (2^{-3.91})^2 \approx 2^{1.9}$ pass to the last round. After the degrouping operation, the message is xor-ed to the rightmost 32-bits of the output and for the compression function of JH for $d = 4$ we have a near-collision for 52 bits.

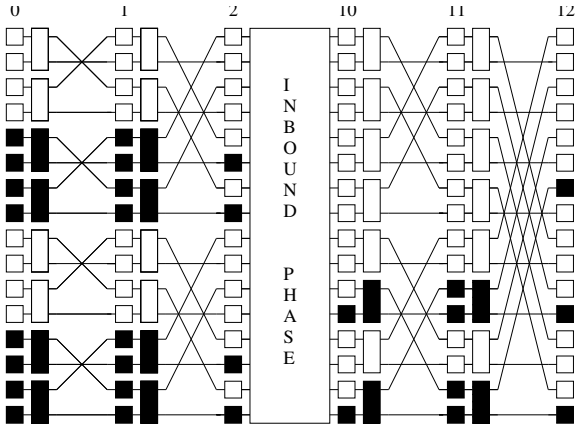


Figure 4.6: Outbound phase of JH ($d = 4$)

Data Complexity

The time complexity of the attack is determined by the first and third inbound phases which is about $2^{32.09}$ each, hence the total time complexity is $2^{32.09} + 2^{32.09} = 2^{33.09}$ U-function calls. The memory complexity is also determined by the third inbound phase which is $2^{24.18}$.

Results

We obtain a 52-bit free-start near-collision for 10 rounds of the JH compression function. The results are unfortunately not better than theoretic bounds for JH with $d = 4$, but it helps us to implement and verify the attack and expand it for the submitted version of JH.

4.4.2 The Improved Rebound Attack on JH with $d = 8$

In this section, the attack in Section 4.4.1 on JH with $d = 4$ is extended to JH with $d = 8$ using more rounds (hence the larger number of steps and the increased complexity) for each of the inbound and outbound phases. The attack is applicable to 19 rounds (out of 35.5) of the compression function. We first explain the attack in detail and then give the calculations for the complexity analysis.

Inbound Phase

For the compression function E_8, the inbound phase of the attack is 16 rounds and is composed of two parts. In the first part, we apply the start-from-the-middle-technique three times for rounds $0-3$, $3-10$ and $10-16$. In the second part, we connect the resulting active bytes (hence the corresponding state values) by a match-in-the-middle step. The number of active S-Boxes in each of the sets is:

$$\begin{aligned} &2 \leftarrow\ 4 \leftarrow\ 8 \rightarrow\ 4 \\ &4 \leftarrow\ 8 \leftarrow 16 \leftarrow 32 \leftarrow 64 \leftarrow 128 \rightarrow 64 \rightarrow 32 \\ &32 \leftarrow 64 \rightarrow\ 32 \rightarrow 16 \rightarrow\ 8 \rightarrow\ 4 \rightarrow 2\,. \end{aligned}$$

For a detailed sketch of the inbound phase one can refer to Figure 4.7. The algorithm for each set is similar to the one of E_4. We again start from the middle and then propagate outwards by computing the cross-product of the sets and by using the filtering conditions. For each list, we try all possible 2^{16} pairs in Step 0. The number of sets, the bit length of the middle values (size) of each list, and the number of filtering conditions followed by the number of pairs in each set are given in Table 4.3.

Merging Inbound Phases

Connecting these three lists is again performed as follows. Whenever a pair is obtained from set 2, we check whether it exists in L_1 or not. If it does, another check is done for L_3. We have $2^{19.54}$ and $2^{138.7}$ elements in lists 1 and 3 respectively, $2^{152.28}$ pairs passing the second inbound phase, and 32-bit and 256-bit conditions for the matches. Then, the total expected number of remaining pairs is $(2^{19.54} \cdot 2^{152.28} \cdot 2^{-32}) \cdot 2^{138.70} \cdot 2^{-256} \cdot (2^{-2}) = 2^{20.52}$.

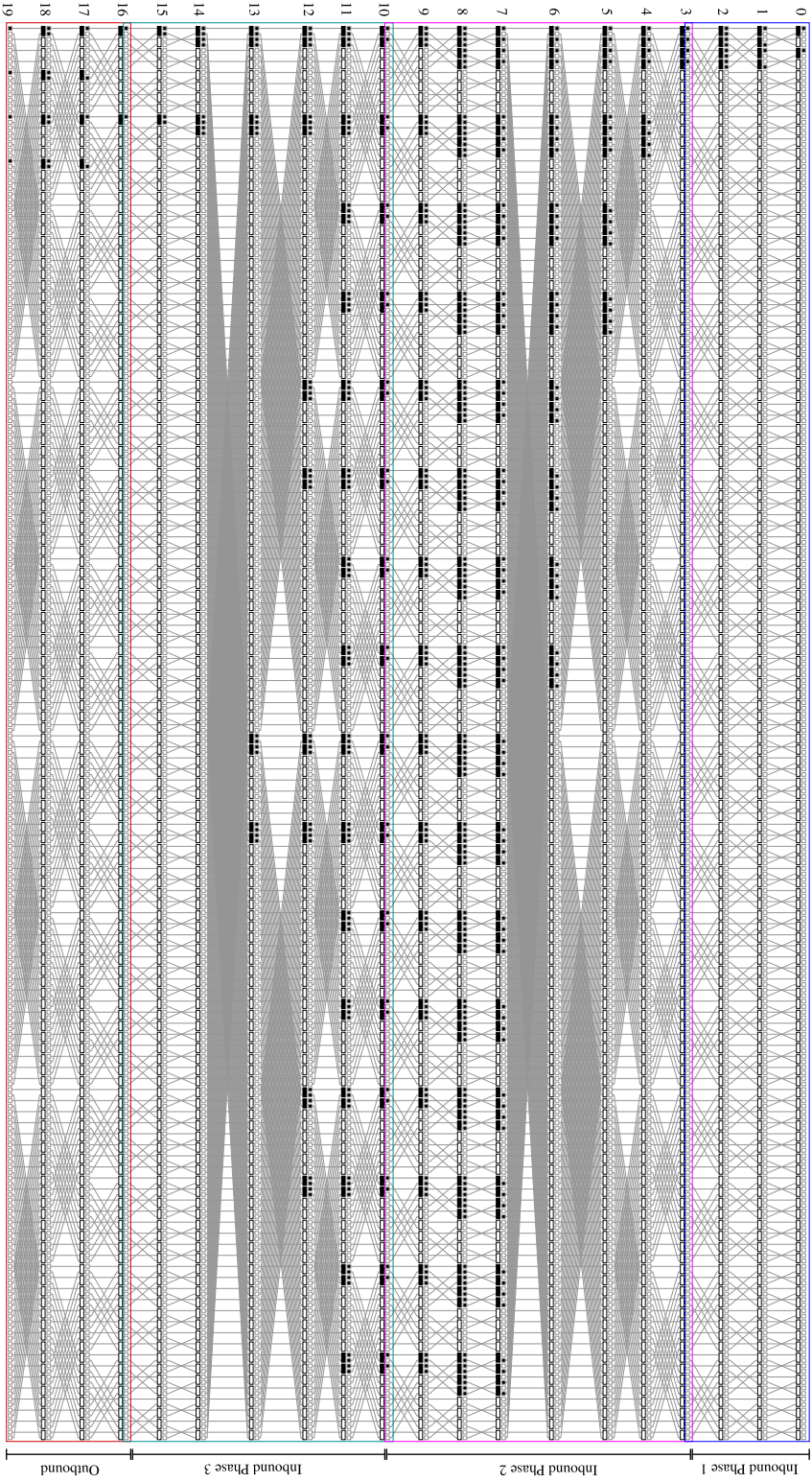


Figure 4.7: Inbound and outbound phases of JH compression function (bit-slice representation)

Table 4.3: Overview of inbound phases for the attack on JH-512

Step	Size	Sets	Filtering Conditions	Pairs Remain	Complexity Backwards	Complexity Forwards
0	8	4	1	$2^{11.91}$	–	$2^{16.00}$
1	16	2	2	$2^{16.00}$	$2^{23.91}$	–
2	32	1	2	$2^{24.18}$	$2^{32.09}$	–
3	32	1	2^{a}	$2^{19.54}$		
0	8	64	1	$2^{11.91}$	–	$2^{16.00}$
1	16	32	2	$2^{16.00}$	$2^{23.91}$	–
2	32	16	2	$2^{24.18}$	–	$2^{32.09}$
3	64	8	4	$2^{32.72}$	$2^{48.46}$	–
4	128	4	4	$2^{49.80}$	$2^{65.54}$	–
5	256	2	4	$2^{83.96}$	$2^{99.70}$	–
6	512	1	4	$2^{152.28}$	$2^{168.02}$	–
0	8	32	1	$2^{11.91}$	–	$2^{16.00}$
1	16	16	2	$2^{16.00}$	$2^{23.91}$	–
2	32	8	2	$2^{24.18}$	–	$2^{32.09}$
3	64	4	2	$2^{40.54}$	–	$2^{48.45}$
4	128	2	2	$2^{73.26}$	–	$2^{81.17}$
5	256	2	2	$2^{138.70}$	–	$2^{146.61}$

[a] Check whether the pairs satisfy the desired input difference

We obtained more pairs than usual due to the additional filtering conditions in the outbound phase, in order to obtain a near-collision (which will be explained in the following part) for 19 rounds of the compression function.

Outbound Phase

The outbound of the attack is composed of 3 rounds in the forward direction. For the pairs that satisfy the inbound phase, we expect to see the following differential trail in the outbound phase:

$$\text{Inbound Phase} \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 4\,.$$

Note that in the last step of the outbound phase we have four filtering conditions. We had $2^{20.52}$ remaining pairs from the inbound phase, thus, we expect $2^{20.52} \cdot (2^{3.91})^4 = 2^{4.88}$ pairs to satisfy the above path. A detailed schema of this trail is shown in Figure 4.7.

The final step of the compression function is xor-ing the message bits after the degrouping operation to the output of the compression function. We have 4 active words in the output and a 4-bit difference in the message, two of which collide in bit positions 512 and 768. Thus, it is possible to cancel them with a probability of 2^{-2} and the number of pairs reduces to $2^{2.88}$. To sum up, we have a difference in $(4 \cdot 4 - 2) + 2 = 16$ bits in total.

Complexity of the Attack

For the inbound phase, the complexity of the attack for $d = 8$ is calculated in a similar manner to that of $d = 4$. The results for preparing the lists are summarized for each step in Table 4.3. The time complexity of the attack is dominated by the second set, L_2, which is about $2^{168.02}$ U-function calls The memory requirements are determined by the largest list, which is L_3 with a size of $2^{138.70}$ 256-bit data.

Results

Note that, in this attack, the complexity requirements are reduced significantly compared to the initial idea that uses only one inbound phase. For 19 rounds of the JH compression function, we obtain a semi-free-start near-collision for 1008 bits. We can simply increase the number of rounds by proceeding forwards in the outbound phase. Our attack still works in this case with the same complexity (U-function calls). The number of bits for the near-collision and the generic attack complexities are given in Table 4.4. As a result, our attack is better than generic attacks up to 22 rounds.

Table 4.4: Complexity of the generic attack for near-collisions

#Rounds	# bits Near Collision	Generic Attack Complexity	Our Results
19	1008	$2^{454.21}$	$2^{168.02}$
20	992	$2^{411.18}$	$2^{168.02}$
21	960	$2^{341.45}$	$2^{168.02}$
22	896	$2^{236.06}$	$2^{168.02}$
23	768	$2^{99.18}$	$2^{168.02}$

4.5 Rebound Attack on JH42

In this section we describe the multi-inbound rebound attack of [158] using 6 inbounds. We first present an outline for the rebound attack on reduced round versions of JH for all hash sizes. We use a differential characteristic that covers 32 rounds, and apply the start-from-the-middle technique by using six inbound phases with partially active states. We first describe how to solve the multi-inbound phase for the active bytes. Contrary to previous attacks on JH, we now have more fixed values from the inbound phases. Hence, in order to find a complete solution, we need to merge these fixed values without contradicting any of them. Therefore, we describe next how to match the passive bytes. Finally, we analyze the outbound part.

4.5.1 Matching the Active Bytes

Multi-inbound Phase

The multi-inbound phase of the attack covers 32 rounds and is composed of two parts. In the first part, we apply the start-from-the-middle-technique six times for rounds $0-4$, $4-10$, $10-16$, $16-20$, $20-26$ and $26-32$. In the second part, we connect the resulting active bytes (hence the corresponding state values) by a match-in-the-middle step. The number of active S-Boxes in each of the sets is:

$$
\begin{aligned}
4 \leftarrow\ 8 \leftarrow 16 \rightarrow\ 8 \rightarrow 4 \\
4 \leftarrow\ 8 \leftarrow 16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \\
16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow\ 8 \rightarrow 4 \\
4 \leftarrow\ 8 \leftarrow 16 \rightarrow\ 8 \rightarrow 4 \\
4 \leftarrow\ 8 \leftarrow 16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \\
16 \leftarrow 32 \leftarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow\ 8 \rightarrow 4\,.
\end{aligned}
$$

For a detailed picture we refer to Figure 4.8. We start from the middle and then propagate outwards by computing the cross-product of the sets and using the filtering conditions. For each inbound we try all possible 2^{16} pairs in Step 0. The number of sets, the bit length of the middle values (size) of each list, and the number of filtering conditions on words followed by the number of pairs in each set are given in Table 4.5. The complexities given in the Table 4.5 are not optimized yet; we will describe the improved complexities later in Section 4.5.1.

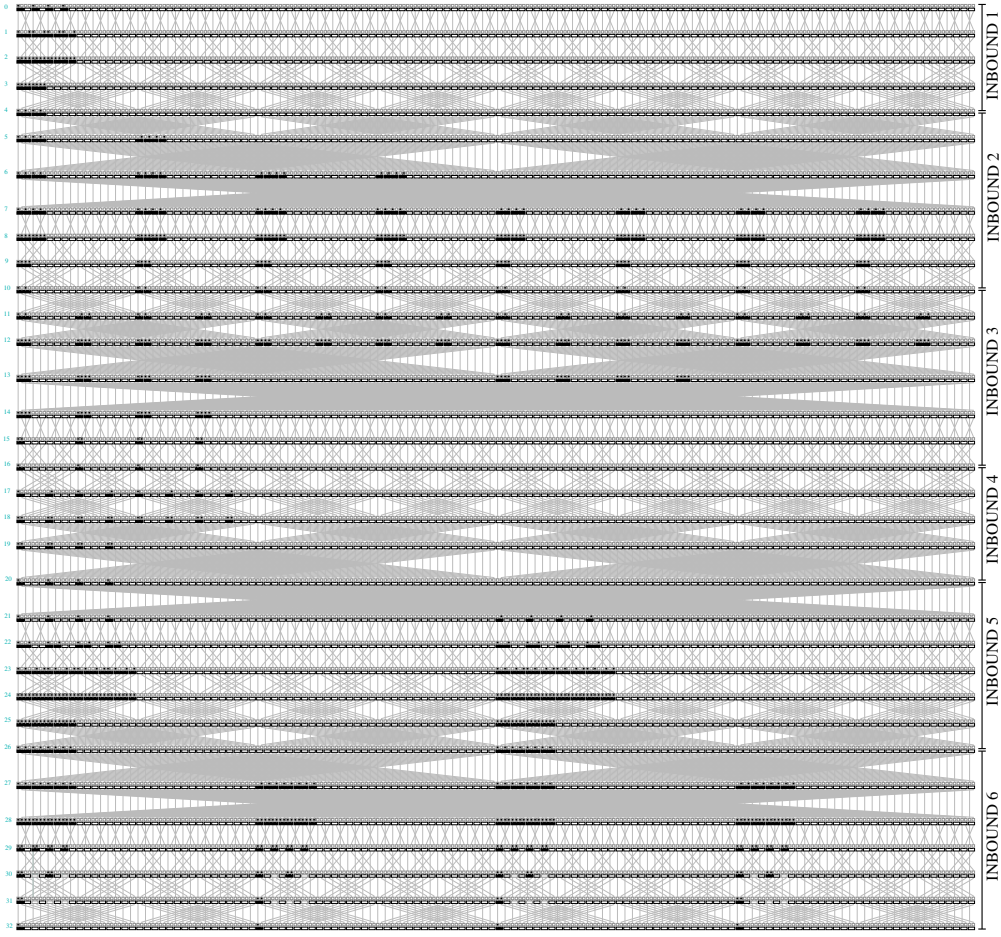


Figure 4.8: Differential characteristic for 32 rounds of the JH42 compression function (bit-slice representation)

Table 4.5: Overview of inbound phases of the attack on 32 rounds of JH

	Step	Size	Sets	Filtering Conditions	Pairs Remaining	Complexity Backwards	Forwards
Inbound 1	0	8	8	1	$2^{11.91}$	–	$2^{16.00}$
	1	16	4	2	$2^{16.00}$	$2^{23.91}$	–
	2	32	2	2	$2^{24.18}$	$2^{32.09}$	–
	3	64	1	4	$2^{32.72}$	$2^{48.46}$	–
	4	64	1	4[a]	$2^{23.44}$		
Inbound 2	0	8	32	1	$2^{11.91}$	–	$2^{16.00}$
	1	16	16	2	$2^{16.00}$	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	–	$2^{32.09}$
	3	64	4	4	$2^{32.72}$	$2^{48.46}$	–
	4	128	2	4	$2^{49.80}$	$2^{65.54}$	–
	5	256	1	4	$2^{83.96}$	$2^{99.70}$	–
Inbound 3	0	8	32	1	$2^{11.91}$	–	$2^{16.00}$
	1	16	16	2	$2^{16.00}$	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	$2^{32.09}$	–
	3	64	4	4	$2^{32.72}$	–	$2^{48.46}$
	4	128	2	4	$2^{49.80}$	–	$2^{65.54}$
	5	256	1	4	$2^{83.96}$	–	$2^{99.70}$
Inbound 4	0	8	8	1	$2^{11.91}$	–	$2^{16.00}$
	1	16	4	2	$2^{16.00}$	$2^{23.91}$	–
	2	32	2	2	$2^{24.18}$	$2^{32.09}$	–
	3	64	1	4	$2^{32.72}$	$2^{48.46}$	–
Inbound 5	0	8	32	1	$2^{11.91}$	–	$2^{16.00}$
	1	16	16	2	$2^{16.00}$	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	–	$2^{32.09}$
	3	64	4	4	$2^{32.72}$	$2^{48.26}$	–
	4	128	2	4	$2^{49.80}$	$2^{65.54}$	–
	5	256	1	4	$2^{83.96}$	$2^{99.70}$	–
Inbound 6	0	8	32	1	$2^{11.91}$	–	$2^{16.00}$
	1	16	16	2	$2^{16.00}$	$2^{23.91}$	–
	2	32	8	2	$2^{24.18}$	$2^{32.09}$	–
	3	64	4	4	$2^{32.72}$	–	$2^{48.46}$
	4	128	2	4	$2^{49.80}$	–	$2^{65.54}$
	5	256	1	4	$2^{83.96}$	–	$2^{99.70}$

[a] Check whether the pairs satisfy the desired input difference

Merging Inbound Phases

The remaining pairs at inbound i are stored on list L_i. Connecting the six lists is performed in three steps as follows:

1. Whenever a pair is obtained from set 2, we check whether it exists in L_3 or not. If it does, another check is done for L_1. Since we have $2^{23.44}$ and $2^{83.96}$ elements in lists 1 and 3 respectively, $2^{83.96}$ pairs passing the second inbound phase, and 32-bit and 128-bit conditions for the matches, the expected number of remaining pairs is $2^{23.44} \cdot 2^{-32} \cdot (2^{83.96} \cdot 2^{-128} \cdot 2^{83.96}) = 2^{31.36}$. We store these these pairs in list A.

2. Similarly, whenever a pair is obtained from set 5, we check whether it exists in L_6 or not. If it does, another check is done for L_4. Since we have $2^{32.72}$ and $2^{83.96}$ elements in lists 4 and 6 respectively, 2^{80} pairs passing the fifth inbound phase, and 32-bit and 128-bit conditions for the matches, the expected number of remaining pairs is $2^{32.72} \cdot 2^{-32} \cdot (2^{83.96} \cdot 2^{-128} \cdot 2^{83.96}) = 2^{40.64}$. We store these pairs in list B.

3. Last step is merging these sets A and B. We have $2^{31.36}$ elements in A and $2^{40.64}$ elements in B and 32 bits of condition. Therefore the total expected number of remaining pairs is $2^{31.36} \cdot 2^{-32} \cdot 2^{40.64} = 2^{40}$.

Improving the Complexity of Finding a Solution for the Differential Parts

We have described how to obtain the existing 2^{40} solutions for the differential part. We will describe here a better way of doing the inbounds, as proposed in [157, Sec.4.1]. This new technique allows us to reduce the previous complexity from $2^{99.70}$ time and $2^{83.96}$ memory to $2^{69.6}$ time and $2^{67.6}$ memory. As in our further analysis we will just use one solution (and not 2^{40}) for the differential part: we will adapt the values being able to finally reduce the complexity of this part of the attack to $2^{59.6}$ time and $2^{57.6}$ memory. This memory is the memory bottleneck of all the analyses presented in this chapter.

1. We consider the six inbounds as described in Section 4.5.1, with the difference that, for inbounds $2, 3, 5$ and 6 we will not perform the last step, but instead we obtain for each inbound $i \in \{2, 3, 5, 6\}$ two lists $L_{A,i}$ and $L_{B,i}$ as a result, each of size $2^{49.80}$ associated to half of the corresponding differential path. As mentioned before, we are only looking for one solution for the whole differential path. Then, instead of the $2^{49.80}$ existing solutions for each list, we can consider $2^{44.8}$ elements in each list.

2. First, we merge lists $L_{A,2}$ and $L_{A,3}$. We have 16-bit conditions on values and 16-bit conditions on differences. We obtain a new list $L_{A,23}$ of size $2^{44.8+44.8-32} = 2^{57.6}$. We do the same with $L_{B,2}$ and $L_{B,3}$ to obtain $L_{B,23}$. Note that this list does not need to be stored, as we can perform the following step whenever an element is found.

3. In order to find a whole solution for the differential part of inbounds 2 and 3, one pair of elements from $L_{A,23}$ and from $L_{B,23}$ still needs to satisfy the following conditions: 32 bits from the parts $L_{A,2}$ and $L_{B,3}$, 32 bits from $L_{B,2}$ and $L_{A,3}$, 3.91×4 bits from the step 5 of inbound 2 that we have not yet verified and 3.91×4 bits from step 5 of inbound 3 that is not yet verified either. Therefore, we have 95.28 bit conditions in total to merge $L_{A,23}$ and $L_{B,23}$. For each element in $L_{B,23}$ we can check with constant cost if the corresponding element appears in $L_{A,23}$ (it can be done by a lookup in a table, representing the differential transitions of L and next by a lookup in the list $L_{A,23}$ to see if the wanted elements appear; see [157] and Figure 4.9 for more details). When we find a good pair, we store it in the list L_{23} that contains about $2^{19.92}$ elements satisfying the differential part of rounds from 4 to 16. The cost of this step is then $2^{57.6+1}$ in time and $2^{59.6}$ in memory.

4. Do the same with inbounds 5 and 6, to obtain list L_{56} of size $2^{19.92}$, with a cost of $2^{57.6+1}$ in time and $2^{57.6}$ in memory.

5. Merge the solutions obtained in the first inbound with the ones in L_{23}, obtaining a new set L_{123} of size $2^{19.92+23.44-32} = 2^{11.36}$.

6. Merge the solutions obtained from Step 4 with list L_{56} obtaining a new one, L_{456} of size $2^{19.92+32.72-32} = 2^{20.64}$.

7. Finally, merging L_{123} and L_{456} gives $2^{11.36+20.64-32} = 1$ partial solution for the differential part of the path from round 0 to round 32.

The complexity of obtaining one partial solution for rounds from 0 to 32 is dominated by Steps $2-4$ of the algorithm. As a result, the complexity of matching the active bytes becomes $2^{59.6}$ time and $2^{57.6}$ memory.

4.5.2 Matching the Passive Bytes

In Figure 4.10, colored boxes denote the S-boxes whose values have already been fixed from the inbound phases. Note that we have not treated the passive bits yet (i.e., found the remaining values that would complete the path). We will propose a way of finding 2^{32} solutions that verify the path from rounds 4 to 26 with time complexity 2^{96} and memory complexity $2^{51.58}$. This can be done in three steps as follows:

1. (Rounds 10 to 14): The sets of groups of 8 bits denoted by a, b, c, d, e, f in round 14 are independent of each other in this part of the path. In round 10, 32 bits are already fixed for each of these sets (groups of 4 bits denoted by A, B, C, D, E, F). By using all possible values of the remaining 96 passive bits (32 bits not fixed from A, B, C, D, E, F plus 64 from the remaining state at round 10), we can easily compute a list of 2^{96} elements with cost 2^{96} that satisfy the 32 bit conditions for each of the groups.

2. (Rounds 14 to 20): In round 20, we have 256 bits (green S-boxes) whose values are fixed from the solutions of the second inbound phase. We can divide the state in round 19 (until the state in round 14) into 4 independent parts (m, n, o, p). In Figure 4.10, the fixed bits coming from round 20 are denoted by green lines and the ones of the first inbound phase are denoted in blue. Note that the three parts m, n, o are identical, while p is different since there are some differences and some additional fixed values in it.

   We fix the parts m and n to some values that satisfy all the conditions of the fixed bits in rounds 19 and 14. This can be done as follows: similar to what we have done in Step 1, we can divide the state of rounds $16-19$ (for each part separately) into four groups (x, y, z, u) such that they are independent of each other when computing forwards.

   In round 16, each group has 16 bits whose values have already been fixed and 48 bits of freedom. We see that each group affects only one fourth of the green lines (16 bits in total) in round 19. Therefore, there exist $2^{48-16} = 2^{32}$ possibilities for each group x, y, z, u but we just need one. This one can then be found with a cost of about 2^{16}.

3. (Merging) Each of the sets $L_a, \ldots, L_f$ has 2^{96} possible values from Step 1, and fixing m and n fixes 64 bits for each of them in round 14. This gives us in average $2^{96-64} = 2^{32}$ possible values for each set in the half of the state associated to o and p in round 14.

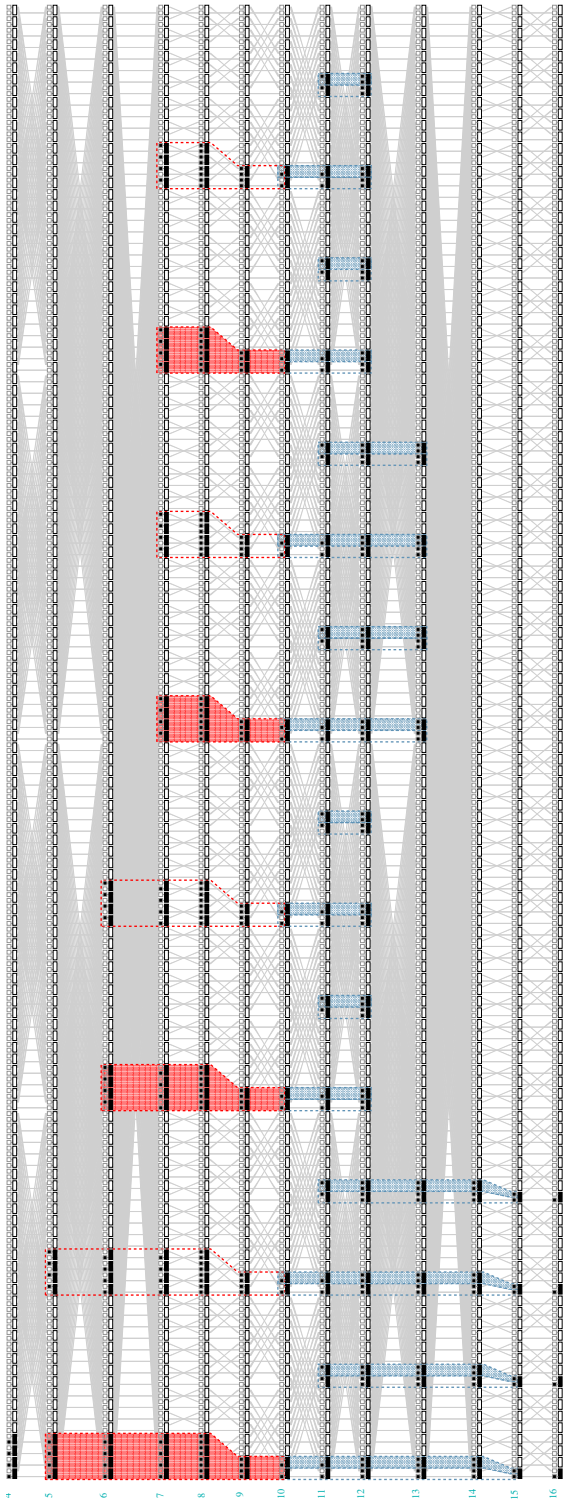


Figure 4.9: Improving the complexity of finding a solution for the differential part for rounds $4-16$ of JH42: Red boxes (above) denote the sets $L_{A,2}$ (filled) and $L_{B,2}$, blue boxes (below) denote the sets $L_{A,3}$ and $L_{B,3}$ (filled).

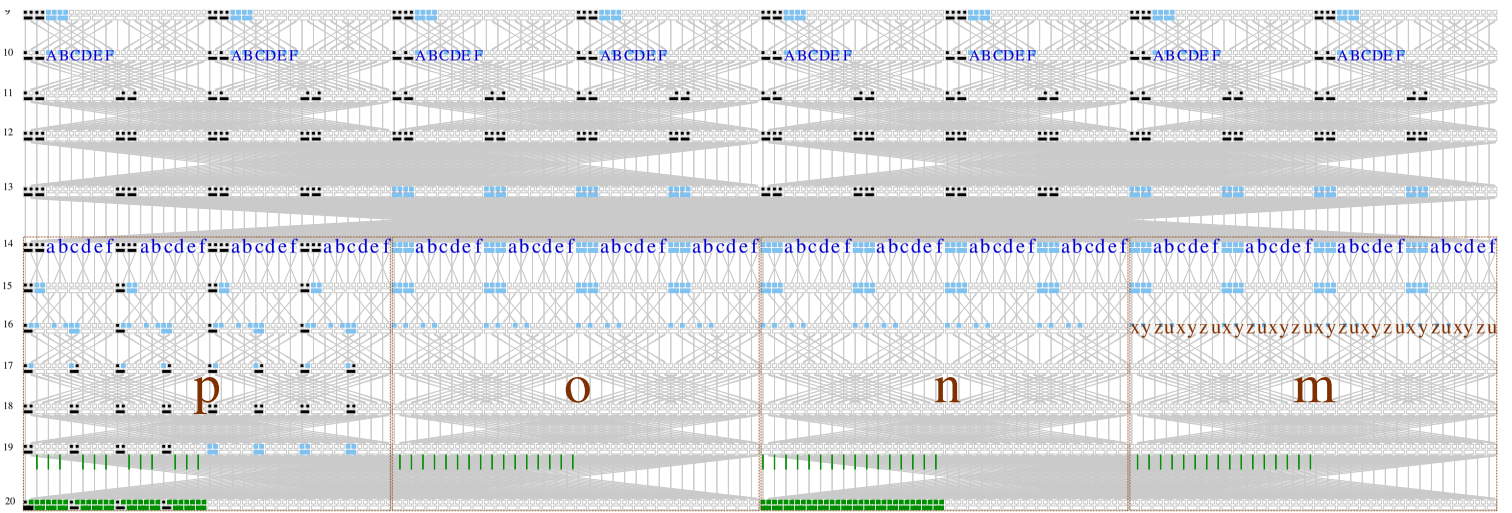


Figure 4.10: Matching the passive bytes of JH42 (bit-slice representation). Blue and green boxes denote the passive words whose values are fixed

For part p we use the same idea explained in Step 2. Group x is completely fixed due to the differential characteristic, and only the groups y, z, u have freedom, so there exists $(2^{32})^3 = 2^{96}$ possibilities. For each possibility, we compute the part of state in round 14 associated to p. We have 32 bits of condition for each of lists, and in average 2^{32} values are associated to each list. Thus, for each of the computed values, we will have only one remaining element that will determine the values at positions $a - f$ in part o.

Now, we have 2^{96} possible o values. The probability that a fixed value verifies the conditions of o in round 19 is $(2^{-4})^{16} = 2^{-64}$. Therefore, we obtain $2^{96-64} = 2^{32}$ solutions that verify the whole path from round 4 to round 26 with a time complexity of 2^{96}.

Note that we do not need to store the lists $L_a, \ldots, L_f$ of elements from round 14 each of size 2^{96}; we can instead store for each of them two lists of size 2^{48} corresponding to the upper and down halves of the corresponding groups in state 13. Then, when fixing a value of m and n we can check with a cost of 2^{32} which will be the list of 2^{32} values for o and p that we obtained in Step 3. Finally, we have obtained 2^{32} complete solutions for the path from 4 to 26 with a cost of 2^{96} in time, and $6 \cdot 2 \cdot 2^{48} \approx 2^{51.58}$ in memory.

Semi-free-start near-collisions up to 32 rounds

Up to now, we have found solutions for the passive bytes from rounds $4 - 26$. If we want a solution for the path from round 0 to round 26, we will have to repeat the previous procedure of matching the passive bytes 2^{16} times (as the probability of passing from round 0 to 4 is 2^{-48} and we have 2^{32} pairs). Then, we can find a solution for rounds $0 - 26$ with time complexity 2^{112}. In order to extend this result to 32 rounds, we have to repeat the previous procedure 2^{192} times (since we have 64 and 128 bits of condition from rounds 26 and 27 respectively). Therefore, the time complexity for finding a complete solution for rounds from 0 to 32 is $2^{112} \cdot 2^{192} = 2^{304}$.

Note that, we still have enough degrees of freedom. In Step 1, we started with 768 bits ($128 \cdot 6$ from the groups $a - f$) in round 14 and matched 192 bits ($32 \cdot 6$ for $A - F$) in round 10. In Step 2, we have 48 bits in round 16 coming from the fourth inbound phase and we matched another 240 bits from the fifth inbound phase in round 19. So in total we have $768 - 192 - 48 - 240 = 288$ bits of degrees of freedom remaining.

4.5.3 Outbound Phase

The outbound phase of the attack is composed of 5 rounds in the forward direction. A detailed schema of this trail is shown in Figure 4.11, and for the pairs that satisfy the inbound phase, we expect to see the following differential trail in the outbound phase:

$$\text{Inbound Phase} \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 8\,.$$

Semi-free-start near-collisions up to 37 rounds

For 32 rounds of the JH compression function, we obtain a semi-free-start near-collision for 1002 bits. We can simply increase the number of rounds by proceeding forwards in the outbound phase. Note that, we have an additional probability of $2^{-32} \cdot 2^{-16}$ coming from the eight filtering conditions in round 34 and the four filtering conditions in round 35. Thus, the complexity of the active part of the attack remains the same: $2^{59.6}$ time and $2^{57.6}$ memory. This is the case as one solution for the differential part is enough for the attack, as it will have different values at the bits with conditions in the outbound part when the passive part is modified. The complexity of the passive part becomes $2^{304} \cdot 2^{48} = 2^{352}$ time and $2^{51.58}$ memory.

The details can be seen in Table 4.6. We also take into account the colliding bits that we obtain at the output of the compression function after the final degrouping with the differences from the message.

Table 4.6: Comparison of complexity of the generic attack for near-collisions and our results on JH42

#Rounds	# Colliding bits	Generic Attack Complexity	Our Results
23	892	2^{231}	$2^{59.6}$
$24-26$	762	2^{99}	$2^{59.6}$ [a]
26	960	2^{342}	2^{112}
27	896	2^{236}	2^{112}
32	1002	2^{437}	2^{304}
33	986	2^{397}	2^{304}
34	954	2^{330}	2^{304}
35	986	2^{397}	2^{336}
36	1002	2^{437}	2^{352}
37	986	2^{397}	2^{352}
38	928	2^{285}	2^{352}

[a] Obtained directly from the solutions of the active part, without need of matching the passive bits

4.6 Distinguishers on JH42

Indifferentiability is considered to be a desirable property for any secure hash function design. Moreover, for many of the designs, the indifferentiability proofs for the mode of operation are based on the assumption that the underlying permutation (function) is ideal (i.e., a random permutation). This is the case for the indifferentiability proof of JH [24], that supposes that E_d is a random permutation.

In this section, we present a distinguisher for E_8 showing that it is distinguishable from a random permutation. Using the differential path that we presented in Section 4.5.1, we can build distinguishers for the full 42 rounds of the internal permutation E_8 with no additional complexity. As a result of our distinguisher, the proof from [24] does not apply to JH as the assumption that E_8 behaves as a random permutation does not hold. Next, we explain how these distinguishers for the internal permutation can be easily extended to distinguishers for the compression function.

There exists also a known trivial distinguisher for the construction of the compression function of JH: if the chaining value has a difference that can be canceled by the message block, then the output will have a difference directly related to the one coming from the message block. This implies that both the message and the chaining values have differences. Contrary to the trivial one, our compression function distinguisher exploits the properties of the internal permutation and only needs differences in the message or in the chaining value.

4.6.1 Distinguishers for the Reduced Round Internal Permutation

Let us remark here briefly that if we find solutions for rounds 4 to 20, and then let them spread freely backwards (difference in 64 bits) and forwards (difference in 256 bits), we can obtain a distinguisher for 26 rounds with a much lower complexity: $2^{59.6}$ time and $2^{57.6}$ memory (the cost of the differential part). As in this chapter the aim is reaching a higher number of rounds, we do not go further into the details.

4.6.2 Distinguishers for the Full Internal Permutation

In the previous sections we showed that a solution for 37 rounds can be obtained with a time complexity of 2^{352} and a memory complexity of $2^{57.6}$. In Figure 4.11, we see how these active words diffuse to the state after 42 rounds

with probability one. Therefore, before the degrouping operation we have 64 active and 192 passive words in the state. The number of active and passive bits still remains the same after the degrouping operation. It is important to remark that the positions of the active bits are fixed, also after the degrouping operation.

We can then build a distinguisher that will distinguish the 42-round permutation E_8 from a random permutation using this path. This distinguisher aims at finding a pair of input states (A, A') such that $E_8(A) \oplus E_8(A')$ collide in the 768 bits mentioned above. Let $A \oplus A' = \Delta_1$ correspond to the input difference of the differential path, then $|\Delta_1| = 8$ bits. Similarly, let $B = E_8(A)$ and $B' = E_8(A')$, then the output difference is $B \oplus B' = \Delta_2$ where $|\Delta_2| = 256$.

For a random function, we calculate the complexity of such a distinguisher as follows: We fix the values of the passive bits in the input; but not the ones of the active bits. Then, we have $2^{|\Delta_1|}$ possibilities for the values from the active bits. We compute the output of E_8 for each one of these values and store them in a list. From this list we can obtain $\binom{2^{|\Delta_1|}}{2}$ pairs with the given input difference pattern. The probability of satisfying the desired output difference pattern is $2^{|\Delta_2|-1024}$ for each pair, so we repeat the procedure with a new value for the input passive bits until we find a solution. The time complexity of finding such an input pair will be:

$$\frac{2^{|\Delta_1|}}{2^{(|\Delta_1|-1)} \cdot (2^{|\Delta_1|} - 1) \cdot 2^{|\Delta_2|-1024}} = 2^{761}.$$

Instead, in our case the complexity of finding such an input pair is the complexity of finding a solution for the path, that is 2^{352} time and $2^{57.6}$ memory.

Another distinguisher of E_8 can be built if we consider the scenario where the differential path for rounds $0 - 4$ does not need to be verified, i.e., $|\Delta_1| = 64$. In this case, we consider that from round 4 to 0 we obtain the differences that propagate with probability one. Therefore, the matching of the passive part does not need to be repeated 2^{208} times but only 2^{160} (as we do not need 2^{48} extra repetitions for verifying rounds 0 to 4). The complexity of this distinguisher will then be 2^{304}, and provides a pair of inputs A and A' that produce an output with 768 colliding bits as the ones represented in Figure 4.11. The complexity of such a generic distinguisher would be $\frac{2^{64}}{(2^{64}-1)\cdot 2^{63-768}} = 2^{705}$, while in our case is 2^{304} in time and $2^{57.6}$ in memory.

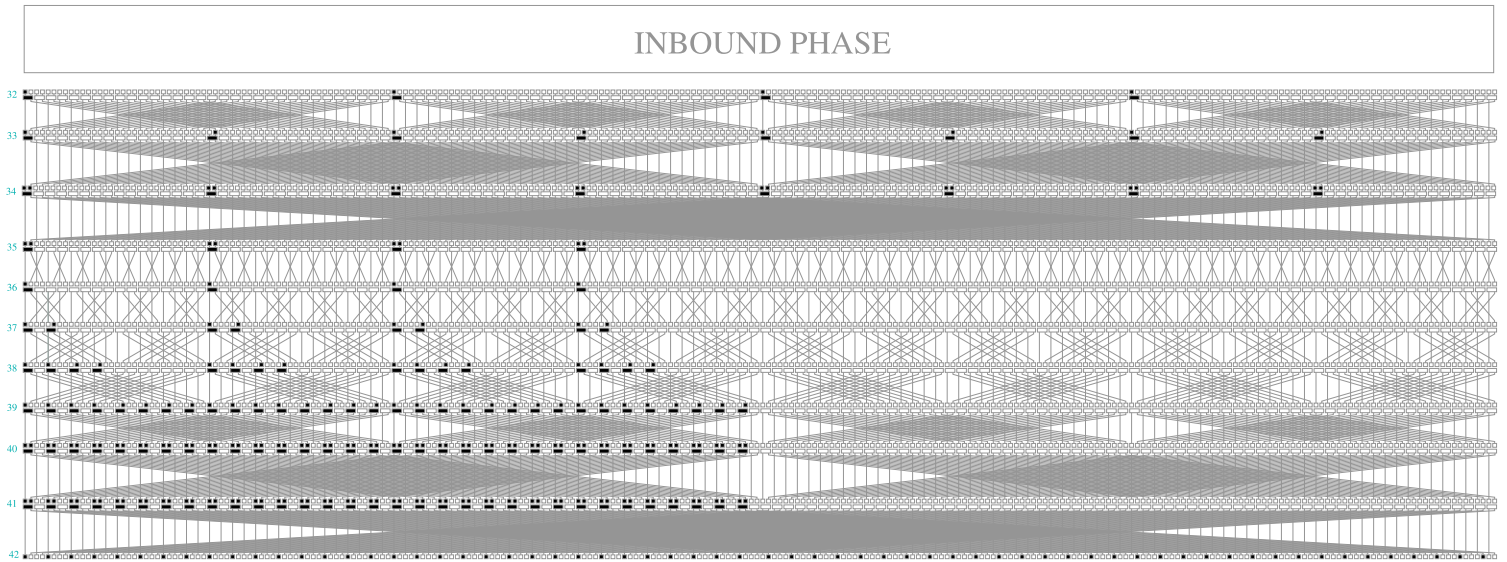


Figure 4.11: Differential characteristic for the outbound phase of JH Compression Function (bit-slice representation)

4.6.3 Distinguishers for the Full Compression Function

We should emphasize that our distinguishers for E_8 can be easily converted to a distinguisher for the full compression function of JH42. We only need to xor this message difference to the output of E_8 as specified.

For our first distinguisher, the input difference is already arranged such that we only have a difference in the message. These active bits coming from the message coincide with the active bits in the output at the xor operation. As a result, we have the same 768 passive bits. The same applies for our second distinguisher when we have differences only in the chaining value.

4.7 Contribution

The author has significant contributions to the security analysis of the attacks mentioned above. A semi-automatic program has been developed to find the characteristics used in the analysis. Furthermore, the author has implemented the attacks for the reduced versions of JH in C. The author is the principal author of the text (except Section 4.6). The names of the authors are given in alphabetical order for the publications.

4.8 Conclusion

In this chapter, we presented the first cryptanalysis results of JH using rebound attack techniques. We first explained our attack on 8 rounds of JH ($d = 4$) in detail and then showed how this attack can be used in order to cryptanalyze 16 rounds of JH hash function. We then presented a 1008-bit semi-free-start near-collision for 19 rounds of the JH compression function by using three inbounds.

We then improved our findings by using six inbound phases, and obtained a 960-bit semi-free-start near-collision for 26 rounds of the JH compression function which can be extended to 37 rounds by repeating the algorithm. Moreover, we have presented distinguishers on the full 42 rounds of the internal permutation E_8 of the tweaked SHA-3 finalist. To the best of our knowledge this is the first distinguisher on the internal permutation of JH faster than a generic attack. Our findings are summarized in Table 4.7.

Table 4.7: Comparison of best attack results on JH (sfs: semi-free-start)

target	rounds	time comp.	memory comp.	attack type	generic comp.	section
hash function	16	$2^{190.2}$	$2^{101.1}$	sfs collision	2^{256}	§4.3
hash function	16	$2^{96.1}$	$2^{96.1}$	sfs collision	2^{256}	[157]
comp. function	$19-22$	2^{168}	$2^{143.7}$	sfs near-collision	2^{236}	§4.4
comp. function	$19-22$	$2^{95.6}$	$2^{95.6}$	sfs near-collision	2^{236}	[157]
comp. function	26	2^{112}	$2^{57.6}$	sfs near-collision	2^{341}	§4.5
comp. function	32	2^{304}	$2^{57.6}$	sfs near-collision	2^{437}	§4.5
comp. function	36	2^{352}	$2^{57.6}$	sfs near-collision	2^{437}	§4.5
comp. function	37	2^{352}	$2^{57.6}$	sfs near-collision	2^{396}	§4.5
internal perm.	42	2^{304}	$2^{57.6}$	distinguisher	2^{705}	§4.6
internal perm.	42	2^{352}	$2^{57.6}$	distinguisher	2^{762}	§4.6

Chapter 5

SPONGENT

As technology is embedded in everyday objects (tools, devices, clothing, homes, healthcare, etc.), the need for security in RFID and sensor networks is dramatically increasing. As a result, lately lightweight cryptography – optimizing the algorithms to fit the most constrained environments – has received a great deal of attention. Previous research was mainly focused on building block ciphers; however, the design of lightweight hash functions is still far from being well-investigated with only few proposals.

During the last two years, some significant work on lightweight hash functions has also been performed: [37] describes ways of using the PRESENT block cipher in hashing modes of operation and [12] and [86] take the approach of designing a dedicated lightweight hash function based on a sponge construction [55, 22] resulting in two hash functions QUARK and PHOTON.

In this chapter, we propose a new family of sponge-based lightweight hash functions SPONGENT with a smaller footprint than most existing dedicated lightweight hash functions: PRESENT in hashing modes and QUARK. Its area is comparable to that of PHOTON.

SPONGENT [35] is a hermetic sponge with a PRESENT-type permutation. We refer to its various parameterizations as SPONGENT-$n/c/r$ for different hash sizes n, capacities c, and rates r. The group of all SPONGENT variants with the same output size of n bits is referred to as SPONGENT-n.

We propose five different hash sizes $n \in \{88, 128, 160, 224, 256\}$, covering most security applications in the field. The SPONGENT-88 functions are designed for extremely restricted scenarios and low preimage security requirements. They can be used e.g. in some RFID protocols and for PRNGs. SPONGENT-128 and SPONGENT-160 might be used in highly constrained applications with low and middle requirements for collision security. The latter also provides compatibility to the SHA-1 interfaces. The parameters of SPONGENT-224 and SPONGENT-256 correspond to those of a subset of SHA-2 and SHA-3 to make SPONGENT compatible to the standard interfaces in usual lightweight embedded scenarios.

5.1 Design Considerations for Lightweight Hashing

The footprint of a hash function is mainly determined by

1. the number of state bits (including the key schedule for block cipher based designs); as well as
2. the size of functional and control logic used in a round function.

Among the recent hash functions, QUARK, while using novel ideas of reducing the state size to minimize (1), does not appear to provide the smallest possible logic size, which is mainly due to the Boolean functions with many inputs used in its round transform. In contrast to that, SPONGENT keeps the round function very simple which reduces the logic size close to the smallest theoretically possible, thus, minimizing (2) and resulting in a significantly more compact design.

As shown in [37], using a lightweight block cipher in a hashing mode (single block length such as Davies-Meyer or double block length such as Hirose) is not necessarily an optimal choice for reducing the footprint, the major restriction being the doubling of the datapath storage requirement due to the feed-forward operation.

At the same time, no feed-forward is necessary for the sponge construction, which is the design approach of choice in this work. In a permutation-based sponge construction, let r be the *rate* (the number of bits input or output per one permutation call), c be the *capacity* (internal state bits not used for input or output), and n be the hash length in bits.

5.2 The Design of SPONGENT

SPONGENT is a sponge construction based on a wide PRESENT-type permutation. Given a finite number of input bits, it produces an n-bit hash value. A design goal for SPONGENT is to follow the hermetic sponge strategy (no structural distinguishers for the underlying permutation are allowed).

5.2.1 Permutation-based Sponge Construction

SPONGENT relies on a sponge construction – a simple iterated design that takes a variable-length input and can produce an output of an arbitrary length based on a permutation π_b operating on a state of a fixed number b of bits. The size of the internal state $b = r + c \geq n$ is called *width*, where r is the *rate* and c the *capacity*.

The sponge construction proceeds in three phases (see also Figure 2.6):

- **Initialization phase:** the message is padded by a single bit 1 followed by a necessary number of 0 bits up to a multiple of r bits (e.g., if $r = 8$, then the 1-bit message '0' is transformed to '01000000'). Then it is cut into blocks of r bits.
- **Absorbing phase:** the r-bit input message blocks are xored into the first r bits of the state, interleaved with applications of the permutation π_b.
- **Squeezing phase:** the first r bits of the state are returned as output, interleaved with applications of the permutation π_b, until n bits are returned.

In SPONGENT, the b-bit 0 is taken as the initial value before the absorbing phase. The message chunks are xored into the r rightmost bit positions of the state. The same r bit positions form parts of the hash output.

5.2.2 Parameters

We propose 13 variants of SPONGENT with five different hash output lengths at multiple security levels, see Table 5.1. We considered (up to) three types of preimage and second-preimage security levels:

Table 5.1: 13 SPONGENT variants

	n (bit)	b (bit)	c (bit)	r (bit)	R number of rounds	security(bit) pre.	2nd pre.	col.
SPONGENT-88/80/8	88	88	80	8	45	80	40	40
SPONGENT-88/176/88	88	264	176	88	135	88	88	44
SPONGENT-128/128/8	128	136	128	8	70	120	64	64
SPONGENT-128/256/128	128	384	256	128	195	128	128	64
SPONGENT-160/160/16	160	176	160	16	90	144	80	80
SPONGENT-160/160/80	160	240	160	80	120	80	80	80
SPONGENT-160/320/160	160	480	320	160	240	160	160	80
SPONGENT-224/224/16	224	240	224	16	120	208	112	112
SPONGENT-224/224/112	224	336	224	112	170	112	112	112
SPONGENT-224/448/224	224	672	448	224	340	224	224	112
SPONGENT-256/256/16	256	272	256	16	140	240	128	128
SPONGENT-256/256/128	256	384	256	128	195	128	128	128
SPONGENT-256/512/256	256	768	512	256	385	256	256	128

- **Full preimage and second-preimage security.** The standard security requirements for a hash function with an n-bit output size are collision resistance of $2^{n/2}$ as well as preimage and second-preimage resistance of 2^n. For this, in SPONGENT, we set $r = n$ and $c = 2n$ to obtain SPONGENT-88/176/88, SPONGENT-128/256/128, SPONGENT-160/320/160, SPONGENT-224/448/224, and SPONGENT-256/512/256.

- **Reduced second-preimage security.** In most embedded scenarios, where a lightweight hash function is likely to be used, the full second-preimage security is not a necessary requirement. By setting $n \approx c$ and r small in a permutation-based sponge, the preimage and second-preimage resistances are reduced to 2^{n-r} and $2^{c/2}$ respectively, while the collision resistance remains at $2^{c/2}$. We use this approach to obtain SPONGENT-88/80/8, SPONGENT-128/128/8, SPONGENT-160/160/16, SPONGENT-224/224/16, and SPONGENT-256/256/16.

- **Reduced preimage and second-preimage security.** In some applications, the collision security is of concern only and one can abandon the requirement of preimage security to be close to 2^n. In a permutation-based sponge, going for $c = n$ and $r = n/2$, results in the reduction of both the preimage security and second-preimage security to $2^{n/2}$, while maintaining the full collision security of $2^{n/2}$. We use this approach

in the design of SPONGENT-160/160/80, SPONGENT-224/224/112, and SPONGENT-256/256/128.

5.2.3 PRESENT-type Permutation

The permutation $\pi_b : \mathbb{F}_2^b \to \mathbb{F}_2^b$ is an R-round transform of the input STATE of b bits that can be outlined at a top-level as:

for $i = 1$ to R **do**
  STATE $\leftarrow$ reverse(lCounter$_b(i)$) $\oplus$ STATE $\oplus$ lCounter$_b(i)$
  STATE $\leftarrow$ sBoxLayer$_b$(STATE)
  STATE $\leftarrow$ pLayer$_b$(STATE)
end for

where sBoxLayer$_b$ and pLayer$_b$ describe how the STATE evolves. For ease of design, only widths b with $4|b$ are allowed. lCounter$_b(i)$ is the state of an LFSR dependent on b at time i which yields the round constant in round i and is added to the rightmost bits of STATE. reverse(lCounter$_b(i)$) is the value of lCounter$_b(i)$ with its bits in reversed order and is added to the leftmost bits of STATE.

The following building blocks are generalizations of the PRESENT structure to larger b-bit widths:

1. sBoxLayer$_b$: This denotes the use of a 4-bit to 4-bit S-box $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ which is applied $b/4$ times in parallel. The action of the S-box in hexadecimal notation is given by the following table:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	E	D	B	0	2	1	4	F	7	A	8	5	9	C	3	6

2. pLayer$_b$: This is an extension of the (inverse) PRESENT bit-permutation and moves bit j of STATE to bit position $P_b(j)$, where

$$P_b(j) = \begin{cases} j \cdot b/4 \mod b-1, & \text{if } j \in \{0,\ldots,b-2\} \\ b-1, & \text{if } j = b-1. \end{cases}$$

and can be seen in Figure 5.1.

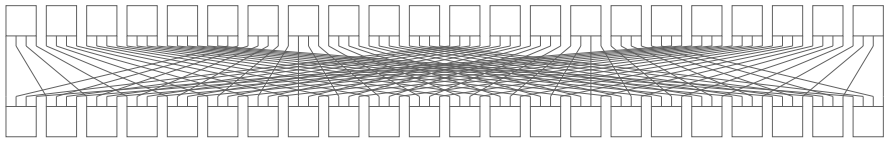


Figure 5.1: The bit permutation layer of SPONGENT-88

3. lCounter$_b$: This is one of the four $\lceil \log_2 R \rceil$-bit LFSRs. The LFSR is clocked once every time its state has been used and its final value is all ones. If ζ is the root of unity in the corresponding binary finite field, the n-bit LFRSs defined by the polynomials given below are used for the SPONGENT variants.

LFSR size (bit)	Primitive Polynomial
6	$\zeta^6 + \zeta^5 + 1$
7	$\zeta^7 + \zeta + 1$
8	$\zeta^8 + \zeta^4 + \zeta^3 + \zeta^2 + 1$
9	$\zeta^9 + \zeta^4 + 1$

Table 5.2 provides sizes and initial values of all the LFSRs.

Table 5.2: Initial values of lCounter$_b$ for all SPONGENT variants

	LFSR size (bit)	Initial Value (hex)
SPONGENT-88/80/8	6	05
SPONGENT-88/176/88	8	D2
SPONGENT-128/128/8	7	7A
SPONGENT-128/256/128	8	FB
SPONGENT-160/160/16	7	45
SPONGENT-160/160/80	7	01
SPONGENT-160/320/160	8	A7
SPONGENT-224/224/16	7	01
SPONGENT-224/224/112	8	52
SPONGENT-224/448/224	9	105
SPONGENT-256/256/16	8	9E
SPONGENT-256/256/128	8	FB
SPONGENT-256/512/256	9	015

5.2.4 Design Rationale

The overall design approach for SPONGENT is to target low area while favoring simplicity.

The 4-bit S-box fulfills the PRESENT design criteria in terms of differential and linear properties [36]. Moreover, any linear approximation over the S-box

involving only single bits both in the input and output masks is unbiased. This aims to restrict the linear hull effect discovered in round-reduced PRESENT.

The function of the bit permutation pLayer is to provide good diffusion, by acting together with the S-box, while having a limited impact on the area requirements. The counters lCounter and ɹǝʇuʍoC are mainly aimed to prevent sliding properties and make prospective cryptanalysis approaches using properties like invariant subspaces [126] more involved.

The structures of the bit permutation and the S-box in SPONGENT make it possible to prove the following differential property (see Section 5.3.1 for the proof):

Theorem 1. *Any 5-round differential characteristic of the underlying permutation of* SPONGENT *with* $b \geq 64$ *has a minimum of* 10 *active S-boxes. Moreover, any 6-round differential characteristic of the underlying permutation of* SPONGENT *with* $b \geq 256$ *has a minimum of* 14 *active S-boxes.*

An important property of the SPONGENT S-box is that its maximum differential probability is 2^{-2}. This fact and the assumption of the independence of difference propagation in different rounds yield an upper bound on the differential characteristic probability of 2^{-20} over 5 rounds and of 2^{-28} over 6 rounds for $b \geq 256$ which follows from the claims of Theorem 1.

Theorem 1 is used to determine the number R of rounds in the permutation π_b: R is chosen in a way that π_b provides at least b active S-boxes. Other types of analysis are performed in the next section.

5.3 Security Analysis

In this section, we discuss the security of SPONGENT against known cryptanalytic attacks by applying the most important state-of-the-art methods of cryptanalysis and investigating their complexity.

5.3.1 Resistance Against Differential Cryptanalysis

Here we analyze the resistance of SPONGENT against differential attacks where Theorem 1 plays a key role providing a lower bound on the number of active S-boxes in a differential characteristic. The similarities of the SPONGENT permutations and the basic PRESENT cipher allow to reuse some of the results obtained for PRESENT in [36]. More precisely, the results on the number of

Table 5.3: Differential characteristics with lowest numbers of differentially active S-boxes (ASN). The probabilities are calculated assuming the independency of round computations.

# of	SPONGENT-88/80/8		SPONGENT-128/128/8		SPONGENT-160/160/16		SPONGENT-224/224/16		SPONGENT-160/160/80	
rounds	ASN	Prob	ASN	Prob	ASN	Prob	ASN	Prob	ASN	Prob
5	10	2^{-21}	10	2^{-22}	10	2^{-21}	10	2^{-21}	14	2^{-21}
10	20	2^{-47}	24	2^{-60}	20	2^{-50}	20	2^{-43}	32	2^{-43}
15	30	2^{-74}	40	2^{-101}	30	2^{-79}	30	2^{-66}	52	2^{-66}
# of	SPONGENT-88/176/88		SPONGENT-128/256/128		SPONGENT-160/320/160		SPONGENT-224/224/112		SPONGENT-224/448/224	
rounds	ASN	Prob	ASN	Prob	ASN	Prob	ASN	Prob	ASN	Prob
6	14	2^{-28}	14	2^{-28}	14	2^{-28}	14	2^{-28}	14	2^{-28}
12	41	2^{-96}	37	2^{-72}	39	2^{-93}	36	2^{-88}		
18	64	2^{-158}	52	2^{-119}	65	2^{-157}	66	2^{-174}		
# of	SPONGENT-256/256/16		SPONGENT-256/256/128		SPONGENT-256/512/256					
rounds	ASN	Prob	ASN	Prob	ASN	Prob				
6	14	2^{-28}	14	2^{-28}	14	2^{-28}				
12	32	2^{-73}	50	2^{-123}	34	2^{-84}				
18	52	2^{-128}	68	2^{-169}	54	2^{-128}				

differentially active S-boxes over 5 and 6 rounds will hold for all respective SPONGENT variants which is reflected in Theorem 1. The proof of Theorem 1 is as follows:

Proof. [*Theorem 1*] The statements for SPONGENT variants with $64 \leq b \leq 255$ can directly be proven by applying the same technique used in [36, Appendix III]. The proof of the 6-round bounds for SPONGENT variants with $b \geq 256$ in Theorem 1 is based on some extended observations. Here, we will only give the proof for when the width, b, is a multiple of 64 bits, i.e., $b = 64n$. The proof for other b values can also be obtained by making use of the observations given below. Since the proof is specific to each b and hence more tedious, we do not present them here.

We obtain n groups and $4n$ subgroups by calling each four consecutive S-boxes a *subgroup* and each sixteen consecutive S-boxes a *group*. To be more specific: subgroup i is comprised of the S-boxes $[4(i-1)\ldots 4i-1]$ and similarly group j has the subgroups $[4(j-1)\ldots 4j-1]$ (see Figure 5.2). By examining the substitution and linear layers, one can make the following observations:

1. The S-box of SPONGENT is such that a difference in a single input bit causes a difference in at least two output bits or vice versa.
2. The input bits to an S-box come from four distinct S-boxes of the same subgroup.
3. The input bits to a subgroup of four S-boxes come from 16 distinct S-boxes of the same group.
4. The input bits to a group of 16 S-boxes come from 64 different S-boxes.
5. The four output bits from a particular S-box enter four distinct S-boxes, each of which belongs to a distinct group of S-boxes in the subsequent round.
6. The output bits of S-boxes in distinct groups go to distinct S-boxes in distinct subgroups.
7. The output bits of S-boxes in distinct subgroups go to distinct S-boxes.

For the latter statement (SPONGENT-256), one has to deal with more cases. Consider six consecutive rounds of SPONGENT ranging from i to $i+5$ for $i \in [1\ldots 155]$. Let D_j be the number of active S-boxes in round j. If $D_j \geq 3$, for $i \leq j \leq i+5$, then the theorem trivially holds. So let us suppose that one of the D_j's is first equal to one and then to two. We have the following cases:

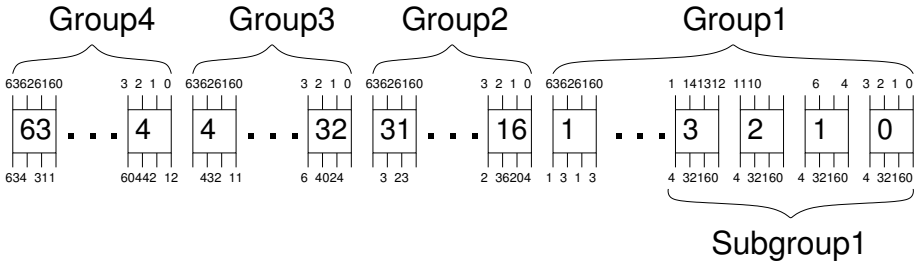


Figure 5.2: The grouping and subgrouping of S-boxes for $b = 256$. The input numbers indicate the S-box origin from the previous round and the output numbers indicate the destination S-box in the following round.

Case $D_{i+2} = 1$**.** By using observation 1, we can deduce that $D_{i+1} + D_{i+3} \geq 3$ and all active S-boxes of round $i + 1$ belong to the same subgroup from observation 2. Each of these active S-boxes has only a single bit difference in their output. So, according to observation 3 we have that $D_i \geq 2D_{i+1}$. Conversely, according to observation 5, all active S-boxes in round $i+3$ belong to distinct groups and have only a single bit difference in their input. So, according to observation 6, we have that $D_{i+4} \geq 2D_{i+3}$. Moreover, all active S-boxes in round $i + 4$ belong to distinct subgroups and have only a single bit difference in their input. Thus, by using observation 7, we obtain that $D_{i+5} \geq 2D_{i+4}$ and can conclude that $\sum_{j=i}^{i+5} D_j \geq 1 + 3 + 2 \cdot 3 + 4D_{i+3} \geq 14$.

Case $D_{i+3} = 1$**.** If $D_{i+2} = 1$ we can refer to the first case. So, suppose that $D_{i+2} \geq 2$. According to observation 2, all active S-boxes of round $i + 2$ belong to the same subgroup and each of these active S-boxes has only a single bit difference in their output. Thus, according to observation 3, $D_{i+1} \geq 2D_{i+2} \geq 4$. Since all active S-boxes in round $i + 1$ belong to distinct S-boxes of the same group and have only a single bit difference in their input, according to observation 4, we have that $D_i \geq 2D_{i+1}$. D_{i+4} and D_{i+5} can get one and two as a minimum value, respectively. Together this gives $\sum_{j=i}^{i+5} D_j \geq 8 + 4 + 2 + 1 + 1 + 2 \geq 18$.

Case $D_{i+1} = 1$**.** If $D_{i+2} = 1$, then we can refer to the first case. Thus, suppose that $D_{i+2} \geq 2$. According to observation 5, all active S-boxes in round $i+2$ belong to distinct groups and have only a single bit difference in their input. Thus, according to observation 6, we have that $D_{i+3} \geq 2D_{i+2}$. Note that all active S-boxes in round $i+3$ belong to distinct subgroups and have only a single bit difference in their input. Therefore, according to observation 7, we have that

$D_{i+4} \geq 2D_{i+3}$. To sum up, $\sum_{j=i}^{i+5} D_j \geq 1+1+2+4+8+D_{i+5} \geq 16+D_{i+5} \geq 17$, since $D_{i+4} > 0$ implies that $D_{i+5} \geq 1$.

Case $D_{i+4} = 1$**.** If $D_{i+3} = 1$, then we can refer to the second case. So, suppose that $D_{i+3} \geq 2$. According to observation 2, all active S-boxes of round $i+3$ belong to the same subgroup and each of those active S-boxes has only a single bit difference in their output. Therefore, according to observation 3, we have that $D_{i+2} \geq D_{i+3}$. Since, all active S-boxes in round $i+2$ belong to distinct S-boxes of the same group and have only a single bit difference in their input, according to observation 4, we have that $D_{i+1} \geq 2D_{i+2}$. Since $D_{i+1} > 0$, $D_i \geq 1$. Thus, we can conclude that $\sum_{j=i}^{i+5} D_j \geq D_i+8+4+2+1+1 \geq D_i+16 \geq 17$.

Cases $D_i = 1$ and $D_{i+5} = 1$ are similar to those for the third and fourth cases.

So far we have considered all paths including one active S-box in one of the rounds and obtained 14 as the minimum number of active S-boxes. But if there exists a path that has two active S-boxes in each round, then the lower bound would be 12. For this purpose, without loss of generality, assume:

$D_{i+1} = D_{i+2} = D_{i+3} = 2$ The two active S-boxes in $i+2$ are either in the same subgroup (i) or in different subgroups (ii).

For (i), from observations 3 and 7, we know that they have single bit of differences coming from two different subgroups of the same group in round $i+1$. From observation 1, these two S-boxes have at least two bits of input difference, hence we obtain $D_i = 4$ by observation 2 and 3. Furthermore the two S-boxes in round $i+2$ have two bits of output difference by observation 1. Hence, in round $i+3$, the active S-boxes have two bits of input and they are in distinct groups by observation 5. Therefore, it is possible to have $D_{i+4} = 2$ in distinct subgroups. Hence by using observation 7, we obtain $D_{i+5} = 4$. Thus, we can conclude that $\sum_{j=i}^{i+5} D_j \geq 4+2+2+2+2+4 \geq 16$.

For (ii), the two active S-boxes in round $i+1$ must have two bits of input and by observation 2 their input bits should be coming from distinct S-boxes in the same subgroup. So, the problem is reduced to the former case with shifted over one round, and we can immediately say that $D_i = 2$ and $D_{i+4} = 4$. Hence by using observation 7, we obtain $D_{i+5} = 4$. Thus, we can conclude that $\sum_{j=i}^{i+5} D_j \geq 2+2+2+2+4+4 \geq 16$.

Based on these results, we conclude that the longest run with two active S-boxes in each round is four rounds, and the number of active S-boxes cannot be less than 14. □

For all SPONGENT variants, we found that those 5- and 6-round bounds are actually tight. We present the characteristics attaining them in Table 5.3. Additionally, we perform a branch-and-bound search for longest characteristics with probabilities in the range of 2^{-b}. The results are given in Table 5.4, most of them based on iterative characteristics.

Table 5.4: The longest characteristics obtained for all SPONGENT variants with probability in the range of 2^{-b}

	# rounds	ASN	Prob
SPONGENT-88/80/8	17	34	2^{-88}
SPONGENT-88/176/88	27	103	2^{-268}
SPONGENT-128/128/8	20	56	2^{-137}
SPONGENT-128/256/128	42	146	2^{-385}
SPONGENT-160/160/16	20	66	2^{-179}
SPONGENT-160/160/80	44	88	2^{-242}
SPONGENT-160/320/160	48	192	2^{-480}
SPONGENT-224/224/16	44	88	2^{-242}
SPONGENT-224/224/112	26	133	2^{-343}
SPONGENT-224/448/224	-	-	-
SPONGENT-256/256/16	30	108	2^{-276}
SPONGENT-256/256/128	31	150	2^{-392}
SPONGENT-256/512/256	85	256	2^{-768}

5.3.2 Collision Attacks

A natural approach to obtain a collision for a sponge construction is to inject a difference in a message block and then cancel the propagated difference by a difference in the next message block, i.e., $(0 \dots 0 || \Delta m_i) \xrightarrow{\pi} (0 \dots 0 || \Delta m_{i+1})$. For this purpose, we follow a narrow trail strategy using truncated differential characteristics. We start from a given input difference (some difference restricted to S-boxes that the message block is xored into) and look for all paths that go to a fixed output difference (also located in the bitrate part of the state). Based on our experiments, even by using truncated differential characteristics, the probability of such a path is quite low and it is not possible to attack the full number of rounds.

Rebound Attack

Compared to the other algorithms the rebound attack has been successfully applied to, the design of SPONGENT imposes some limitations. First of all, since the permutation is bit-oriented, and not byte-oriented, it might be non-trivial to find the path followed by a given input difference and to determine the number of active S-boxes after several rounds. This is mainly due to the difference propagation that strongly depends on the values of the passive part of the state. Moreover, the probability that two inbound phases match requires more detailed analysis. Below we attempt to develop rebound attacks on several SPONGENT variants. Rebound analysis applies similarly to the remaining variants.

For SPONGENT-88/80/8, we looked for characteristics that match in the middle with the available degrees of freedom coming from the message bits. For 5 and more rounds, when the whole state is active in the matching phase, we would not be able to generate enough pairs by using only a difference in the message bits. Since the expected probability of matching the inbound phases is $2^{-b/4}$ (where $b/4$ is the number of S-boxes) and the available degree of freedom is only 2^{2r}, this argument is also valid for SPONGENT-128/128/8, SPONGENT-160/160/16, SPONGENT-224/224/16, and SPONGENT-256/256/16. For other SPONGENT variants there exist enough degrees of freedom and we decided to explore it with one of the SPONGENT variants.

It is trivial to find one round inbound phase in SPONGENT and then by applying the outbound phase for several rounds, which technically yields a differential characteristic. Since, one third of the state is xored with the message value for the variants whose rate is different from 8 or 16, we have enough flexibility to diffuse the difference through forward and backward direction. But then, merging these differential characteristics seems difficult due to the limited number of pairs generated in the inbound phase.

In our example which is given in Figure 5.3, we focused on SPONGENT-128/256/128 and found a five-round trail by following the strategy outlined above. In our attack, we fix the input and output differences of sBoxLayer in the fourth round. For half of the differences, we fix the difference to $1_x \rightarrow 3_x$ and for the other half it is possible to fix the difference to either $4_x \rightarrow 3_x$ or $8_x \rightarrow 3_x$, but not both together. Then, we let the differences diffuse for three rounds in the backward direction and for one round in the forward direction. All possible positions of the active bits are shown in black in Figure 5.3. Note that in round 5, we impose a restriction on the outputs of the SBoxLayer such that the differences occur only in the bitrate part.

It is possible to generate $4^{11} \cdot 2^{11} = 2^{33}$ pairs in the inbound phase and a

pair can satisfy the desired differential trail with a probability of $Pr[B_x \rightarrow \{1_x, 2_x\}]^6 \cdot Pr[D_x \rightarrow \{1_x, 2_x, 3_x\}]^6 \cdot Pr[6_x \rightarrow \{1_x, 2_x\}]^4 = 2^{-26.15}$. Therefore, in total, we expect to have $2^{6.85}$ valid pairs that satisfy the given path.

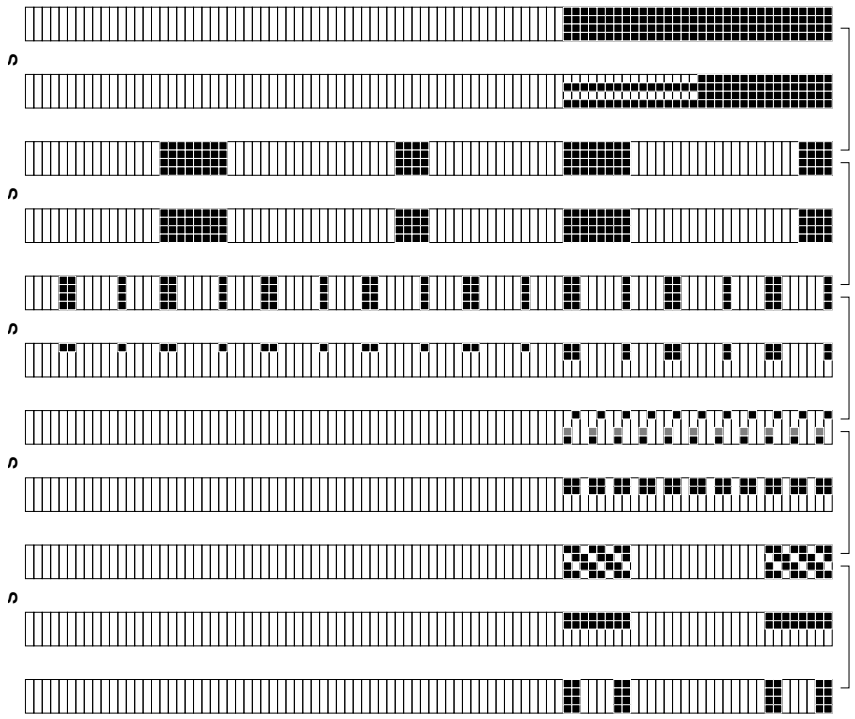


Figure 5.3: Differential path for the rebound attack on SPONGENT-128/256/128 (S: sBoxLayer_{384}, P: pLayer_{384})

Bound considerations for the rebound attack

The adversary might try to find an attack that uses multiple inbounds with a sparse differential. Therefore, to explore the security against multiple inbound phases, we put the adversary into a best-case scenario as follows.

We know that there exists no differential characteristic over five rounds with the number of active S-boxes less than 10 for all SPONGENT variants. We can also deduce lower bounds on the number of active S-boxes for $1, 2, 3$, and 4 rounds as $1, 2, 4$ and 6, respectively. Then a bound on the minimum number

of active S-boxes, hence the probability of a differential characteristic, for any number of rounds can be approximated by combining these bounds.[1]

The desired bit security level for a sponge construction with respect to collision attacks is $c/2$. From now on we assume that the complexity of each inbound phase is equal to $c/2$ and at least one active S-box matches between two inbound phases (with probability 2^{-8}). Let n_{in} be the number of inbound phases then we have to generate $n_{elm} = 2^{8\cdot(n_{in}-1)/n_{in}}$ elements for each inbound phase. Let p denote the probability of each inbound phase, then p can be at least $2^{-(c/2-\lceil log_2(n_{elm})\rceil)}$ and we can compute the number of rounds in each inbound phase by using the bounds given above.

Under these assumptions, the maximum number of rounds per inbound phase and the percentage of the total number of rounds attacked is given in Table 5.5.

Table 5.5: Bounds for rebound attack

	2 Inbounds		3 Inbounds	
	rounds /inbound	attacked rounds(%)	rounds /inbound	attacked rounds(%)
SPONGENT-88/80/8	9	40.00	9	60.00
SPONGENT-88/176/88	10	14.81	9	20.00
SPONGENT-128/128/8	15	42.86	14	60.00
SPONGENT-128/256/128	14	14.36	13	20.00
SPONGENT-160/160/16	19	42.22	19	63.33
SPONGENT-160/160/80	19	31.67	19	47.50
SPONGENT-160/320/160	17	14.17	16	20.00
SPONGENT-224/224/16	28	46.67	27	67.50
SPONGENT-224/224/112	23	27.06	23	40.59
SPONGENT-224/448/224	23	13.53	23	20.29
SPONGENT-256/256/16	28	40.00	27	57.86
SPONGENT-256/256/128	28	28.72	27	41.54
SPONGENT-256/512/256	28	14.55	27	21.04

5.3.3 Preimage Resistance

Here we apply a meet-in-the-middle approach to obtain preimages on SPONGENT. The attack has two main steps: pre-computation and matching phase. The complexity of the attack is dominated by the pre-computation phase.

[1] Note that, Table 5.3 shows that these bounds might be optimistic.

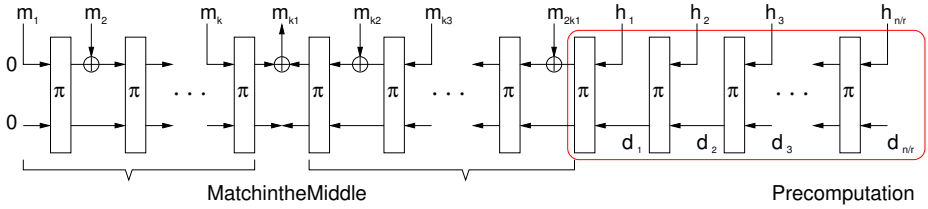


Figure 5.4: Meet-in-the-middle attack against sponge construction

Since the hash size is n bits, and the data is extracted in r-bit chunks, there exists n/r rounds in the squeezing phase. To be able to compute the data backwards in the absorbing phase, we need to know not only the h_i's but also d_i values to obtain the input value of the permutation π, where h_i denotes the part of the hash value and d_i is the concatenated part to h_i. The algorithm is as follows:

1. **Pre-computation:** We know that $\pi^{-1}(h_{i+1}, d_{i+1}) = (h_i, d_i)$ for each i in the squeezing phase. Since h_i (r-bits) is already fixed, the probability of finding such d_i is 2^{-r}. Therefore, we start with $2^{((n/r)-1)\cdot r} = 2^{n-r}$ different $d_{n/r}$ values to have a solution for d_1.
2. **Match-in-the-middle:** Choose k such that $k \cdot r \geq c/2$. Then
   - Generate $2^{c/2}$ elements in the backward direction by using (h_1, d_1) and possible values for $m_{k+2}, \ldots, m_{2k+1}$ and store them in a table.
   - Generate $2^{c/2}$ elements in the forward direction by using possible values for $m_1, \ldots, m_k$ and compare with list in the previous step to find a match of c bits (corresponding to capacity) in the middle.
   - Obtain m_{k+1} by xor-ing the r bits (corresponding to bitrate) for the matching elements.

In the pre-computation part, we obtain the required value d_1 to compute the data backwards in the absorbing phase by 2^{n-r} computations. We need $2^{c/2}$ memory to store the elements generated in the second step and $2^{c/2}$ computations are needed to find a full match. These complexities are exactly given in [193] which extends the bounds given in [55] for $c > n$. We have derived those once again here for completeness. The preimage attack complexities together with the parameter k are given in Table 5.6.

Note that, if $c \leq n - r$, it is sufficient to try all possible 2^c values to construct the whole state in order to obtain a preimage, hence it provides

Table 5.6: Meet-in-the-middle attack results for SPONGENT

	k	Time Complexity $max(2^{n-r}, 2^{c/2})$	Memory Complexity $(2^{c/2})$
SPONGENT-88/80/8	5	2^{80}	2^{40}
SPONGENT-88/176/88	1	2^{88}	2^{88}
SPONGENT-128/128/8	8	2^{120}	2^{64}
SPONGENT-128/256/128	1	2^{128}	2^{128}
SPONGENT-160/160/16	5	2^{144}	2^{80}
SPONGENT-160/160/80	1	2^{80}	2^{80}
SPONGENT-160/320/160	1	2^{160}	2^{160}
SPONGENT-224/224/16	7	2^{208}	2^{112}
SPONGENT-224/224/112	1	2^{112}	2^{112}
SPONGENT-224/448/224	1	2^{224}	2^{224}
SPONGENT-256/256/16	8	2^{240}	2^{128}
SPONGENT-256/256/128	1	2^{128}	2^{128}
SPONGENT-256/512/256	1	2^{256}	2^{256}

an upper bound for the preimage resistance. If we combine the results we obtain $max(2^{min(n-r,c)}, 2^{c/2})$ and it can be generalized into the form: $min(2^{min(n,\ c+r)}, max(2^{min(n-r,\ c)}, 2^{c/2}))$. Here, $2^{min(n,\ c+r)}$ computations will be necessary depending on the permutation size when the generic attack, defined above, fails.

5.3.4 Linear Attacks

The most successful attacks, in terms of number of rounds cryptanalyzed, for the block cipher PRESENT are those based on linear approximations. In particular the multi-dimensional linear attack [46] and the statistical saturation attack [47] claim to break up to 26 rounds. It was shown in [125] that both attacks are closely related. Moreover, the main reason why these attacks are the most successful attacks on PRESENT so far, is the existence of many linear trails with only one active S-box in each round. It is not immediately clear how linear distinguishers on the SPONGENT permutation π_b could be transferred into collision or (second) pre-image attacks on the hash function. However, as we claim that SPONGENT is a hermetic sponge construction, the existence of such distinguishers has to be excluded. So the SPONGENT S-box was chosen in a way that allows for at most one trail with this property given a linear approximation.

Unlike for the block cipher PRESENT, where the key determines the actual linear correlation between an input and an output mask, for the permutation π_b we can compute the actual linear trail contribution for all trails with only one active S-box in every round. Each such trail over w rounds has a correlation of $\pm 2^{-2w}$ and for each trail determining the sign is easy. More concretely, one can easily compute a $b \cdot b$ matrix M_t over the rationals such that the entry at position i, j is the correlation coefficient for round t for the linear trail with input mask e_i and output mask e_j. Here e_i (resp. e_j) is the unit vector with a single 1 at position i (resp. j). Note that the matrices M_t are sparse and all very similar, the only difference is caused by the round constant, which induces sign changes at a few positions only.

Given those matrices, it is now possible to compute the maximal linear correlation contribution for those one bit intermediate masks for all one bit input and output masks. For w rounds we simply compute $M^{(w)} = \prod_{i=1}^{w} M_i$ and the maximal correlation is given by $c_w := \max_{i,j} |M_{ij}^{(w)}|$. We compute this value for all SPONGENT variants. Table 5.7 summarizes those results. Most importantly, this table shows the maximal number of rounds w where the trail contributions is still larger than or equal to $2^{-b/2}$. Beyond this number of rounds, it seems unlikely that distinguishers based on linear approximations exist. For most SPONGENT variants, the best linear hull based on single-bit masks has exactly one linear trail.

5.4 Contribution

The names of the authors are given in alphabetical order for the publications. The author has major contributions to the security analysis of SPONGENT (except the linear attacks os Section 5.3.4). For this purpose two tools has been developed. The first tool used in our analysis performs a tree search to find not only the best characteristics/ differentials (in terms of probability) but also the iterative characteristics. The second tool, aims to find the maximum number of rounds that the security bound 2^b are reached. These tools differ in the way that they perform the search: The first tool is based on a breadth-first search in which the tree is limited to characteristics with at most three active Sboxes, whereas the second tool is based on depth-first search with upto six active Sboxes in each round.

Table 5.7: Results of linear trail correlation based on one bit masks for SPONGENT

	b	max w with $c_w \geq 2^{-b/2}$	R	$\log_2 c_R$
SPONGENT-88/80/8	88	22	45	−90
SPONGENT-88/176/88	264	66	135	−270
SPONGENT-128/128/8	136	34	70	−140
SPONGENT-128/256/128	384	96	195	−388.4
SPONGENT-160/160/16	176	44	90	−180
SPONGENT-160/160/80	240	60	120	−240
SPONGENT-160/320/160	480	122	240	−473.7
SPONGENT-224/224/16	240	60	120	−240
SPONGENT-224/224/112	336	84	170	−340
SPONGENT-224/448/224	673	169	340	−675.3
SPONGENT-256/256/16	272	68	140	−280
SPONGENT-256/256/128	384	96	195	−388.4
SPONGENT-256/512/256	768	192	385	−770

5.5 Conclusion

In this work, we have explored the design space of lightweight cryptographic hashing by proposing the family of new hash functions SPONGENT for resource-constrained applications. We consider 5 hash sizes for SPONGENT – ranging from the ones offering mainly preimage resistance only to those complying to (a subset of) the SHA-2 and SHA-3 parameters. For each parameter set, we instantiate SPONGENT using up to three competing security paradigms (all of them offering full collision security): reduced second-preimage security, reduced preimage and second-preimage security, as well as full preimage and second-preimage security. We also perform security analysis in terms of differential properties, linear distinguishers, and rebound attacks.

Chapter 6

Conclusion

Cryptographic hash functions are fundamental components of many applications used in our daily lives and hence are of great importance to their security. Consequently, the design and cryptanalysis of cyrptographic hash functions is one of the most popular and active areas of cryptography.

Recent advances in cryptanalysis have shown that several popular hash functions, including MD5 and SHA-1, are not as secure as claimed. Although there was no serious threat against their successor SHA-2, the National Institute of Standards and Technology (NIST) started a public competition to choose the next cryptographic hash function standard SHA-3.

Furthermore, in parallel to the evolution of technology, RFID and sensor networks are being embedded in everyday objects such as tools, devices, clothing, homes, etc. which require secure yet efficiently implementable cryptographic primitives including secret-key ciphers and hash functions. As a result, lately lightweight cryptography – optimizing the algorithms to fit the most constrained environments – has received a great deal of attention.

6.1 Contributions of This Thesis

In this thesis we focus both on the analysis and design of hash functions. The cryptanalysis of JH is strongly related with the SHA-3 competition as JH is one of the finalists. Our attacks are based on the rebound attack and they apply to

both the hash function and the compression function of JH. We have shown that a semi-free-start collision and semi-free-start near-collisions can be found for the reduced round JH. Based on these results, we described a distinguisher for the full internal permutation, that also applies to the full compression function.

Our design of SPONGENT contributes to the area of lightweight cryptography. SPONGENT is a family of sponge-based lightweight hash functions that uses a PRESENT-type permutation. We proposed thirteen SPONGENT variants covering most security applications in the field ranging from extremely restricted scenarios to the standard interfaces at different security levels. In our work, we evaluated the security of SPONGENT against known cryptanalytic attacks by applying the most important state-of-the-art methods of cryptanalysis and investigating their complexity.

6.2 Directions for Future Work

The research on cryptographic hash functions is a perpetual interplay in which each new design triggers new attack strategies and vica versa. No matter how hard the researchers work, many questions will remain to be answered. We conclude this thesis with some directions for future work.

- Since the winner of the SHA-3 competition is finally chosen, a natural target for future research is KECCAK. Although there were several attacks published, there is a very limited improvement so far in breaking the hash function. The maximum number of rounds for which a practical collision can be found is 4 (out of 24) for KECCAK-224 and 5 for KECCAK-256, whereas it is just 3 for KECCAK-384 and KECCAK-512. These results are improved by one more round for the case of near-collisions. However the distinguishers and second preimage attacks covering larger number of rounds are far from being practical.
- In SPONGENT, we mainly focused on giving bounds against differential cryptanalysis and rebound type of attacks in our analysis. However, it would be interesting to investigate the security of SPONGENT against other attacks such as biclique attacks or algebraic cryptanalysis. If applicable, bicliques can be used to construct efficient meet-in-the-middle attacks. Moreover algrebraic relations for a small number of rounds can be found as SPONGENT uses 4×4 S-boxes and a bit-oriented permutation. Perhaps it is possible to find efficient attacks which was not considered during the design process.

- The importance of lightweight cryptography is increasing over the time, so does the need for good lightweight cryptographic primitives. Although there are many hash function designs introduced recently, how to optimize the security, area and power consuption simultaneously is not yet well understood and remains an open problem.
- Although the analysis given in the scope of this thesis does not cover hash functions based on ARX, there are numerous designs (including the standards SHA-1 and SHA-2) that use this approach. Unfortunately, we believe that ARX based designs did not get as much attention as SPN and Feistel networks from the cryptographic community. Due to their complex structure there is no efficient method to perform the analysis or to give bounds against differential and linear cryptanalysis.

Bibliography

[1] Al-Kadi, I. A. Selections from Cryptologia. Artech House, Inc., 1998, ch. Origins of cryptology: the Arab contribution, pp. 93–122.

[2] Andreeva, E., Bouillaguet, C., Dunkelman, O., and Kelsey, J. Herding, Second Preimage and Trojan Message Attacks beyond Merkle-Damgård. In Jr. et al. [106], pp. 393–414.

[3] Andreeva, E., and Mennink, B. Provable chosen-target-forced-midfix preimage resistance. In Miri and Vaudenay [150], pp. 37–54.

[4] Andreeva, E., and Stam, M. The symbiosis between collision and preimage resistance. In *IMA Int. Conf.* (2011), L. Chen, Ed., vol. 7089 of *Lecture Notes in Computer Science*, Springer, pp. 152–171.

[5] Aoki, K., and Sasaki, Y. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In *Selected Areas in Cryptography* (2008), R. M. Avanzi, L. Keliher, and F. Sica, Eds., vol. 5381 of *Lecture Notes in Computer Science*, Springer, pp. 103–119.

[6] Aoki, K., and Sasaki, Y. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In Halevi [87], pp. 70–89.

[7] Ars, G., Faugère, J.-C., Imai, H., Kawazoe, M., and Sugita, M. Comparison Between XL and Gröbner Basis Algorithms. In *ASIACRYPT* (2004), P. J. Lee, Ed., vol. 3329 of *Lecture Notes in Computer Science*, Springer, pp. 338–353.

[8] Ashur, T., and Dunkelman, O. Linear Analysis of Reduced-Round CubeHash. In *ACNS* (2011), J. Lopez and G. Tsudik, Eds., vol. 6715 of *Lecture Notes in Computer Science*, pp. 462–478.

[9] Atalay, A., Kara, O., Karakoc, F., and Manap, C. SHAMATA Hash Function Algorithm Specifications. Submission to NIST, 2008.

[10] Aumasson, J.-P., Çagdas Çalik, Meier, W., Özen, O., Phan, R. C.-W., and Varici, K. Improved Cryptanalysis of Skein. In Matsui [138], pp. 542–559.

[11] Aumasson, J.-P., Dinur, I., Meier, W., and Shamir, A. Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In Dunkelman [75], pp. 1–22.

[12] Aumasson, J.-P., Henzen, L., Meier, W., and Naya-Plasencia, M. Quark: A Lightweight Hash. In Mangard and Standaert [135], pp. 1–15.

[13] Aumasson, J.-P., Henzen, L., Meier, W., and Phan, R. C.-W. SHA-3 proposal BLAKE. Submission to NIST (Round 1/2), 2008.

[14] Babbage, S., and Dodd, M. The MICKEY Stream Ciphers. In Robshaw and Billet [173], pp. 191–209.

[15] Barreto, P. S. L. M., and Rijmen, V. The WHIRLPOOL Hashing Function. In *Proceedings of the 1st NESSIE Workshop* (Leuven, Belgium, 2000), p. 15.

[16] Bellare, M., Canetti, R., and Krawczyk, H. Keying Hash Functions for Message Authentication. In *CRYPTO* (1996), N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, Springer, pp. 1–15.

[17] Bellare, M., and Micciancio, D. A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. In *EUROCRYPT* (1997), W. Fumy, Ed., vol. 1233 of *Lecture Notes in Computer Science*, Springer, pp. 163–192.

[18] Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., and Seurin, Y. SHA-3 Proposal: ECHO. Submission to NIST (updated), 2009.

[19] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. RadioGatún, a belt-and-mill hash function. *IACR Cryptology ePrint Archive 2006* (2006), 369.

[20] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. Sufficient conditions for sound tree and sequential hashing modes. Cryptology ePrint Archive, Report 2009/210, 2009. `http://eprint.iacr.org/`.

[21] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. The Keccak SHA-3 submission. Submission to NIST (Round 3), 2011.

[22] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. On the Indifferentiability of the Sponge Construction. In *EUROCRYPT'08* (2008), N. P. Smart, Ed., vol. 4965 of *LNCS*, Springer, pp. 181–197.

[23] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Sponge-Based Pseudo-Random Number Generators. In Mangard and Standaert [135], pp. 33–47.

[24] Bhattacharyya, R., Mandal, A., and Nandi, M. Security Analysis of the Mode of JH Hash Function. In Hong and Iwata [96], pp. 168–191.

[25] Biham, E., Biryukov, A., and Shamir, A. Miss in the Middle Attacks on IDEA and Khufu. In Knudsen [118], pp. 124–138.

[26] Biham, E., and Dunkelman, O. A Framework for Iterative Hash Functions - HAIFA. *IACR Cryptology ePrint Archive 2007* (2007), 278.

[27] Biham, E., Dunkelman, O., and Keller, N. The Rectangle Attack - Rectangling the Serpent. In *EUROCRYPT* (2001), B. Pfitzmann, Ed., vol. 2045 of *Lecture Notes in Computer Science*, Springer, pp. 340–357.

[28] Biham, E., and Shamir, A. Differential Cryptanalysis of DES-like Cryptosystems. In Menezes and Vanstone [145], pp. 2–21.

[29] Biryukov, A., Ed. *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers* (2007), vol. 4593 of *Lecture Notes in Computer Science*, Springer.

[30] Biryukov, A., Lamberger, M., Mendel, F., and Nikolic, I. Second-order differential collisions for reduced sha-256. In Lee and Wang [128], pp. 270–287.

[31] Biryukov, A., Nikolic, I., and Roy, A. Boomerang Attacks on BLAKE-32. In Joux [103], pp. 218–237.

[32] Black, J., Rogaway, P., and Shrimpton, T. Black-box analysis of the block-cipher-based hash-function constructions from pgv. In *CRYPTO* (2002), M. Yung, Ed., vol. 2442 of *Lecture Notes in Computer Science*, Springer, pp. 320–335.

[33] Blackburn, S. R., Stinson, D. R., and Upadhyay, J. On the Complexity of the Herding Attack and Some Related Attacks on Hash Functions. *Des. Codes Cryptography 64*, 1-2 (2012), 171–193.

[34] Bogdanov, A., Khovratovich, D., and Rechberger, C. Biclique Cryptanalysis of the Full AES. In Lee and Wang [128], pp. 344–371.

[35] Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., and Verbauwhede, I. SPONGENT: A Lightweight Hash Function. In *CHES'11* (2011), B. Preneel and T. Takagi, Eds., vol. 6917 of *LNCS*, Springer, pp. 312–325.

[36] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsoe, C. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES* (2007), P. Paillier and I. Verbauwhede, Eds., vol. 4727 of *Lecture Notes in Computer Science*, Springer, pp. 450–466.

[37] Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., and Seurin, Y. Hash Functions and RFID Tags: Mind the Gap. In *CHES* (2008), E. Oswald and P. Rohatgi, Eds., vol. 5154 of *Lecture Notes in Computer Science*, Springer, pp. 283–299.

[38] Bosselaers, A., and Preneel, B., Eds. *Integrity Primitives for Secure Information Systems, Final Report of RACE Integrity Primitives Evaluation RIPE-RACE 1040*, vol. 1007 of *Lecture Notes in Computer Science*. Springer, 1995.

[39] Boura, C., Canteaut, A., and De Cannière, C. Higher-order differential properties of keccak and *luffa*. In Joux [103], pp. 252–269.

[40] Brachtl, B. O., Coppersmith, D., Hyden, M. M., Jr., S. M. M., Meyer, C. H. W., Oseas, J., Pilpel, S., and Schilling, M. Data Authentication Using Modification Detection Codes Based on a Public One-way Encryption Function. U. S. Patent No. 4,908,861 (March 1990).

[41] Brassard, G., Ed. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings* (1990), vol. 435 of *Lecture Notes in Computer Science*, Springer.

[42] Bresson, E., Canteaut, A., Chevallier-Mames, B., Clavier, C., Fuhr, T., Gouget, A., Icart, T., Misarsky, J.-F., Naya-Plasencia, M., Paillier, P., Pornin, T., Reinhard, J.-R., Thuillet, C., and Videau, M. Shabal. Submission to NIST (Round 1), 2008.

[43] 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. *Journal of Symbolic Computation 41*, 3-4 (2006), 475–511.

[44] Chabaud, F., and Joux, A. Differential Collisions in SHA-0. In *CRYPTO* (1998), H. Krawczyk, Ed., vol. 1462 of *Lecture Notes in Computer Science*, Springer, pp. 56–71.

[45] Chang, D., Gupta, K. C., and Nandi, M. RC4-Hash: A New Hash Function Based on RC4. In *INDOCRYPT* (2006), R. Barua and T. Lange, Eds., vol. 4329 of *Lecture Notes in Computer Science*, Springer, pp. 80–94.

[46] Cho, J. Y. Linear Cryptanalysis of Reduced-Round PRESENT. In Pieprzyk [162], pp. 302–317.

[47] Collard, B., and Standaert, F.-X. A Statistical Saturation Attack against the Block Cipher PRESENT. In *CT-RSA* (2009), M. Fischlin, Ed., vol. 5473 of *Lecture Notes in Computer Science*, Springer, pp. 195–210.

[48] Contini, S., Lenstra, A. K., and Steinfeld, R. VSH, an Efficient and Provable Collision-Resistant Hash Function. In Vaudenay [195], pp. 165–182.

[49] Courtois, N. The Security of Hidden Field Equations (HFE). In *CT-RSA* (2001), D. Naccache, Ed., vol. 2020 of *Lecture Notes in Computer Science*, Springer, pp. 266–281.

[50] Courtois, N., Klimov, A., Patarin, J., and Shamir, A. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *EUROCRYPT* (2000), B. Preneel, Ed., vol. 1807 of *Lecture Notes in Computer Science*, Springer, pp. 392–407.

[51] Courtois, N., and Pieprzyk, J. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *ASIACRYPT* (2002), Y. Zheng, Ed., vol. 2501 of *Lecture Notes in Computer Science*, Springer, pp. 267–287.

[52] Cramer, R., Ed. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings* (2005), vol. 3494 of *LNCS*, Springer.

[53] Daemen, J., and Assche, G. V. Producing Collisions for Panama, Instantaneously. In Biryukov [29], pp. 1–18.

[54] Daemen, J., and Clapp, C. S. K. Fast Hashing and Stream Encryption with PANAMA. In *FSE* (1998), S. Vaudenay, Ed., vol. 1372 of *Lecture Notes in Computer Science*, Springer, pp. 60–74.

[55] Daemen, J., Peeters, M., and Van Assche, G. Sponge Functions. Ecrypt Hash Workshop 2007, 2007.

[56] Daemen, J., and Rijmen, V. The Wide Trail Design Strategy. In *IMA Int. Conf.* (2001), B. Honary, Ed., vol. 2260 of *Lecture Notes in Computer Science*, Springer, pp. 222–238.

[57] Daemen, J., and Rijmen, V. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[58] Damgård, I. A Design Principle for Hash Functions. In Brassard [41], pp. 416–427.

[59] Davies, D., and Price, W. Digital Signatures, an update, 1994.

[60] De Cannière, C. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In *ISC* (2006), S. K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, and B. Preneel, Eds., vol. 4176 of *Lecture Notes in Computer Science*, Springer, pp. 171–186.

[61] De Cannière, C., Dunkelman, O., and Knezevic, M. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *CHES* (2009), C. Clavier and K. Gaj, Eds., vol. 5747 of *Lecture Notes in Computer Science*, Springer, pp. 272–288.

[62] De Cannière, C., Mendel, F., and Rechberger, C. Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In *Selected Areas in Cryptography* (2007), C. M. Adams, A. Miri, and M. J. Wiener, Eds., vol. 4876 of *Lecture Notes in Computer Science*, Springer, pp. 56–73.

[63] De Cannière, C., and Preneel, B. Trivium. In Robshaw and Billet [173], pp. 244–266.

[64] De Cannière, C., and Rechberger, C. Finding SHA-1 Characteristics: General Results and Applications. In *ASIACRYPT* (2006), X. Lai and K. Chen, Eds., vol. 4284 of *LNCS*, Springer, pp. 1–20.

[65] De Cannière, C., Sato, H., and Watanabe, D. Hash Function Luffa: Specification. Submission to NIST (Round 2), 2009.

[66] Dean, R. D. *Formal Aspects of Mobile Code Security*. PhD thesis, 1999.

[67] den Boer, B., and Bosselaers, A. Collisions for the Compression Function of MD5. In Helleseth [91], pp. 293–304.

[68] Denning, D. E. *Cryptography and Data Security*. Addison-Wesley, 1982.

[69] Diffie, W., and Hellman, M. E. New Directions in Cryptography. *IEEE Transactions on Information Theory 22*, 6 (1976), 644–654.

[70] Dinur, I., and Shamir, A. Cube Attacks on Tweakable Black Box Polynomials. *IACR Cryptology ePrint Archive 2008* (2008), 385.

[71] Dinur, I., and Shamir, A. Breaking Grain-128 with Dynamic Cube Attacks. In Joux [103], pp. 167–187.

[72] Dobbertin, H. Cryptanalysis of MD4. In Gollmann [85], pp. 53–69.

[73] Dobbertin, H., Bosselaers, A., and Preneel, B. RIPEMD-160: A Strengthened Version of RIPEMD. In Gollmann [85], pp. 71–82.

[74] Dodis, Y., Reyzin, L., Rivest, R. L., and Shen, E. Indifferentiability of Permutation-Based Compression Functions and Tree-Based Modes of Operation, with Applications to MD6. In Dunkelman [75], pp. 104–121.

[75] Dunkelman, O., Ed. *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers* (2009), vol. 5665 of *Lecture Notes in Computer Science*, Springer.

[76] Faugère, J.-C. A New Efficient Algorithm for Computing Gröbner basis (F4), 2000.

[77] Faugère, J.-C. A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F5). In *In: ISSAC'02: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation* (2002), pp. 75–83.

[78] Feistel, H. Block Cipher Cryptographic System. U.S. Patent No. 3,798,359 (Filed June 30, 1971).

[79] Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., and Walker, J. The Skein Hash Function Family. Submission to NIST (Round 1), 2008.

[80] Flajolet, P., and Odlyzko, A. M. Random Mapping Statistics. In *EUROCRYPT* (1989), J.-J. Quisquater and J. Vandewalle, Eds., vol. 434 of *Lecture Notes in Computer Science*, Springer, pp. 329–354.

[81] Gauravaram, P., and Knudsen, L. R. On randomizing hash functions to strengthen the security of digital signatures. In Joux [102], pp. 88–105.

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

[83] Gilbert, H., and Peyrin, T. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In Hong and Iwata [96], pp. 365–383.

[84] Gligoroski, D., Klima, V., Knapskog, S. J., El-Hadedy, M., Amundsen, J., and Mjolsnes, S. F. Cryptographic Hash Function BLUE MIDNIGHT WISH. Submission to NIST (Round 1), 2008.

[85] Gollmann, D., Ed. *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings* (1996), vol. 1039 of *Lecture Notes in Computer Science*, Springer.

[86] Guo, J., Peyrin, T., and Poschmann, A. The PHOTON Family of Lightweight Hash Functions. In Rogaway [174], pp. 222–239.

[87] Halevi, S., Ed. *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings* (2009), vol. 5677 of *Lecture Notes in Computer Science*, Springer.

[88] Halevi, S., and Krawczyk, H. Strengthening Digital Signatures Via Randomized Hashing. In *CRYPTO* (2006), C. Dwork, Ed., vol. 4117 of *Lecture Notes in Computer Science*, Springer, pp. 41–59.

[89] Hell, M., Johansson, T., Maximov, A., and Meier, W. The Grain Family of Stream Ciphers. In Robshaw and Billet [173], pp. 179–190.

[90] Hell, M., Johansson, T., and Meier, W. Grain: A Stream Cipher for Constrained Environments. *IJWMC 2*, 1 (2007), 86–93.

[91] Helleseth, T., Ed. *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings* (1994), vol. 765 of *Lecture Notes in Computer Science*, Springer.

[92] Hellman, M. E. A Cryptanalytic Time-Memory Trade-Off. *IEEE Transactions on Information Theory 26*, 4 (1980), 401–406.

[93] Herodotus. *The Histories*. 1992. translated by George Rawlinson.

[94] Hirose, S. Some plausible constructions of double-block-length hash functions. In *FSE* (2006), M. J. B. Robshaw, Ed., vol. 4047 of *Lecture Notes in Computer Science*, Springer, pp. 210–225.

[95] Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., and Chee, S. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In *CHES* (2006), L. Goubin and M. Matsui, Eds., vol. 4249 of *Lecture Notes in Computer Science*, Springer, pp. 46–59.

[96] Hong, S., and Iwata, T., Eds. *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers* (2010), vol. 6147 of *Lecture Notes in Computer Science*, Springer.

[97] Ideguchi, K., Tischhauser, E., and Preneel, B. Improved Collision Attacks on the Reduced-Round Grøstl Hash Function. In *ISC* (2010), M. Burmester, G. Tsudik, S. S. Magliveras, and I. Ilic, Eds., vol. 6531 of *Lecture Notes in Computer Science*, Springer, pp. 1–16.

[98] Indesteege, S. The LANE hash function. Submission to NIST, 2008.

[99] Indesteege, S., and Preneel, B. Collisions for RC4-Hash. In *ISC* (2008), T.-C. Wu, C.-L. Lei, V. Rijmen, and D.-T. Lee, Eds., vol. 5222 of *Lecture Notes in Computer Science*, Springer, pp. 355–366.

[100] International Organization for Standardization. *ISO/IEC 10118-3:2004: Information Technology - Security Techniques - Hashfunctions - Part 3: Dedicated Hash-Functions*. Geneva, Switzerland, 2004.

[101] Joux, A. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In *CRYPTO* (2004), M. K. Franklin, Ed., vol. 3152 of *Lecture Notes in Computer Science*, Springer, pp. 306–316.

[102] Joux, A., Ed. *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings* (2009), vol. 5479 of *LNCS*, Springer.

[103] Joux, A., Ed. *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers* (2011), vol. 6733 of *Lecture Notes in Computer Science*, Springer.

[104] Joux, A., and Peyrin, T. Hash Functions and the (Amplified) Boomerang Attack. In *CRYPTO* (2007), A. Menezes, Ed., vol. 4622 of *Lecture Notes in Computer Science*, Springer, pp. 244–263.

[105] Jr., B. S. K., and Robshaw, M. J. B. Linear Cryptanalysis Using Multiple Approximations. In *CRYPTO* (1994), Y. Desmedt, Ed., vol. 839 of *Lecture Notes in Computer Science*, Springer, pp. 26–39.

[106] Jr., M. J. J., Rijmen, V., and Safavi-Naini, R., Eds. *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers* (2009), vol. 5867 of *Lecture Notes in Computer Science*, Springer.

[107] Kahn, D. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. 1996.

[108] Kelsey, J., and Kohno, T. Herding Hash Functions and the Nostradamus Attack. In Vaudenay [195], pp. 183–200.

[109] Kelsey, J., Kohno, T., and Schneier, B. Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In *FSE* (2000), B. Schneier, Ed., vol. 1978 of *Lecture Notes in Computer Science*, Springer, pp. 75–93.

[110] Kelsey, J., and Schneier, B. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In Cramer [52], pp. 474–490.

[111] Khovratovich, D. Bicliques for Permutations: Collision and Preimage Attacks in Stronger Settings. *IACR Cryptology ePrint Archive 2012* (2012), 141.

[112] Khovratovich, D., Leurent, G., and Rechberger, C. Narrow-Bicliques: Cryptanalysis of Full IDEA. In *EUROCRYPT* (2012), D. Pointcheval and T. Johansson, Eds., vol. 7237 of *Lecture Notes in Computer Science*, Springer, pp. 392–410.

[113] Khovratovich, D., Rechberger, C., and Savelieva, A. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family. *IACR Cryptology ePrint Archive 2011* (2011), 286.

[114] Kim, J., Biryukov, A., Preneel, B., and Lee, S. On the Security of Encryption Modes of MD4, MD5 and HAVAL. In *ICICS* (2005), S. Qing, W. Mao, J. Lopez, and G. Wang, Eds., vol. 3783 of *Lecture Notes in Computer Science*, Springer, pp. 147–158.

[115] Kipnis, A., and Shamir, A. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In *CRYPTO* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 19–30.

[116] Knudsen, L. DEAL - A 128-bit Block Cipher. In *NIST AES Proposal* (1998).

[117] Knudsen, L. R. Truncated and Higher Order Differentials. In *FSE* (1994), B. Preneel, Ed., vol. 1008 of *Lecture Notes in Computer Science*, Springer, pp. 196–211.

[118] Knudsen, L. R., Ed. *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings* (1999), vol. 1636 of *Lecture Notes in Computer Science*, Springer.

[119] Knudsen, L. R., Rechberger, C., and Thomsen, S. S. The Grindahl Hash Functions. In Biryukov [29], pp. 39–57.

[120] Knuth, D. E. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison-Wesley, 1969.

[121] Lai, X. Higher Order Derivatives and Differential Cryptanalysis. In *Proc. "Symposium on Communication, Coding and Cryptography", in honor of J. L. Massey on the occasion of his 60'th birthday* (1994), Kluwer Academic Publishers.

[122] Lai, X., and Massey, J. L. Markov Ciphers and Differentail Cryptanalysis. In *EUROCRYPT* (1991), D. W. Davies, Ed., vol. 547 of *Lecture Notes in Computer Science*, Springer, pp. 17–38.

[123] Lai, X., and Massey, J. L. Hash function based on block ciphers. In *EUROCRYPT* (1992), R. A. Rueppel, Ed., vol. 658 of *Lecture Notes in Computer Science*, Springer, pp. 55–70.

[124] Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., and Schläffer, M. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Matsui [138], pp. 126–143.

[125] Leander, G. On Linear Hulls, Statistical Saturation Attacks, PRESENT and a Cryptanalysis of PUFFIN. In *EUROCRYPT* (2011), K. G. Paterson, Ed., vol. 6632 of *Lecture Notes in Computer Science*, Springer, pp. 303–322.

[126] Leander, G., Abdelraheem, M. A., AlKhzaimi, H., and Zenner, E. A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In Rogaway [174], pp. 206–221.

[127] Leander, G., Paar, C., Poschmann, A., and Schramm, K. New Lightweight DES Variants. In Biryukov [29], pp. 196–210.

[128] Lee, D. H., and 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 (2011), vol. 7073 of *Lecture Notes in Computer Science*, Springer.

[129] Li, Y., and Ailan, W. Linear Cryptanalysis for the Compression Function of Hamsi-256. In *Proceedings of the 2011 International Conference on Network Computing and Information Security - Volume 01* (Washington, DC, USA, 2011), NCIS '11, IEEE Computer Society, pp. 302–306.

[130] Lim, C. H., and Korkishko, T. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In *WISA* (2005), J. Song, T. Kwon, and M. Yung, Eds., vol. 3786 of *Lecture Notes in Computer Science*, Springer, pp. 243–258.

[131] Lipmaa, H., and Moriai, S. Efficient Algorithms for Computing Differential Properties of Addition. In Matsui [137], pp. 336–350.

[132] Lipmaa, H., Wallén, J., and Dumas, P. On the Additive Differential Probability of Exclusive-Or. In Roy and Meier [178], pp. 317–331.

[133] Lucks, S. A Failure-Friendly Design Principle for Hash Functions. In *ASIACRYPT* (2005), B. K. Roy, Ed., vol. 3788 of *Lecture Notes in Computer Science*, Springer, pp. 474–494.

[134] Lyubashevsky, V., Micciancio, D., Peikert, C., and Rosen, A. SWIFFT: A Modest Proposal for FFT Hashing. In *FSE* (2008), K. Nyberg, Ed., vol. 5086 of *LNCS*, Springer, pp. 54–72.

[135] Mangard, S., and Standaert, F.-X., Eds. *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings* (2010), vol. 6225 of *LNCS*, Springer.

[136] Matsui, M. Linear Cryptoanalysis Method for DES Cipher. In Helleseth [91], pp. 386–397.

[137] Matsui, M., Ed. *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers* (2002), vol. 2355 of *Lecture Notes in Computer Science*, Springer.

[138] Matsui, M., Ed. *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings* (2009), vol. 5912 of *Lecture Notes in Computer Science*, Springer.

[139] Matyas, S., Meyer, C., and Oseas, J. Generating Strong One-Way Functions with Cryptographic Algorithm. IBM Techn. Disclosure Bull., Vol. 27, No. 10A, 1985. pp. 5658-5659.

[140] Maurer, U. M. Indistinguishability of random systems. In *EUROCRYPT* (2002), L. R. Knudsen, Ed., vol. 2332 of *Lecture Notes in Computer Science*, Springer, pp. 110–132.

[141] Mendel, F., Peyrin, T., Rechberger, C., and Schläffer, M. Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In Jr. et al. [106], pp. 16–35.

[142] Mendel, F., Rechberger, C., Schläffer, M., and Thomsen, S. S. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Dunkelman [75], pp. 260–276.

[143] Mendel, F., Rechberger, C., Schläffer, M., and Thomsen, S. S. Rebound Attacks on the Reduced Grøstl Hash Function. In Pieprzyk [162], pp. 350–365.

[144] Menezes, A., van Oorschot, P. C., and Vanstone, S. A. *Handbook of Applied Cryptography*. CRC Press, 1996.

[145] Menezes, A., and Vanstone, S. A., Eds. *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings* (1991), vol. 537 of *Lecture Notes in Computer Science*, Springer.

[146] Mennink, B., and Preneel, B. Hash Functions Based on Three Permutations: A Generic Security Analysis. *IACR Cryptology ePrint Archive 2011* (2011), 532.

[147] Merkle, R. C. A Certified Digital Signature. In Brassard [41], pp. 218–238.

[148] Merkle, R. C. One Way Hash Functions and DES. In Brassard [41], pp. 428–446.

[149] Merkle, R. C. A fast software one-way hash function. *J. Cryptology 3*, 1 (1990), 43–58.

[150] Miri, A., and Vaudenay, S., Eds. *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers* (2012), vol. 7118 of *Lecture Notes in Computer Science*, Springer.

[151] Miyaguchi, S., Iwata, M., and Ohta, K. New 128-bit Hash Function. In *Proceeding of 4th International Joint Workshop on Computer Communications* (1989), pp. 279–288.

[152] Murphy, S. The Return of the Cryptographic Boomerang. *IEEE Transactions on Information Theory 57*, 4 (2011), 2517–2521.

[153] National Institute of Standards and Technology. Cryptographic Hash Algorithm Competition. `http://www.nist.gov/hash-competition`.

[154] National Institute of Standards and Technology. *Secure Hash Standard*. Washington, 1995. Note: Federal Information Processing Standard 180-1.

[155] National Institute of Standards and Technology. Announcing Development of a Federal Information Processing Standard for Advanced Encryption Standard. `http://csrc.nist.gov/archive/aes/pre-round1/aes_9701.txt`.

[156] National Institute of Standards and Technology. Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition. `http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf`.

[157] Naya-Plasencia, M. How to Improve Rebound Attacks. In Rogaway [174], pp. 188–205.

[158] Naya-Plasencia, M., Toz, D., and Varici, K. Rebound Attack on JH42. In Lee and Wang [128], pp. 252–269.

[159] Nikolic, I., Biryukov, A., and Khovratovich, D. Hash family LUX - Algorithm Specifications and Supporting Documentation. Submission to NIST, 2008.

[160] Nyberg, K. Linear Approximation of Block Ciphers. In *EUROCRYPT* (1994), A. D. Santis, Ed., vol. 950 of *Lecture Notes in Computer Science*, Springer, pp. 439–444.

[161] Peyrin, T. Improved Differential Attacks for ECHO and Grøstl. In *CRYPTO* (2010), T. Rabin, Ed., vol. 6223 of *Lecture Notes in Computer Science*, Springer, pp. 370–392.

[162] Pieprzyk, J., Ed. *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings* (2010), vol. 5985 of *Lecture Notes in Computer Science*, Springer.

[163] Preneel, B. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993. René Govaerts and Joos Vandewalle (promotors).

[164] Preneel, B. MASH Hash Functions (Modular Arithmetic Secure Hash). In *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg, Ed. Springer, 2005.

[165] Preneel, B., Govaerts, R., and Vandewalle, J. Hash Functions Based on Block Ciphers: A Synthetic Approach. In *CRYPTO* (1993), D. R. Stinson, Ed., vol. 773 of *Lecture Notes in Computer Science*, Springer, pp. 368–378.

[166] Quisquater, J.-J., and Delescaille, J.-P. How Easy is Collision Search? New Results and Applications to DES. In Brassard [41], pp. 408–413.

[167] Rabin, M. Digitalized signatures, 1978.

[168] Rijmen, V., Toz, D., and Varici, K. Rebound Attack on Reduced-Round Versions of JH. In Hong and Iwata [96], pp. 286–303.

[169] Rijmen, V., Van Rompay, B., Preneel, B., and Vandewalle, J. Producing Collisions for PANAMA. In Matsui [137], pp. 37–51.

[170] Rivest, R. The MD5 Message-Digest Algorithm, 1992.

[171] Rivest, R. L. The MD4 Message Digest Algorithm. In Menezes and Vanstone [145], pp. 303–311.

[172] Rivest, R. L. The MD6 Hash Function – A Proposal to NIST for SHA-3. Submission to NIST, 2008.

[173] Robshaw, M. J. B., and Billet, O., Eds. *New Stream Cipher Designs - The eSTREAM Finalists*, vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008.

[174] Rogaway, P., Ed. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings* (2011), vol. 6841 of *LNCS*, Springer.

[175] Rogaway, P., and Shrimpton, T. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Roy and Meier [178], pp. 371–388.

[176] Rogaway, P., and Steinberger, J. P. Constructing Cryptographic Hash Functions from Fixed-Key Blockciphers. In *CRYPTO* (2008), D. Wagner, Ed., vol. 5157 of *Lecture Notes in Computer Science*, Springer, pp. 433–450.

[177] Rogaway, P., and Steinberger, J. P. Security/Efficiency Tradeoffs for Permutation-Based Hashing. In *EUROCRYPT* (2008), N. P. Smart, Ed., vol. 4965 of *Lecture Notes in Computer Science*, Springer, pp. 220–236.

[178] Roy, B. K., and Meier, W., Eds. *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers* (2004), vol. 3017 of *Lecture Notes in Computer Science*, Springer.

[179] Sasaki, Y. Boomerang Distinguishers on MD4-Family: First Practical Results on Full 5-Pass HAVAL. In Miri and Vaudenay [150], pp. 1–18.

[180] Sasaki, Y., and Aoki, K. Preimage Attacks on 3, 4, and 5-Pass HAVAL. In *ASIACRYPT* (2008), J. Pieprzyk, Ed., vol. 5350 of *Lecture Notes in Computer Science*, Springer, pp. 253–271.

[181] Sasaki, Y., and Aoki, K. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In Joux [102], pp. 134–152.

[182] Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., and Ohta, K. Non-full-active Super-Sbox Analysis: Applications to ECHO and Grøstl. In *ASIACRYPT* (2010), M. Abe, Ed., vol. 6477 of *Lecture Notes in Computer Science*, Springer, pp. 38–55.

[183] Schläffer, M. Subspace Distinguisher for 5/8 Rounds of the ECHO-256 Hash Function. In *Selected Areas in Cryptography* (2010), A. Biryukov, G. Gong, and D. R. Stinson, Eds., vol. 6544 of *LNCS*, Springer, pp. 369–387.

[184] Shannon, C. Communication Theory of Secrecy Systems. *Bell System Technical Journal, Vol 28* (October 1949), 656–715.

[185] Shannon, C. E. A Mathematical Theory of Communication. *Bell System Technical Journal 27* (1948), 379–423.

[186] Shoup, V., Ed. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings* (2005), vol. 3621 of *LNCS*, Springer.

[187] Singh, S. *The Code Book: The Evolution of Secrecy from Mary, Queen of Scots, to Quantum Cryptography*, 1st ed. Doubleday, 1999.

[188] Staffelbach, O., and Meier, W. Cryptographic Significance of the Carry for Ciphers Based on Integer Addition. In Menezes and Vanstone [145], pp. 601–614.

[189] Standaert, F.-X., Piret, G., Gershenfeld, N., and Quisquater, J.-J. SEA: A Scalable Encryption Algorithm for Small Embedded Applications. In *CARDIS* (2006), J. Domingo-Ferrer, J. Posegga, and D. Schreckling, Eds., vol. 3928 of *Lecture Notes in Computer Science*, Springer, pp. 222–236.

[190] Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A. K., Molnar, D., Osvik, D. A., and de Weger, B. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In Halevi [87], pp. 55–69.

[191] Stinson, D. Some Observations on the Theory of Cryptographic Hash Functions. Tech. rep., 2001.

[192] Suzuki, K., Tonien, D., Kurosawa, K., and Toyota, K. Birthday Paradox for Multi-collisions. In *ICISC* (2006), M. S. Rhee and B. Lee, Eds., vol. 4296 of *Lecture Notes in Computer Science*, Springer, pp. 29–40.

[193] Van Assche, G. Errata for Keccak presentation. E-mail sent to the NIST SHA-3 mailing list on Feb 7 2011, on behalf of the Keccak team, 2011.

[194] van Oorschot, P. C., and Wiener, M. J. Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology 12*, 1 (1999), 1–28.

[195] Vaudenay, S., Ed. *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings* (2006), vol. 4004 of *Lecture Notes in Computer Science*, Springer.

[196] Velichkov, V., Mouha, N., De Cannière, C., and Preneel, B. UNAF: A Special Set of Additive Differences with Application to the Differential Analysis of ARX. In *Lecture Notes in Computer Science* (2012), Springer-Verlag.

[197] Vielhaber, M. Breaking one.fivium by aida an algebraic iv differential attack. *IACR Cryptology ePrint Archive 2007* (2007), 413.

[198] Wagner, D. The Boomerang Attack. In Knudsen [118], pp. 156–170.

[199] Wang, X., Lai, X., Feng, D., Chen, H., and Yu, X. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer [52], pp. 1–18.

[200] Wang, X., Yin, Y. L., and Yu, H. Finding Collisions in the Full SHA-1. In Shoup [186], pp. 17–36.

[201] Wang, X., and Yu, H. How to Break MD5 and Other Hash Functions. In Cramer [52], pp. 19–35.

[202] Wang, X., Yu, H., and Yin, Y. L. Efficient Collision Search Attacks on SHA-0. In Shoup [186], pp. 1–16.

[203] Watanabe, D., Hatano, Y., Yamada, T., and Kaneko, T. Higher Order Differential Attack on Step-Reduced Variants of *Luffa* v1. In Hong and Iwata [96], pp. 270–285.

[204] Wolf, C., and Preneel, B. Taxonomy of public key schemes based on the problem of multivariate quadratic equations. *IACR Cryptology ePrint Archive 2005* (2005), 77.

[205] Wu, H. The Hash Function JH. Submission to NIST, 2008. `http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/jh_round2.pdf`.

[206] Yu, H., Chen, J., and Wang, X. The Boomerang Attacks on the Round-Reduced Skein-512. *IACR Cryptology ePrint Archive 2012* (2012), 238.

[207] Yuval, G. How to Swindle Rabin. *Cryptologia 3* (1979), 187–189.

[208] Zheng, Y., Pieprzyk, J., and Seberry, J. HAVAL - A One-Way Hashing Algorithm with Variable Length of Output. In *AUSCRYPT* (1992), J. Seberry and Y. Zheng, Eds., vol. 718 of *Lecture Notes in Computer Science*, Springer, pp. 83–104.

Appendix A

Appendix to Chapter 4

We performed several experiments to obtain semi-free-start collisions and near-collisions for reduced round version of JH with $d = 4$ using different S-Box setups. The implementation results shown below are obtained by using the same S-Box ($S0$) in each round.

Table A.1: Example for rebound attack with one inbound phase ($d = 4$)

	Bit-Slice Results			Reference Results		
	P_1	P_2	Difference	P_1	P_2	Difference
0	6ddf8804acec67ef	addf8804acec67ef	c000000000000000	6dacdfec886704ef	adacdfec886704ef	c000000000000000
1	53d4792d4304231c	03d4792d4104231c	5000000002000000	53d4792d4304231c	03d4792d4104231c	5000000002000000
2	c7b01dd79c2227e3	e7ba1dd7cc2927e3	200a0000500b0000	c71d9c27b0d722e3	e71dcc27bad729e3	200050000a000b00
3	55c5a1f5a7f8a0bc	3595acfe6708a9b2	60500d0bc0f0090e	55a7c5f8a1a0f5bc	35679508aca9feb2	60c050f00d090b0e
4	2b3ead8b7712b433	a9f04e08299bd357	82cee3835e896764	2b3ead8b7712b433	a9f04e08299bd357	82cee3835e896764
5	bd2e0491b4824e41	d62ef09101827441	6b00f400b5003a00	bd04b44e2e918241	d6f001742e918241	6bf4b53a00000000
6	7c47294041d9f1c6	46472940b0d9f1c6	3a000000f1000000	7c4147d929f140c6	46b047d929f140c6	3af1000000000000
7	89100fddca5632e0	a5100fddca5632e0	2c00000000000000	89100fddca5632e0	a5100fddca5632e0	2c00000000000000
8	29184f8fc8fc9af1	19184f8fc8fc9af1	3000000000000000	294fc89a188ffcf1	194fc89a188ffcf1	3000000000000000

Table A.2: Example for rebound attack with three inbound phases ($d = 4$)

	Bit-Slice Results			Reference Results		
	P_1	P_2	Difference	P_1	P_2	Difference
0	1f7515504810c889	a2f615509bb7c889	bd830000d3a70000	1f7515504810c889	a2f615509bb7c889	bd830000d3a70000
1	f1deb28ce918a82f	2c50b28c8301a82f	dd8e00006a190000	f1b2e9a8de8c182f	2cb283a8508c012f	dd006a008e001900
2	38155fb295268981	98855fb205b68981	a090000090900000	389515265f89b281	980585b65f89b281	a090909000000000
3	3bbb6852f41c2558	6def685206be2558	56540000f2a20000	3bbb6852f41c2558	6def685206be2558	56540000f2a20000
4	da13df17a7e2cab4	5d13df17d4e2cab4	8700000073000000	dadfa7ca1317e2b4	5ddfd4ca1317e2b4	8700730000000000
5	645fd8db7c61cb47	345fdedbbc61c747	50000600c0000c00	647c5f61d8cbdb47	34bc5f61dec7db47	50c00000060c0000
6	6a6608e68bf54b1c	8b6611e638f5521c	e1001900b3001900	6a6608e68bf54b1c	8b6611e638f5521c	e1001900b3001900
7	b5949080938407ef	95940080338417ef	20009000a0001000	b5909307948084ef	95003317948084ef	2090a01000000000
8	6edb504a1ecd6c1f	f8dbab4a4fcdde1f	9600fb005100b200	6e1edbcd506c4a1f	f84fdbcdabde4a1f	96510000fbb20000
9	c8eb818e3e2c12fb	17eb628e3e2c12fb	df00e30000000000	c8eb818e3e2c12fb	17eb628e3e2c12fb	df00e30000000000
10	a52f5a622f9d89f4	d52f7a622f9d89f4	7000200000000000	a55a2f892f629df4	d57a2f892f629df4	7020000000000000
11	1982d87688e226a2	9f82267688e226a2	8600fe0000000000	198882e2d82676a2	9f8882e2262676a2	86000000fe000000
12	73d70b007a661b2e	b3d71b0079661b2e	c000100003000000	73d70b007a661b2e	b3d71b0079661b2e	c000100003000000

Appendix B

Appendix to Chapter 5

In this section we give the best iterative characteristics we obtained for SPONGENT variants based on depth-first search. In the tables below, the non-zero values are given in hexadecimal whereas the symbol '.' denotes the zero values and the probabilities are given in minus the binary logarithm.

Table B.1: Sample differential path for SPONGENT-160/160/16

Round	Difference	Prob
0	.....9.....9	0
1	.....8.1.....	6
2	.....28.....28..	5
3	.....6..C.....6..C	12
4	.....24....9	10
5	.....6.....61.....	9
6	.....8..8.....18..	7
7	.....C....9.....9.....C	11
8	.....42..41.....	10
9	.....6.....66.....6.....	11
10	.....8.1.....8.1.....	10
11	.....A.A.....A.A.	10
12	.....5..A	12
13	.....9.....9	6
	Total	119

Table B.2: Sample differential paths for SPONGENT-160/160/80 and SPONGENT-224/224/16

SPONGENT-160/160/80 and SPONGENT-224/224/16

Round	Difference	Prob
0	.....9.....9....	0
1	.....8..1.....	6
2	.....9.....9....	5
	Total	11

SPONGENT−256/256/16

Round	Difference	Prob
0	.....6...3.....6...3	0
1	.....44.....11.....	10
2	.....6.6.....6.6....	10
3	.....A...5.	8
4	.....22.....22	6
5	.....6...3.....6...3	12
	Total	46

Table B.3: Sample differential paths for SPONGENT-128/256/128

Round	Difference	Prob
0	.....1..1.....1..1.....	0
1	.....9..9.....9..9.....	8
2	.....24.....24.....	12
3	.....6..6.....6..6.....	12
4	.....12...12.....	8
5	.....C3.....C3..	10
6	.....C.....C.....C.....C	12
7	.....1..1.....1..1	12
8	.....9..9.....9..9	8
9	.....9....9.....	12
10	.....41.....	6
11	.....C.....C.....	5
12	.....8.....8.	6
13	.....82.....82	6
14	.....3.....3.....3.....3	12
15	.....1..1.....1..1.....	8
	Total	137

Table B.4: Sample differential paths for SPONGENT-256/256/128

Round	Difference	Prob
0	.....2.8...2.8.....	0
1	.....145.....145.....	12
2	.....7.....7.....7.....7...	16
3	.....8..8..8..8	8
4	.....249.....249	12
5	.....6.....6.....7.....7.....	18
6	.....41....41.....	8
7	.....3.C.....3.C.	10
8	.....8.....8.....A.....A	12
9	.....41.....41.....41.....	12
10	.....3..3.....3.....3..3.....3...	15
11	.....48..8.48..8.....	12
12	.....659.....659.....	18
13	.....1C....1C.....	16
14	.....1.4.....186.....	10
15	.....5.....6.....5.....7...	12
16	.....2.8...2.8.....	10
	Total	201

Table B.5: Sample differential paths for SPONGENT-160/320/160

SPONGENT-160/320/160

Round	Difference	Prob
0	.....6...C.....6...C.....	0
1	.....88.....88.	10
2	.....3...6.....3...6	12
3	.....11.....11.....	10
4	.....6...C.....6...C.....	8
	Total	40

SPONGENT-160/320/160

Round	Difference	Prob
0	.....6...C.....6...C.....	0
1	.....11.....44.....	10
2	.....6.6.....6.....6...	10
3	.....A.....8.....8	8
4	.....81.....8.81	9
5	.....C.....B.....C.....B	13
6	.....81.....81.....	12
7	.....6...C.....6...C.....	10
	Total	72

Curriculum Vitae

Deniz Toz was born on 19th August, 1980 in Ankara, Turkey. In June 2002, she graduated as a Civil Engineer from the Middle East Technical University (METU), Ankara, Turkey and she completed her double major in Mathematics in February 2003. She she obtained her Master of Science degree in Cryptography from METU in June 2005. Between 2003 and 2009, she worked as a software engineer at Prota Engineering R&D Office - Ankara. In February 2009, she joined the research group COSIC (COmputer Security and Industrial Cryptography) at the Department of Electrical Engineering (ESAT) of KU Leuven. Her PhD research was sponsored by the Research Fund of the KU Leuven.

List of publications

International Journals

1. A. Bogdanov, M. Knezevic, G. Leander, K. Varici, D. Toz, I. Verbauwhede, "SPONGENT: The Design Space of Lightweight Cryptographic Hashing," *IEEE Transactions on Computers*, 14 pages, 2011.

LNCS Conferences

1. F. Mendel, V. Rijmen, D. Toz, K. Varıcı, "Collisions for the WIDEA-8 Compression Function", *Topics in Cryptology - CT-RSA 2013, The Cryptographers' Track at the RSA Conference, Lecture Notes in Computer Science 7779*, E. Dawson (ed.), Springer-Verlag, pp. 162-175, 2013.

2. F. Mendel, V. Rijmen, Deniz Toz, Kerem Varıcı, "Differential Analysis of the LED Block Cipher", *Advances in Cryptology - ASIACRYPT 2012, Lecture Notes in Computer Science 7658*, K. Sako, and X. Wang (eds.), Springer-Verlag, pp. 190-207, 2012.

3. A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varıcı, I. Verbauwhede, "SPONGENT: A Lightweight Hash Function", *Cryptographic Hardware and Embedded Systems - CHES 2011, Lecture Notes in Computer Science 6917*, B.Preneel, T. Takagi, Eds., Springer-Verlag, pp. 312-327, 2011.

4. M. Naya-Plasencia, D. Toz, K. Varıcı, "Rebound Attack on JH42", *Advances in Cryptology - ASIACRYPT 2011, Lecture Notes in Computer Science 7073*, D. Lee, X. Wang, Eds., Springer-Verlag, pp. 252-269, 2011.

5. V. Rijmen, D. Toz, K. Varıcı, "Rebound Attack on Reduced-Round Versions of JH", *Fast Software Encryption, FSE 2010, Lecture Notes in Computer Science 6147*, S. Hong, T. Iwata, Eds., Springer-Verlag, pp. 286-303, 2010.

Other Conferences and Workshops

1. V. Rijmen, D. Toz, K. Varıcı, "On the Four-Round AES Characteristics", *International Workshop on Coding and Cryptography (WCC 2013)*, 14 pages, 2013.
2. A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varıcı, I. Verbauwhede,"SPONGENT: The Design Space of Lightweight Cryptographic Hashing", *Lightweight Cryptography*, 23 pages, 2011.

Technical Reports

1. C. Rechberger, T. E. Bjørstad, J. Daemen, C. De Cannière, P. Gauravaram, D. Khovratovich, W. Meier, T. Nad, I. Nikolic, M. Robshaw, M. Schläffer, S. S. Thomsen, E. Tischhauser, D. Toz, G. Van Assche, K. Varıcı, "ECRYPT II SHA-3 Design and Cryptanalysis Report", 2010.
2. P. Gauravaram, F. Mendel, M. Naya-Plasencia, V. Rijmen, D. Toz, "ECRYPT II D.SYM.7 Intermediate Status Report", 2011

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING (ESAT)
COMPUTER SECURITY AND INDUSTRIAL CRYPTOGRAPHY (COSIC)
Kasteelpark Arenberg 10, Bus: 2446
B-3001 Heverlee
www.esat.kuleuven.be/scd