KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Electrical Engineering (ESAT)

Design and Analysis of Cryptographic Hash Functions

Özgül KÜÇÜK

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

April 2012

Design and Analysis of Cryptographic Hash Functions

Özgül KÜÇÜK

Jury:
Prof. dr. Adhemar Bultheel, chairman
Prof. dr. ir. Bart Preneel, promotor
Prof. dr. ir. Vincent Rijmen, promotor
Prof. dr. Lars R. Knudsen
(Technical University of Denmark)
Dr. Christian Rechberger
(Technical University of Denmark)
Prof. dr. ir. Joos Vandewalle
Prof. dr. ir. Luc Van Eycken

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

April 2012

"A certain type of perfection can only be realized through a limitless accumulation of the imperfect."

Haruki Murakami, Kafka on the Shore

© Katholieke Universiteit Leuven – Faculty of Engineering
Arenbergkasteel, B-3001 Heverlee (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotocopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/2012/7515/35
ISBN 978-94-6018-499-4

Acknowledgements

It would have been impossible to complete this dissertation without the help and encouragement of several people, to only some of whom it is possible to express my gratitude here.

I would like to start with my promotors, prof. Bart Preneel and prof. Vincent Rijmen. I would like to thank Bart for giving me a chance to do a PhD at COSIC, and for proposing hash functions as a challenging topic while I was searching for a research problem. His approvals to attend conferences enabled me to be part of an active scientific community and to broaden my knowledge. I am especially indebted to him for his extensive feedback which helped to turn the text into a (more) readable form. I would like to thank Vincent, first of all for being a driving force in the symmetric key group. Completing a PhD study requires a certain degree of discipline, and I am grateful for Vincent's reminders about schedule whenever necessary. He also provided valuable feedback on the thesis, guidance and advice. Above all, I am grateful for the expertise that my promotors have shared with me.

I am grateful to prof. Joos Vandewalle and prof. Luc Van Eycken who followed my entire PhD process and provided feedback as jury members. I would like to thank prof. Adhemar Bultheel, as chairman, and prof. Lars Knudsen and dr. Christian Rechberger for accepting to serve as jury members. I thank all jury members for their valuable comments, their intriguing questions, and their patience for the last minute updates.

There are numerous colleagues who enlightened me while I was wandering around in the dark, sometimes by pointing out directions, sometimes by asking questions, making comments, or by giving a hand. I would like to thank prof. Alex Biryukov for giving me the assignment to apply slide attacks to stream ciphers, which resulted in the first result of this PhD. It is my pleasure to thank my co-authors: Christophe De Canniére, Sebastiaan Indesteege, Jun Kitahara, Nicky Mouha, Katsuyuki Okeya, Bart Preneel, Markus Ullrich, Dai Watanabe, Hongjun Wu, Hirotaka Yoshida. I am grateful to Christophe De Cannière not only for being a colleague and a co-author but also for being a friend. Thank you for the many

insightful discussions we had.

The submission of Hamsi to the SHA-3 competition gave me a unique chance to be involved in a exciting and important project, and I would like to thank the people who made this possible. I thank Bart and Vincent and the symmetric key group for their feedback during the regular design meetings that were held before the start of the competition. I would like to thank the people who helped me to finalize the submission: Alp for writing the first version of the reference code, Junfeng for the hardware implementation, and Antoon for the extra coding support, my neighbours Marie-Paule and Chris for allowing me to use their credit card to send the submission via DHL at the last minute. Thank you all! Finally, it is my pleasure to thank many distinguished researchers who analyzed Hamsi and implemented it in hardware and software, and thank you many carefull readers who pointed out the bugs in the document and the implementations. I appreciate their time and effort.

In the fall of 2010, I visited Sony Japan, as part of the Global Internship Program. I would like to thank System Security and System Technologies Laboratories for making this visit possible and for hosting me for three months.

I would like to thank my colleagues at COSIC whom I have met during this PhD journey: Hongjun, Markulf, Gautham, Nicky, Vesselin, Elena, Andrey, Elmar, Jan, Begül, Gabriel, Nessim, Paul, Sebastiaan, Claudia, Seda, Jonsung, Brecht, Junfeng, Fré, Li, Quingju, Lejla, Jiazhe, Deniz, Kerem, Nele, Orr, Mina, Bart, and many others.

A PhD degree is not only earned by doing good research but also by completing a not so trivial pile of administrative tasks and paperwork. Without the help of Péla Noë who handled every possible question with great care and patience, I would have been lost after the very first steps. The financial issues were handled by Elvira Wouters and Elsy Vermoesen. Thank you!

Finally, I would like to thank my parents and brothers for their understanding and encouragement through those years, during which I could not spend enough time with them.

Özgül Küçük
Matongé, April 2012

Abstract

The topic of this thesis is the design and analysis of cryptographic hash functions. A hash function is a map from variable-length input bit strings to fixed-length output bit strings. Despite their simple definition, hash functions play an essential role in a wide area of applications such as digital signature algorithms, message authentication codes, password verification, and key derivation.

The main contribution of this thesis is a novel and elegant proposal of a cryptographic hash function. In this thesis, we approach the problem of the design and analysis of cryptographic hash functions with a particular example, the hash function Hamsi. The design of Hamsi is based on the use of a relatively light underlying primitive in each iteration of the mode of operation, combined with a strong message expansion function. We investigate the design constraints of this approach by analyzing Hamsi. In the first part, we cover the design aspects of Hamsi and also propose a variant called Hamsi$^{\oplus}$. In the sequent parts we provide analysis results, namely indifferentiability analysis and collision analysis. Finally, as a separate research study we analyze the initialization of the stream cipher Grain.

Samenvatting

Het onderwerp van deze thesis is het ontwerp en analyse van cryptografische hashfuncties. Een hashfunctie is een afbeelding met als invoer een bitstring van variabele lengte en als uitvoer een bitstring van vaste lengte. Ondanks hun eenvoudige definitie, spelen hashfuncties een essentiële rol in een groot aantal toepassingen zoals digitale handtekeningen, data integriteit, paswoord verificatie en sleutelafleiding.

De voornaamste bijdrage van deze thesis is een nieuw en elegant ontwerp van een cryptografische hashfunctie. In deze thesis bestuderen we het ontwerp en analyse van cryptografische hashfuncties aan de hand van een welbepaald voorbeeld, namelijk de hashfunctie Hamsi. Hamsi is gebaseerd op een relatief eenvoudig primitief dat herhaaldelijk wordt uitgevoerd, gecombineerd met een sterke expansiefunctie voor het bericht. We onderzoeken de beperkingen die dit ontwerp met zich meebrengt door Hamsi te analyseren.

In het eerste deel behandelen we de verschillende aspecten van het ontwerp van Hamsi en stellen we ook een variant voor die we Hamsi$^{\oplus}$ noemen. In de daaropvolgende delen geven we de resultaten van twee analyses, namelijk een onderscheidsanalyse en een botsingsanalyse. In een apart onderzoek analyseren we ook de initialisatie van het stroomcijfer Grain.

Contents

Acknowledgements **i**

Abstract **iii**

Samenvatting **v**

Contents **vii**

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**

1.1 Cryptography 1

1.2 Cryptographic Hash Functions 2

1.2.1 Types 2

1.2.2 SHA-3 4

1.3 About this Thesis 4

2 Background on Hash Functions **7**

2.1 Cryptographic Hash Functions 7

2.2 Security Requirements 8

2.3 Modes Of Operation 9

2.3.1 The Merkle-Damgård Construction 9
2.3.2 The Sponge Construction 10
2.4 Generic Attacks 11
2.4.1 Preimage and 2nd Preimage Attacks 12
2.4.2 Generic Collision Attack: The Birthday Attack 13
2.5 Differential Cryptanalysis 15
2.6 Applications of Hash Functions 16
2.6.1 Digital Signatures 16
2.6.2 Protection of Passwords 17
2.6.3 Confirmation of Knowledge 17
2.6.4 Pseudo-random bit generation 18
2.6.5 Key derivation 18
2.6.6 Construction of MAC Algorithms 18

3 The Hash Function Hamsi 19

3.1 Introduction 20
3.2 Design Choices 21
3.3 Preliminaries 23
3.3.1 Endianness 24
3.4 General Design 25
3.5 Initial Values 26
3.6 Message Padding 26
3.7 Message Expansion 28
3.7.1 Hamsi-256/Hamsi-224 29
3.7.2 Hamsi-512/Hamsi-384 29
3.7.3 Implementation 30
3.8 Concatenation 31
3.9 The Non-Linear Permutation P 32

3.9.1 Addition of Constants and Counter 32
3.9.2 Substitution Layer 34
3.9.3 Diffusion Layer 35
3.10 Truncation T 37
3.11 The Non-Linear Permutation P_f 38
3.12 Truncations T_{224}, T_{384} 38
3.13 Number of Rounds 39
3.14 Hamsi$^{\oplus}$ 40
3.14.1 Truncations T_{256}, T_{224} 43
3.15 Implementation Aspects 43
3.15.1 Software Implementation 43
3.15.2 Hardware Implementation 44
3.16 Conclusion 47

4 Security Analysis of Hamsi 49

4.1 Collision Attacks on Hash Functions 50
4.2 Resistance of Hamsi to Differential Attacks 51
4.2.1 Number of Active Sboxes 51
4.2.2 An Upper Bound on the Probability of 1-round Differential Characteristics 53
4.2.3 Pseudo-collisions 54
4.3 External Analysis Results on Hamsi 55
4.3.1 Analysis of the Compression Function of Hamsi-256 55
4.3.2 Analysis Results on the Hash Function 56
4.4 Collision Analysis of Hamsi-256 57
4.4.1 Linearization 58
4.4.2 Imposing a Zero-Difference on the Sboxes 60
4.4.3 The Effect of Message Expansion 64

4.4.4 A Bound on the Number of Active Sboxes 64
4.5 On the Security of Hamsi$^{\oplus}$ 65
4.6 Conclusion 66

5 Indifferentiability of Hamsi and Hamsi$^{\oplus}$ 67

5.1 Indifferentiability 68
5.2 Indifferentiability of Hamsi 70
5.2.1 Hamsi Mode of Operation 70
5.2.2 Graph Representation 71
5.2.3 The Distinguisher's Setting 72
5.2.4 Simulators $P[RO]$ and $P_f[RO]$ 73
5.2.5 Indifferentiability Proofs 75
5.3 Indifferentiability of Hamsi$^{\oplus}$ 82
5.3.1 Description of Hamsi$^{\oplus}$. 82
5.3.2 Graph Representation 82
5.3.3 Indifferentiability Proofs 83
5.4 Impact of the Indifferentiability Bound 87
5.5 Conclusion 87

6 Analysis of Grain's Initialization Algorithm 89

6.1 Description of Grain 89
6.1.1 Keystream Generation 90
6.1.2 Key and IV Initialization 90
6.1.3 Grain v1 91
6.1.4 Grain-128 91
6.2 Slide Attacks 92
6.2.1 Related (K, IV) Pairs 92
6.2.2 A Related-Key Slide Attack 94

6.2.3 Speeding up Exhaustive Key Search 94
6.2.4 Avoiding the Sliding Property 95
6.3 Conclusion 96

7 Conclusion **97**
7.1 Contributions of this Thesis 97
7.2 Open Problems 99

A Linear Codes Used for the Message Expansion of Hamsi **103**
A.1 Construction of the Linear Code [128,16,70] over F_4 103
A.2 Construction of the Linear Code [256,32,131] over F_4 104
A.3 Generator Matrices 105

Bibliography **113**

Curriculum Vitae **127**

List of Figures

2.1 Security requirements for hash functions 8

2.2 The Merkle-Damgård strengthening 9

2.3 The Sponge Construction 10

2.4 A digital signature scheme using a hash function 16

3.1 Representation of the Hamsi-256 state 20

3.2 Representation of the Hamsi-512 state 20

3.3 The general design of Hamsi 27

3.4 Concatenation in Hamsi-256/Hamsi-224 31

3.5 Concatenation in Hamsi-512/Hamsi-384 32

3.6 Hamsi Sbox acting on the columns in bitslice implementation . . . 35

3.7 Application of L over the diagonals of the state matrix. 36

3.8 Truncation in Hamsi-256 38

3.9 The general design of Hamsi$^{\oplus}$ 41

3.10 An iteration of Hamsi$^{\oplus}$ before the finalization. 42

3.11 Cycles/byte versus message length. Platform: amd64, Sandy Bridge, 2011 Intel Core i7-2600K (eBASH). 44

4.1 An example of a collision producing path before finalization in Hamsi 54

4.2 Form of collision producing differentials in Hamsi-256 57

4.3 Pseudo-collision path after linearization 60

4.4 A sketch of our method 61
4.5 Finding a right pair without the message expansion for Hamsi-256 63
4.6 A right pair without the message expansion for Hamsi$^{\oplus}$-256 64

5.1 The indifferentiability setting 69
5.2 The Hamsi mode of operation 70
5.3 The distinguishers setting 73

6.1 Grain during the keystream generation phase 90
6.2 Grain during the initialization phase 91

List of Tables

1.1 The second round SHA-3 candidates, the final round candidates are indicated in bold. 4

3.1 Hamsi variants and security claims in bits 20

3.2 Notation 24

3.3 Hamsi variants and parameters in bits 24

3.4 Initial Values of Hamsi 28

3.5 Constants of P 33

3.6 Constants of P suitable for bitsliced implementation 33

3.7 Truth table of the Hamsi Sbox 34

3.8 Equations of the Hamsi Sbox suitable for bitslice implementation [106]. 35

3.9 Application of L in Hamsi-256/Hamsi-224 35

3.10 Application of L in Hamsi-512/Hamsi-384 36

3.11 Constants of P_f 39

3.12 Constants of P_f suitable for bitslice implementation 39

3.13 Number of rounds of permutations P and P_f for Hamsi 40

3.14 Hamsi$^{\oplus}$ variants and parameters in bits 40

3.15 Number of rounds of permutations P and P_f for Hamsi$^{\oplus}$ 40

3.16 Number of 128-bit vector operations for Hamsi-256 44

3.17 Ranking of the 14 SHA-3 designs in terms of Throughput-to-Area ratio metric [63] in ASIC. 45

3.18 Summary of performance of 256-bit and 512-bit variants of SHA-3 2nd round candidates in FPGAs [69]. The performance is measured by four metrics: throughput to area ratio (T/A), throughput (Thr), area (Area) and execution time for short messages (Short M). The symbols ✓, ≈, × refer to best, medium and worst respectively. . . 46

4.1 Collision attacks on round-1 SHA-3 competitors 51

4.2 The difference distribution table of the Hamsi Sbox 52

4.3 Number of input differentials 54

4.4 Choice of differences from the difference distribution table 59

Chapter 1

Introduction

1.1 Cryptography

Cryptography is the science of constructing schemes or protocols that enable users to communicate over an insecure channel without sacrificing the privacy or the authentication of the transmission. Cryptanalysis is the science (and art) of breaking those schemes where as cryptology or crypto refers to the study of both. Cryptology is an interdisciplinary area that uses tools from mathematics and computer science.

Modern cryptography has emerged from the Enigma [1] machine to the AES [42] through a history of design and analysis of ciphers. Modern cryptography has diversified into many applications. Nowadays it is not only used for securing data, but also for electronic payments and voting schemes, for proving that certain information is known without revealing it, or for sharing a secret in such a way that only qualified subsets of the shares can reconstruct the secret.

The symmetric cryptographic primitives are the basic building blocks of cryptography; they can be *block ciphers*, *stream ciphers* and *hash functions*; they play an important role in all those applications. Today's computing technologies (secure email, online banking, ATMs, mobile applications) require secure and efficient crypto schemes.

The tradeoff between security and efficiency is the most important issue in the design of cryptographic algorithms. Cryptographic algorithms should be suitable to implement in a variety of platforms and have reasonable performance with an adequate security margin.

Confidentiality deals with keeping the secrecy of information whereas authentication is about whether someone or something is, what it is declared to be. Protecting the confidentiality and authenticity of information is an important concept in cryptography and encryption on its own does not provide authenticity. Cryptographic hash functions are important tools for providing authentication of data and entities. For a good hash function it should be infeasible to find two messages with the same hash value, or to find the corresponding message given the hash value. Hence those properties form the basis for a secure authentication with a cryptographic hash function.

1.2 Cryptographic Hash Functions

In practice, cryptographic hash functions are defined as fixed mappings from variable input bit strings to fixed length output bit strings. In theory, they are defined as "keyed" mappings, namely as a family of functions parameterized with a "key" which is known, unlike cryptographic keys which are secret. This distinction creates a separation between practical applications and theory of cryptographic hash functions.

1.2.1 Types

It is possible to classify hash functions based on the design approaches. We will first make two main distinctions; hash functions based on mathematical hard problems and others. In the first type of designs, the security problem is reduced to a mathematical problem which is believed to be hard to solve. This approach has several problems; most of this type of designs suffer from efficiency, and sometimes the security reductions are valid for certain parameters only. It might be the case that the proofs do not hold for the proposed parameters. Also some examples may not meet all security criteria required for hash functions. However, despite all those facts this is a promising area of research with examples, VSH [38] – based on a number-theoretic problem, FSB [9] – based on coding theory, ECOH [33] – using elliptic curves, SWIFFTX [8] and LASH [18] – based on certain lattice problems.

The second design approach is more ad hoc and is based on experience and know-how. Most designs fall in this category. The security of those designs can be verified to an extent by showing the strength with respect to certain classes of attacks. Within this category we identify a variety of approaches. It is hard to make a clear distinction between those. When used in a special mode block ciphers can be used to construct hash functions as demonstrated by Preneel et al. in [108]. These constructions reduce the security of the hash function to the security of the block cipher shown by Black et al. in [29]. Recently, there have been proposals

that use a permutation (or transformation) in a special mode, namely the sponge construction [21].

Some hash functions following the second approach are called "dedicated" designs, namely, MD and SHA family, which are based on a block cipher in Davies-Meyer mode. They are designed from scratch to be efficient in software. At the rump session of CRYPTO 2004 with the announcement of results by Chinese researchers [129, 130], there has been a series of attacks on the most widely used dedicated hash functions, SHA-0/1 as well as the MD4 family. Those attacks are based on differential cryptanalysis techniques applied to hash functions, combined with complex message modification techniques.

We now consider a second approach to classify hash functions based on a different perspective: hash functions based on strong primitives and hash functions based on light primitives.

Since most hash function proposals are sequential, we can state in general that there are two necessary ingredients for any hash function: an iteration mode, and a chaining transformation. The chaining transformation takes at least two inputs: the previous chaining variable (or a fixed IV), and a message block. The initial value (IV) is part of the description of the hash function, and it is a fixed input to the first chaining transformation. In combination with an appropriate padding method, this iteration mode, which is known as the Merkle-Damgård construction, has determined the basic structure of most hash functions. The most elegant part of this construction is that the problem of designing a collision-resistant hash function can be reduced to that of designing a collision-resistant chaining transformation. This is not sufficient, however. Proofs, such as the one presented by Merkle and Damgård, show that an attack on the hash function immediately leads to an attack on the chaining transformation. Even though the converse is not true, it is still a general belief that any structural weakness in the chaining transformation should be avoided. This results in an approach of designing hash functions based on strong primitives. However, as it is shown by the latest attacks on dedicated hash functions, designing a secure and efficient hash function is not an easy task. We are interested in the other approach where the chaining transformation is based on a light primitive (small number of rounds for example). By a light primitive we mean a chaining transformation that does not have the strength of a block cipher: it is more like a reduced round version of a block cipher or a permutation with a small number of rounds. Typically mixing of message blocks into the state takes several iterations. The construction enforces that the security of the compression function should be considered in the context of the iteration mode. The cryptographic hash function Hamsi, to which a large fraction of this thesis is devoted, is the result of such an approach. The stream based hash functions such as RadioGatun [20], Grindahl [83], and Fugue [64] are some other examples.

1.2.2 SHA-3

In 2007, NIST announced a public competition [102] to develop a new cryptographic hash algorithm as a response to the recent advances in the cryptanalysis of hash functions. Out of the 64 submitted algorithms, 51 were accepted as complete submissions, after which the 1st round of the competition started. The 1st-round candidates were presented in February 2009 at the First SHA-3 Candidate Conference held by NIST in Leuven. This was followed by an active period of analysis, in which many candidates were broken or shown to have weaknesses. The fourteen 2nd-round candidates (see Table 1.1), which were announced in July 2009, are: Blake, BMW, CubeHash, Echo, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SIMD, and Skein. One of those 2nd-round candidates, Hamsi [85], was designed as part of this thesis project. In December 2010, five algorithms have been selected for the final round, these are: BLAKE, Grøstl, JH, Keccak and Skein. NIST published a report [122] explaining their decision for each candidate, Hamsi was not chosen because of ROM requirements and 2nd-preimage attacks [51, 59].

Table 1.1: The second round SHA-3 candidates, the final round candidates are indicated in bold.

Candidate	Principal Submitter
BLAKE	Jean-Philippe Aumasson
Blue Midnight Wish	Svein Johan Knapskog
CubeHash	Daniel J. Bernstein
ECHO	Henri Gilbert
Fugue	Charanjit S. Jutla
Grøstl	Lars R. Knudsen
Hamsi	Özgül Küçük
JH	Hongjun Wu
Keccak	The Keccak Team
Luffa	Dai Watanabe
Shabal	Jean-François Misarsky
SHAvite-3	Orr Dunkelman
SIMD	Gaëtan Leurent
Skein	Bruce Schneier

1.3 About this Thesis

This thesis investigates the design and analysis of cryptographic hash functions, and illustrates this with a specific example named Hamsi. Most of the results are related to this algorithm. The general structure of this thesis is outlined below:

- Chapter 1 provides a brief introduction to cryptography and summarizes this thesis.
- In Chapter 2, we zoom in on cryptographic hash functions. We mention some of their applications and discuss the basic concepts.
- In Chapter 3, we present the hash function Hamsi. Hamsi has been submitted to the NIST SHA-3 competition and has made it to the second round of the competition. Hamsi has some unique design features that distinguish it from other designs. It combines a light compression function with a strong message expansion. In this chapter, we concentrate on the design aspects of Hamsi. We also present a wide-pipe variant, called Hamsi$^{\oplus}$.
- In Chapter 4, we provide some cryptanalysis results on Hamsi. We first briefly mention the resistance of Hamsi to differential attacks, then we summarize the external analysis results on Hamsi. Finally, we present our collision analysis of the compression function of Hamsi-256, which is so far the best collision analysis result.
- In Chapter 5, we prove the indifferentiability of Hamsi and Hamsi$^{\oplus}$. Collision resistance is one of the essential criteria for hash functions. Given a collision resistant chaining transformation, the Merkle-Damgård construction provides a collision resistant hash function; however, it does not guarantee other properties. Indifferentiability is an approach to show the strength of the hash function with respect to many criteria. Assuming that the underlying primitives are ideal, an indifferentiability proof shows that the hash function is as strong as a random oracle truncated to a fixed length. In this chapter, we introduce the indifferentiability concept, and next we define the necessary representations for our proofs such as the Hamsi mode of operation, the graph representation and the distinguisher's setting. We describe our simulators and present the indifferentiability proofs. Following the same construction and skipping the repetitions we also prove the indifferentiability of Hamsi$^{\oplus}$. After a discussion about the impact of the indifferentiability bound we conclude this chapter.
- In Chapter 6, we focus on one particular aspect of another important class of cryptographic algorithms: stream ciphers. We show how to apply slide attacks to the initialization of the stream cipher Grain. Grain is a family of ciphers with two members, Grain v1 and Grain-128. Our results applies to both versions. They were submitted to the eSTREAM competition and one of its members Grain v1 has been selected for the eSTREAM portfolio together with three other ciphers. After a brief introduction to the Grain family of ciphers and the slide attacks,we show the sliding property of the initialization and describe how to exploit this fact to speed up the exhaustive key search by a factor of two. We conclude this chapter with a simple fix to avoid the sliding property. The results of this chapter are published in [48].

- We conclude this thesis with a summary of our contributions and some open problems of varying difficulty in Chapter 7.

Results contributed during this thesis study, but which are not included in this thesis can be found in [123, 132].

Chapter 2

Background on Hash Functions

In this chapter we present a brief introduction to the basic concepts of cryptographic hash functions and their applications. For a comprehensive study of the subject we refer to the thesis of Preneel [107].

2.1 Cryptographic Hash Functions

Definition 2.1. *A cryptographic hash function H is a map from variable-length input bit strings to fixed-length output bit strings,*

$$H : \{0,1\}^* \to \{0,1\}^n \ .$$

On the other hand, a cryptographic hash function can be defined more formally as an instance from a family of functions. Let $H : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^n$ be a family of functions. For a particular key $K \in \{0,1\}^k$, $H_K : \{0,1\}^* \to \{0,1\}^n$ is defined for every $m \in \{0,1\}^*$ by $H_K(m) = H(K,m)$. In practice when we refer to a hash function we mean this instance. If the key K is secret then the hash function is used for authentication and it is called a *message authentication code* abbreviated as MAC. To simplify the notations we will drop K most of the time.

Hash functions compress the input, that is the domain of the input is larger than the range, hence collisions are unavoidable. However, a secure hash function should be collision-resistant, meaning that it should be hard to find collisions. But a collision can be found accidentally or computed in advance; to overcome this problem in formal proofs one has to find a collision for each member of the family, which makes it harder to pre-compute the collision for each key. Hence, in order to define formal security notions, cryptographic hash functions are defined

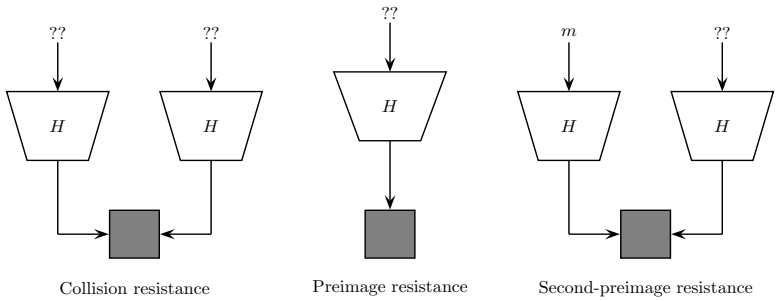


Figure 2.1: Security requirements for hash functions

as families. Namely, in [45] Damgård introduces infinite family of hash functions that captures his definition of computational infeasibility. But this method is not applicable to practical concrete constructions, because it gives asymptotic results. Hence finite families of hash functions are used in formal security proofs.

2.2 Security Requirements

Hash functions should satisfy the following three fundamental security properties that can be informally defined as follows:

- *Collision resistance*: it should be computationally infeasible to find distinct input values mapping to the same output value, that is $m \neq m^*$ but $H(m) = H(m^*)$.
- *Preimage resistance*: it should be computationally infeasible to find an input m which hashes to a specified output $y = H(m)$.
- *Second-preimage resistance*: Given m such that $H(m) = y$, it should be computationally infeasible to find a second distinct input m^* that hashes to the same output as specified input, that is $H(m^*) = y$.

Note that collision resistance implies 2nd-preimage resistance, this follows by showing that any 2nd-preimage also gives a colliding pair. However, the generic complexities of finding a 2nd-preimage are much higher than finding a colliding pair. Formal definitions and non-trivial relations between the security requirements and enhancements are examined in several papers by Rogaway and Shrimpton [115], Rogaway [114], Andreeva et al. [4], and Reyhanitabar et al. [110].

2.3 Modes Of Operation

Hash functions process variable length inputs. While they typically consist of building blocks that take fixed length inputs, a mode of operation of a hash function is an algorithm that handles the variable input length. In most of the constructions in the literature this is achieved in a sequential manner.

2.3.1 The Merkle-Damgård Construction

The well-known Merkle-Damgård construction [46, 98] has determined the basic structure of iterated hash functions. In combination with a proper padding method, the Merkle-Damgård construction iterates sequentially a chaining transformation that takes as input a message block and the previous chaining value. The iteration algorithm is also referred to as compression function. The input lengths to the iteration algorithm are determined by the length of the message block and the chaining value. Padding is an algorithm to extend inputs of arbitrary bit-length to a string with a length that is a multiple of the input message block length. One of the common ways is obtained by appending a single '1′ bit and '0' bits as many as needed and the binary representation of the length of the message.

Definition 2.2. *Let f be a chaining transformation and IV the fixed initial value. Let $m_1, m_2, \ldots, m_l$ be a sequence of padded message blocks as described above. The following iteration is called Merkle-Damgård strengthening since Lai and Massey [86].*

$$h_i = f(h_{(i-1)}, m_i), \quad h_0 = IV, \quad 1 \leq i \leq l.$$

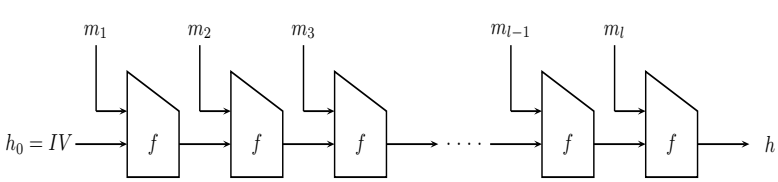


Figure 2.2: The Merkle-Damgård strengthening

Given $H(m)$, if one can compute $H(m||m^*)$ for an arbitrary m^*, then the scheme is vulnerable to a length extension attack. Note that having a final transformation different from the chaining transformation precludes such an attack.

The most elegant part of the Merkle-Damgård construction is that it reduces the problem of designing a collision-resistant hash function to that of designing a collision-resistant chaining transformation. Namely, if the chaining transformation

is collision resistant then the hash function is collision resistant. The proof follows by showing that a collision in the hash function requires a collision in the chaining transformation at some iteration.

The relation between the chaining transformation and the hash function is a challenging area to explore. While it seems easier for the chaining transformation to inherit properties from the hash function, the other way around is not easy. Namely, it is typically difficult to reason which properties of the chaining transformation might yield weaknesses in the hash function.

The Merkle-Damgård construction is quite well studied and several weaknesses (generic attacks) such as multi-collisions [73], long-message second preimages [49], [75] and differentiability [93] have been shown for this construction.

Following the attacks on Merkle-Damgård several other iterated constructions have been proposed such as EMD [15], ROX [7], HAIFA [24], and, Sponge [21, 22].

A different construction method is based on directed trees. These are known as tree-based hash algorithms and they can be implemented in parallel [46, 116]. MD6 [113] and ESSENCE [91], are examples for such hash algorithms.

2.3.2 The Sponge Construction

Another example of an iterative construction is the *sponge construction* which is proposed as an alternative to Merkle-Damgård by Bertoni et al. [21]. It is a generalization of hash functions to more general functions whose output length is arbitrary. Namely, by using a fixed-length transformation (or permutation) f (see Figure 2.3) that operates on a fixed number of $b = r + c$ bits, i.e, r is called the bitrate and c is called the capacity, the sponge construction builds a function F with variable-length input and arbitrary output length.

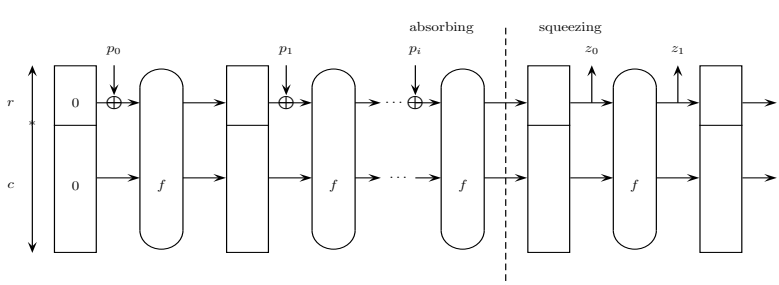


Figure 2.3: The Sponge Construction

The sponge construction as can be seen in Figure 2.3 operates in two phases:

- **Absorbing phase:** The r-bit message blocks (p_i) are XORed with the first r bits of the state. When all the message blocks are processed the squeezing phase starts.
- **Squeezing phase:** The first r bits of the state are returned as output blocks. The number of output blocks is chosen by the user.

The security of the sponge construction is proved by Bertoni et al. [22] by using the indifferentiability framework introduced by Maurer et al. [93]. The capacity c determines the security namely, it is proven in [22] that the success probability of differentiating a sponge construction calling a random permutation or transformation from a random oracle is upper bounded by $N^2 \cdot 2^{-(c+1)}$ where N is the number of calls to f and $N \ll 2^{c/2}$. Note that the bound is independent of the output length. The c bits are not affected by the input blocks and are never output during the squeezing phase.

The *hermetic sponge strategy* executes the sponge construction based on an underlying permutation f that should not have any structural distinguishers. The hash function Keccak, which is currently a final round candidate in the SHA-3 competition, follows the hermetic design strategy. However, Aumasson and Meier [13] showed the existence of zero-sum distinguishers for 16 rounds of the underlying permutation f of Keccak hash function. Boura and Canteaut [31] extended the zero-sum distinguishers to 18 rounds and 20 rounds. Finally, Boura, Canteaut and De Cannière [32] showed the existence of zero-sums on the full permutation (24 rounds). Although, the distinguishers seem to be very weak, i.e., with high complexity, and there seem to be no evidence of threat to the hash function Keccak, they reveal a conflict to the assumptions made by the hermetic sponge strategy.

The features mentioned above and the security proofs make the sponge construction an attractive alternative to Merkle-Damgård for building hash functions and stream ciphers.

2.4 Generic Attacks

An attack on a hash function is called *generic* if it is valid regardless of the inner components of the hash function. For example if the hash function uses a permutation, replacing the permutation with another one should not affect the complexity of a generic attack on that hash function.

2.4.1 Preimage and 2nd Preimage Attacks

For many hash constructions the digest length (most of the time denoted with n) determines the security against several generic attacks. Namely,

- Generic preimage: Given a hash value h, it takes approximately 2^n evaluations of the hash function to find a preimage m such that $H(m) = h$, see Algorithm 1.

Algorithm 1 Generic preimage attack

```
Given h ∈ {0,1}^n
for all messages m, do
  if H(m) = h then
    return m
  end if
end for
return false
```

- Generic 2nd preimage: Given $h = H(m)$, it takes approximately 2^n evaluations of the hash function to find a 2nd preimage $H(m^*) = h$, see Algorithm 2.

Algorithm 2 Generic 2nd preimage attack

```
Given m and h ∈ {0,1}^n, such that h = H(m)
for all messages m* ≠ m, do
  if H(m*) = h then
    return m*
  end if
end for
return false
```

If one can construct an attack faster than the generic attacks either by exploiting the structure of inner primitives or mode of operation, then the hash function is called *broken*.

Note that in sponge construction the security against generic attacks does not depend on the digest length but on another parameter called the capacity, see Section 2.3.2.

2.4.2 Generic Collision Attack: The Birthday Attack

The birthday attack is a generic collision attack on any hash function. It is based on the birthday bound. The general setting is as follows: assume that we have q balls and N bins where $q \leq N$ and we throw the balls one by one into the bins. The probability of each ball to land in any of the N bins is equal. A collision occurs as soon as a bin contains more than one ball. We want to compute the probability of a collision denoted with $C(N, q)$.

Lemma 2.1 ([16])**.** *The inequality*

$$(1 - \frac{1}{e}) \cdot x \leq 1 - e^{-x} \leq x$$

is true for any real number x with $0 \leq x \leq 1$.

Theorem 2.2 ([16])**.** *The probability of a collision $C(N, q)$ when we throw $1 \leq q$ balls to $q \leq N$ bins is bounded as follows:*

$$1 - e^{-q(q-1)/2N} \leq C(N, q) \leq \frac{q(q-1)}{2N} \quad .$$

Proof. We throw q balls, let Q_i denote the probability for the i-th ball to collide with any of the previous ones. When we throw the i-th ball there are at most $i-1$ occupied slots and it is equally likely to land in any of them. The probability $Pr[Q_i]$ is at most $i - 1/N$, so

$$\begin{aligned} C(n, q) &\leq Pr[Q_1] + Pr[Q_2] + \ldots + Pr[Q_q] \\ &\leq 0 + \frac{1}{N} + \ldots + \frac{q-1}{N} \\ &= \frac{q(q-1)}{2N} \quad . \end{aligned}$$

For the lower bound we will compute the probability of no collision after throwing the i-th ball, we denote this by P_i. If there is no collision this means that all the balls should be occupying different slots, so the probability of no collision after throwing the $(i+1)$-st ball is $Pr[P_i + 1|P_i] = \frac{N-i}{N} = 1 - \frac{i}{N}$. On the other hand

the probability of no collision after throwing q balls is $1 - C(N, q)$, hence we have:

$$\begin{aligned}
1 - C(N,q) &= Pr[P_q] \\
&= Pr[P_q|P_{q-1}] \cdot Pr[P_{q-1}] \\
&= Pr[P_q|P_{q-1}] \cdot Pr[P_{q-1}|P_{q-2}] \ldots Pr[P_2|P_1] \cdot Pr[P_1] \\
&= \prod_{i=1}^{q-1} Pr[P_i + 1|P_i] \\
&= \prod_{i=1}^{q-1} (1 - \frac{i}{N}) \\
&\leq \prod_{i=1}^{q-1} e^{-i/N} \\
&= e^{-1/N - 2/N - \ldots - (q-1)/N} \\
&= e^{-q(q-1)/2N} \quad .
\end{aligned}$$

□

Note that $i/N \leq 1$ implies that we can use the inequality from Lemma 2.1, that is $1 - x \leq e^{-x}$ for each term of the above expression. Hence we have:

$$1 - e^{-q(q-1)/2N} \leq C(N, q) \leq \frac{q(q-1)}{2N} \quad .$$

We can conclude that the collision probability grows roughly proportional to q^2/N. The problem can be formulated trivially for hash functions. For a hash function H with digest length n the expected number of hash evaluations to find a colliding message pair, i.e $H(m) = H(m^*)$, where $m \neq m^*$ is about $2^{n/2}$, the time complexity of the birthday attack is $O(2^{n/2})$.

The birthday attack can be used to create meaningful collisions as shown by Yuval [133]. Start with two messages, first one the original, second one which we want to fake. Make changes at $n/2$ places of each message without changing the meaning. Then produce $2^{n/2}$ variants of each message, with high probability we will expect a collision.

The birthday attack has memory requirements $O(2^{n/2})$, namely one has to keep a list of messages and the corresponding hash values, i.e $(h(m), m)$, and the size of

Algorithm 3 Collision search with the birthday paradox

```
for about 2^{n/2} many m do
  compute y = H(m)
  if there is a (y, m*) pair in the hash table then
    return (m, m*)
  end if
  add (y, m) in the hash table
end for
return false
```

this list is $2^{n/2}$. Pollard's rho-method and the cycle finding algorithms [105, 84] can be used to significantly reduce the memory requirements. It is also possible to run the collision search in several processors in parallel, as described by van Oorschot and Wiener in [125, 126].

2.5 Differential Cryptanalysis

Differential cryptanalysis is a cryptanalysis method based on the analysis of the effect of specific differences on a pair of inputs on the output of the cipher. This is usually achieved by analyzing the propagation of differences through the cipher. It was introduced by Biham and Shamir in 1990 [117]. It was first applied to DES [25, 26], and FEAL [101].

The most common differences are defined with respect to the XOR operation; however the difference operation can be subtraction modulo 2^n or difference with respect to any group operation. For block ciphers it is most of the time determined by how the key is incorporated to the cipher. For ciphers which consists of XOR, addition and rotation operations it is a nontrivial task to analyze the propagation of differences through the cipher [77].

The propagation of a XOR difference through an affine function can be determined with probability 1; for nonlinear functions this is determined with a certain probability smaller than 1. The propagation of differences through the rounds of an iterated cipher is represented by *differential characteristics* and an r-round differential characteristic is defined as the set of differential characteristics of r consecutive rounds. The input and corresponding output difference pair of a cipher is called a *differential*. Another way of describing a differential is as the set of all differential characteristics with the same input and output difference as the differential. If we assume that the rounds are independent then the probability of an r-round differential characteristic is computed by multiplying the probabilities of each round.

Differential cryptanalysis is a very powerful technique that can be applied to modern symmetric primitives such as block ciphers, stream ciphers and hash functions. It has also been extended to other cryptanalysis methods such as higher-order differential cryptanalysis [81], truncated differential cryptanalysis [81], boomerang attacks [127], impossible differential cryptanalysis [82, 23].

For block ciphers the differences are propagated without the knowledge of the key. The analysis might reveal the key values or enable to distinguish the cipher from a random block cipher. For hash functions there is no secret key; but any message pair satisfying the given input difference and zero output difference results in a collision. This gives more freedom when applying differential cryptanalysis to hash functions; once a satisfying message pair is found there is no further need to search for the correct "key" and also choosing correct values for the initial rounds may enable to bypass those rounds with probability one. Those facts have the potential to reduce the complexity of such an attack when applied to hash functions.

2.6 Applications of Hash Functions

2.6.1 Digital Signatures

A digital signature is a string obtained by applying a digital signature scheme, that depends on a secret value known to the signer and on the content of the message being signed. Many signatures are created by the use of a private signing function and a hash function. In most cases public key algorithms are used for signing but they are quite slow compared to their symmetric counterparts, e.g., block ciphers and stream ciphers. For efficiency reasons instead of signing the message, the hash of the message which is much shorter, is signed. The application of a hash function to the message before signing is also crucial to destroy the structure of the message in the signature scheme which might be exploited [56]. As shown in Figure 2.4, in a digital signature scheme the hash of the message is signed and appended to the message and the result is sent to the recipient.

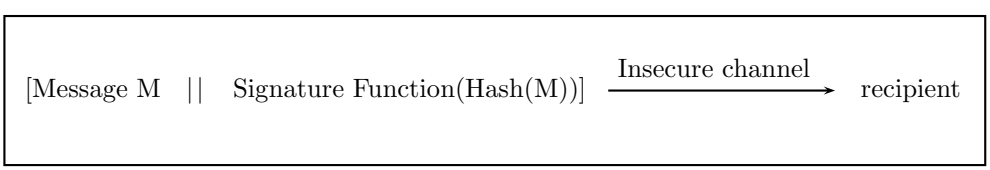


Figure 2.4: A digital signature scheme using a hash function

Signatures are generated with the signer's private key and verified with the public key of the signer. If one can construct messages that hash to the same value then this creates a security weakness in the signature scheme, namely the same signature will be valid for different messages. For this reason, collision resistance is a necessary requirement for the usage of hash functions in digital signature schemes. The same holds for pre-image and 2nd pre-image resistance.

2.6.2 Protection of Passwords

In many operating systems a password file is stored to enable users to get access to certain privileges. Cryptographic hash functions are used to keep the hash of passwords instead of keeping the password itself. If the file is revealed then the security of passwords is still protected if the hash function is pre-image resistant.

Similarly hash functions can be used for intrusion detection and virus detection in a system. The hash of files are stored and secured for future comparison of alteration.

One-time passwords

One-time passwords are passwords that are valid for only one login session or transaction. Given an initial key hash functions can be used as one-way functions to compute new passwords by applying the hash as many times as necessary. Hash chains, suggested by Lamport [88], are an example of usage of hash functions to generate one time passwords for insecure environments. Authentication of the user by the server works as follows: the server maintains $h^n(password)$ provided in advance by the user, when the user wants to authenticate she supplies $h^{n-1}(password)$ to the server, the server computes $h(h^{n-1}(password))$ and verifies that it matches with the hash chain stored. For the next authentication $h^{n-1}(password)$ is stored.

2.6.3 Confirmation of Knowledge

Cryptographic hash functions can be used to prove the knowledge of information without revealing it. This can be established by revealing the hash of the secret. For example confirmation of the key in an authenticated key establishment protocol, document-dating or time stamping by hash-code registration.

2.6.4 Pseudo-random bit generation

Given a random seed a cryptographic hash function can be used to generate pseudo-random bits by successively applying the function to the seed. It can also be applied to the output seed of a hardware random number generator to generate unbiased pseudo-random bits.

2.6.5 Key derivation

A key derivation function is an algorithm that is used to derive keys from a secret value such as a value obtained by Diffie-Hellman key establishment. A hash function can be used as a part of a key derivation algorithm.

2.6.6 Construction of MAC Algorithms

Hash functions can be used to provide the authenticity of data or entities. Namely, the authentication of a message sent over an insecure channel can be established with the use of MAC (Message Authentication Code) algorithms, which can be built from hash functions with an additional input – the key. The key is shared between the communicating partners and kept secret. The communication works as follows, whenever one of the parties sends a message to the other, she computes an authentication tag for the message by using the MAC algorithm and the secret key. She sends this authentication tag together with the message. The recipient on the other end uses the MAC algorithm and the secret key to verify the authentication tag of the message. Moreover, it should be impossible to forge the MAC result without the knowledge of the key.

Chapter 3

The Hash Function Hamsi

"The ideal situation occurs when the things that we regard as beautiful are also regarded by other people as useful."

Donald Knuth

In this chapter we introduce the hash function Hamsi. In October 2008, the National Institute of Standards and Technology (NIST) has launched a competition to select a new hash function standard, called SHA-3 [102]. Hamsi which was designed during our doctoral research, has been submitted to this competition. Hamsi has been selected for the second round together with 13 other candidates out of 64 submissions. Five algorithms have been selected for the final round, namely BLAKE, Grøstl, JH, Keccak and Skein. NIST has published a report [122] explaining their decision for each candidate. As stated in this report Hamsi is not chosen because of ROM requirements and 2nd-preimage attacks [59, 51].

Note that the version of Hamsi in this thesis is different from the 2nd round proposal in the following; we have corrected a bug in the constants of P_f by replacing them with the new ones and corrected another bug in the IV values. None of those affect the existing analysis results.

At the end of this chapter we propose a wide-pipe variant of Hamsi, called Hamsi$^{\oplus}$.

3.1 Introduction

Hamsi is a family of cryptographic hash functions. There are two instances of Hamsi, Hamsi-256 and Hamsi-512. Table 3.1 summarizes the variants, corresponding parameters and security claims in bits. Hamsi-224 and Hamsi-384 are very similar to Hamsi-256 and Hamsi-512 respectively; they only differ in initial values, and a final truncation. Thus, here we will mainly mention Hamsi-256 and Hamsi-512. Unless explicitly mentioned, operations and data structures for Hamsi-256 and Hamsi-512 apply for their stripped down counterparts, Hamsi-224 and Hamsi-384. When we use the term Hamsi we mean all variants.

Table 3.1: Hamsi variants and security claims in bits

Variant	Digest length	Collision resistance	Preimage resistance	2nd-preimage resistance	Message size per iteration
Hamsi-256	256	128	256	256	32
Hamsi-512	512	256	512	512	64
Hamsi-224	224	112	224	224	32
Hamsi-384	384	192	384	384	64

At the core of Hamsi are the expansion function and round transformations. The round transformation operates on a state matrix of 4 rows. The number of columns is 4 for Hamsi-256 and 8 for Hamsi-512. Any entry in the matrix is a word of 32 bits. The representation of Hamsi state is shown in Figure 3.1 and in Figure 3.2.

Figure 3.1: Representation of the Hamsi-256 state

s_0	s_1	s_2	s_3
s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}
s_{12}	s_{13}	s_{14}	s_{15}

Figure 3.2: Representation of the Hamsi-512 state

s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
s_{16}	s_{17}	s_{18}	s_{19}	s_{20}	s_{21}	s_{22}	s_{23}
s_{24}	s_{25}	s_{26}	s_{27}	s_{28}	s_{29}	s_{30}	s_{31}

In every round, 4 operations change the matrix. The first is a constant XOR into the whole matrix. The second is a simple XOR of the round number into the least significant bits of the entry in the first row and second column of the state matrix of Hamsi. The third is a nonlinear substitution (Sbox), and the fourth is a diffusion operation on the matrix.

The substitution layer uses a simple Sbox to operate on groups of 4 bits taken from the same bit position in each 4 rows of the state matrix. The result is written back into the same positions. The diffusion layer operates on 4 words from different positions in the matrix, and the result is written back to those positions.

The round transformations build the permutations P and P_f which are used as a chaining transformation and as a final transformation. The permutations P and P_f consist of operations XOR, shift and rotation resulting from the linear transformation and operations AND, OR and NOT resulting from the Sboxes. The only place where finite field operations take place is the message expansion; these can be implemented over F_2 by using operations XOR and AND. We discuss the implementation aspects of the message expansion in Section 3.7.3. In Appendix A the generator matrices of the linear codes suitable for the bitslice implementation are also included.

In the following sections we give the specification of the Hamsi hash function suitable for bitsliced implementations.

3.2 Design Choices

This section summarizes the main design choices of Hamsi. In the following sections we will cover the design in detail.

The Mode of Operation

Hamsi is inspired by stream based hash algorithms (such as Grindahl [83]). The mode of operation of Hamsi can be visualized as a variant of the Merkle-Damgård, Sponge or Concatenate-Permute-Truncate design approach. Although many of those constructions can be merged into each other, there are specific properties that can be emphasized for each. Hamsi is based on several properties of those modes of operations. Below we list the main properties of the Hamsi mode of operation:

- Hamsi processes short message blocks in each iteration, namely (depending on the variant see Table 3.3) inputs 32-bit or 64-bit message blocks. In software, for short messages of length 64 bytes or 8 bytes, Hamsi is

faster than the designs that process larger message blocks in each iteration. See eBASH [52] (ECRYPT Benchmarking of All Submitted Hashes) for comparisons.

- Assuming that the underlying compression function is collision-resistant, the Merkle-Damgård iteration builds a collision-resistant hash function. Many hash functions are built on strong block ciphers. Moreover, it is a traditional design approach to try to secure the compression function from all possible "strange" properties and to make it "random". But most of the time it is not evident what to avoid and what not to. The Hamsi mode of operation tries to achieve randomness in the hash function but not in the compression function or permutations. Structural properties in the output of the compression function are tolerable as long as they don't yield weaknesses in the hash function. It is possible to exhibit non-random properties in one iteration, hence we believe that the security of the compression function should be considered in the context of the iteration mode.

- Although there are many attacks with high complexity to narrow-pipe designs such as Joux's multicollision attack [73] and herding attacks [74], for efficiency reasons we chose Hamsi to be a narrow-pipe design, namely the chaining value has the same size as the digest length. For those who are concerned about 2nd-preimage or preimage attacks with complexity over the birthday bound, we propose at the end of this chapter a wide-pipe variant called Hamsi$^{\oplus}$, which can be obtained from Hamsi with a small tweak.

Message Expansion

One of the crucial parts of the design of Hamsi is the message expansion. Hamsi has a strong linear message expansion based on Best Known Linear Codes (BKLC) over F_4. We have chosen the best possible code in terms of minimum distance to achieve the best results for the number of active Sboxes. The Concatenation map ensures that the minimum distance of the linear code gives the number of active Sboxes for the first round. Namely, Hamsi-256 has 70 active Sboxes and Hamsi-512 has 131 active Sboxes for the first round. The linear expansion is also flexible, i.e., it has several implementation possibilities and it does not depend on the chaining value.

The Compression Function

The choice of the 4-bit Sboxes and the application of the linear transformation make the Hamsi compression function suitable for bitslice implementations. The constants are chosen to achieve asymmetry in the Sboxes. The counter value

exhibits asymmetry between the rounds as a caution for slide attacks and fixed-point attacks.

The Finalization

In general, stream-based hash algorithms have a light compression function, e.g., based on a permutation of one round. The permutation of Grindahl has 1 round, the permutation P used in Hamsi-256 has 3 rounds, whereas the permutation P used in Hamsi-512 has 6 rounds. A consequence of this approach is that mixing of the message block into the state takes several iterations. The effect of this on the last message block is resolved by processing the last message block with a finalization which is a stronger permutation, typically similar or the same permutation used in the previous iterations but strengthened with more rounds. The permutation P_f used as finalization in Hamsi prevents slide attacks and length-extension attacks. Hence, it is obvious that the Hamsi hash function without the finalization would be insecure.

3.3 Preliminaries

In this section we will give some definitions that will be used in the following sections.

Definition 3.1. *A linear code C over a vector space A^n is a subspace of A^n.*

We will always deal with codes over finite fields and A will be F_2.

Definition 3.2. *Elements of a code are called codewords and the length of the code is n, where $C \subset A^n$.*

Definition 3.3. *The dimension of a linear code C is defined to be the dimension of C as a vector space over A.*

Definition 3.4. *A $k \times n$ matrix obtained by taking basis elements of C as the rows is called a generator matrix of the code C.*

Definition 3.5. *The Hamming distance between the vectors $x, y \in A^n$ is defined as the number of different components, formulated as follows:*

$$hd(x, y) = \#\{i | x_i \neq y_i\} \ .$$

The Hamming weight, ($hw(\cdot)$) of a vector is defined as its distance to $(0, \ldots, 0)$.

Definition 3.6. *The minimum distance of the code C is defined as the minimum of the Hamming distance between all codewords:*

$$d = min\{hd(x, y) | x, y \in C and\ x \neq y\} \ .$$

The length n, dimension k and the minimum distance d are called *parameters* of the code. Note that for a linear code the minimum distance is the same as the minimum weight, $d = min\{hw(x)|x \in C, x \neq 0\}$.

Definition 3.7 ([41])**.** *The branch number of a linear transformation ϕ defined over a vector space A^n is defined as follows:*

$$br(\phi) = min\{hw(a) + hw(\phi(a)|a \neq 0, a \in A^n\} \ .$$

Table 3.2: Notation

F_n	Finite Field with n elements, where n is a prime power
$<<<$	left rotation
$\oplus$	exclusive or, XOR
$<<$	left shift
$[n, m, d]$	code with length n, dimension m and minimum distance d
kB	Kilobytes

Table 3.3: Hamsi variants and parameters in bits

Variant	Digest length	Size of chaining value	State Size	Expanded message size	Message size per iteration
Hamsi-256	256	256	512	256	32
Hamsi-512	512	512	1024	512	64
Hamsi-224	224	256	512	256	32
Hamsi-384	384	512	1024	512	64

3.3.1 Endianness

All operations in Hamsi are on words (at least 32 bits) and therefore are independent of endianness. The only places that require endianness are when the message bytes are converted into bits and when the output words are converted into bytes. We explain the endiannes for Hamsi-256 and Hamsi-224, for all other variants same principles hold.

The 4-byte message `0x00`, `0x01`, `0x02`, `0x03` is converted to `00000000`, `00000001`, `00000010`, `00000011`, where the leftmost bit is 0 and the rightmost is 31. The message expansion is a linear map from 32 bits to eight 32-bit words. The expanded

message words, `0x00010203`, `0x04050607`, `0x08090A0B`, `0x0C0D0E0F`, `0x10111213`, `0x14151617`, `18191A1B`, `0x1C1D1E1F` are converted to 00000000 00000001 00000010 00000011,..., 00011010 00011101 00011110 00001111, where the leftmost bit is 0 and the rightmost bit is 255.

3.4 General Design

In this section we describe the general design, namely the iteration mode of Hamsi. In each iteration Hamsi uses a permutation, a message expansion and a feedforward of the chaining value. The non-linear permutation required for the design uses the linear transformation and one of the Sboxes of the block cipher Serpent [3].
The general design is shown in Figure 3.3, but more precisely, Hamsi can be described as the composition of the following mappings:

Message Expansion	$E : \{0,1\}^m \to \{0,1\}^r$
Concatenation	$C : \{0,1\}^r \times \{0,1\}^c \to \{0,1\}^{r+c}$
Non-linear Permutations	$P, P_f : \{0,1\}^{r+c} \to \{0,1\}^{r+c}$
Truncations	$T : \{0,1\}^{r+c} \to \{0,1\}^n$
	$T_{224} : \{0,1\}^{256} \to \{0,1\}^{224}$
	$T_{384} : \{0,1\}^{512} \to \{0,1\}^{384}$

$m = 32, r = 256, c = 256, n = 256$ for Hamsi-256 and Hamsi-224.
$m = 64, r = 512, c = 512, n = 512$ for Hamsi-512 and Hamsi-384.

Specifications of the mappings for different variants of Hamsi are given in the following sections. Let $(M_1||M_2||M_3||\ldots||M_l||)$ be a padded message, then Hamsi variants can be described as follows:
Hamsi-256:

$$h_i = (T \circ P \circ C(E(M_i), h_{i-1})) \oplus h_{i-1}, \quad h_0 = iv_{256}, \quad 0 < i < l \tag{3.1}$$

$$h = (T \circ P_f \circ C(E(M_l), h_{l-1})) \oplus h_{l-1} \ . \tag{3.2}$$

Hamsi-224:

$$h_i = (T \circ P \circ C(E(M_i), h_{i-1})) \oplus h_{i-1}, \quad h_0 = iv_{224}, \quad 0 < i < l \tag{3.3}$$

$$h = (T_{224} \circ P_f \circ C(E(M_l), h_{l-1})) \oplus h_{l-1} \ . \tag{3.4}$$

Hamsi-512:

$$h_i = (T \circ P \circ C(E(M_i), h_{i-1})) \oplus h_{i-1}, \quad h_0 = iv_{512}, \quad 0 < i < l \tag{3.5}$$

$$h = (T \circ P_f \circ C(E(M_l), h_{l-1})) \oplus h_{l-1} \ . \tag{3.6}$$

Hamsi-384:

$$h_i = (T \circ P \circ C(E(M_i), h_{i-1})) \oplus h_{i-1}, \quad h_0 = iv_{384}, \quad 0 < i < l \tag{3.7}$$

$$h = (T_{384} \circ P_f \circ C(E(M_l), h_{l-1})) \oplus h_{l-1} \ . \tag{3.8}$$

3.5 Initial Values

The initial values are used as the initial chaining value, h_0. Hamsi has 4 initial values: iv_{256}, iv_{224}, iv_{512}, iv_{384} used in Hamsi-256, Hamsi-224, Hamsi-512 and Hamsi-384 respectively. The initial values are obtained from the UTF-8 encoding of the text “Özgül Küçük, Katholieke Universiteit Leuven, Departement Elektrotechniek, Computer Security and Industrial Cryptography, Kasteelpark Arenberg 10, bus 2446, B-3001 Leuven-Heverlee, Belgium.”

The initial values are obtained in the following manner. The encoding of the address string is UTF-8. iv_{224} is the first 256 bits, iv_{256} is the second 256 bits, totalling 512. iv_{384} is the second 512 bits and iv_{512} is the third 512 bits. The *IV* values consist of 32 bit words, each read in a *Big endian* fashion. Thus, the first word of iv_{224} is 0xc3967a67, which is 0xc396 for "Ö" UTF-8 encoded, 0x7a for "z", and 0x67 for "g", giving us the beginning 4 bytes of the address string "Özg".

3.6 Message Padding

Hamsi operates on 32-bit and 64-bit message blocks in Hamsi-256 and Hamsi-224 and Hamsi-512 and Hamsi-384, respectively. If the message blocks are not a multiple of 32 or 64 we apply the following procedure: append a ‘1’-bit to the message and number of ‘0’-bits filling the last message block. Append the message length as 64-bit unsigned integer as the last message block. As required by NIST Hamsi has maximum message length $2^{64} - 1$.

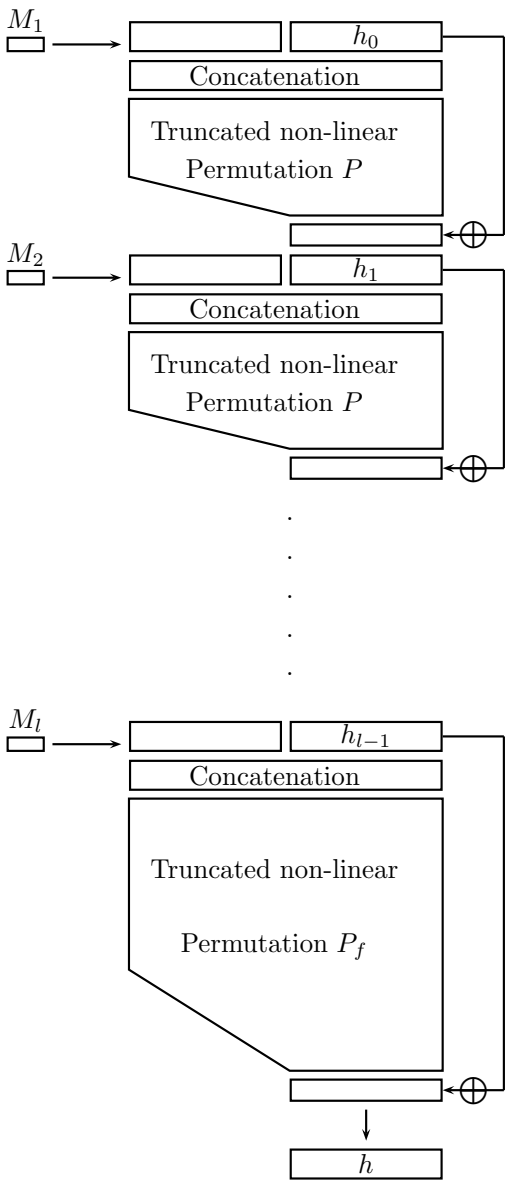


Figure 3.3: The general design of Hamsi

Table 3.4: Initial Values of Hamsi

iv_{224}	0xc3967a67, 0xc3bc6c20, 0x4bc3bcc3, 0xa7c3bc6b 0x2c204b61, 0x74686f6c, 0x69656b65, 0x20556e69
iv_{256}	0x76657273, 0x69746569, 0x74204c65, 0x7576656e 0x2c204465, 0x70617274, 0x656d656e, 0x7420456c
iv_{384}	0x656b7472, 0x6f746563, 0x686e6965, 0x6b2c2043 0x6f6d7075, 0x74657220, 0x53656375, 0x72697479 0x20616e64, 0x20496e64, 0x75737472, 0x69616c20 0x43727970, 0x746f6772, 0x61706879, 0x2c204b61
iv_{512}	0x73746565, 0x6c706172, 0x6b204172, 0x656e6265 0x72672031, 0x302c2062, 0x75732032, 0x3434362c 0x20422d33, 0x30303120, 0x4c657576, 0x656e2d48 0x65766572, 0x6c65652c, 0x2042656c, 0x6769756d

3.7 Message Expansion

In each iteration, the message blocks are expanded before being processed with the chaining value. Hamsi uses linear codes [124] for this purpose. We had several criteria when we designed the message expansion:

- Choose codes with high minimum distance. This is because the minimum distance of the code affects the number of active Sboxes for the first round.
- We choose codes over F_4. In order to have the effect of the minimum distance as strong as possible we could choose codes either over F_4 or F_2 (because we concatenate 2 bits from the expanded message words with 2 bits from the chaining value). The best codes we could choose over F_2 didn't achieve better than what we chose from F_4. Namely, we implemented the quasi-cyclic code [256,32,96] over F_2 and counted that it activates 67 Sboxes whereas the code [128,16,70] over F_4 activates 70 Sboxes.
- Codes should be simple and efficient to implement. In Section 3.7.3 we will discuss several implementation possibilities.
- The message expansion is independent of the chaining variable. This allows to expand the message words in advance. Namely, while the processor is busy with the current iteration, the message expansion of the next iteration can be executed.

3.7.1 Hamsi-256/Hamsi-224

The message expansion of Hamsi-224 and Hamsi-256 expands 32-bits to 256-bits with the code [128,16,70] over F_4. This is defined by the map $E : \{0,1\}^{32} \rightarrow \{0,1\}^{256}$, here and below G is the generator matrix of the code:

$$\begin{aligned} E(M_i) &= (M_i \times G),\ M_i \in F_4^{16} \\ &= (m_0, m_1, \ldots, m_7),\ m_i \in F_2^{32}\ . \end{aligned}$$

The linear code [128,16,70] can be constructed in several ways; we chose the one that has a better weight distribution of the codewords. It is obtained by the truncation of two coordinates from each codeword of the best known linear code [130,16,72] over F_4; this can be achieved by truncating the last two columns from the generator matrix. The linear code [128,16,70] can be constructed with the tool Magma [30] as follows:

$$F < w >: \ = GF(4); \tag{3.9}$$

$$B: \ = BestKnownLinearCode(GF(4), 130, 16); \tag{3.10}$$

$$E: \ = PunctureCode(C, \{129..130\}); \tag{3.11}$$

Note that 83 is an upper bound for the minimum distance of the code [128,16,70] over F_4 [62]. In order to fix the code we give the detailed construction and the generator matrix suitable for bitsliced implementation in Appendix A.

3.7.2 Hamsi-512/Hamsi-384

The message expansion of Hamsi-384 and Hamsi-512 expands 64 bits to 512 bits with the code [256,32,131] over F_4 [40]. $E : \{0,1\}^{64} \rightarrow \{0,1\}^{512}$, defining the expansion is applied as follows:

$$\begin{aligned} E(M_i) &= (M_i \times G),\ M_i \in F_4^{32} \\ &= (m_0, m_1, \ldots, m_{15}),\ m_i \in F_2^{32}\ . \end{aligned}$$

Again we use the best known linear code for the message expansion of Hamsi-512 (Hamsi-384 respectively). An upper bound for the minimum distance is 168 [40]. The linear code [256,32,131] over F_4 can be generated with Magma [30]. The detailed construction is given in the Appendix A.

3.7.3 Implementation

The message expansion can be implemented in several ways suitable for software or for hardware. The expanded message words can be produced by multiplying the message blocks with a suitable generator matrix of the linear code. For Hamsi-256 and Hamsi-512 this is a matrix of size 1kB and 4kB respectively. On the other hand this multiplication can be represented by a lookup table of size 32kB for Hamsi-256 and 128kB for Hamsi-512, aiming for fast software. It may also be possible to exploit the structure of the code, that will reduce the size of the matrices; we leave this as an open problem. Below we explain these methods in detail.

Lookup tables

The Hamsi expansion can be performed in a method suitable for any arbitrary linear transformation: for every *b*yte of the input, depending on the position of the byte, a table is generated offline. This table contains the result of the multiplication of the byte (where as other values of the message block is taken to be zero) with the generator matrix. For every value of the byte and for each position a table is generated. During runtime, for every byte position, the value is looked up in the table corresponding to that position, and all the output contributions obtained are XORed to get the final output. The ordering of the bits in the array should be suitable with the ordering of message bits and the chaining value. We perform the expansion in the above mentioned method, but the effect of it is taken to be the same with the method explained below.

We take as input, 16 (or 32 for Hamsi-512) values from F_4. These values, as a vector are multiplied by the generator matrix (which is 16×128 or 32×256). The resulting value is termed M, a vector of 128 (or 256) values from F_4. This is the contribution of the input bytes to the iteration function. These bytes, together with the chaining values (of the same length) are used to initialize the internal state of the iteration function. The M vector is used to obtain the vector m (which is of the same size) by a simple bit permutation. The m vector consists of q words of 32 bit (4 words for Hamsi-256, 8 for Hamsi-512). As can be seen in Figure 3.4, for each $i < q$, every bit of m_i is teamed with the bit in the same position in m_{q+i} to enter the same Sbox. These pairs of bits come from the F_4 values in M. To this effect, M_i and M_{q+i} are used to obtain m_i and m_{q+i}, where all even positioned bits (e.g. for $i < 16$, in bit positions $2 * i$ of M_i and M_{q+i}) are placed in m_i, and all odd positioned bits are placed in m_{q+i}. If we denote the j-th bit of a vector a with $a[j]$, then for $0 \leq j < 16$, $m_i[j] = M_i[2 * j]$, $m_i[16 + j] = M_{q+i}[2 * j]$, $m_{q+i}[j] = M_i[2 * j + 1]$, and $m_{q+i}[16 + j] = M_{q+i}[2 * j + 1]$.

Multiplication with the Generator Matrix

Given a message block the corresponding expanded message can be computed by multiplying the message block with the generator matrix over F_4. Instead of multiplying over F_4 we can simplify the generation by multiplying over F_2: the corresponding matrices over F_2 are in Appendix A. The multiplication with the given matrices outputs expanded message words suitable for bitsliced implementations.

3.8 Concatenation

The expanded message words $(m_0, m_1, \ldots, m_i)$ are concatenated to the chaining value $(c_0, c_1, \ldots, c_i)$, $(i = \{7, 15\})$ with a certain order. The result is afterwards input to the nonlinear permutation P. The concatenation map C determines the ordering of bits (and words) input to P. With this ordering 2 bits from the chaining value and 2 bits from the expanded message block are input to the 4-bit Sboxes.

In Hamsi-256 and Hamsi-224, $C : \{0,1\}^{256} \times \{0,1\}^{256} \to \{0,1\}^{512}$ is:

$$C(m_0, m_1, \ldots, m_7, c_0, c_1, \ldots, c_7) =$$

$$(m_0, m_1, c_0, c_1, c_2, c_3, m_2, m_3, m_4, m_5, c_4, c_5, c_6, c_7, m_6, m_7),\ m_i, c_i \in F_2^{32} \ .$$

In Hamsi-512 and Hamsi-384, $C : \{0,1\}^{512} \times \{0,1\}^{512} \to \{0,1\}^{1024}$ is:

$$C(m_0, m_1, \ldots, m_{14}, m_{15}, c_0, c_1, \ldots, c_{14}, c_{15}) =$$

$$(m_0, m_1, c_0, c_1, m_2, m_3, c_2, c_3, c_4, c_5, m_4, m_5, c_6, c_7, m_6, m_7, m_8,$$

$$m_9, c_8, c_9, m_{10}, m_{11}, c_{10}, c_{11}, c_{12}, c_{13}, m_{12}, m_{13}, c_{14}, c_{15}, m_{14}, m_{15}), m_i, c_i \in F_2^{32} \ .$$

$$(m_0, m_1, \ldots, m_7, c_0, c_1, \ldots, c_7) \xrightarrow{C} \begin{array}{|c|c|c|c|} \hline m_0 & m_1 & c_0 & c_1 \\ \hline c_2 & c_3 & m_2 & m_3 \\ \hline m_4 & m_5 & c_4 & c_5 \\ \hline c_6 & c_7 & m_6 & m_7 \\ \hline \end{array}$$

Figure 3.4: Concatenation in Hamsi-256/Hamsi-224

$$(m_0, m_1, \ldots, m_{15}, c_0, c_1, \ldots, c_{15}) \xrightarrow{C}$$

m_0	m_1	c_0	c_1	m_2	m_3	c_2	c_3
c_4	c_5	m_4	m_5	c_6	c_7	m_6	m_7
m_8	m_9	c_8	c_9	c_{10}	c_{11}	m_{10}	m_{11}
c_{12}	c_{13}	m_{12}	m_{13}	m_{14}	m_{15}	c_{14}	c_{15}

Figure 3.5: Concatenation in Hamsi-512/Hamsi-384

3.9 The Non-Linear Permutation P

The non-linear permutation consists of 3 layers; input bits are first XORed with the constants and a counter; this is followed by the application of 4-bit Sboxes and several applications of the linear transformation L; this is repeated for each round. We represent a state of the permutation with $(s_0, s_1, s_2, \ldots, s_j)$, $j = 15, 31$ and $s_i \in F_2^{32}, \quad i = 0, 1, \ldots, j$. This can be visualized with a 4×4 and 4×8 matrix in Hamsi-256/Hamsi-224 (Figure 3.1) and Hamsi-512/Hamsi-384 (Figure 3.2), respectively.

3.9.1 Addition of Constants and Counter

The constants $\alpha_i \in F_2^{32}, i = 0, 1, 2, \ldots, 31$ are XORed with the input state before the substitution layer together with the counter. We use the round number as the counter c; for the first round $c = 0$ and second round $c = 1$, etc. We use constants to ensure asymmetry in the same round within the Sboxes and the counter in between the rounds. The constants are permutations of the sequence $0, 1, 2, \ldots, 15$ (each 4 bits), see Table 3.5. In Table 3.6 the representation of the constants corresponding to the bitsliced implementation is given, in all implementations this table will be used.

The Sboxes in Hamsi acts over the columns of a Hamsi state (see Figure 3.6) consequently the bits of the constants in Table 3.5 are arranged in Table 3.6 such that when α_i are XORed with the state words of Hamsi, the effect is the same as XORing the elements of p_i column-wise with the inputs of the Sboxes, e.g., α_0 contains the most significant bits of the elements of the permutations p_0 and p_1.

In Hamsi-256 and Hamsi-224:

$$(s_0, s_1, \ldots, s_{15}) :=$$

$$(s_0 \oplus \alpha_0, s_1 \oplus \alpha_1 \oplus c, s_2 \oplus \alpha_2, s_3 \oplus \alpha_3, s_4 \oplus \alpha_8, s_5 \oplus \alpha_9, s_6 \oplus \alpha_{10}, s_7 \oplus \alpha_{11},$$

$$s_8 \oplus \alpha_{16}, s_9 \oplus \alpha_{17}, s_{10} \oplus \alpha_{18}, s_{11} \oplus \alpha_{19}, s_{12} \oplus \alpha_{24}, s_{13} \oplus \alpha_{25}, s_{14} \oplus \alpha_{26}, s_{15} \oplus \alpha_{27}) \ .$$

Table 3.5: Constants of P

$p_0 = \{15, 7, 11, 3, 13, 5, 9, 1, 14, 6, 10, 2, 12, 4, 8, 0\}$
$p_1 = \{15, 11, 13, 9, 14, 10, 12, 8, 7, 3, 5, 1, 6, 2, 4, 0\}$
$p_2 = \{15, 13, 14, 12, 7, 5, 6, 4, 11, 9, 10, 8, 3, 1, 2, 0\}$
$p_3 = \{15, 14, 7, 6, 11, 10, 3, 2, 13, 12, 5, 4, 9, 8, 1, 0\}$
$p_4 = \{15, 7, 13, 5, 14, 6, 12, 4, 11, 3, 9, 1, 10, 2, 8, 0\}$
$p_5 = \{15, 11, 14, 10, 7, 3, 6, 2, 13, 9, 12, 8, 5, 1, 4, 0\}$
$p_6 = \{15, 13, 7, 5, 11, 9, 3, 1, 14, 12, 6, 4, 10, 8, 2, 0\}$
$p_7 = \{15, 14, 11, 10, 13, 12, 9, 8, 7, 6, 3, 2, 5, 4, 1, 0\}$
$p_8 = \{15, 13, 14, 12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2, 0\}$
$p_9 = \{15, 14, 7, 6, 13, 12, 5, 4, 11, 10, 3, 2, 9, 8, 1, 0\}$
$p_{10} = \{15, 7, 11, 3, 14, 6, 10, 2, 13, 5, 9, 1, 12, 4, 8, 0\}$
$p_{11} = \{15, 11, 13, 9, 7, 3, 5, 1, 14, 10, 12, 8, 6, 2, 4, 0\}$
$p_{12} = \{15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0\}$
$p_{13} = \{15, 7, 14, 6, 13, 5, 12, 4, 11, 3, 10, 2, 9, 1, 8, 0\}$
$p_{14} = \{15, 11, 7, 3, 14, 10, 6, 2, 13, 9, 5, 1, 12, 8, 4, 0\}$
$p_{15} = \{15, 13, 11, 9, 7, 5, 3, 1, 14, 12, 10, 8, 6, 4, 2, 0\}$

Table 3.6: Constants of P suitable for bitsliced implementation

$\alpha_0 =$ 0xff00f0f0	$\alpha_1 =$ 0xccccaaaa	$\alpha_2 =$ 0xf0f0cccc	$\alpha_3 =$ 0xff00aaaa
$\alpha_4 =$ 0xccccaaaa	$\alpha_5 =$ 0xf0f0ff00	$\alpha_6 =$ 0xaaaacccc	$\alpha_7 =$ 0xf0f0ff00
$\alpha_8 =$ 0xf0f0cccc	$\alpha_9 =$ 0xaaaaff00	$\alpha_{10} =$ 0xccccff00	$\alpha_{11} =$ 0xaaaaf0f0
$\alpha_{12} =$ 0xaaaaf0f0	$\alpha_{13} =$ 0xff00cccc	$\alpha_{14} =$ 0xccccf0f0	$\alpha_{15} =$ 0xff00aaaa
$\alpha_{16} =$ 0xccccaaaa	$\alpha_{17} =$ 0xff00f0f0	$\alpha_{18} =$ 0xff00aaaa	$\alpha_{19} =$ 0xf0f0cccc
$\alpha_{20} =$ 0xf0f0ff00	$\alpha_{21} =$ 0xccccaaaa	$\alpha_{22} =$ 0xf0f0ff00	$\alpha_{23} =$ 0xaaaacccc
$\alpha_{24} =$ 0xaaaaff00	$\alpha_{25} =$ 0xf0f0cccc	$\alpha_{26} =$ 0xaaaaf0f0	$\alpha_{27} =$ 0xccccff00
$\alpha_{28} =$ 0xff00cccc	$\alpha_{29} =$ 0xaaaaf0f0	$\alpha_{30} =$ 0xff00aaaa	$\alpha_{31} =$ 0xccccf0f0

In Hamsi-512 and Hamsi-384:

$$(s_0, s_1, \ldots, s_{31}) := (s_0 \oplus \alpha_0, s_1 \oplus \alpha_1 \oplus c, s_2 \oplus \alpha_2, \ldots, s_{31} \oplus \alpha_{31}) \ .$$

Note that the constants are XORed in a different order in Hamsi-256 (Hamsi-224) and Hamsi-512 (Hamsi-384).

Table 3.7: Truth table of the Hamsi Sbox

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S[x]	8	6	7	9	3	C	A	F	D	1	E	4	0	B	5	2

3.9.2 Substitution Layer

Hamsi uses a 4×4-bit Sbox $S : F_2^4 \rightarrow F_2^4$. We borrowed the Sbox from the block cipher Serpent [3] because of its good differential and linear properties and its efficient implementation. Hence, the Hamsi Sbox satisfies the following properties:

- The highest differential probability of the difference distribution table is $1/4$.
- Each linear characteristic has a probability in the range $1/2 \mp 1/4$ and a linear relation between one single bit in the input and one single bit in the output has a probability in the range $1/2 \mp 1/8$.
- A one-bit input difference always leads to at least two-bit output differences.
- The algebraic-degree of three out of four output bits is 3. The algebraic normal form (ANF) of the Sbox is:

$$
\begin{aligned}
y_0 &= x_1 \oplus x_2 \oplus x_3 \oplus x_0x_2 \\
y_1 &= x_0 \oplus x_1 \oplus x_2 \oplus x_1x_2 \oplus x_0x_3 \oplus x_2x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \\
y_2 &= x_0 \oplus x_1 \oplus x_3 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \\
y_3 &= 1 \oplus x_0 \oplus x_1 \oplus x_2 \oplus x_1x_3 \oplus x_0x_1x_2
\end{aligned}
\tag{3.12}
$$

Hamsi is conveniently designed for bitsliced implementation. There are 128 (or 256 for Hamsi-512) parallel and identical Sboxes; all can be executed at the same time in computer words of up to 128 bits (or 256 for Hamsi-512). Hence, if one has registers of size 128 bits, one can make use of it to make Hamsi faster. But registers of 32 bits are sufficient for the basic implementation. The Hamsi Sbox can be described by using 16 operations and 5 registers of size up to 128-bits. In Table 3.8 we give equations suitable for bitslice implementations [106]. The bits of the Sbox are positioned to the Hamsi state matrix as follows; the first row is considered the least significant bit (LSB), and similarly the fourth, the most significant bit (MSB), see Figure 3.6.

Table 3.8: Equations of the Hamsi Sbox suitable for bitslice implementation [106].

1. $r = x_0$	2. $x_0 = x_0 \& x_2$
3. $x_0 = x_0 \oplus x_3$	4. $x_2 = x_2 \oplus x_1$
5. $x_2 = x_2 \oplus x_0$	6. $x_3 = x_3 \vert r$
7. $x_3 = x_3 \oplus x_1$	8. $r = r \oplus x_2$
9. $x_1 = x_3$	10. $x_3 = x_3 \vert r$
11. $x_3 = x_3 \oplus x_0$	12. $x_0 = x_0 \& x_1$
13. $r = r \oplus x_0$	14. $x_1 = x_1 \oplus x_3$
15. $x_1 = x_1 \oplus r$	16. $r = Not(r)$
$x_0 = x_2$, $x_2 = x_1$	$x_1 = x_3$, $x_3 = r$

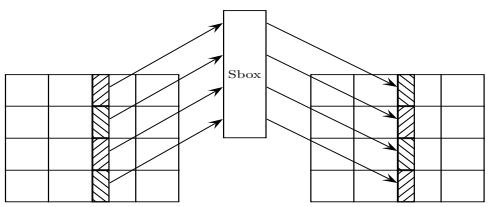


Figure 3.6: Hamsi Sbox acting on the columns in bitslice implementation

3.9.3 Diffusion Layer

The diffusion layer of Hamsi is based on several applications of the linear transformation $L : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ [3]. L operates on 32-bit words; inputs and outputs are 4 32-bit words.

Diffusion in Hamsi-256 and Hamsi-224

L acts over the diagonals of the state matrix as shown in Figure 3.7.

Table 3.9: Application of L in Hamsi-256/Hamsi-224

1.$(s_0, s_5, s_{10}, s_{15}) := L(s_0, s_5, s_{10}, s_{15})$
2.$(s_1, s_6, s_{11}, s_{12}) := L(s_1, s_6, s_{11}, s_{12})$
3.$(s_2, s_7, s_8, s_{13}) := L(s_2, s_7, s_8, s_{13})$
4.$(s_3, s_4, s_9, s_{14}) := L(s_3, s_4, s_9, s_{14})$

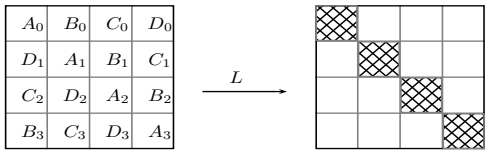


Figure 3.7: Application of L over the diagonals of the state matrix.

Diffusion in Hamsi-512 and Hamsi-384

Table 3.10: Application of L in Hamsi-512/Hamsi-384

1.$(s_0, s_9, s_{18}, s_{27}) := L(s_0, s_9, s_{18}, s_{27})$
2.$(s_1, s_{10}, s_{19}, s_{28}) := L(s_1, s_{10}, s_{19}, s_{28})$
3.$(s_2, s_{11}, s_{20}, s_{29}) := L(s_2, s_{11}, s_{20}, s_{29})$
4.$(s_3, s_{12}, s_{21}, s_{30}) := L(s_3, s_{12}, s_{21}, s_{30})$
5.$(s_4, s_{13}, s_{22}, s_{31}) := L(s_4, s_{13}, s_{22}, s_{31})$
6.$(s_5, s_{14}, s_{23}, s_{24}) := L(s_5, s_{14}, s_{23}, s_{24})$
7.$(s_6, s_{15}, s_{16}, s_{25}) := L(s_6, s_{15}, s_{16}, s_{25})$
8.$(s_7, s_8, s_{17}, s_{26}) := L(s_7, s_8, s_{17}, s_{26})$
9.$(s_0, s_2, s_5, s_7) := L(s_0, s_2, s_5, s_7)$
10.$(s_{16}, s_{19}, s_{21}, s_{22}) := L(s_{16}, s_{19}, s_{21}, s_{22})$
11.$(s_9, s_{11}, s_{12}, s_{14}) := L(s_9, s_{11}, s_{12}, s_{14})$
12.$(s_{25}, s_{26}, s_{28}, s_{31}) := L(s_{25}, s_{26}, s_{28}, s_{31})$

Note that in Hamsi-512 (and Hamsi-384) L is applied 12 times (3 times more than in Hamsi-256). L diffuses over 128-bits and the Hamsi-512 state has 256 bits in each row of the state matrix hence L is applied 3 times more in order to achieve diffusion in between the two 128-bit words. $L(s_9, s_{11}, s_{12}, s_{14})$ and $L(s_{25}, s_{26}, s_{28}, s_{31})$ need not be applied in the last round because they are already truncated, see Section 3.10.

Description of L

Let $a, b, c, d \in F_2^{32}$, below we give the construction of the linear transformation $L(a, b, c, d)$. L consists of XOR, shift and rotation operations:

$$a := a \lll 13$$

$$c := c \lll 3$$

$$b := b \oplus a \oplus c$$

$$d := d \oplus c \oplus (a \ll 3)$$

$$b := b \lll 1$$

$$d := d \lll 7$$

$$a := a \oplus b \oplus d$$

$$c := c \oplus d \oplus (b \ll 7)$$

$$a := a \lll 5$$

$$c := c \lll 22$$

The branch number of the linear transformation is 3. Recall that the Hamsi state is represented by a matrix as shown in Figure 3.1 and in Figure 3.2. Each entry of this matrix is 32 bits. The Sboxes are applied over the columns of this matrix and the linear transformation L is applied over the words in the diagonal, namely over the rows. The way that the Sboxes and the linear transformation are applied to the Hamsi state has the following consequence: any input bit difference to L results in 2 output bit differences because of the branch number and this will be input bit difference to 2 Sboxes. Moreover, we can state the following generalization:

Property 1. *If L has n output bit differences, this will affect n Sboxes of the next round, for $0 < n \leq 128$.*

3.10 Truncation T

In Hamsi-256 and Hamsi-224, $T : \{0,1\}^{512} \rightarrow \{0,1\}^{256}$ is defined as follows:

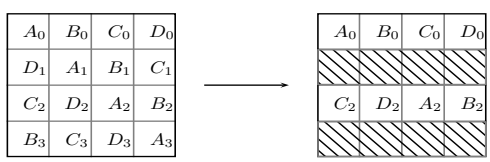
$$T(s_0, s_1, s_2, \ldots, s_{14}, s_{15}) = (s_0, s_1, s_2, s_3, s_8, s_9, s_{10}, s_{11}) \;\; \forall s_i \in F_2^{32} \;\; .$$

In Hamsi-512 and Hamsi-384, $T : \{0,1\}^{1024} \rightarrow \{0,1\}^{512}$ is defined as follows:

$$T(s_0, s_1, s_2, \ldots, s_{30}, s_{31}) = (s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_{16}, s_{17}, s_{18}, s_{19},$$
$$s_{20}, s_{21}, s_{22}, s_{23}) \ \forall s_i \in F_2^{32} \ .$$

The truncation map T is applied after the last round of the nonlinear permutation P. Figure 3.8 shows the state after the application of the linear transformation L and truncation T. Similar letters correspond to the words input to L, e.g. $L(A_0, A_1, A_2, A_3)$, etc.

Figure 3.8: Truncation in Hamsi-256



3.11 The Non-Linear Permutation P_f

P and P_f differ only in the number of rounds and constants. The constants of P_f are given in Table 3.11. The representation of constants suitable for bitslice implementation is given in Table 3.12. P_f is applied to the last message block as the final transformation.

3.12 Truncations T_{224}, T_{384}

T_{224} and T_{384} define the truncation method applied to Hamsi-256 and Hamsi-512 to obtain the digest sizes of 224 and 384 bits.

$T_{224} : \{0,1\}^{256} \rightarrow \{0,1\}^{224}$ is defined as follows:

$$T_{224}(s_0, s_1, \ldots, s_7) = (s_0, s_1, s_2, s_3, s_4, s_5, s_6) \ .$$

$T_{384} : \{0,1\}^{512} \rightarrow \{0,1\}^{384}$ is:

$$T_{384}(s_0, s_1, \ldots, s_{15}) = (s_0, s_1, s_3, s_4, s_5, s_6, s_8, s_9, s_{10}, s_{12}, s_{13}, s_{15}) \ .$$

Table 3.11: Constants of P_f

$p_0 = \{9, 13, 5, 1, 11, 2, 14, 7, 15, 3, 10, 6, 12, 4, 0, 8\}$
$p_1 = \{3, 11, 10, 2, 7, 4, 13, 14, 15, 6, 5, 12, 9, 8, 0, 1\}$
$p_2 = \{6, 7, 5, 4, 14, 8, 11, 13, 15, 12, 10, 9, 3, 1, 0, 2\}$
$p_3 = \{12, 14, 10, 8, 13, 1, 7, 11, 15, 9, 5, 3, 6, 2, 0, 4\}$
$p_4 = \{3, 11, 9, 1, 7, 4, 14, 13, 15, 5, 6, 12, 10, 8, 0, 2\}$
$p_5 = \{6, 7, 3, 2, 14, 8, 13, 11, 15, 10, 12, 9, 5, 1, 0, 4\}$
$p_6 = \{12, 14, 6, 4, 13, 1, 11, 7, 15, 5, 9, 3, 10, 2, 0, 8\}$
$p_7 = \{9, 13, 12, 8, 11, 2, 7, 14, 15, 10, 3, 6, 5, 4, 0, 1\}$
$p_8 = \{5, 7, 6, 4, 13, 8, 11, 14, 15, 12, 9, 10, 3, 2, 0, 1\}$
$p_9 = \{10, 14, 12, 8, 11, 1, 7, 13, 15, 9, 3, 5, 6, 4, 0, 2\}$
$p_{10} = \{5, 13, 9, 1, 7, 2, 14, 11, 15, 3, 6, 10, 12, 8, 0, 4\}$
$p_{11} = \{10, 11, 3, 2, 14, 4, 13, 7, 15, 6, 12, 5, 9, 1, 0, 8\}$
$p_{12} = \{9, 11, 10, 8, 13, 4, 7, 14, 15, 12, 5, 6, 3, 2, 0, 1\}$
$p_{13} = \{3, 7, 5, 1, 11, 8, 14, 13, 15, 9, 10, 12, 6, 4, 0, 2\}$
$p_{14} = \{6, 14, 10, 2, 7, 1, 13, 11, 15, 3, 5, 9, 12, 8, 0, 4\}$
$p_{15} = \{12, 13, 5, 4, 14, 2, 11, 7, 15, 6, 10, 3, 9, 1, 0, 8\}$

Table 3.12: Constants of P_f suitable for bitslice implementation

$\alpha_0 =$ 0xcaa9639c	$\alpha_1 =$ 0x0ff0f9c0	$\alpha_2 =$ 0x639c0ff0	$\alpha_3 =$ 0xcaa9f9c0
$\alpha_4 =$ 0x0ff0f9c0	$\alpha_5 =$ 0639ccaa9	$\alpha_6 =$ 0xf9c00ff0	$\alpha_7 =$ 0x639ccaa9
$\alpha_8 =$ 0x639c0ff0	$\alpha_9 =$ 0xf9c0caa9	$\alpha_{10} =$ 0x0ff0caa9	$\alpha_{11} =$ 0xf9c0639c
$\alpha_{12} =$ 0xf9c0639c	$\alpha_{13} =$ 0xcaa90ff0	$\alpha_{14} =$ 0x0ff0639c	$\alpha_{15} =$ 0xcaa9f9c0
$\alpha_{16} =$ 0x0ff0f9c0	$\alpha_{17} =$ 0xcaa9639c	$\alpha_{18} =$ 0xcaa9f9c0	$\alpha_{19} =$ 0x639c0ff0
$\alpha_{20} =$ 0x639ccaa9	$\alpha_{21} =$ 0x0ff0f9c0	$\alpha_{22} =$ 0x639ccaa9	$\alpha_{23} =$ 0xf9c00ff0
$\alpha_{24} =$ 0xf9c0caa9	$\alpha_{25} =$ 0x639c0ff0	$\alpha_{26} =$ 0xf9c0639c	$\alpha_{27} =$ 0x0ff0caa9
$\alpha_{28} =$ 0xcaa90ff0	$\alpha_{29} =$ 0xf9c0639c	$\alpha_{30} =$ 0xcaa9f9c0	$\alpha_{31} =$ 0x0ff0639c

3.13 Number of Rounds

The number of rounds recommended for different variants of Hamsi is given in Table 3.13. We would like to emphasize that number of rounds is the tunable parameter of Hamsi; it can be decreased to obtain weaker versions and increased in order to achieve better security as long as the performance values are not changed drastically.

The variants Hamsi-256/8 and Hamsi-224/8 differs only in the number of rounds of finalization P_f from Hamsi-256 and Hamsi-224, respectively.

Table 3.13: Number of rounds of permutations P and P_f for Hamsi

Variant	Hamsi-256 Hamsi-224	Hamsi-512 Hamsi-384	Hamsi-256/8 Hamsi-224/8
Rounds of P	3	6	3
Rounds of P_f	6	12	8

3.14 Hamsi$^{\oplus}$

In this section we define a variant of Hamsi, denoted with Hamsi$^{\oplus}$ and pronounced as Hamsi-XOR. The only difference between Hamsi and Hamsi$^{\oplus}$ is in the way the expanded message words are incorporated into each iteration. Instead of overwriting, the expanded message words are XORed with the chaining value. This change results in a wide-pipe design where the size of the chaining values is larger than the digest length. This increases the margin of the generic attacks that target inner collisions, i.e., collisions before the finalization.

Hamsi$^{\oplus}$ is defined only for the digest sizes 256 and 224. In the specification of Hamsi$^{\oplus}$ we use the same maps and representations as of Hamsi. We give the definition of Hamsi$^{\oplus}$ in Algorithm 4.

In Figure 3.10, we define the XOR of the expanded message words with the chaining value and the XOR of a part of the previous chaining value with the output of the permutation. We represent the state with a 4 by 4 matrix with each entry 32-bit words.

Table 3.14: Hamsi$^{\oplus}$ variants and parameters in bits

Variant	Digest length	Size of chaining value	State Size	Expanded message size	Message size per iteration
Hamsi$^{\oplus}$-256	256	512	512	256	32
Hamsi$^{\oplus}$-224	224	512	512	256	32

Table 3.15: Number of rounds of permutations P and P_f for Hamsi$^{\oplus}$

Variant	Hamsi$^{\oplus}$-256 Hamsi$^{\oplus}$-224	Hamsi$^{\oplus}$-256/8 Hamsi$^{\oplus}$-224/8
Rounds of P	3	3
Rounds of P_f	6	8

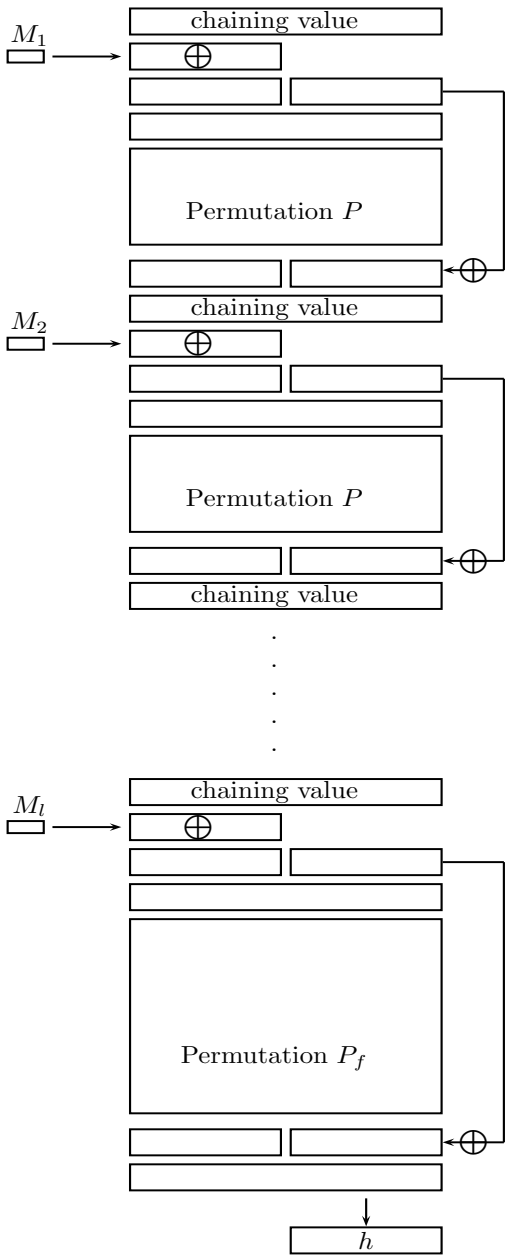


Figure 3.9: The general design of Hamsi$^{\oplus}$

XOR of expanded message words

c_0	c_1	c_2	c_3
c_4	c_5	c_6	c_7
c_8	c_9	c_{10}	c_{11}
c_{12}	c_{13}	c_{14}	c_{15}

$\longrightarrow$

$c_0 \oplus m_0$	$c_1 \oplus m_1$	c_2	c_3
c_4	c_5	$c_6 \oplus m_2$	$c_7 \oplus m_3$
$c_8 \oplus m_4$	$c_9 \oplus m_5$	c_{10}	c_{11}
c_{12}	c_{13}	$c_{14} \oplus m_6$	$c_{15} \oplus m_7$

Application of P

$c_0 \oplus m_0$	$c_1 \oplus m_1$	c_2	c_3
c_4	c_5	$c_6 \oplus m_2$	$c_7 \oplus m_3$
$c_8 \oplus m_4$	$c_9 \oplus m_5$	c_{10}	c_{11}
c_{12}	c_{13}	$c_{14} \oplus m_6$	$c_{15} \oplus m_7$

$\longrightarrow$

s_0	s_1	s_2	s_3
s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}
s_{12}	s_{13}	s_{14}	s_{15}

XOR of part of a chaining value

s_0	s_1	s_2	s_3
s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}
s_{12}	s_{13}	s_{14}	s_{15}

$\longrightarrow$

s_0	s_1	$s_2 \oplus c_2$	$s_3 \oplus c_3$
$s_4 \oplus c_4$	$s_5 \oplus c_5$	s_6	s_7
s_8	s_9	$s_{10} \oplus c_{10}$	$s_{11} \oplus c_{11}$
$s_{12} \oplus c_{12}$	$s_{13} \oplus c_{13}$	s_{14}	s_{15}

Figure 3.10: An iteration of Hamsi$^{\oplus}$ before the finalization.

Algorithm 4 Definition of Hamsi$^{\oplus}$

Let M be a padded message, $(M_1||M_2||M_3||\ldots||M_l||)$ $M_i \in F_2^m$.
$s_m \leftarrow 0^r, h_0 \leftarrow IV$
for $i = 0$ to $l-1$ **do**
  $(t_m, t_c) \leftarrow P \circ C(E(m_i) \oplus s_m, h_{i-1})$
  $(s_m, h_i) \leftarrow (t_m, t_c \oplus h_{i-1})$
end for
Finalization
$(t_m, t_c) \leftarrow P_f(E(m_l) \oplus s_m, h_{l-1})$
$(s_m, h_i) \leftarrow (t_m, t_c \oplus h_{l-1})$
Truncate to $n \in \{256, 224\}$.

3.14.1 Truncations T_{256}, T_{224}

Below, we define the truncation maps T_{256}, T_{224} used to obtain the digest sizes 256 and 224 at the end of the finalization P_f.

$T_{256} : \{0,1\}^{512} \rightarrow \{0,1\}^{256}$ is defined as follows:

$$T_{256}(s_0, s_1, \ldots, s_{15}) = (s_2, s_3, s_4, s_5, s_{10}, s_{11}, s_{12}, s_{13}) \ .$$

$T_{224} : \{0,1\}^{512} \rightarrow \{0,1\}^{224}$ is defined as follows:

$$T_{224}(s_0, s_1, \ldots, s_{15}) = (s_2, s_3, s_4, s_5, s_{10}, s_{11}, s_{12}) \ .$$

3.15 Implementation Aspects

In this section, we summarize implementation aspects and present some comparisons with other hash algorithms.

3.15.1 Software Implementation

Hamsi is designed to be suitable for bitsliced implementations, hence the fast software implementations of Hamsi take advantage of vector unit operations. In Table 3.16 we give the number and type of operations used by the building blocks of Hamsi-256. The software performance of Hamsi-256 for short messages is considerably better than for long messages. We give an example comparison in Figure 3.11, the measurements are taken from eBASH. In eBASH execution times for all sizes of messages from 0-bytes to 4096 bytes are given, however five specific sizes 8, 64, 576, 1536, 4096 are reported in the tables. For more detailed performance results on different platforms we refer to eBASH [52].

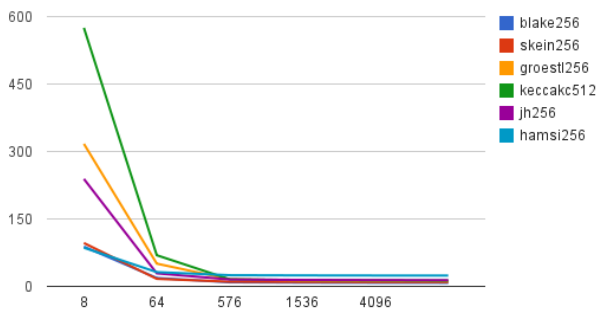


Figure 3.11: Cycles/byte versus message length. Platform: amd64, Sandy Bridge, 2011 Intel Core i7-2600K (eBASH).

Table 3.16: Number of 128-bit vector operations for Hamsi-256

Sbox		L		Message Expansion		Constants and Counter	
XOR	10	XOR	8	XOR	6	XOR	5
AND	2	rotation	6				
OR	2	shift	2				
MOVE	5	MOVE	2				

3.15.2 Hardware Implementation

Fair evaluation of the hardware aspects of a crypto algorithm is more complicated than software evaluation; there is a huge variety of platforms with different design targets and technologies, implementation and comparison methods. Also there is no standardized approach. The variety of usage scenarios of hash functions makes it difficult to set a common evaluation criteria. Hence, the evaluation methodology for hardware requires to fix the following aspects: environment, implementation method and performance comparison method.

ASIC Implementation

In this section, we present results from [63] in Table 3.1, obtained with the UMC 130nm standard cell technology. For more detailed comparison and exact results one should refer to the original paper. Here we include the Table 3.17 to give

a basic understanding and comparison of hardware performance of Hamsi-256 in ASIC. Below, we give the metrics used for comparison. The tradeoff points are chosen to reveal the performance of the designs while moving from MinArea to MaxSpeed:

- **MinArea:** This design approach targets to minimize the logic resources (gates) at the expense of performance.
- **MaxSpeed:** This design approach targets to minimize the computational delay of the design at the expense of area.
- **Tradeoff0:** A computational delay of two-thirds between the MinArea and MaxSpeed design points.
- **Tradeoff1:** A computational delay of five-sixths between the MinArea and MaxSpeed design points.

Table 3.17: Ranking of the 14 SHA-3 designs in terms of Throughput-to-Area ratio metric [63] in ASIC.

Rank	MinArea	Tradeoff0	Tradeoff1	MaxSpeed
1	Luffa	Luffa	Luffa	Luffa
2	Keccak	Keccak	Keccak	Keccak
3	**Hamsi**	**Hamsi**	**Hamsi**	CubeHash
4	Grøstl	CubeHash	CubeHash	SHA256
5	CubeHash	Grøstl	Grøstl	**Hamsi**
6	SHAvite-3	SHAvite-3	SHA256	Blake
7	SHA256	SHA256	SHavite-3	Grøstl
8	JH	JH	Blake	SHAvite-3
9	Blake	Blake	JH	JH
10	BMW	BMW	BMW	BMW
11	Shabal	Shabal	Shabal	Shabal
12	Skein	Skein	Skein	Skein
13	Echo	Echo	Echo	Echo
14	Fugue	Fugue	Fugue	Fugue
15	SIMD	SIMD	SIMD	SIMD

FPGA Implementation

In this section, we present a summary of the implementation results of SHA-3 2nd round candidates in 10 FPGA families from Xilinx and Altera published in [69], in Table 3.18. This gives a good source of comparison for the performance of

Table 3.18: Summary of performance of 256-bit and 512-bit variants of SHA-3 2nd round candidates in FPGAs [69]. The performance is measured by four metrics: throughput to area ratio (T/A), throughput (Thr), area (Area) and execution time for short messages (Short M). The symbols ✓, ≈, × refer to best, medium and worst respectively.

	T/A	Thr	Area	Short M	T/A	Thr	Area	Short M
BLAKE	≈	≈	≈	≈	≈	≈	≈	≈
BMW	×	✓	×	×	≈	✓	×	≈
CubeHash	✓	≈	✓	×	✓	≈	✓	×
ECHO	×	✓	×	✓	×	✓	×	✓
Fugue	≈	≈	✓	≈	≈	×	✓	×
Grøstl	≈	✓	≈	✓	≈	✓	≈	✓
Hamsi	≈	≈	✓	≈	≈	×	≈	≈
JH	≈	×	≈	×	×	≈	×	×
Keccak	✓	✓	≈	✓	✓	✓	✓	✓
Luffa	✓	✓	✓	✓	✓	✓	≈	✓
Shabal	✓	≈	✓	×	✓	×	✓	×
SHAvite-3	≈	≈	≈	≈	≈	≈	≈	≈
SIMD	×	≈	×	×	×	≈	×	×
Skein	≈	≈	≈	≈	≈	≈	≈	✓

Hamsi-256 and Hamsi-512 in FPGAs. These families include FPGAs optimized for minimum cost (Spartan 3 from Xilinx and Cyclone II, III and IV from Altera) and FPGAs optimized for high performance (Virtex 4, 5, and 6 from Xilinx and Stratix II, III and IV from Altera). All hash functions are implemented using VHDL language. We again refer to the original paper for detailed results.

The optimization target is Throughput to Area Ratio, where throughput is the number of input bits process per unit of time for long messages and Area is expressed in terms of the number of basic programmable logic blocks specific to a given FPGA family.

The 15 hash functions implemented are divided into 6 groups as follows [69]:

- Group 1: area and throughput are not affected by the change of the output size: CubeHash, JH, Shabal, Skein.
- Group 2: area and throughput both double: BMW, SIMD.
- Group 3: area and throughput both increase, but area increases more: BLAKE, Grøstl, SHAvite-3 and SHA-2.
- Group 4: area stays the same and throughput decreases: ECHO, Keccak.

- Group 5: area increases and throughput stays the same: Hamsi, Luffa.
- Group 6: area increases and throughput decreases: Fugue.

The groups 1 and 2 are best according to throughput to area ratio followed by groups 3, 4 and 5 and group 6, the last.

3.16 Conclusion

In this chapter we proposed the cryptographic hash functions Hamsi and Hamsi$^{\oplus}$. We have concentrated on the design aspects and specifications of Hamsi because Hamsi$^{\oplus}$ can be obtained from Hamsi with a small tweak. Hence, it is sufficient to describe the differences.

Many well-known hash functions are built on strong primitives following the design approach of Merkle-Damgård proof, i.e., a collision resistant compression function implies a collision resistant hash function. However the Merkle-Damgård proof does not state that the hash function is insecure if the compression function is not collision resistant. Hamsi (and Hamsi$^{\oplus}$) fulfill a cryptographers demand of a secure hash function based on light and consequently nonideal primitives.

Moreover, although Hamsi (and Hamsi$^{\oplus}$) is neither the first nor the last example of such hash function, it has some unique design features, namely, the expansion of small message blocks with strong message expansion, the way 4-bit Sboxes and the linear layer integrate in the permutation, etc.

As a second-round candidate in the SHA-3 competition, Hamsi had the opportunity to be analyzed and implemented by many researchers in the field. The results have led to interesting new insights and have strengthened our belief that our design approach could provide a useful alternative in some practical applications.

Chapter 4

Security Analysis of Hamsi

"Out of the most secure things, the most secure is to doubt."

Simón Bolívar

There are numerous attacks of different nature to cryptographic hash functions in literature, ranging from attacks on the mode of operation to attacks on the underlying primitives. In this chapter, we are interested in attacks aiming to exploit properties of the underlying primitives in order to attack the hash function itself. For example differential cryptanalysis, truncated differential cryptanalysis, high-order differential cryptanalysis, algebraic attacks, analysis of reduced versions.

The focus of this chapter is the security analysis of Hamsi with respect to collision resistance by applying differential cryptanalysis. Collision resistance is one of the main security requirements of hash functions and the complexity of a generic collision attack, i.e., the birthday attack, is much lower then the complexity of generic (second) preimage attacks.

While the SHA-3 competition was launched as a response to a series of attacks on widely deployed hash functions, it has also initiated the wide application of known techniques of differential cryptanalysis (attacks on SHA-3 candidates) as well as the invention of new attacks such as rotational cryptanalysis [76] and rebound attacks [96].

Hamsi is designed to be resistant to known attacks with a security margin for future advances. However, apart from designing with security in mind, the amount of resources that can be dedicated to the security analysis and the analysis efforts that arose during the SHA-3 competition is not comparable. Hence, in this chapter we also go over the invaluable external analysis results on Hamsi.

After a brief introduction in Section 4.1 to collision attacks on hash functions we discuss the resistance of Hamsi to differential attacks based on the number of active Sboxes and compute the probability of a 1-round differential characteristic for Hamsi-256 and Hamsi-512 in Section 4.2. We summarize external analysis results on Hamsi in Section 4.3. The main contribution of this chapter is the collision analysis of Hamsi-256 based on two different approaches of imposing conditions on Sboxes detailed in Section 4.4. The results included in this section have been presented in [47]. Finally, we briefly mention analysis of Hamsi$^{\oplus}$ in Section 4.5 and conclude the chapter.

4.1 Collision Attacks on Hash Functions

In a collision attack on a hash function an adversary tries to find two different messages that hash to the same value. The generic complexity of such an attack is equivalent to about $2^{n/2}$ hash evaluations and any attack with a complexity lower than this, is accepted as a *break* of the hash function (see Chapter 2.4). The break of a hash function can be either a theoretical or a practical one. Whether an attack is practical or theoretical depends on the available computing power. If it is computationally possible to construct examples of for instance colliding message pairs, or given a hash value constructing the preimage then it is a practical break of the hash function. Most of the attacks in literature are theoretical, namely, only the complexity of the attack is given.

Differential cryptanalysis is the most popular and the most successful technique used in the collision attacks on hash functions. The important milestones of collision attacks to hash functions date back to the first analyses of MD4 [50] by den Boer and Bosselaers [50] and by Dobbertin [53] and to the analysis of reduced RIPEMD [54] by Dobbertin and to the series of attacks by Chinese researchers in 2004 on dedicated hash functions [129]. The application of differential cryptanalysis to find collisions is based on the construction of differential characteristics. A survey of techniques used for finding differential characteristics can be found in the thesis of Rechberger [109].

In the SHA-3 competition, 17 of the first round candidates out of 51 have been *broken* with collision attacks, 11 of which with practical collisions, namely by showing examples of messages that collide. Although different applications of hash functions might require different levels of security, a practical collision attack nevertheless is a serious security flaw.

In the table below, we list round 1 SHA-3 candidates which are broken by collision attacks. We distinguish the hash functions for which examples of colliding messages are constructed under practical collisions.

Table 4.1: Collision attacks on round-1 SHA-3 competitors

Attacks	Hash Functions	References
Collision	Blender, Dynamic SHA2, ESSENCE LUX, NaSHA, TIB3, Waterfall	[78], [11], [103] [44], [134], [97] [58]
Practical collision	Boole, DCH, Dynamic SHA, EnRUPT Khichidi-1, Sgail, SHAMATA, Spectral Hash StreamHash, Tangle, WaMM	[95], [87], [70] [72], [100], [94] [71], [65], [120] [119], [131]

4.2 Resistance of Hamsi to Differential Attacks

Given an input difference to the Hamsi hash function, the Sboxes influenced by this difference are called *active*. The classical differential cryptanalysis of hash functions typically relies on the existence of colliding differential characteristics, i.e., differential characteristics with zero output differential, of sufficiently high probability. A protection against this from the point of view of the designer is to have as many active Sboxes as possible such that the probability of such a differential characteristic is low. Even though this probability can often be increased by the attacker by using the degrees of freedom in the message, we can still say as a rule of thumb that a high number of active Sboxes makes differential cryptanalysis harder. The number of active Sboxes is directly related with the diffusion properties of the algorithm.

If we examine the design of Hamsi from that perspective, we can make the following straightforward calculations.

4.2.1 Number of Active Sboxes

Recall that the Hamsi state is the concatenation of the expanded message words with the chaining value. Namely, the 512-bit state of Hamsi-256 contains 256 bits from the expanded message and 256 bits from the chaining value. The message is expanded via a linear code with minimum distance 70. Note that the linear code is defined over F_4 and any difference in the message will result in 70 differences over the expanded message over F_4. The 4-bit Sboxes take 2 bits from the expanded message and 2 bits from the chaining value. Hence any bit difference in the message will result in 70 differences over F_4, i.e., over each 2-bit input to the Sboxes from the expanded message. Hence the minimum distance of the code gives the number of active Sboxes for the first round assuming that at least a one bit difference is

applied to the message at any position. This holds for the entire Hamsi family. Now we can proceed to calculate the probability of a 1-round differential characteristic.

The Sboxes are the only components that contribute to the differential probability, and the differential behaviour of the Sbox is shown by *the difference distribution table*. The difference distribution table counts the number of input pairs with a certain input difference that gives a certain output difference. The first column and the first row corresponds to the possible input differences and the possible output differences, respectively.

Table 4.2: The difference distribution table of the Hamsi Sbox

Δ	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	2	0	2	0	0	2	2	2	0	4	2
2	0	0	0	4	0	4	0	0	0	4	0	0	0	0	0	4
3	0	4	2	0	0	0	2	0	0	2	0	0	2	0	2	2
4	0	0	0	0	0	0	4	0	0	0	4	4	0	4	0	0
5	0	4	0	2	2	2	2	0	2	0	0	0	2	0	2	2
6	0	0	2	2	2	2	0	0	2	2	0	0	0	0	2	2
7	0	0	0	0	4	2	0	2	0	0	2	2	2	0	0	2
8	0	0	0	2	0	2	0	4	0	2	0	0	0	4	0	2
9	0	0	0	2	0	0	0	2	4	2	2	2	2	0	0	0
A	0	0	2	0	2	0	4	0	2	0	4	0	0	0	2	0
B	0	4	0	0	2	0	2	0	2	2	0	0	2	0	0	2
C	0	0	2	0	2	0	0	0	2	0	0	4	0	4	2	0
D	0	4	2	2	0	2	2	0	0	0	0	0	2	0	2	0
E	0	0	2	0	2	0	0	4	2	0	0	0	0	4	2	0
F	0	0	4	2	0	0	0	2	0	2	2	2	2	0	0	0

The highest entry, corresponding to nonzero input-output differences in the difference distribution table of the Hamsi Sbox is 4, the number of possible input (output) differences is 16, hence the differential probability at most is $4/16 = 2^{-2}$ for each active Sbox. Any nonzero difference in the message activates at least 70 Sboxes, this results in a probability $(2^{-2})^{70} = 2^{-140}$. If we choose a message pair then the differential probability that it is a right pair is 2^{-140}. If we try all possible message pairs then the probability that there is at least one right pair is $2^{-140} \times 2^{32} = 2^{-108}$.

The existence of a colliding differential characteristic in the compression function (when there is nonzero difference on the message and the chaining value is fixed to

the IV) after several iterations, with probability higher than 2^{-128} would be a break of Hamsi-256. Indeed, this would imply that it is possible to construct colliding messages with a complexity lower than the generic birthday bound which is 2^{128}. If there exist 80 active Sboxes after 3 rounds, the probability of a differential characteristic (taking into account the message freedom) would be $(2^{-2})^{80} \times 2^{32} = 2^{-128}$. Hence we want to make sure that there exists no collision producing paths with less than 80 active Sboxes. However, this is not a sufficient security measure since there are many attacks of different nature that can exploit the properties of a hash function. Also we don't have a rigorous proof that Hamsi-256 achieves this goal.

We can do same the calculations for Hamsi-512. Namely, the minimum distance of the linear code used for message expansion of Hamsi-512 is 131, so given a difference on the message, there exist 131 active Sboxes for the first round. The probability of a 1-round differential characteristic is $(2^{-2})^{131} = 2^{-262}$. The probability of finding a right message pair after trying all possible pairs is $2^{-262} \times 2^{64} = 2^{-198}$. Note that Hamsi-512 processes 64-bit message blocks. Similarly, it is reasonable to target more than 155 active Sboxes since $(2^{-2})^{155} \times 2^{64} = 2^{-256}$.

4.2.2 An Upper Bound on the Probability of 1-round Differential Characteristics

As we have mentioned several times and as indicated from the computations of the probability of a 1-round differential characteristic, the linear code is an important parameter of the design with respect to differential type of attacks in general. This raises the question whether better codes exist. More precisely, what is the highest minimum distance that can be achieved?

In Section 3.7, we discuss the message expansion and describe the linear codes used. Hamsi-256 uses the linear code with the parameters [128,16,70] over F_4, the upper bound for this code in terms of minimum distance is 83. If one constructs a better code with the same parameters but higher minimum distance (upper bounded by 83), it can be used to improve Hamsi-256. However, this change would lead to a new design and should be analyzed. But since none of the existing analysis results so far on Hamsi exploits a property of the message expansion, improving the minimum distance would likely to improve the design because it will increase the number of active Sboxes. Similarly, for Hamsi-512 the minimum distance of the linear code [256,32,131] over F_4 is upper bounded by 168.

The improved linear codes would give better probabilities of 1-round differential characteristics, namely upper bounded by $(2^{-2})^{83} \cdot 2^{32} = 2^{-134}$ for Hamsi-256 and upper bounded by $(2^{-2})^{168} \cdot 2^{64} = 2^{-272}$ for Hamsi-512.

4.2.3 Pseudo-collisions

In any colliding differential characteristic in the hash function Hamsi there will be an iteration where one has to introduce a difference on the message and an iteration where there exists collisions in the chaining value. This can happen either in one iteration or in several iterations. In other words, it can take several iterations of the compression function to reach a collision. We have discussed above the probability of a 1-round differential characteristic of the compression function when there is a difference on the message. In Section 4.4 we will examine the case when there is a difference on the chaining transformation and no difference on the message.

In the hash function literature the type of collision where the attacker can choose the chaining value is called a *pseudo-collision*. An attacker can either set the chaining value to a fixed value different from the initial value or allow a difference. The first case is called a *semi-free start collision* where the other is a *free start collision*. If there exists a partial collision of the bits of the chaining value then this is called a *near collision*. Pseudo-collisions or near-collisions can be considered as a threat for the hash function as long as it is possible to use them as a part of a *collision producing path*. If this is not satisfied they should be considered as the properties of the compression function. A collision producing path is a colliding differential characteristics of several iterations, we have shown an example in Figure 4.1.

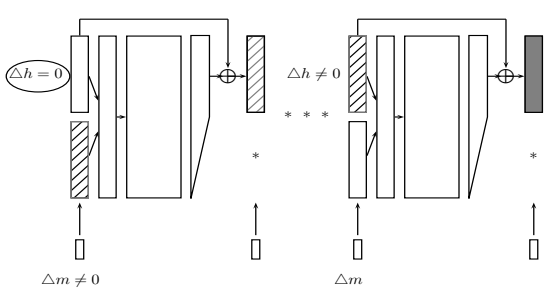


Figure 4.1: An example of a collision producing path before finalization in Hamsi

Table 4.3: Number of input differentials

Type of collision	$\triangle M_i$	$\triangle h_{i-1}$	Possible input differentials
free start	0	$\neq 0$	2^{256}
semi-free start	$\neq 0$	0	2^{32}
free start	$\neq 0$	$\neq 0$	2^{288}

We focus on the collisions before the finalization. It is harder to construct collisions for the finalization since it has more rounds, and any collision that happens before, can be combined with the finalization to construct a collision for the hash function.

4.3 External Analysis Results on Hamsi

Our initial analysis on Hamsi in Section 4.2 computes trivially the probability of a 1-round differential characteristic based on the number of active Sboxes when there is a nonzero difference on the message. However, all external analysis results mainly concentrate on the case when there is zero difference on the message and nonzero difference on the chaining value. This is because the message expansion ensures that the probability of differential characteristics when there is a difference on the message is low, as intended during the design process. Moreover, most of the analysis results on Hamsi-256 focus on the nonrandomness of the compression function exploiting the light compression function, another design choice. Additionally, there are results exhibiting differential distinguishers on the finalization.

The most effective external analysis results so far seem to be second-preimage attacks [34, 59, 51] that exploit the algebraic degree and some properties of the Sboxes or the low diffusion on some input bits into a theoretical attack to the hash function with marginal complexities.

There are almost no results on Hamsi-512 except a short notice in [34]. Hamsi-512 is a conservative design namely, although the parameters of Hamsi-512 are twice that of Hamsi-256, i.e, 64-bit message block size and 1024-bit state size and the primitives used in the permutations P and P_f are the same, the linear layer of Hamsi-512 is heavier which makes it harder to observe similar properties as in the case of the compression function of Hamsi-256.

4.3.1 Analysis of the Compression Function of Hamsi-256

On the degree of the compression function. The compression function of Hamsi is built from a permutation that consists of addition of constants and a counter, application of Sboxes and a linear transformation layer. The nonlinearity of the permutation is inherited from the nonlinearity of the Sboxes. The Hamsi Sbox has 4 input and 4 output bits, 3 out of 4 output bits have algebraic degree 3 (see Equation (3.12)). Now, we can proceed to compute the degree of the output bits after n rounds, for different scenarios of different variables chosen as constant, without confusing the reader. Note that a similar simple computation is also made in [12].

If we assume that all the input bits are variable then since each round increases the algebraic degree by 3, after n rounds, the algebraic degree of the output bits is at most 3^n. As we have mentioned above not all the output bits of the Sboxes have algebraic degree 3 (see Equation (3.12)) so it is possible that some output bits after n rounds have algebraic degree less than 3^n.

If we choose some of the input variables as constants then the above rule, as expected, may not hold. The algebraic degree of the output bits depend on the following parameters; the algebraic degree of the Sboxes, number of input variables, number of rounds. As a consequence of the number of rounds neither the compression function nor the finalization of Hamsi-256 (and Hamsi-256/8) achieves the algebraic degree of a pseudorandom function as noted by Aumasson in [10] and Aumasson et al. in [12]. So far we are not aware of any consequence of the fact that the algebraic degree of the compression function or the finalization is not ideal, on the security of the hash function.

Near-Collisions. There are several results that report near-collisions on the compression function of Hamsi-256 by Nikolić [104], Wang et al. [128] and Aumasson et al. [12], by only applying difference on the chaining value. In all those analysis results the hamming weight of the output is larger than the hamming weight of the input, hence it is not clear in which context they can be useful. However, the low complexity of those results indicate that it is relatively easy to construct near-collisions for the compression function of Hamsi-256 given only difference on the chaining value. But apparently it is not too evident to construct near colliding paths that lower the hamming weight of the input difference.

Note that the manually constructed near colliding path of Nikolić [104] exploits the fact that the branch number of the Hamsi Sbox is 3, i.e., given a 2-bit difference on the Sbox the output difference can be at most 1. The idea is to start from a low hamming weight difference after the second application of the Sbox layer and extend it forward and backward.

Distinguishers. Aumasson et al. [12] exhibits differential distinguishers for the compression function and finalization of Hamsi-256. Moreover, Boura et al. [31] and Aumasson et al. [13] show that it is relatively easy to find zero-sums for the finalization of Hamsi-256. However, this analysis does not take into account the message expansion which means that those subsets can never be input to the Hamsi permutation.

4.3.2 Analysis Results on the Hash Function

The first analysis result on the hash function is a pseudo second-preimage attack with complexity $2^{254.25}$ by Çalık et al. in [34]. In this paper, it is shown that given

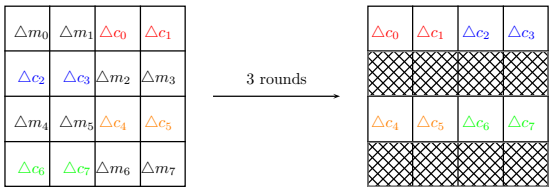


Figure 4.2: Form of collision producing differentials in Hamsi-256

a difference on certain input bits of the chaining value the difference on one of the output bit of the compression function can be predicted with probability one.

Fuhr [59] breaks the generic bound (for short messages) of finding preimages for Hamsi-256 with complexity equivalent to $2^{251.3}$ compression function evaluations. In his result, Fuhr sets conditions on the message block and the chaining variable in order to find affine relations between the output of the compression function and some bits of the chaining variable.

Dinur et al. [51] describes an algebraic attack that finds second preimages with complexity about 2^{247} compression function evaluations for short messages. The complexity computation of the attack is based on the comparison of bit operations required for the attack with the best available implementation of Hamsi-256, which raises the question whether a better implementation of Hamsi-256 would effect the result.

4.4 Collision Analysis of Hamsi-256

In this section, we will present our collision analysis of Hamsi-256. Our method mainly relies on imposing conditions on the Sboxes, i.e., choosing a certain input-output difference pair, and searching for sparse colliding differential characteristics that satisfy those conditions. Once we have the differential characteristics, we search for the satisfying message pairs.

About the differential structure. Figure 4.2 shows the nature of the input, output differentials of Hamsi-256 in order to obtain collisions. We are searching for differential characteristics that have the initial and final form shown in Figure 4.2. This is due to the XOR (feedforward) of the chaining values with the truncated Hamsi state after 3 rounds.

In the first part, we focus on linearization of the difference distribution table of the Sboxes, hence we impose linear relations. In the second part we impose zero differences on randomly chosen Sboxes.

As we have noted earlier, the nonlinear parts of a cipher play an important role in the application of differential cryptanalysis. Namely, when a difference is input to a nonlinear function there exist several output differences with different probabilities. Hence, choosing an input/output difference that will yield a collision with complexity lower than a generic attack after several rounds is not straightforward. In order to reduce the complexity of the analysis one approach is to search for differential characteristics with the number of active Sboxes as low as possible. These are called "sparse" differential characteristics. A good differential characteristic in the context of hash function cryptanalysis is a collision producing and a sparse one. Nevertheless, there is no-"generally successful method" to follow in the construction of sparse collision producing differential characteristics.

An approach used to tackle this problem is to replace the search for nonlinear collision producing differential characteristics with the search of linear ones. In this approach, given an input difference, the output difference is fixed such that the mapping from the input differences to output differences is linear. Consequently, this map constitutes a linear code, hence, searching for colliding sparse differential characteristics in a linear space can be formulated as the search of linear codewords of low Hamming weight in a linear code.

The linearization method has for the first time been used by Chabaud and Joux in [37] to attack SHA-0 and was subsequently applied to SHA-1 by Rijmen and Oswald in [111]. Note that in these hash functions linearizing the differential characteristics coincides with the linearization of the nonlinear components in the hash function. Later, Indesteege et al. have applied this approach to attack several SHA-3 candidates such as EnRUPT and SHAMATA-256 in [72] and [71].

4.4.1 Linearization

In this section we investigate collision producing differential characteristics which are linear. First, we fix a set of linear differential characteristics, then we solve for collision producing characteristics. Below we describe the method in detail.

As shown in Table 4.4 we choose a subset of input and output differences that constitute a linear map with respect to the XOR operation. Let 2 and 8 be the input differences chosen; from our table the corresponding output differences are 3 and 9 and it follows that

$$\begin{aligned}\Delta(2) \oplus \Delta(8) &= 3 \oplus 9 \\ &= \Delta(2 \oplus 8) \ .\end{aligned}$$

Note that we don't linearize the Sbox as a function but we linearize the difference distribution table. However, this gives a linearization of the Sbox as well. By

linearizing, we fix the output difference given an input difference, which enables to follow the characteristic trivially. Since XOR over F_2 is equivalent to addition in F_2 we can combine this linearization with the linear layer. Given an input differential we can compute the output differential uniquely. Hence, we can represent the set of differential characteristics satisfying this particular linearization as a linear transformation.

The next step is to solve for colliding linear differential characteristics. The kernel, i.e., the set of input differences mapped to 0, of this transformation gives the colliding differentials. This linear transformation takes as input a 288-bit input difference, where 32 bits comes from the message words and 256 bits from the chaining value, and outputs 256 bits. When we solve the linear system for colliding differential characteristics there exist 2^{32} solutions. For each of the solution we can check for valid message pairs.

Table 4.4: Choice of differences from the difference distribution table

Δ	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	$\boxed{16}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	$\boxed{2}$	0	2	0	0	2	2	2	0	4	2
2	0	0	0	$\boxed{4}$	0	4	0	0	0	4	0	0	0	0	0	4
3	0	4	2	0	0	0	$\boxed{2}$	0	0	2	0	0	2	0	2	2
4	0	0	0	0	0	0	4	0	0	0	4	4	0	$\boxed{4}$	0	0
5	0	4	0	2	2	2	2	0	$\boxed{2}$	0	0	0	2	0	2	2
6	0	0	2	2	2	2	0	0	2	2	0	0	0	0	$\boxed{2}$	2
7	0	0	0	0	4	2	0	2	0	0	2	$\boxed{2}$	2	0	0	2
8	0	0	0	2	0	2	0	4	0	$\boxed{2}$	0	0	0	4	0	2
9	0	0	0	2	0	0	0	2	4	2	2	2	$\boxed{2}$	0	0	0
A	0	0	2	0	2	0	4	0	2	0	$\boxed{4}$	0	0	0	2	0
B	0	4	0	0	2	0	2	0	2	2	0	0	2	0	0	$\boxed{2}$
C	0	0	2	0	$\boxed{2}$	0	0	0	2	0	0	4	0	4	2	0
D	0	$\boxed{4}$	2	2	0	2	2	0	0	0	0	0	2	0	2	0
E	0	0	2	0	2	0	0	$\boxed{4}$	2	0	0	0	0	4	2	0
F	0	0	$\boxed{4}$	2	0	0	0	2	0	2	2	2	2	0	0	0

In Figure 4.3 we present a 3 round pseudo-collision path that we have found by using the linearization method. The differences are stored on a 4 by 4 matrix similar to the representation of a Hamsi state. The differences on the expanded message words and the chaining value are concatenated as in Figure 3.4 and the Sboxes are applied as in Figure 3.6. We indicate the differences that cancel each other (by the feedforward) with the same colors. Note that there is no difference in the message and the differences are only in the chaining values. The values that are truncated after 3 rounds are shown with $*$. The differential characteristic is very dense, namely linearization does not help to construct sparse differential

characteristics and consequently the possibility of valid message pairs is very low.

There are 128 Sboxes in the first round of Hamsi-256; for each Sbox there exist 16 possible linearizations of the difference distribution table, this makes 16^{128} possible linearizations for one round. The number of possible linearizations is huge and there is no method to find the best one. Hence, it is unlikely that the linearization method will result in a successful attack.

```
00000000 00000000 6D28F7FB 86E815F6
F092E563 E320D437 00000000 00000000
00000000 00000000 0D1AF482 39FE3C3C
AAA2C7A0 0FF68052 00000000 00000000

E6F837D3 B8B910F2 E828A190 C97F7E58
6B576502 CDD14975 647BE79E 80DE580C
C18A370D 04B0E5D2 B1AFE290 F5BF9C04
3B329283 47F03106 6825C912 56BEE38D

5C5B80A6 08FC4F18 C1BB9795 E9298860
0046E2A7 FE017F2B 27AD2775 2DFFA88D
BEDA81EF 4D73C2BA 13BFA370 EA71B502
F640A3C8 AE43FE8A 0C3413B0 32084DCC

6D28F7FB 86E815F6 F092E563 E320D437
******** ******** ******** ********
0D1AF482 39FE3C3C AAA2C7A0 0FF68052
******** ******** ******** ********
```

Figure 4.3: Pseudo-collision path after linearization

4.4.2 Imposing a Zero-Difference on the Sboxes

We continue our search for sparse colliding differential characteristics with another method in this section. In the linearization approach described above, we impose linear relations on the Sboxes. It is relatively easy to construct colliding differential characteristics; however, since they are not sparse, they have a very low probability. In this section, we will examine the case when we impose zero differences. Note that in the difference distribution table the highest probability occurs for the zero difference. By randomly choosing Sboxes and imposing zero differences we try to hit a sparse colliding differential characteristics. If there exists a sparse colliding differential characteristics we should be able to find it with our method; compared to linearization it is harder to construct colliding differential characteristics but they are sparser.

An Algorithmic Description

We can describe our method briefly as follows:

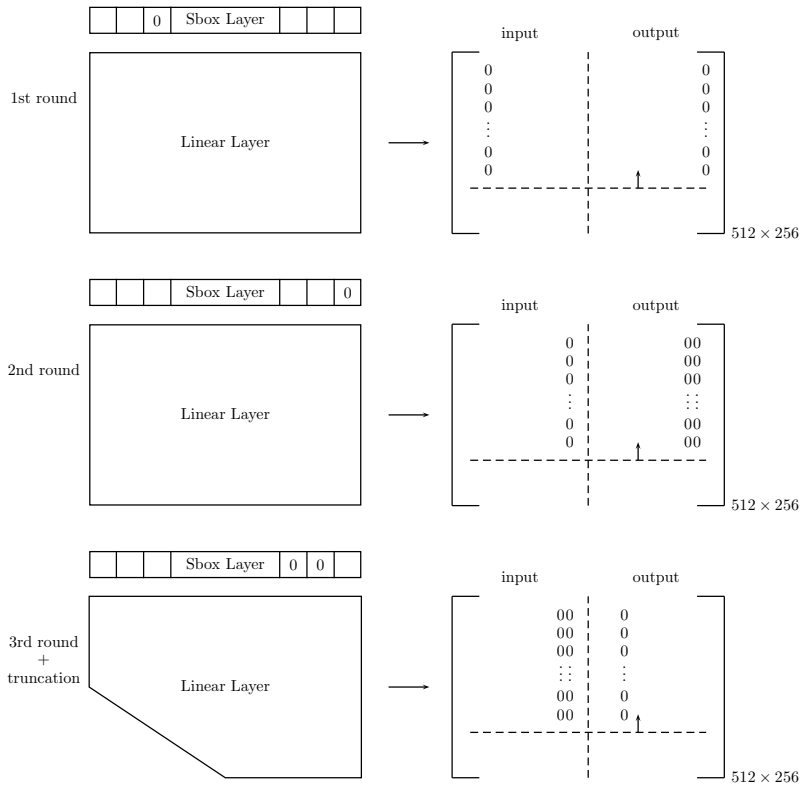


Figure 4.4: A sketch of our method

- Keep a search space of all possible differential characteristics for 3 rounds.
- Restrict the space by imposing zero differences on randomly chosen Sboxes, one at a time.
- Once the search space is small enough, enumerate the remaining differential characteristics and if one of them is sufficiently sparse, search for a satisfying message.

Since the search space becomes complicated when we start imposing zero differences, we approximate it as a product of 3 linear spaces. This product set includes the actual search space, but also contains a large number of elements that are not valid differential characteristics.

Our approach is illustrated in Figure 4.4. For each round we build a matrix that represents the linear space of all possible input and output differences of the linear

layer. The matrix has 512 rows, each of which consists of two halves: the first half represents an input difference to the linear layer and the second half represents the corresponding output difference. Both differences are represented by 128 4-bit values, where each value is the output resp. input difference of a single Sbox in the previous resp. next Sbox layer.

In order to obtain a valid 3-round differential characteristic, the output difference of each linear layer should be consistent with the input difference of the next layer. Namely, each Sbox between two consecutive linear layers should have compatible input and output differences. In particular, zero differences in the same positions should be preserved in consecutive layers.

Each time we impose a zero difference to a randomly chosen Sbox, we will update the linear spaces by eliminating all non-zero entries in the corresponding 4-bit column. This reduces the size of the linear space, as the dashed horizontal line in the matrices illustrates. For instance, if we impose a zero difference on the last Sbox of the second Sbox layer then we need to eliminate the last 4-bit column in the first half of the second matrix and also the last 4-bit column of the second half of the first matrix.

Because of the feedforward, in any colliding differential characteristic, the input difference on the chaining value should be the same as the truncated output difference after 3 rounds. Hence, when we impose a zero difference in the first Sbox layer (as shown in Figure 4.4), we will not only eliminate a 4-bit column in the first part of the first matrix but also in the second part of the last matrix. Note that two out of the 4 bits of each column in this part will already be zero because of the truncation.

In order to reduce the number of invalid characteristics in the approximated search space, we will try to eliminate inconsistencies between the 3 linear spaces as soon as we detect them. For instance, when a zero column appears in one of the matrices at a position at which we did not explicitly impose a zero difference, we will eliminate the corresponding column in the adjacent layer as well. This elimination may on its turn create new zero columns, resulting in a chain reaction in which the linear spaces reduce very quickly, either to a small subspace that we can exhaustively search, or to a space containing only the all zero characteristic, in which case we have to restart the process.

Pseudo-Collision Path

We present the pseudo-collision path (and the values) that we have found by imposing zero differences on the Sboxes in Figure 4.5. The left column of the Figure 4.5 contains the values and the right column the differences. In each layer the values and the differences are positioned on a 4 by 4 matrix representing the

```
105A411D B212EDD2 09B2C1FE 9CEEF906 | 00000000 00000000 00000400 04004000
1C2A6B5B 1D87131F 6982D873 48D5A272 | 00020084 00000000 00000000 00000000
849070A9 FAB73D35 FD1B3091 96DA604C | 00000000 00000000 00000000 04000000
B95A869E 767D2C94 AA451472 3AE05B7C | 00020084 00000000 00000000 00000000

FD15E236 845622B3 8CFA5F7A 48401089 | 04000000 00810000 00100000 00001100
E2BCDB5A F00CED01 C46CF5FF 5982659E | 00000008 00000000 00040800 00008000
CA20EB50 40B3C463 BDDD1C36 FBB4853C | 00001000 00000001 00000800 00008180
68D602FA 87DAA392 C3A119D4 95653DF5 | 00000000 00000000 00000000 00200000

4701D9F5 3085615E 942D4868 7597A62A | AD78300A 5D67064C 180142F8 C8868BDF
C3D4D560 E9DFBF04 9114FFD4 3B37438A | 20702018 01008100 40295820 40002285
082FA01A 88C35F6D B614141F FA7D2673 | 9408347A 12BC8706 00352AA0 00859AB5
C9AB478B A76C45A6 32B11C01 568D2B50 | 02C24002 00C02890 CE041442 546A4000

0147CC75 B77AD1C9 5AA88025 929E6636 | 00000400 04004000 00020084 00000000
89DB5FCF D0EE9A75 0511751B 72DA5D44 | ******** ******** ******** ********
A205F0E9 A846FABF AA4EEA5F 9F0CE200 | 00000000 04000000 00020084 00000000
35064178 22C980DD ADED0D84 8C96468E | ******** ******** ******** ********
```

Figure 4.5: Finding a right pair without the message expansion for Hamsi-256

Hamsi state. The first layer consists of the input differences to the first round and the corresponding values. Note that there are no differences on the message. The second layer consists of the differences input to the second round and the corresponding values. The last layer consists of output differences from the 3rd round and the corresponding values. The colliding differences by the feedforward are shown with the same colors.

Construction of the Values

Assume that we had a differential characteristic through one Sbox layer, then it would be easy to construct values because we could just assign values that satisfy the differences for each Sbox independently. If we had two consecutive Sbox layers, we can make use of the following observation on the difference distribution table of the Hamsi Sbox: given any valid non-zero input/output difference the number of values satisfying this difference is either 2 or 4. Since these set of values satisfy a fixed difference they form an affine space, this fact is also common to AES Sbox and noted in [43]. We can write the possible values for the output of the first Sbox layer and the input of the second Sbox layer as a system of affine equations taking into account the linear layer in between and solve for the values.

If we look at the differential path we have, the first Sbox layer contains the fewest active Sboxes, hence it makes sense to construct the values satisfying the pseudo-collision path starting from the 2nd and 3rd layer (using the method described above). Once we construct the values for the 2nd and 3rd layer we compute backwards to the 1st layer. The first layer is quite sparse (6 active Sboxes),

```
105A411D B212EDD2 09B2C1FE 9CEEF906 | 00000000 00000000 00000400 04004000
1C2A6B5B 1D87131F 6982D873 48D5A272 | 00020084 00000000 00000000 00000000
849070A9 FAB73D35 FD1B3091 96DA604C | 00000000 00000000 00000000 04000000
B95A869E 767D2C94 AA451472 3AE05B7C | 00020084 00000000 00000000 00000000

FD15E236 845622B3 8CFA5F7A 48401089 | 04000000 00810000 00100000 00001100
E2BCDB5A F00CED01 C46CF5FF 5982659E | 00000008 00000000 00040800 00008000
CA20EB50 40B3C463 BDDD1C36 FBB4853C | 00001000 00000001 00000800 00008180
68D602FA 87DAA392 C3A119D4 95653DF5 | 00000000 00000000 00000000 00200000

4701D9F5 3085615E 942D4868 7597A62A | AD78300A 5D67064C 180142F8 C8868BDF
C3D4D560 E9DFBF04 9114FFD4 3B37438A | 20702018 01008100 40295820 40002285
082FA01A 88C35F6D B614141F FA7D2673 | 9408347A 12BC8706 00352AA0 00859AB5
C9AB478B A76C45A6 32B11C01 568D2B50 | 02C24002 00C02890 CE041442 546A4000

0147CC75 B77AD1C9 5AA88025 929E6636 | 00000400 04004000 00020084 00000000
89DB5FCF D0EE9A75 0511751B 72DA5D44 | 1F12729F 4E7C352C 9D969B90 CB0D0181
A205F0E9 A846FABF AA4EEA5F 9F0CE200 | 00000000 04000000 00020084 00000000
35064178 22C980DD ADED0D84 8C96468E | E8059152 67C34144 41053112 06F171D6
```

Figure 4.6: A right pair without the message expansion for Hamsi$^{\oplus}$-256

hence we will simply try until the values that we computed backwards satisfy the differences.

4.4.3 The Effect of Message Expansion

The values in Figure 4.5 constructed from the pseudo-collision path do not coincide with a valid expanded message, namely there are 2^{32} possible messages, and none of them expands to the value that we have found. If there were no message expansion in Hamsi-256 this path would be a pseudo-collision. This can be interpreted in two ways, the message expansion is a strength of Hamsi against collision attacks however even without it, it is not so easy to construct collisions, namely we could only construct a pseudo-collision.

4.4.4 A Bound on the Number of Active Sboxes

As we have noted in Section 4.2.1 the number of active Sboxes in a collision producing path can be considered as a basic evaluation of security against differential cryptanalysis. We describe below how to estimate a very rough lower bound on the number of active Sboxes by using the number of tries (how many times we run the program) and the number of restrictions we impose on the Sboxes.

- Every round consists 128 Sboxes. Since there are 3 rounds, in total it makes $3 \cdot 128 = 384$ Sboxes. There are 97 possible input/output differences (nonzero entries) out of 256 in the difference distribution table of the Hamsi Sbox.

The probability of a valid difference is 97/256. The number of all possible colliding differential characteristics can be estimated as: $2^{32+256} \times 2^{512} \times 2^{512} \times 2^{256} \times (97/256)^{3 \cdot 128} = 2^{1030}$.

- Each time we impose a zero difference we reduce the search space roughly 97 times. Since $2^{1030}/(97^x) = 1 \Leftrightarrow x = 156$, if we impose 156 zeros this will leave one sparse differential characteristic.
- Suppose that the sparsest characteristic has w active Sboxes. Assume that we have imposed 156 zero differences (out of $3 \cdot 128$). We will find the characteristic if the active Sboxes do not collide with the 156 imposed zeros, the probability of this to happen is $(1 - \frac{w}{3 \cdot 128})^{156}$.

Note that this is a very rough probability, however we can use it to estimate answers to certain questions such as; How many trials do we expect to do in order to find a characteristic with 32 active Sboxes? The probability of this to happen is $(1 - 32/3 \cdot 128)^{156} \approx 2^{-20}$. Hence we need to make about 2^{20} trials. Assume that we did 2^{24} trials and we don't find a characteristic with 32 active Sboxes, how likely would be if there really existed such a characteristic? This we can compute as $(1 - 2^{-20})^{2^{24}} \approx \exp(-2^{-20} \times 2^{24}) = 0.0000001$. So it is very unlikely that such characteristic exist.

4.5 On the Security of Hamsi$^{\oplus}$

The security analysis of Hamsi$^{\oplus}$ is similar to Hamsi in many aspects, namely, the number of active Sboxes can be computed in the same way as with Hamsi and consequently the probability of a 1-round differential characteristic is the same and the algebraic degree of the compression function is the same.

The external analysis results can be modified for Hamsi$^{\oplus}$. The second-preimage attacks on the hash function can be applied in exactly the same way however since the chaining values are larger the complexity of the attacks are beyond the generic bounds.

The pseudo-collision path without a valid expanded message of Figure 4.5 is a differential characteristic for Hamsi$^{\oplus}$-256 as well, shown in Figure 4.6. Note that the differences after the 3rd round are not truncated, they will be XORed with the differences from the expanded message words of the next iteration. However the differences constructed are not valid, i.e., there exist only 2^{32} valid differences. Hence, in the case of Hamsi$^{\oplus}$ neither the values satisfying the zero difference on the message nor the differences of the expanded message words of the next iteration are valid.

4.6 Conclusion

In this chapter we presented the security analysis of Hamsi by examining the design aspects and by focusing on the collision analysis of Hamsi-256.

We have shown how the design choices of Hamsi affect its resistance against differential attacks by computing the probability of 1-round differential characteristics based on the number of active Sboxes. We also presented an upper bound on this probability computed by the upper bound on the minimum distance of the linear codes used for message expansion. This flexibility shows that if the linear codes with higher minimum distance is constructed then this will contribute to the security of Hamsi in general.

We have also presented a summary of external analysis results on Hamsi. There are many interesting external analysis results on Hamsi indicating the nonrandomness of the compression function and the finalization. Although we think that those analyzes are necessary since by design Hamsi is based on a light compression function whether those properties can be used to mount an attack on the hash function is unclear. There are also several marginal second-preimage attacks on the hash function with very high complexities. We believe that those results also show the strength of Hamsi-256.

The main result of this chapter is the search for colliding differential characteristics of the compression function of Hamsi-256 before finalization. We have investigated the case when there is a zero difference on the message and a non-zero difference on the chaining value. We followed two approaches, namely linearization and imposing zero differences on the Sboxes. By applying linearization we have constructed a colliding differential characteristic which is very dense, hence it is very unlikely that there exist satisfying values. By applying the second approach we have constructed sparser colliding differential characteristics and corresponding values. However the values that satisfy the differential path do not constitute a valid expanded message. The results presented in this chapter on Hamsi-256 are the best collision analysis results so far. Our results indicate the strength of Hamsi-256 in terms of collisions.

At the end of the chapter we also briefly mentioned the security of Hamsi$^{\oplus}$. Despite the fact that Hamsi$^{\oplus}$ is a tweak of Hamsi, a concrete understanding requires more analysis, hence we leave it as a future study.

Chapter 5

Indifferentiability of Hamsi and Hamsi$^{\oplus}$

"Any sufficiently advanced technology is indistinguishable from magic."

Arthur C.Clarke

In this chapter, we analyze the security of the mode of operation of Hamsi and Hamsi$^{\oplus}$ against generic attacks. Cryptographic hash functions should satisfy several security criteria. One can approach this problem by investigating the properties of the underlying primitives, e.g., the compression function or the block cipher, that are preserved by the construction (i.e, the mode of operation) [7, 15], or analyze the construction in the indifferentiability framework [93]. In [22] Bertoni et al. prove for the first time the indifferentiability of a construction (the sponge construction) calling a random permutation rather than an ideal compression function or ideal block cipher. Although Hamsi is not a sponge construction we follow a similar approach in our proof. Namely, we prove the indifferentiability of Hamsi and Hamsi$^{\oplus}$ from a random oracle truncated to a fixed output length, assuming that the underlying primitives, i.e., permutations P and P_f, are ideal and we establish concrete bounds. There are several hash functions which have been submitted to the SHA-3 competition that show some resemblance with the sponge construction. In a recent work of Andreeva et al. [5], a generalization of "sponge like" functions is introduced and analyzed under the indifferentiability framework. Whereas this general approach could probably be adopted for Hamsi and Hamsi$^{\oplus}$, it does not seem that it would make the proof easier than a dedicated approach. Moreover, by comparing our indifferentiability proofs of Hamsi and Hamsi$^{\oplus}$ it is

possible to follow the contribution of different components of the design in the proof, hence we believe that this makes it easier to understand the ideas and method.

5.1 Indifferentiability

Proving the security of a cryptosystem with an idealized primitive is a generic methodology followed in many cryptographic security proofs. The idealized primitive can be a random oracle or an ideal block cipher, a random permutation or a transformation, or any idealization of a primitive used in the system. Nevertheless this approach has different implications for systems with public interfaces compared to systems with secret interfaces. Namely, there are many results (e.g, [35, 55, 36]), showing that various cryptographic systems are secure in the random oracle model but insecure for any instantiation of the random oracle. Actually, a formal proof in the random oracle model indicates that there are no structural flaws in the design of the system.

Random oracles are idealizations of cryptographic hash functions. A random oracle denoted as RO is a map from $\{0,1\}^*$ to $\{0,1\}^\infty$ chosen by selecting each bit of $RO(x)$ uniformly and independently, for every input x [17, 57]. For the same query it responds always in the same way. It is trivial to distinguish a random oracle from a hash function given its interface and public parameter. For any value we can compute the hash function and compare the result with what we would get from the random oracle. Moreover, Canetti et al. prove in [35] that no hash function can implement a random oracle. More precisely, there exists a cryptosystem $C()$ such that $C(RO)$ is secure however $C(H)$ is insecure for any hash algorithm H.

Two systems are said to be indistinguishable if there exists no efficient algorithm that is able to decide which system it is interacting with. *Indifferentiability* is an extension of indistinguishability to systems which have public interfaces. It has been introduced by Maurer et al. [93]. Informally, it means that if a cryptographic primitive U is *indifferentiable* from another primitive V then the security of the cryptosystem based on U, i.e C(U), is not effected if we replace V by U. Coron et al. [39] applied this notion to iterated hash function constructions. Thus, when an adversary queries a concrete hash function it has also access to its inner primitives, namely compression functions or permutations. Hence when we argue the indistinguishability of a hash function from the random oracle we need some more tools to simulate the primitives and complete the setting. The purpose of the simulator shown in the setting in Figure 5.1 as evident from its name, is to simulate the random primitive.
On the right is the black box that contains the RO (truncated to a fixed output) and the simulator $P[RO]$, and on the left are the public values which are the hash function $S[F]$ and the underlying primitive F. The adversary tries to distinguish

the two systems by querying the black box with two interfaces and comparing the result with what it could obtain from the public values. The black box should give responses consistent with its components and with the public values. This is called the indifferentiability setting. More precisely, a hash function $S[F]$ is indifferentiable from a random oracle RO if there exists a simulator $P[RO]$ such that no adversary can distinguish the two systems with negligible probability.

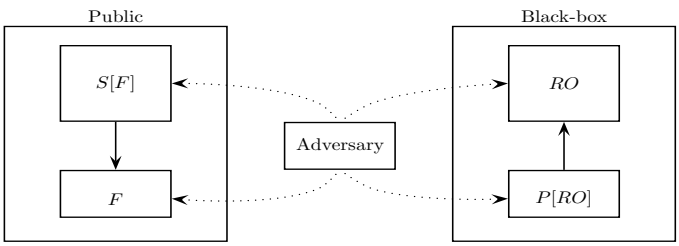


Figure 5.1: The indifferentiability setting

Below we present several definitions that state our indifferentiability setting formally.

Definition 5.1. *A function f is negligible, if for every polynomial p() there exists an integer $N > 0$ such that for all $n > N$, $f < \frac{1}{p(n)}$.*

Definition 5.2. *A security parameter in cryptography is a measure of resource requirements to break a cryptographic algorithm or a protocol, or the adversary's probability of breaking the security of a cryptosystem.*

Definition 5.3 ([39])**.** *A Turing machine C with oracle access to an ideal primitive F is said to be (t_D, t_S, q, ϵ) indifferentiable from an ideal primitive G if there exists a simulator $P[G]$, such that for any distinguisher D it holds that:*

$$|Pr[D[C[F]], F] = 1| - Pr[D[G, P[G]] = 1] < \epsilon \ .$$

The simulator has oracle access to G and runs in time at most t_S. The distinguisher runs in time t_D and makes at most q queries. Similarly, $C[F]$ is said to be indifferentiable from G if ϵ is a negligible function of the security parameter k.

In our setting the Turing machine C corresponds to the hash function, the ideal primitives F and G are the ideal compression function and the random oracle respectively. To summarize, if $C[F]$ is indifferentiable from a random oracle then we can replace the random oracle by $C[F]$ in a cryptosystem and the result is as secure in the ideal compression function model as in the random oracle model. Note that in a recent study of Ristenpart et al. [112] it is shown that this composition

statement is not necessarily valid for "any" cryptosystem, namely there exists a hash-based storage auditing scheme which is provably secure in the random-oracle model but can easily be broken in indifferentiability hash construction. This indicates that one must be careful about the limitations of this composition when performing security analyses.

5.2 Indifferentiability of Hamsi

In this section, after giving necessary definitions and representations, we will define the simulators for the permutations P and P_f and eventually prove the indifferentiability of Hamsi.

5.2.1 Hamsi Mode of Operation

The Hamsi mode of operation can be visualized as a variant of the Merkle-Damgård, stream based hash, Sponge or Concatenate-Permute-Truncate design approach. Although it is not a sponge construction, there are sufficient similarities that allows to adapt the indifferentiability proofs [22] to Hamsi, as we do in this chapter. In the indifferentiability framework it is assumed that the underlying primitives are ideal, although we clearly know that they are not. But as we have mentioned before those proofs show the strength of the mode of operation of a hash function against generic attacks if the underlying building blocks are ideal, so there is no conflict in this assumption.

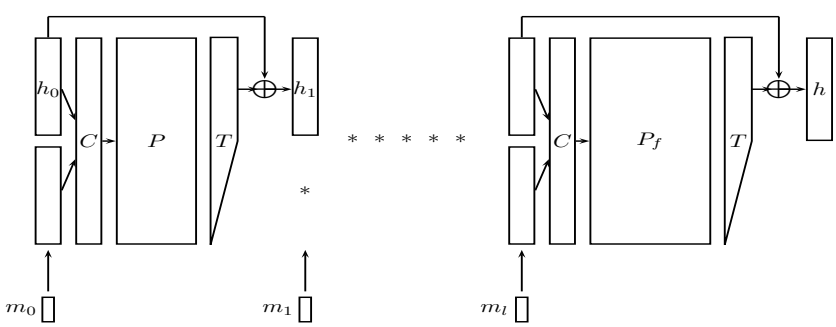


Figure 5.2: The Hamsi mode of operation

Description of Hamsi. Hamsi is constructed from the mappings below; we use simplified representations in the algorithmic description where we don't specify the mappings used for concatenation and truncation, for details see Chapter 3.

Message Expansion	$E : \{0,1\}^m \to \{0,1\}^r$
Non-linear Permutations	$P, P_f : \{0,1\}^{r+c} \to \{0,1\}^{r+c}$

Algorithm 5 The Hash Function Hamsi

Let M be a padded message, $M = m_0 m_1 \ldots m_{l-2} m_{l-1}$, $m_i \in F_2^m$
$s_c \leftarrow IV$
for $i = 0$ to $l-2$ **do**
  $(t_m, t_c) \leftarrow P(E(m_i), s_c)$
  $s_c \leftarrow t_c \oplus s_c$
end for
Finalization
$(t_m, t_c) \leftarrow P_f(E(m_{l-1}), s_c)$
$s_c \leftarrow t_c \oplus s_c$

5.2.2 Graph Representation

We represent the Hamsi mode of operation with a graph using the algorithmic description above. The directed graph is constructed by using two permutations P and P_f and XOR of the previous chaining value. We represent our graph with $G = (v, e)$, v corresponding to nodes, and e to edges, $v \in \{0,1\}^c$. Chaining values are stored in the nodes. Initially the graph contains only the initial value IV. Edges e are constructed in two steps: first by evaluating the permutation (P or P_f) labeled by valid expanded message words $E(m_i)$, second by XORing the result with the feedforward of the chaining variable; $s_c \xrightarrow{E(m_i)} t_c \oplus s_c$. In the first two levels of the graph we can apply only P and from then on we can apply either P or P_f for evaluating the next node. P_f can only be applied once and the nodes computed are not continued. This follows from the padding rule and by design P_f is only applied to the last padded message block to compute the hash value. For each node there are 2^c values that can be chosen with 2^m message labels by applying P or P_f.

Definition 5.4. *A path p from node s_c to a node t_c is an ordered set of messages denoted with $E(m_i)$, that can be followed by applying P or P_f.*

Given a path p to the node s_c we can construct a path p' from s_c to s'_c as follows: if s_c is not from a node where P_f is applied, then we compute $P(E(m_i), s_c)$ or $P_f(E(m_i), s_c)$ and the result of the computation is XORed with s_c to compute the next node s'_c. We add $E(m_i)$ to our set p and obtain $p' = p \cup \{E(m_i)\}$.

Definition 5.5. *A rooted node is a node that can be reached from the node IV.*

As we note in the following definition, there may exist several paths to a node.

Definition 5.6. *Given a path p from the initial node IV to the node s_c, a* collision *is a path p' from IV to s_c where there exists a message label $E(m_i) \in p'$ but $E(m_i) \notin p$.*

We will use those definitions when we construct our simulators. The graph of the simulators can be constructed similar to the graph of Hamsi (or Hamsi$^{\oplus}$).

5.2.3 The Distinguisher's Setting

The underlying primitive of the Hamsi mode of operation is constructed from the permutations P and the finalization P_f, and the transformations concatenation (C), truncation (T) and feedforward. Hence, we can prove the indifferentiability for the cases when the underlying primitive is a transformation or a permutation. By a transformation we mean any combination of the permutations with the concatenation, the truncation or the feedforward. We believe that choosing the smallest component would give more insight about the soundness of the construction and the indifferentiability bounds, hence we give the proof for the case when the underlying primitives are random permutations.

Definition 5.7. *A random permutation defined over a certain domain is a permutation chosen randomly and uniformly from the set of all permutations defined over this domain.*

The adversary tries to distinguish the two systems $(RO, P[RO], P_f[RO])$, and $(S[F_P, F_{P_f}], F_P, F_{P_f})$ as shown in Figure 5.3. RO is the random oracle, $S[F_P, S_{P_f}]$ is the Hamsi (or Hamsi$^{\oplus}$) hash function returning requested bit output values, (256, 224, 384 or 512 bits). $P[RO]$ and $P_f[RO]$ denotes simulators of P and P_f respectively. The two systems both have the same interfaces. H is the interface for $S[F_P, F_{P_f}]$ and the random oracle RO. H takes variable length input $x \in \{0,1\}^*$ and an integer $n \in \{256, 224, 384, 512\}$ and returns $y \in \{0,1\}^n$. The public primitives F_{P_f}, F_P and corresponding simulators $P[RO], P_f[RO]$ are queried with the interfaces F and F_f. The inverse of the primitives and the simulators are queried by using the interfaces F^{-1} and F_f^{-1}.

Let X be a sequence of queries to either $(S[F_P, F_{P_f}], F_P, F_{P_f})$ or $(RO, P[RO], P_f[RO])$. The sequence X consists of queries Q^0 to the interface H and queries Q^1 to the interfaces F, F_f, F^{-1}, F_f^{-1}. The adversary can query the random oracle, simulators, the hash function and the underlying primitives by querying the corresponding interfaces. In our setting the simulator $P_f[RO]$ queries the random oracle.

The adversary can check for consistency of the responses of F_f (i.e,$(P_f[RO])$) by querying the random oracle. The adversary tries to distinguish the two systems

by finding an inconsistency or by comparing the probability distributions. The systems should give consistent responses with the internal parts to the queries made by the adversary. The simulator should give consistent answers with what the adversary could get by querying the RO. The random oracle RO and the simulators should give consistent answers with what the adversary could get by querying the hash function and the ideal primitives. By construction $(S[F_P, F_{P_f}], F_P, F_{P_f})$ is consistent. The consistency of $(RO, P[RO], P_f[RO])$ is proved by Lemma 5.3.

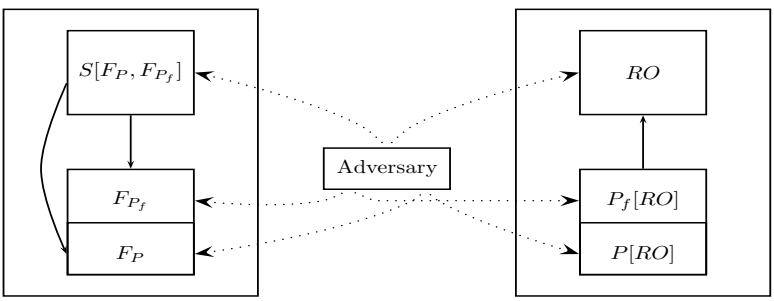


Figure 5.3: The distinguishers setting

Definition 5.8. *The cost N of a sequence X of queries is the total number of calls to F, F_f, F^{-1} or F_f^{-1} either due to queries Q^1 or via queries Q^0 to $S[F_P, F_{P_f}]$.*

Each query to F, F_f, F^{-1} or F_f^{-1} contributes 1 to the cost. An l-bit query to H contributes $\lfloor \frac{l}{m} \rfloor + 3$ to the cost, due to the padding rule. This corresponds to $\lfloor \frac{l}{m} \rfloor + 2$ calls to F and one call to F_f.

Padding. The Hamsi compression function process m-bit message blocks. This is enabled by the padding which is a mapping that inputs variable length messages and outputs a sequence of m-bit message blocks. The Hamsi hash function with padding is denoted with H' and

$$H'[F_P, F_{P_f}](x) \equiv S[F_P, F_{P_f}](pad(x)) \ .$$

5.2.4 Simulators $P[RO]$ and $P_f[RO]$

In this section we give precise definitions of $P[RO]$ and $P_f[RO]$, that simulate the ideal primitives and their inverses. As shown in Figure 5.3 the adversary queries the interfaces. Arrows represent the direction of the queries. Simulators give responses to minimize the probability that two systems can be distinguished from

each other. In order to achieve this, the simulators keep track of those queries in a directed graph similar to the one from Section 5.2.2 and also maintain several data sets which are described below. Moreover, the adversary can also rebuild the simulator graph from the responses. The domain and image of queries to the interfaces F and F_f are represented by the sets S_F, T_F, S_{F_f}, T_{F_f}. $S_F = \{$ set of (s_m, s_c) queried to F or returned as a response by $F^{-1}\}$, $T_F = \{$ set of (t_m, t_c) queried to F^{-1} or returned as a response by $F\}$, $S_{F_f} = \{$ set of (s_m, s_c) queried to F_f or returned as a response by $F_f^{-1}\}$, $T_{F_f} = \{$ set of (t_m, t_c) queried to F_f^{-1} or returned as a response by $F_f\}$. Obviously, $|S_F| = |T_F|$, $|S_{F_f}| = |T_{F_f}|$. The definitions of rooted node and path from Section 5.2.2 can be straightforwardly borrowed. The set of expanded message words is $M = \{E(m_i)|m_i \in \{0,1\}^m\}$ and $R = \{s_c|s_c$ is rooted $\}$ is the set of rooted nodes. R contains IV initially. Note that the number of rooted nodes cannot be larger than the number of queries. The set of nodes with an outgoing edge is $O = \{s_c|s_c$ has an outgoing edge$\}$. O initially has no elements. A and C are the domains of the queries (s_m, s_c), $A = \{s_m|s_m \in \{0,1\}^r\}$, $C = \{s_c|s_c \in \{0,1\}^c\}$.

For the rooted nodes for which there is a path from the initial node IV, the adversary can query the random oracle in order to check the consistency of the responses returned by the simulators. Such sequences of responses are called *consistent* when they don't yield inconsistencies. If $R \cup O = C$ then the simulator can no longer ensure consistency, and it is called *saturated*. The simulator fails when this happens. We try to avoid inner collisions because those can be extended to collisions in the hash function which yield an inconsistency with the definition of a random oracle.

Description of the Simulator $P[RO]$ of Hamsi. Here we explain the pseudocode description of the simulator for the permutation P given in Algorithm 6. Assume that the adversary queries $(s_m, s_c) \in A \times C$ from the interface F. The simulator checks if it has been queried or returned (by F^{-1}) before. If yes then it returns the image of the query from T_F. If not, then if s_m is not a valid expanded message word or s_c is not rooted then it returns (t_m, t_c) randomly and uniformly from the set $A \times C$ excluding the set T_F. In both cases on s_c the adversary cannot check for consistency by calling the random oracle. Hence the simulator can return a random and uniform value from $(A \times C) \setminus T_F$. Note that since P is a permutation, it is bijective, and the simulator, although it returns random and uniform values, should return each value only once. In order to preserve the properties of the permutation, the returned value should not be an element of T_F. To keep track of the queries and responses, we add (s_m, s_c) to S_F and (t_m, t_c) to T_F and update the graph with an edge from s_c to $t_c \oplus s_c$. We also add s_c to the set of nodes with an outgoing edge O.
If s_m is a valid message, s_c is rooted, and the simulator is not saturated, then the simulator chooses t_m randomly and uniformly such that (t_m, t_c) is chosen excluding the set T_F and $s_c \oplus t_c$ is chosen randomly and uniformly excluding the nodes which

are rooted (R) and those with an outgoing edge (O). This is to ensure that any call of a rooted node should result only in the path of a single node known and to avoid multiple paths. Hence $t_c \oplus s_c$ should be added to the set of rooted nodes R and the simulator updates the graph with an edge from s_c to $t_c \oplus s_c$. The simulator adds (s_m, s_c) to S_F and (t_m, t_c) to T_F and s_c to the set of nodes with an outgoing edge O. The simulator fails if it is saturated.
Assume that the adversary queries (t_m, t_c) from the interface F^{-1}. The simulator checks if it has been returned or queried (by F) before. If yes, then it returns the pre-image of (t_m, t_c). If not then it returns (s_m, s_c) randomly and uniformly excluding the set S_F and adds an edge from s_c to $t_c \oplus s_c$ in the simulator graph. The probability that s_m is a valid expanded message word is $2^m/2^r$, where $2^m \ll 2^r$. Hence, the simulator of the inverse permutation P^{-1} returns consistent responses with the RO and the ideal permutation F_P with high probability because they can not be checked and it does not lead to new rooted nodes or new paths.

Description of the Simulator $P_f[RO]$ of Hamsi. The simulator $P_f[RO]$ behaves similar to the simulator $P[RO]$. Here we will only describe the differences. Assume that the adversary queries (s_m, s_c) from the interface F_f. If s_m is a valid expanded message word and s_c is rooted and the simulator is not saturated then the adversary can query the random oracle to check for the consistency of the response of the simulator. Hence, the simulator calls the random oracle to assign to t_c the value $RO(s_m) \oplus s_c$. The simulator chooses t_m such that $(t_m, t_c) \notin T_{F_f}$, and adds t_c to the set of rooted nodes R. To keep track of the queries and responses, the simulator adds (s_m, s_c) to S_{F_f}, (t_m, t_c) to T_{F_f} and s_c to the set of nodes with an outgoing edge O. The simulator adds an edge from s_c to $s_c \oplus t_c$ and returns (t_m, t_c). The queries to F_f^{-1} are processed similar to F^{-1}.

5.2.5 Indifferentiability Proofs

Lemma 5.1. *To every node in the simulator graph of the permutation P there is at most one path, unless the simulator is saturated.*

Proof. Assume that there exists a node t_c in the simulator graph of P such that $\exists$ multipaths to that node. By definition paths are constructed for valid expanded message words. And by definition of the simulator (Algorithm 6, steps 8-14), we can only construct paths from rooted nodes to rooted nodes. It follows that t_c is rooted. From step 10 of Algorithm 6 there exists s_c, t'_c and t_m such that $t_c = s_c \oplus t'_c$ and $(t_m, t'_c) \notin T_F$, we can obtain t_c uniquely from s_c and t'_c. This is a contradiction to our assumption of multipaths hence the simulator should be saturated (by step 8). And by definition, F^{-1} does not lead to new rooted nodes or new paths, hence queries to F^{-1} do not yield multipaths either. □

Algorithm 6 Simulator $P[RO]$ of Hamsi

```
Query (s_m, s_c) from the interface F
if (s_m, s_c) ∉ S_F then
   if (s_m ∉ M OR s_c ∉ R) then
      choose (t_m, t_c) ∈ (A × C) \ T_F randomly and uniformly
      add (s_m, s_c) to S_F and (t_m, t_c) to T_F and s_c to O
      add an edge from s_c to s_c ⊕ t_c
      return (t_m, t_c)
   else if (R ∪ O ≠ C) then
      choose t_m randomly and uniformly
      choose (t_m, t_c) ∉ T_F and s_c ⊕ t_c ∈ C \ (R ∪ O) randomly and uniformly
      add s_c ⊕ t_c to the set of rooted nodes R
      add (s_m, s_c) to S_F and (t_m, t_c) to T_F and s_c to O
      add an edge from s_c to s_c ⊕ t_c
      return (t_m, t_c)
   else
      simulator fails
   end if
end if
else return (t_m, t_c) ∈ T_F
```

```
Query (t_m, t_c) from the interface F^{-1}
if (t_m, t_c) ∉ T_F then
   choose (s_m, s_c) ∉ S_F randomly and uniformly
   add (s_m, s_c) to S_F and (t_m, t_c) to T_F and s_c to O
   add an edge from s_c to s_c ⊕ t_c
   return (s_m, s_c)
end if
else return (s_m, s_c) ∈ S_F
```

Lemma 5.2. *To every node in the simulator graph of P_f there is at most one path, unless the simulator is saturated.*

Proof. The proof follows similar steps as the proof of Lemma 5.1. The only difference is that t_c is obtained by querying the RO, but the random oracle RO returns the same answer given the same query. The rest follows the same reasoning of Lemma 5.1. □

Lemma 5.3. *Given queries to the simulator $P_f[RO]$ and RO, the simulator returns consistent responses, unless it is saturated.*

Proof. The adversary can only search for inconsistencies by calling the random oracle RO for the nodes which are rooted and queried to $P_f[RO]$. By Lemma 5.1

Algorithm 7 Simulator $P_f[RO]$ of Hamsi

1: Query (s_m, s_c) from the interface F_f
2: **if** $(s_m, s_c) \notin S_{F_f}$ **then**
3: **if** ($s_m \notin M$ or $s_c \notin R$) or (path cannot be unpadded) **then**
4: choose $(t_m, t_c) \in (A \times C) \setminus T_{F_f}$ randomly and uniformly
5: add (s_m, s_c) to S_{P_f} and (t_m, t_c) to T_{F_f} and s_c to O
6: add an edge from s_c to $s_c \oplus t_c$
7: return (t_m, t_c)
8: **else if** $(R \cup O \neq C)$ **then**
9: assign $t_c = RO(\text{path}) \oplus s_c$
10: choose $(t_m, t_c) \notin T_{F_f}$ randomly and uniformly
11: add t_c to the set of rooted nodes R
12: add (s_m, s_c) to S_{F_f} and (t_m, t_c) to T_{F_f} and s_c to O
13: add an edge from s_c to $s_c \oplus t_c$
14: return (t_m, t_c)
15: **else**
16: simulator fails
17: **end if**
18: **end if**
19: else return $(t_m, t_c) \in T_{F_f}$

1: Query (t_m, t_c) from the interface F_f^{-1}
2: **if** $(t_m, t_c) \notin T_{P_f}$ **then**
3: choose $(s_m, s_c) \notin S_{F_f}$ randomly and uniformly
4: add (s_m, s_c) to S_{F_f} and (t_m, t_c) to T_{F_f} and s_c to O
5: add an edge from s_c to $s_c \oplus t_c$
6: return (s_m, s_c)
7: **end if**
8: else return $(s_m, s_c) \in S_{F_f}$

for each such node there exists a unique path (Algorithm 7 steps 8-14). The answers returned from the simulator $P_f[RO]$ will be consistent with what the adversary could obtain from the random oracle RO as long as the simulator is not saturated. □

The following lemma is about the equivalence of queries in between Q^0 and Q^1 up to the cost 2^c. Recall that Q^0 denotes queries to H and Q^1 denotes queries to F, F_f, F^{-1}, F_f^{-1}.

Lemma 5.4. *Any sequence of queries of type Q^0 up to cost 2^c can be converted to a sequence of queries of type Q^1. Q^1 gives at least the same amount of information to the adversary and has no higher cost than Q^0.*

Proof. Let $(x, n) \in Q^0$ and $x' = pad(x)$. Given the algorithm for the mode of operation of Hamsi, we can convert x' into $x'/m - 1$ queries to F and 1 query to F_f each with cost 1. By Lemma 5.3 all queries up to 2^c are consistent with the responses returned from the RO. Since we count the cost of queries to F and F_f equivalent for the sake of simplicity we can assume that x can be converted to x'/m queries of type Q^1 each of cost 1. And by Definition 5.8 the cost of queries to Q^0 is $\frac{x'}{m}$. This can be repeated for all queries $(x, n) \in Q^0$. □

Definition 5.9. *The advantage of an adversary is a measure of how successfully it can attack a cryptographic algorithm, by distinguishing it from an idealized version of that type of algorithm.*

In the following lemma we show that the advantage of an adversary to distinguish the permutations P and P_f from the simulators $P[RO]$ and $P_f[RO]$ is negligible.

Lemma 5.5. *The advantage of an adversary to distinguish between the two systems ($S[F_P, F_{P_f}]$,F_P,F_{P_f}) and (RO,$P[RO]$,$P_f[RO]$) with the responses to a sequence of $(q + q_f) < 2^{c/2}$ queries Q^1 is upper bounded by:*

$$f(q, q_f) = 1 - \frac{\prod_{i=0}^{q-1}(1 - \frac{2i+1}{2^c}) \cdot \prod_{i=0}^{q_f-1}(1 - \frac{i}{2^r})}{\prod_{i=0}^{q-1}(1 - \frac{i}{2^{r+c}}) \cdot \prod_{i=0}^{q_f-1}(1 - \frac{i}{2^{r+c}})} \ ,$$

where q and q_f denote the number of queries to F_P and F_{P_f} respectively.

Proof. Let F represent the permutations (F_P or F_{P_f}) and S represent the simulators ($P[RO]$ or $P_f[RO]$). The advantage of the adversary is defined as follows:

$$Adv(A) = |Pr[A[F] = 1)] - Pr[A[S] = 1]| \ .$$

Let $X \in A \times C$, denote a sequence of responses to a sequence of $(q + q_f)$ different queries to F or S. The set N of all X sequences has $(2^{c+r})^{\cdot(q+q_f)}$ elements. The adversary tries to distinguish the two systems F and S by comparing their probability distributions, namely it computes 1 for the response sequence X if $Pr(X|F) > Pr[X|S]$ and 0 otherwise. Note that the simulators give non-uniform responses only for the nodes which are rooted, hence to obtain maximum variational difference it makes sense for the adversary to query rooted nodes.

The advantage of the adversary is upper bounded by the variational difference which can be formulated as follows:

$$Adv(A) \leq \sum_{X \in N} |Pr[F|X] - Pr[S|X]| \cdot Pr[X] = \frac{1}{2} \sum_{X \in N} |Pr[X|F] - Pr[X|S]| \ .$$

where

$$Pr[S|X] = \frac{Pr[X|S] \cdot Pr[S]}{Pr[X]} \quad , \quad Pr[F|X] = \frac{Pr[X|F] \cdot Pr[F]}{Pr[X]} \quad ,$$

and

$$Pr[F] = Pr[S] = \frac{1}{2} \quad .$$

For the rest of the proof we explain and compute the associated probabilities.

Compute $Pr[X|F] = Pr[X|F_P] \cdot Pr[X|F_{P_f}]$**:** Let q and q_f be the number of queries to the interfaces F_P and F_{P_f} respectively, $(q + q_f) < 2^{c/2}$. Since P and P_f are permutations their image set is uniformly distributed. Recall that the domain and the image of the queries is the set $A \times C$. Let N_P and N_{P_f} denote the possible number of answers to q queries to P and q_f queries to P_f. For ith query there exists $2^{r+c} - (i-1)$ possible values, hence we have:
$N_P = 2^{r+c} \cdot (2^{r+c} - 1) \ldots (2^{r+c} - q + 1)$ and
$N_{P_f} = 2^{r+c} \cdot (2^{r+c} - 1) \ldots (2^{r+c} - q_f + 1)$.
The associated probabilities are:

$$Pr[X|F] = Pr[X|F_P] \cdot Pr[X|F_{P_f}] = \frac{1}{N_P} \cdot \frac{1}{N_{P_f}} = [\frac{(2^{r+c})!}{(2^{r+c} - q)!}]^{-1} \cdot [\frac{(2^{r+c})!}{(2^{r+c} - q_f)!}]^{-1} \quad .$$

If we denote $\frac{a!}{(a-n)!}$ with $a_{(n)}$, we have

$$Pr[X|F] = [(2^{r+c})_{(q)} \cdot (2^{r+c})_{(q_f)}]^{-1} \quad .$$

Compute $Pr[X|S] = Pr[X|P[RO]] \cdot Pr[X|P_f[RO]]$**.**

Compute $Pr[X|P[RO]]$ (Algorithm 6): The number of possible answers from step 4 is $(2^{r+c}) \cdot (2^{r+c} - 1) \cdot (2^{r+c} - 2) \ldots (2^{r+c} - q + 1)$ with probability 0, (since we assume that all the queries are rooted).
The number of possibilities from step 8 is computed as follows: S_P, T_P have initially zero elements. For each query we increase the number of elements by one for each set. $t_m \in A$ is chosen randomly and uniformly hence for q queries there are $(2^r)^q$ possible responses.

The number of possibilities from $(t_m, t_c) \notin T_P$ and $s_c \oplus t_c \in C \setminus (R \cup O)$ for q queries is computed as follows:

For each query of a rooted node, the elements of the set R, is increased by one. R initially has only the IV. The set of nodes with an outgoing edge O and the set T_P initially has zero elements. For the first query there are $2^c - 1$ possible responses.

Each time a response is returned to a rooted node the number of elements of the set $R \cup O$ and T_P is increased by one. Hence number of possible responses is at least: $(2^c - 1) \cdot (2^c - 3) \ldots (2^c - 2q + 1)$. For q queries to the simulator $P[RO]$ the probability is:

$$Pr[X|P[RO]] \leq \frac{1}{(2^r)^q \cdot (2^c - 1) \cdot (2^c - 3) \ldots \cdot (2^c - 2q + 1)} \quad .$$

Compute $Pr[X|P_f[RO]]$ (Algorithm 7): The number of possible answers from step 4 is $(2^{r+c}) \cdot (2^{r+c} - 1) \cdot (2^{r+c} - 2) \ldots (2^{r+c} - q_f + 1)$ with probability 0. The number of possibilities for t_c, determined by the random oracle in step 9, is $(2^c)^{q_f}$. The number of possibilities for t_m in step 10 is at least: $2^r \cdot (2^r - 1 \ldots (2^r - q_f + 1)$ (Since t_c is assigned a value by step 9, we compute the probability that $(t_m, t_c) \notin T_{P_f}$ by counting t_m such that the desired condition holds.)

$$Pr[X|P_f[RO]] \leq \frac{1}{(2^c)^{q_f} \cdot 2^r \cdot (2^r - 1) \ldots (2^r - q_f + 1)} = [(2^c)^{q_f} \cdot (2^r)_{(q_f)}]^{-1} \quad .$$

Hence we have:

$$Pr[X|S] \leq \frac{1}{(2^r)^q \cdot [(2^c - 1) \cdot (2^c - 3) \ldots (2^c - 2q + 1) \cdot (2^c)^{q_f} \cdot (2^r)_{(q_f)}} \quad .$$

Compute $Pr[X|P^{-1}[RO]$**:** $(s_m, s_c) \in S_p$ with probability 2^{m-c} since $m \ll c$, $(s_m, s_c) \notin S_p$ most of the time (with probability $1 - 2^{m-c}$). The same holds for $Pr[x|P_f^{-1}[RO]]$. Queries to $P^{-1}[RO]$ and $P_f^{-1}[RO]$ never yield new rooted nodes, hence they are either with probability 0 or already counted before.

Compute Adv(A):

$$Adv(A) \leq \frac{1}{2} \sum_{X \in N} |Pr[X|F] - Pr[X|S]|$$

We can divide the set N into 3 subsets N_0, N_A, N_B defined as follows:

- $N_0 = \{X|Pr[X|F] = Pr[X|S] = 0\}$
- $N_A = \{X|Pr[X|S] = 0,\ Pr[X|F] \neq 0\}$
- $N_B = \{X|Pr[X|F] \neq 0 \text{ and } Pr[X|S] \neq 0\}$

If a sequence has the same response for different queries then this cannot be returned by the random permutations or the simulators, hence such sequences are

represented by the subset N_0. The set N_A represents the set of responses that can be returned only by the ideal permutations and the set N_B represents the responses that can be returned by the permutations or the simulators. Note that N_B is a subset of N_A.

$$\begin{aligned}
Adv(A) \leq & \frac{1}{2}(\sum_{X \in N_0} |Pr[X|F] - Pr[X|S]| \\
& + \sum_{X \in N_A \setminus N_B} |Pr[X|F] - Pr[X|S]| \\
& + \sum_{X \in N_B} |Pr[X|F] - Pr[X|S]|) \\
& = \frac{1}{2}(0 + (|N_A| - |N_B|) \cdot \frac{1}{|N_A|} + |N_B| \cdot (\frac{1}{|N_B|} - \frac{1}{|N_A|})) \\
& = 1 - \frac{|N_B|}{|N_A|} \ .
\end{aligned}$$

$$\begin{aligned}
Adv(A) \leq & 1 - \frac{(2^r)^q \cdot [(2^c - 1) \cdot (2^c - 3) \ldots (2^c - 2q + 1) \cdot (2^c)^{q_f} \cdot (2^r)_{(q_f)}}{(2^{r+c})_{(q)} \cdot (2^{r+c})_{(q_f)}} \\
= & 1 - \frac{\prod_{i=0}^{q-1} (1 - \frac{2i+1}{2^c}) \cdot \prod_{i=0}^{q_f - 1} (1 - \frac{i}{2^r})}{\prod_{i=0}^{q-1} (1 - \frac{i}{2^{r+c}}) \cdot \prod_{i=0}^{q_f - 1} (1 - \frac{i}{2^{r+c}})} \ .
\end{aligned}$$

□

Theorem 5.6. *Let $H'[F_P, F_{P_f}]$ be the Hamsi hash function with the padding transformation where F_P and F_{P_f} are random permutations. $H'[F_P, F_{P_f}]$ is $(t_D, t_S, q, q_f, \epsilon)$ indistinguishable from a random oracle, for any t_D,$t_S = O((q + q_f)^2)$, $q + q_f < 2^{\frac{c}{2}}$ and any $\epsilon > f(q, q_f)$.*

Proof. By Lemma 5.4 it is sufficient to consider sequences of type Q^1. Lemma 5.5 gives an upper bound $f(q, q_f)$ on the distinguishing advantage of an adversary of a sequence of queries of cost $q + q_f < 2^{c/2}$. For each query which is rooted the simulator should find the path to that query and call the *RO* with a cost equal to length of this path. The length of the path can be at most the number of rooted nodes which is $q + q_f$, hence $t_s = (q + q_f) \cdot (q + q_f)$. □

Without loss of generality we can take $q = q_f$ and if $q \ll 2^{c/2}$ then we can use $1 - x \approx e^{-x}$ for $x \ll 1$ to simplify $f(q, q_f)$:

$$\begin{aligned} f(q, q_f) \equiv f(q) &\approx 1 - \exp[-(\frac{q^2}{2^c} + \frac{q \cdot (q-1)}{2^{r+1}} - 2 \cdot \frac{q \cdot (q-1)}{2^{r+c+1}})] \\ &\approx \frac{q^2}{2^c} + \frac{q \cdot (q-1)}{2^{r+1}} \quad . \end{aligned} \tag{5.1}$$

5.3 Indifferentiability of Hamsi$^{\oplus}$

In this section we will examine the indifferentiability of a variant of Hamsi denoted with Hamsi$^{\oplus}$. All the definitions and proofs from the previous sections apply to here. Below we give description of Hamsi$^{\oplus}$, the simulators and prove Lemma 5.5.

5.3.1 Description of Hamsi$^{\oplus}$.

In this construction expanded message blocks are XORed to the state instead of overwriting. This makes Hamsi wide-pipe where the chaining values are twice the hash size. In Algorithm 8, we give the pseudo-code description of the mode of operation of Hamsi$^{\oplus}$. This is a simplified description of Algorithm 4, where we have omitted the concatenation map in order to have a higher level of abstraction, which is enough for our proofs.

Algorithm 8 The Hash Function Hamsi$^{\oplus}$

Let M be a padded message, $M = m_0 m_1 \ldots m_{l-2} m_{l-1}$, $m_i \in F_2^m$
$s_m \leftarrow 0^r, s_c \leftarrow IV$
for $i = 0$ to $l - 2$ **do**
  $(t_m, t_c) \leftarrow P(E(m_i) \oplus s_m, s_c)$
  $(s_m, s_c) \leftarrow (t_m, t_c \oplus s_c)$
end for
Finalization
$(t_m, t_c) \leftarrow P_f(E(m_{l-1}) \oplus s_m, s_c)$
$(s_m, s_c) \leftarrow (t_m, t_c \oplus s_c)$

5.3.2 Graph Representation

We construct the directed graph that represents the mode of operation of Hamsi$^{\oplus}$ by following the algorithmic description as follows: Let $G = (v, e)$ be the

corresponding graph where v represent the nodes and e the edges. In Hamsi$^{\oplus}$ chaining values are of size $(r+c)$ bits and they are stored in the nodes, $v \in \{0,1\}^{r+c}$. We distinguish between r and c because, r bits are XORed with the expanded message words and c bits are XORed with the feedforward of the c bits of the previous chaining value. Initially the graph contains $(0^r, IV)$. Let s_m represent the first r bits and s_c the last c bits. The edges are constructed in three steps: first by moving from (s_m, s_c) to $(s_m \oplus E(m), s_c)$ then by applying the permutation (P or P_f) and finally by XORing the last c bits by the last c-bits of the previous chaining variable; $(s_m, s_c) \rightarrow (s_m \oplus E(m), s_c) \xrightarrow{P/P_f} (t_m, t_c \oplus s_c)$. This also shows how to construct a *path* from the node (s_m, s_c) to the node $(t_m, t_c \oplus s_c)$. Note that similar to Hamsi a path is uniquely determined by the choice of the message.

The set of all nodes can be partitioned into subsets of nodes denoted by $\mathcal{N}_{(s_m,s_c)}$. This set can be obtained from the same c-bit part of a node by XORing all possible expanded message labels to the r-bit part. Those subsets are called supernodes. Initially IV is the only supernode.

Definition 5.10. *The supernode $\mathcal{N}_{(s_m,s_c)}$ obtained from the node (s_m, s_c) is $\{(s_{m'}, s_c) | s_{m'} = s_m \oplus E(m)\}$. The set of all 2^{r+c-m} supernodes is denoted by U.*

Definition 5.11. *A rooted node is a node that can be reached from the node IV, by following supernodes.*

The definitions and setting related to the distinguisher's setting is similar to Hamsi hence we will not repeat them but refer to them whenever needed.

Description of the Simulator $P[RO]$ of Hamsi$^{\oplus}$. The only difference from the simulator of $P[RO]$ of Hamsi is that we now work with supernodes.

Description of the Simulator $P_f[RO]$ of Hamsi$^{\oplus}$. The only difference from the simulator of $P_f[RO]$ of Hamsi is that we now work with supernodes.

5.3.3 Indifferentiability Proofs

The proofs of Lemma 5.2, Lemma 5.3 and Lemma 5.4 can be adapted for Hamsi$^{\oplus}$. In Lemma 5.1 the paths are constructed from supernodes to supernodes. Below, we prove Lemma 5.7 corresponding to Lemma 5.5.

Lemma 5.7. *The advantage of an adversary to distinguish between the two systems ($S[F_P, F_{P_f}]$,F_P,F_{P_f}) and (RO,$P[RO]$,$P_f[RO]$) with the responses to a sequence of $(q + q_f) < 2^{c/2}$ queries Q^1 is upper bounded by:*

$$f(q, q_f) = 1 - \frac{\prod_{i=0}^{q-1}(1 - \frac{i+1}{2^{r+c-m}}) \cdot \prod_{i=0}^{q_f-1}(1 - \frac{i}{2^r})}{\prod_{i=0}^{q-1}(1 - \frac{i}{2^{r+c}}) \cdot \prod_{i=0}^{q_f-1}(1 - \frac{i}{2^{r+c}})} \ .$$

where q and q_f denotes the number of queries to F_P and F_{P_f} respectively.

Algorithm 9 Simulator $P[RO]$ of Hamsi$^{\oplus}$

Query (s_m, s_c) from the interface F
if $(s_m, s_c) \notin S_P$ **then**
  if $\mathcal{N}_{(s_m,s_c)} \notin R)$ **then**
    choose $(t_m, t_c) \in (A \times C) \setminus T_F$ randomly and uniformly
    add (s_m, s_c) to S_F and (t_m, t_c) to T_F and $\mathcal{N}_{(s_m,s_c)}$ to O
    add an edge from $\mathcal{N}_{(s_m,s_c)}$ to $\mathcal{N}_{(t_m,t_c \oplus s_c)}$
    return (t_m, t_c)
  else if $(R \cup O \neq U)$ **then**
    choose (t_m, t_c) randomly and uniformly from the set of all nodes satisfying $(t_m, t_c) \notin T_F$ and $\mathcal{N}_{(t_m,t_c \oplus s_c)} \in U \setminus (R \cup O)$
    add $\mathcal{N}_{(t_m,t_c \oplus s_c)}$ to the set of rooted supernodes R
    add (s_m, s_c) to S_F and (t_m, t_c) to T_F and $\mathcal{N}_{(s_m,s_c)}$ to O
    add an edge from $\mathcal{N}_{(s_m,s_c)}$ to $\mathcal{N}_{(t_m,t_c \oplus s_c)}$
    return (t_m, t_c)
  else
    simulator fails
  end if
end if
else return $(t_m, t_c) \in T_F$

Query (t_m, t_c) from the interface F^{-1}
if $(t_m, t_c) \notin T_F$ **then**
  choose $(s_m, s_c) \notin S_F$ randomly and uniformly from the set of nodes satisfying $(s_m, s_c) \notin S_F$ and $\mathcal{N}_{(s_m,s_c)} \in U \setminus (R \cup O)$
  add (s_m, s_c) to S_F and (t_m, t_c) to T_F and $\mathcal{N}_{(s_m,s_c)}$ to O
  add an edge from $\mathcal{N}_{(s_m,s_c)}$ to $\mathcal{N}_{(t_m,t_c \oplus s_c)}$
  return (s_m, s_c)
end if
else return $(t_m, t_c) \in S_F$

Proof. We follow the similar steps of the proof of Lemma 5.5. We will compute:

$$Adv(A) \leq \frac{1}{2} \sum_{X} |Pr[X|F] - Pr[X|S]| \ .$$

Compute $Pr[X|F] = Pr[X|F_P] \cdot Pr[X|F_{P_f}]$.

$$Pr[X|F] = [(2^{r+c})_{(q)} \cdot (2^{r+c})_{(q_f)}]^{-1} \ .$$

Compute $Pr[X|S] = Pr[X|P[RO]] \cdot Pr[X|P_f[RO]]$.

Algorithm 10 Simulator $P_f[RO]$ of Hamsi$^{\oplus}$

1: Query (s_m, s_c) from the interface F_f
2: **if** $(s_m, s_c) \notin S_{F_f}$ **then**
3: **if** $(\mathcal{N}_{(s_m,s_c)} \notin R)$ or (path cannot be unpadded) **then**
4: choose $(t_m, t_c) \in (A \times C) \setminus T_{F_f}$ randomly and uniformly
5: add (s_m, s_c) to S_{F_f} and (t_m, t_c) to T_{F_f} and $\mathcal{N}_{(s_m,s_c)}$ to O
6: add an edge from $\mathcal{N}_{(s_m,s_c)}$ to $\mathcal{N}_{(t_m,t_c \oplus s_c)}$
7: return (t_m, t_c)
8: **else if** $(R \cup O \neq U)$ **then**
9: assign $t_c = RO(\text{path}) \oplus s_c$
10: choose $(t_m, t_c) \notin T_{F_f}$ randomly and uniformly
11: add $\mathcal{N}_{(t_m,t_c)}$ to the set of rooted nodes R
12: add (s_m, s_c) to S_{F_f} and (t_m, t_c) to T_{F_f} and $\mathcal{N}_{(s_m,s_c)}$ to O
13: add an edge from $\mathcal{N}_{(s_m,s_c)}$ to $\mathcal{N}_{(t_m,t_c \oplus s_c)}$
14: return (t_m, t_c)
15: **else**
16: simulator fails
17: **end if**
18: **end if**
19: else return $(t_m, t_c) \in T_{F_f}$

1: Query (t_m, t_c) from the interface F_f^{-1}
2: **if** $(t_m, t_c) \notin T_{P_f}$ **then**
3: choose $(s_m, s_c) \notin S_{F_f}$ randomly and uniformly from the set of nodes satisfying $(s_m, s_c) \notin S_{F_f}$ and $\mathcal{N}_{(s_m,s_c)} \in U \setminus (R \cup O)$
4: add (s_m, s_c) to S_{F_f} and (t_m, t_c) to T_{F_f} and $\mathcal{N}_{(s_m,s_c)}$ to O
5: add an edge from $\mathcal{N}_{(s_m,s_c)}$ to $\mathcal{N}_{(t_m,t_c \oplus s_c)}$
6: return (s_m, s_c)
7: **end if**
8: else return $(t_m, t_c) \in S_{F_f}$

Compute $Pr[X|P[RO]]$**:** We count the possibilities from step 9 of Algorithm 7. Since the number of elements of the sets $R \cup O$ and T_P is increased by one after each query, and since each supernode in $R \cup O$ corresponds to 2^m nodes, the number of possibilities is $(2^{c+r} - 2^m) \cdot (2^{c+r} - 2 \cdot 2^m - 1) \cdot (2^{c+r} - 3 \cdot 2^m - 2) \ldots (2^{c+r} - q \cdot (2^m + 1) - q + 1)$. Hence, assuming that $1 \ll 2^m$, we have

$$Pr[X|P[RO]] \approx \frac{1}{(2^{c+r} - 2^m) \cdot (2^{c+r} - 2 \cdot 2^m) \ldots (2^{c+r} - q \cdot 2^m)} \quad .$$

Compute $Pr[X|P_f[RO]]$**:** The number of possibilities from step 9 and step 10 of Algorithm 7 is the same as for Hamsi$^{\oplus}$. Hence, we have

$$Pr[X|P_f[RO]] \leq \frac{1}{(2^c)^{q_f} \cdot 2^r \cdot (2^r - 1) \ldots (2^r - q_f + 1)} = [(2^c)^{q_f} \cdot (2^r)_{(q_f)}]^{-1} \ .$$

Combining all probabilities we get,

$$Pr[X|S] \leq \frac{1}{(2^{c+r} - 2^m) \cdot (2^{c+r} - 2 \cdot 2^m) \ldots (2^{c+r} - q \cdot 2^m) \cdot (2^c)^{q_f} \cdot (2^r)_{(q_f)}} \ .$$

Now we can compute Adv(A) as follows:

$$\begin{aligned} Adv(A) &\leq \frac{1}{2} \sum_X |Pr[X|F] - Pr[X|S]| \\ &= 1 - \frac{(2^{c+r} - 2^m) \cdot (2^{c+r} - 2 \cdot 2^m) \ldots (2^{c+r} - q \cdot 2^m) \cdot (2^c)^{q_f} \cdot (2^r)_{(q_f)}}{(2^{r+c})_{(q)} \cdot (2^{r+c})_{(q_f)}} \\ &= 1 - \frac{\prod_{i=0}^{q-1} (1 - \frac{i+1}{2^{r+c-m}}) \cdot \prod_{i=0}^{q_f-1} (1 - \frac{i}{2^r})}{\prod_{i=0}^{q-1} (1 - \frac{i}{2^{r+c}}) \cdot \prod_{i=0}^{q_f-1} (1 - \frac{i}{2^{r+c}})} \ . \end{aligned}$$

□

Theorem 5.8. *Let $H'[F_P, F_{P_f}]$ be the Hamsi$^{\oplus}$ hash function with the padding transformation where F_P and F_{P_f} are random permutations. $H'[F_P, F_{P_f}]$ is $(t_D, t_S, q, q_f, \epsilon)$ indistinguishable from a random oracle, for any t_D,$t_S = O((q + q_f)^2)$, $q + q_f < 2^{c/2}$ and any $\epsilon > f(q, q_f)$.*

Proof. Follows the same arguments as in Theorem 5.6. □

Again we can take $q = q_f$ and if $q \ll 2^{c/2}$ then we can use $1 - x \approx \exp(-x)$ for $x \ll 1$ to simplify $f(q, q_f)$:

$$\begin{aligned} f(q, q_f) \equiv f(q) &\approx 1 - e^{-(\frac{q \cdot (q+1)}{2^{r+c-m+1}} + \frac{q \cdot (q-1)}{2^{r+1}} - 2 \cdot \frac{q \cdot (q-1)}{2^{r+c+1}})} \\ &\approx \frac{q \cdot (q+1)}{2^{r+c-m+1}} + \frac{q \cdot (q-1)}{2^{r+1}} \ . \end{aligned} \tag{5.2}$$

5.4 Impact of the Indifferentiability Bound

The effect of this result on the success probability of any generic attack on Hamsi (Hamsi$^{\oplus}$) is as follows: assume that there exists a generic attack with a success probability higher than the differentiating advantage then this would imply a method to differentiate with a higher probability which is a contradiction to our bound, hence the success probability of any generic attack should be less than the differentiating advantage. More generally, as it is proven by Andreeva et al. in [6]: the success probability of any generic attack on a construction is upper bounded by the sum of the success probability of the same attack on the random oracle and the *RO* differentiating advantage.

Note that indifferentiability bound is valid under the assumption that the underlying primitives are ideal. So any attack below this bound would have to exploit the properties of the primitives.

5.5 Conclusion

In this chapter we analyzed the security of the hash functions Hamsi and Hamsi$^{\oplus}$ in the indifferentiability framework. We have proven that Hamsi and Hamsi$^{\oplus}$ are indifferentiable from a random oracle truncated to a fixed output length, up to the indifferentiability bounds given in Equation (5.1) and in Equation (5.2), under the assumption that the underlying permutations are ideal.

The results we have obtained show that the success probability of any generic attack to Hamsi should be lower than the bound $(\frac{q^2}{2^c}) + (\frac{q\cdot(q-1)}{2^{r+1}})$, and the success probability of any generic attack to Hamsi$^{\oplus}$ should be lower than the bound $(\frac{q\cdot(q+1)}{2^{r+c-m+1}}) + (\frac{q\cdot(q-1)}{2^{r+1}})$, where q denotes the number of queries and c denotes the size of the chaining value for Hamsi, $r + c$ is the size of the chaining value for Hamsi$^{\oplus}$, r is the size of the expanded message words and m is the message block length.

The main difference between Hamsi and Hamsi$^{\oplus}$ is the length of the chaining value, namely, in Hamsi$^{\oplus}$ chaining values are $r+c$ bit, whereas in Hamsi c bit. Moreover, the expanded message words are XORed to the state in Hamsi$^{\oplus}$, whereas in Hamsi they overwrite the state. This effects the generic success probability of finding inner collisions, for example while the success probability of finding an inner collision after q queries for Hamsi is $(\frac{q^2}{2^c})$ where c is the size of the chaining value, for Hamsi$^{\oplus}$ it is $(\frac{q^2}{2^{r+c-m}})$ where $c + r$ is the size of the chaining value and m is the message block length (there are 2^m possible expanded message words which are XORed to the chaining value).

The indifferentiability bound we have obtained for Hamsi is optimal, however the indifferentiability bound of Hamsi$^{\oplus}$ is dominated by the second term $\frac{q\cdot(q-1)}{2^{r+1}}$, which indicates that it could be improved. Namely, it would be optimal as well if we had only the first term ($\frac{q\cdot(q+1)}{2^{r+c-m+1}}$). This is due to the worst case assumptions we made during our probability computations. Hence, we leave the improvement of the indifferentiability bound of Hamsi$^{\oplus}$ as an open problem.

Chapter 6

Analysis of Grain's Initialization Algorithm

In this chapter, we analyze the initialization algorithm of Grain, one of the eSTREAM Stream Cipher Project candidates which made it to the third phase of the project. We point out the existence of a sliding property in the initialization algorithm, and show that it can be used to reduce by half the cost of exhaustive key search. The results of this chapter are published in [48].

6.1 Description of Grain

Grain is a family of stream ciphers, proposed by Hell, Johansson, and Meier in 2005 [66]; it was designed to be particularly efficient and compact in hardware. Its two members, Grain v1 and Grain-128, accept 80-bit and 128-bit keys respectively. The original version of the cipher, later referred to as Grain v0, was submitted to the eSTREAM project, but contained a serious flaw, as was demonstrated by several researchers [19, 118]. As a response, the initial submission was tweaked and extended to a family of ciphers.

In the next two sections we first describe the building blocks common to all members of the Grain family. Afterwards, we will show how these blocks are instantiated for the specific ciphers Grain v1 and Grain-128.

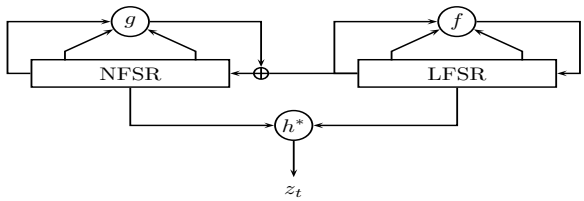


Figure 6.1: Grain during the keystream generation phase

6.1.1 Keystream Generation

All Grain members consist of three building blocks: an n-bit nonlinear feedback shift register (NFSR), an n-bit linear feedback shift register (LFSR), and a nonlinear filtering function. If we denote the content of the NFSR and the LFSR at any time t by $B_t = (b_t, b_{t+1}, \ldots, b_{t+n})$ and $S_t = (s_t, s_{t+1}, \ldots, s_{t+n})$, then the keystream generation process is defined as

$$s_{t+n} = f(S_t)\,,$$

$$b_{t+n} = g(B_t) \oplus s_t\,,$$

$$z_t = h^*(B_t, S_t)\,,$$

where g and f are the update functions of the NFSR and LFSR respectively, and h^* is the filtering function (see Figure 6.1).

6.1.2 Key and IV Initialization

The initial state of the shift registers is derived from the key and the IV by running an initialization process, which uses the same building blocks as for key stream generation, and will be the main subject of this paper. First, the key and the IV are loaded into the NFSR and LFSR respectively, and the remaining last bits of the LFSR are filled with ones. The cipher is then clocked for as many times as there are state bits. This is done in the same way as before, except that the output of the filtering function is fed back to the shift registers, as shown in Fig. 6.2 and in the equations below.

$$r_t = h^*(B_t, S_t) \oplus s_t\,,$$

$$s_{t+n} = f(r_t, s_{t+1}, \ldots, s_{t+n-1})\,,$$

$$b_{t+n} = g(B_t) \oplus r_t\,.$$

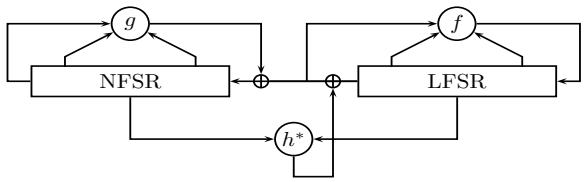


Figure 6.2: Grain during the initialization phase

6.1.3 Grain v1

Grain v1 is an 80-bit stream cipher which accepts 64-bit IVs. The NFSR and the LFSR are both 80 bits long, and therefore, as explained above, the initialization takes 160 cycles. The different functions are instantiated as follows:

$$f(S_t) = s_t \oplus s_{t+13} \oplus s_{t+23} \oplus s_{t+38} \oplus s_{t+51} \oplus s_{t+62}\,,$$

$$g(B_t) = b_t \oplus b_{t+14} \oplus b_{t+62}$$

$$\oplus\, g'(b_{t+9}, b_{t+15}, b_{t+21}, b_{t+28}, b_{t+33}, b_{t+37}, b_{t+45}, b_{t+52}, b_{t+60}, b_{t+63})\,,$$

$$h^*(B_t, S_t) = \sum_{i \in \mathcal{A}} b_{t+i} \oplus h(s_{t+3}, s_{t+25}, s_{t+46}, s_{t+64}, b_{t+63})\,,$$

with $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$, g' a function of degree 5, and h a function of degree 3. The exact definitions of these functions can be found in [66].

6.1.4 Grain-128

Grain-128 is the 128-bit member of the Grain family. The IV size is increased to 96 bits, and the shift registers are now both 128 bits long. The initialization takes 256 cycles, and the functions are defined as follows:

$$f(S_t) = s_t \oplus s_{t+7} \oplus s_{t+38} \oplus s_{t+70} \oplus s_{t+81} \oplus s_{t+96}\,,$$

$$g(B_t) = b_t \oplus b_{t+26} \oplus b_{t+56} \oplus b_{t+91} \oplus b_{t+96}$$

$$\oplus\, b_{t+3}b_{t+67} \oplus b_{t+11}b_{t+13} \oplus b_{t+17}b_{t+18}$$

$$\oplus\, b_{t+27}b_{t+59} \oplus b_{t+40}b_{t+48} \oplus b_{t+61}b_{t+65} \oplus b_{t+68}b_{t+84},$$

$$h^*(B_t, S_t) = \sum_{i \in \mathcal{A}} b_{t+i} \oplus h(s_{t+8}, s_{t+13}, s_{t+20}, s_{t+42}, s_{t+60}, s_{t+79}, s_{t+95}, b_{t+12}, b_{t+95})\,.$$

In the equations above, $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$, and h is a very sparse function of degree 3. Again, we refer to the specifications [67] for the exact definition.

6.2 Slide Attacks

In this section we discuss a first class of attacks on Grain's initialization phase, which are based on a particular sliding property of the algorithm. Slide attacks have been introduced by Biryukov and Wagner [28] in 1999, and have since then mainly been used to attack block ciphers. A rather unique property of this cryptanalysis technique is that its complexity is not affected by the number of rounds, as long as they are all (close to) identical. This will also be the case in the attacks presented below: the attacks apply regardless of how many initialization steps are performed.

Note that although we will illustrate the attacks using Grain v1, the discussion in the next sections applies to Grain-128 just as well.

6.2.1 Related $(K, \mathbf{IV})$ Pairs

The sliding property exploited in the next sections is a consequence of the similarity of the operations performed in Grain at any time t, both during initialization and key generation, as well as of the particular way in which the key and IV bits are loaded. More specifically, let us consider a secret key $K = (k_0, \ldots, k_{79})$, used in combination with an initialization vector $\mathrm{IV} = (v_0, \ldots, v_{63})$. During the first 161 cycles (160 initialization steps and 1 key generation step), the registers will contain the following values:

$$\text{init. phase}\left\{\begin{array}{rl} B_0 = (k_0, \ldots\ldots, k_{78}, k_{79}) & S_0 = (v_0, \ldots, v_{62}, \quad v_{63}, \quad 1, \ldots, 1, \quad 1) \\ B_1 = (k_1, \ldots\ldots, k_{79}, b_{80}) & S_1 = (v_1, \ldots, v_{63}, \quad 1, \quad 1, \ldots, 1, \quad s_{80}) \\ \vdots & \vdots \\ B_{160} = (b_{160}, \ldots, b_{238}, b_{239}) & S_{160} = (s_{160}, \ldots\ldots\ldots\ldots\ldots\ldots, s_{238}, s_{239}) \\ B_{161} = (b_{161}, \ldots, b_{239}, b_{240}) & S_{161} = (s_{161}, \ldots\ldots\ldots\ldots\ldots\ldots, s_{239}, s_{240}) \end{array}\right.$$

Let us now assume that $s_{80} = 1$. Note that if this is not the case, it suffices to flip any element from the set $\{v_0, v_{13}, v_{23}, v_{38}, v_{51}, v_{62}\}$ for the assumption to hold. This follows from the way s_{80} is computed in Grain v1. Namely, $s_{80} = h^*(B_t, S_t) \oplus v_0 \oplus v_{13} \oplus v_{23} \oplus v_{38} \oplus v_{51} \oplus v_{62}$. We then consider a second key $K^* = (k_1, \ldots, k_{79}, b_{80})$ together with the initialization vector $\mathrm{IV}^* = (v_1, \ldots, v_{63}, 1)$. After loading this pair into the registers, we obtain:

$$B_0^* = (k_1, \ldots\ldots, k_{79}, b_{80}) \qquad S_0^* = (v_1, \ldots, v_{63}, \quad 1, \quad 1, \ldots, 1, \quad 1)$$

This, however, is identical to the content of B_1, and since the operations during the initialization are identical as well, the equality $B_t^* = B_{t+1}$ is preserved until step 159, as shown below.

$$\text{init. phase}\left[\begin{array}{ll} B_0^* = (k_1, \ldots\ldots, k_{79}, b_{80}) & S_0^* = (v_1, \ldots, v_{63}, \quad 1, \quad 1, \ldots, 1, \quad 1) \\ \vdots & \vdots \\ B_{159}^* = (b_{160}, \ldots, b_{238}, b_{239}) & S_{159}^* = \quad (s_{160}, \ldots\ldots\ldots\ldots, s_{238}, s_{239}) \\ B_{160}^* = (b_{161}, \ldots, b_{239}, b_{239}^*) & S_{160}^* = \quad (s_{161}, \ldots\ldots\ldots\ldots, s_{239}, s_{239}^*) \end{array}\right.$$

$$\begin{array}{ll} B_{161}^* = (b_{162}, \ldots, b_{239}^*, b_{240}^*) & S_{161}^* = \quad (s_{162}, \ldots\ldots\ldots\ldots, s_{239}^*, s_{240}^*) \end{array}$$

In step 160, b_{239}^* and s_{239}^* are not necessarily equal to b_{240} and s_{240}, since the former are computed in initialization mode, whereas the latter are computed in key stream generation mode. Nevertheless, and owing to the tap positions of Grain v1, the equality will still be detectable in the first 15 keystream bits.

Moreover, if $h^*(B_{159}^*, S_{159}^*) = h^*(B_{160}, S_{160}) = 0$ (this happens with probability 1/2), then both modes of Grain are equivalent, and hence the equality is preserved in the last step as well. After this point, (B_t^*, S_t^*) and (B_{t+1}, S_{t+1}) are both updated in key stream generation mode; their values will therefore stay equal till the end, leading to identical but shifted key streams.

With an appropriate choice of IVs, similar sliding behaviors can also be observed by sliding the keys over more bit positions. In general, we have the following property for $1 \leq n \leq 16$:

Property 2. *For a fraction $2^{-2 \cdot n}$ of pairs (K, IV), there exists a related pair (K^*, IV^*) which produces an identical but n-bit shifted key stream.*

Note that the existence of different (K, IV) pairs which produce identical but shifted key streams is in itself not so uncommon in stream ciphers. When a stream cipher is iterated, its internal state typically follows a huge predefined cycle, and the role of the initialization algorithm is to assign a different starting position for each (K, IV) pair. Obviously, if the total length of the cycle(s) is smaller than the number of possible (K, IV) pairs multiplied by the maximum allowed key stream length, then some overlap between the key stream sequences generated by different (K, IV) pairs is unavoidable. This is the case in many stream ciphers, including Grain. However, what is shown by the property above, is that the initialization algorithm of Grain has the particularity that it tends to cluster different starting positions together, i.e, producing shifted keystreams, instead of distributing them evenly over the cycle(s).

6.2.2 A Related-Key Slide Attack

A first straightforward application of the property described in the previous section is a related-key attack. Suppose that the attacker somehow suspects that two (K, IV) pairs are related in the way explained earlier. In that case, he knows that the corresponding key stream sequences will be shifted over one bit with probability 1/4, and if that happens, he can conclude that $s_{80} = 1$. This allows him to derive a simple equation in the secret key bits. Note that if the (K, IV) pairs are shifted over $n > 1$ positions, then with probability $2^{-2 \cdot n}$ the attacker will be able to obtain n equations.

As is the case for all related key attacks, the simple attack just described is admittedly based on a rather strong supposition. In principle, however, one could imagine practical situations where different session keys are derived from a single master key in a funny way, making this sort of related keys more likely to occur, or where the attacker has some means to transform the keys before they are used (for example by causing synchronization errors).

6.2.3 Speeding up Exhaustive Key Search

A second application, which is definitely of more practical relevance, is to use the sliding property of the initialization algorithm to speed up exhaustive key search by a factor two.

The straightforward way to run an exhaustive search on Grain v1 is described by the pseudo-code below:

for $K = 0$ to $2^{80} - 1$ **do**
  perform 160 initialization steps;
  generate first few key stream bits $(z_0, \ldots, z_t)$;
  check if $(z_0, \ldots, z_t)$ matches given key stream;
end for

Let us now analyze the special case where the given key stream sequence was generated using an IV which equals $I = (1, \ldots, 1)$. In this case, which can easily be enforced if we assume a chosen-IV scenario, the algorithm above can be improved by exploiting the sliding property. In order to see this, it suffices to analyze the contents of the registers during the initialization:

$$
\text{init. phase}\left\{\begin{array}{rl}
B_0 = (k_0, \ldots\ldots, k_{78}, k_{79}) & S_0 = (1, \ldots, \quad 1, \quad 1, \ldots\ldots, 1 \quad 1) \\
B_1 = (k_1, \ldots\ldots, k_{79}, b_{80}) & S_1 = (1, \ldots, \quad 1, \quad 1, \ldots\ldots, 1 \quad s_{80}) \\
B_2 = (k_2, \ldots\ldots, b_{80}, b_{81}) & S_2 = (1, \ldots, \quad 1, \quad 1, \ldots, s_{80}, \quad s_{81}) \\
\vdots & \vdots \\
B_{160} = (b_{160}, \ldots, b_{238}, b_{239}) & S_{160} = \quad (s_{160}, \ldots\ldots\ldots\ldots, s_{238}, s_{239}) \\
B_{161} = (b_{161}, \ldots, b_{239}, b_{240}) & S_{161} = \quad (s_{161}, \ldots\ldots\ldots\ldots, s_{239}, s_{240})
\end{array}\right.
$$

The improvement is based on the observation that if $s_{80} = 1$, we can check two keys without having to recalculate the initialization. If $s_{81} = 1$ as well, then we can simultaneously verify three keys, and so on. In order to use this property to cover the key space in an efficient way, we need to change the order in which the keys are searched, though. This is done in the pseudo-code below:

```
K = 0;
repeat
   perform 160 initialization steps;
   generate first 16 key stream bits (z_0, ..., z_15);
   for t = 0 to [largest n < 16 for which S_n = I] do
      check if (z_t, ..., z_15) matches given key stream;
      K = B_{t+1};
   end for
until K = 0
```

This code will not run through all possible keys, but only check a cycle of keys with an expected length of 2^{79}. This, however, is done by performing only 2^{78} initializations, making it twice as fast as the standard exhaustive search algorithm. If we are unlucky, and the secret key is not found, then the algorithm can simply be repeated with a different starting key in order to cover a different cycle.

Finally note that the algorithm retains a number of useful properties of the regular exhaustive search algorithm: it can be used to attack several keys simultaneously, and it can easily be parallelized (using distinguished points to limit overlap). One limitation however, is that it can only be applied if the attacker can get hold of a keystream sequence corresponding to $\mathrm{IV} = (1, \ldots, 1)$, or of a set of keystream sequences corresponding to a number of related IVs. Depending on how the IVs are picked, the latter might be easier to obtain.

6.2.4 Avoiding the Sliding Property

As discussed earlier, the existence of related (K, IV) pairs in Grain cannot be avoided without increasing the state size. However, in order to avoid the particular sliding behavior of the initialization algorithm, one could try to act on the two

factors that lead to this property: the similarity of the computations performed at different times t, and the self-similarity of the constant loaded into the last bits of the LFSR. The similarity of computations could be destroyed by involving a counter in each step. This would effectively increase the size of the state, but one could argue that this counter needs to be stored anyway to decide when the initialization algorithm finishes. An easier modification, however, would be to eliminate the self-similarity of the initialization constant. If the last 16 bits of the LFSR would for example have been initialized with $(0, \ldots, 0, 1)$, then this would already have significantly reduced the probability of the sliding property.

6.3 Conclusion

In this chapter we have studied a sliding property in the initialization algorithm of Grain. We have shown that the sliding property can be used to reduce by half the cost of exhaustive key search. While this might not be significant for Grain-128, it could have some impact on Grain v1, given its relatively short 80-bit key. Moreover, we have shown that this attack could be avoided by making a minor change in the constant used during the initialization.

Chapter 7

Conclusion

In this chapter, we will summarize the contributions of this thesis and present some open problems of varying difficulty and of varying generality. Some open problems are a continuation of the specific topics covered in this thesis and some of them address the general open problems related to the design and analysis of cryptographic hash functions.

7.1 Contributions of this Thesis

The design and analysis of cryptographic algorithms is an active and challenging area of cryptography; *active* because modern digital society requires secure communication, *challenging* because the existence of a variety of platforms and the imposed constraints make it hard to specify generically the term "efficient". The design of cryptographic hash functions being the "Swiss army knife" of cryptography has received notable attention for the past years, especially with the start of the SHA-3 competition. The SHA-3 competition launched a variety of designs; many of them have been analyzed extensively and the ones shown to have weaknesses were eliminated during the first round.

Designing a sound cryptographic algorithm requires a considerable research activity and effort. Designs are merely influenced from existing and possible future attacks. Consequently, it requires a lot of time and effort to validate the security and the soundness of a design. There are variety of paths followed in the design approach and hot discussions arise about the implications of analysis results.

To make it simpler, we focused on a natural and a basic question that is argued in the crypto community, which is "how should a cryptographic hash function look like?". However, the simple questions are most of the time hardest to answer. We can state several properties, listed below which are not so easy to achieve simultaneously.

- simple to understand and analyze;
- efficient to implement in different platforms;
- secure;
- should be analyzed with enough attention and effort by good cryptographers.

In this thesis we attempt to design a hash function meeting those requirements. We follow a particular approach in our design. Below we discuss contributions presented in related chapters of this thesis:

- The main topic of this thesis is the design of the cryptographic hash function Hamsi. With its outside the box construction we aimed to explore new design approaches without sacrifying the security and the performance. We had the chance to test our algorithm by submitting it to the SHA-3 competition. Hamsi made it to the second round of the competition together with 13 other algorithms. It has been analyzed by many cryptographers all over the world [10, 104, 128, 13, 34, 12, 59, 121, 31, 51], implemented in different platforms [68, 60, 92, 63, 14], and compared with other algorithms. We believe that with its innovative design Hamsi has been tested with respect to many aspects. It is a contribution to the design approaches of cryptographic hash functions. We also propose a wide-pipe variant called Hamsi$^{\oplus}$. Hamsi$^{\oplus}$ increases the security margin towards generic attacks, by increasing the size of the chaining variable. To summarize, in the first part of this thesis we present two hash functions Hamsi and Hamsi$^{\oplus}$.
- In the second part of this thesis we focused on the security analysis of Hamsi. This chapter summarizes the security analysis of Hamsi by covering measures taken during the design process and external analysis results. Moreover, the main contribution of this chapter is the collision analysis of Hamsi-256. We have presented a pseudo-collision path for Hamsi-256 without valid expanded message words; so far this is the best collision analysis.
- In the third part of this thesis we analyzed Hamsi and Hamsi$^{\oplus}$ in the indifferentiability framework and established concrete bounds. We showed that assuming that the underlying primitives are ideal Hamsi and Hamsi$^{\oplus}$ are indifferentiable from a random oracle truncated to a fixed length. We have computed concrete bounds for the differentiating advantage; in the case of

Hamsi our bound is optimal. The bound for Hamsi$^{\oplus}$ is not optimal, however, the structure of the bound suggests a possibility of future improvements. We leave it as an open problem.

- The fourth part of this thesis contains the analysis of the initialization algorithm of the stream cipher Grain. In this part we showed the sliding property and exploited this fact to reduce the exhaustive key search by a factor of 2. At the time our attack was published, it was one of the few results on Grain. It exploits a simple property of the initialization algorithm and it can be fixed very easily, namely, the recently proposed version of Grain-128 with authentication benefits from our analysis [2].

7.2 Open Problems

- Investigate other design choices for Hamsi. For example, keep everything the same in Hamsi-256 but increase the message block length to 64 bits, use appropriate message expansion for this size, i.e., the linear code with parameters [128,32,47] over F_4, and increase the number of rounds to 6. In order to distinguish the two designs we will call this hash function "Sardalya" which is another fish of similar characteristics with Hamsi. Sardalya has a weaker message expansion (because of the minimum distance) but the number of rounds is increased. We would expect that by doubling both the size of the message block length and the number of rounds the performance of two designs should be approximately the same. However, many analysis results on Hamsi fail because the message block length (32 bits) does not enable enough freedom. Note that the message freedom in Sardalya is increased which may have implications on security. Hence, it would be interesting to compare the security of Hamsi versus Sardalya.
  In Section 4.2.2 we discuss implications of the upper bound on the minimum distance of the linear code used for message expansion on the security of Hamsi. This is another direction that can be followed, namely increasing the strength of the message expansion and reducing the size of the message block length and decreasing the number of rounds. Note that in this approach one should keep in mind the low algebraic degree possessed by 4×4 bit Sboxes. These possibilities indicate a more general problem of finding the best trade-off for Hamsi type of designs.

- Hamsi benefits from the best known linear codes with highest minimum distance for the message expansion. However, one drawback of this method is the size of the lookup tables used to represent the expansion suitable for fast software implementations. Especially in the case of the Hamsi-512 the size of this table is 128kB (see Section 3.7.3). The construction methods of the linear codes are given in Appendix A. The linear code used for the message

expansion of Hamsi-256 uses the method "ConstructionX" [90] to combine two quasi-cyclic codes and one cyclic code. It would be interesting to analyze the possibilities to exploit the cyclic structure and the construction method. The linear code used for Hamsi-512 is constructed by the concatenation of two codes. We leave it as open problem to exploit the construction methods of the linear codes that can yield more compact implementations of Hamsi for constrained software environments. For hardware implementations the message expansion can be implemented by a matrix multiplication over F_2. The generator matrices of size 1kB and 4kB for Hamsi-256 and Hamsi-512, used for this multiplication, are included in Appendix A. We believe that, there is also room for more compact hardware implementations of those matrices by exploiting the linear code.

- There are several studies about constructing the most efficient and secure 4×4-bit Sbox suitable for software and hardware [27, 89, 123]. However there is not much study about optimal linear layers except to our knowledge studies on T-functions [79, 80]. It seems that for hash functions based on a light compression function it is crucial to have an optimal and secure linear layer. Note that some of the analysis results on Hamsi-256 exploit the properties of the linear transformation [34, 51]. In general, it is not trivial to construct optimal diffusion layers for symmetric primitives. Following the wide trail design strategy employed in AES [42] many symmetric primitives use MDS (Maximum Distance Separable) codes in the diffusion layer, such as the SHA-3 candidates Grøstl and JH. However, construction of optimal linear layers that use the logical operations XOR, shift and rotation is an open problem. The definition of optimal depends on the structure of the design or on the properties of the other building blocks used, such as the branch number of the Sboxes or the number of rounds. In AES the branch number of the MDS code ensures that given a difference on an Sbox it takes 2 rounds to spread this difference to all Sboxes. However in the case of Hamsi the branch number of the Sbox combined with the branch number of the linear transformation enables one to construct near-colliding differential paths, i.e., starting from the middle and going forward and backward, see the result of Nikolić [104]. Moreover in the algebraic attack by Dinur and Shamir [51] the fact that "several output bits of the compression function depend only on a small number of inputs from the second round" is exploited. This suggests a more enhanced and context related definition for the term *optimal linear layer*.

- When processing short messages, hash functions with light chaining transformation may have an advantage in performance. For comparisons see eBASH [52]. It is an open problem to investigate the suitability of such hash functions for constrained environments. Moreover, the state size of the chaining transformation is an important parameter for hardware implementations, besides the implications on security. One can aim lower

security levels for preimage attacks compared to collision attacks. Namely, the state size of Hamsi-256 (512 bits) chosen such that it is optimal, however without the feedforward Hamsi would not satisfy NIST requirements for preimage and 2nd preimage resistance. Hence, one can trade the state size and security level, and this is what makes some designs more suitable for lightweight cryptography. Note that there are also other aspects that should be considered in designs aiming the lightweight cryptography. It is a general open problem to design cryptographic algorithms that will suit the needs of lightweight applications. Moreover, those solutions need not necessarily adopt the traditional tools of symmetric primitives.

- During the SHA-3 competition many candidates are analyzed extensively and as a result of those efforts many properties of the primitives of the hash functions are shown. For example, it is relatively easy to find 4-sums for the compression function of the SHA-3 final round candidate Grøstl, see the updated submission document [61]. Some of those properties are called "distinguishers", however given the fact that in a hash function everything is public, this term may be confusing and/or controversial. Hence, it is an open problem to set a theoretical approach to distinguish the properties of the compression function that should be avoided from the ones that are not relevant, in relation with the mode of operation.

  Attacks on the underlying blocks sometimes can be used to attack the hash function. Is it safer, if we try to avoid all possible weaknesses from the compression function? This may also result in the following situation; with the current know-how this is not possible since we neither have a rigorous definition of "weakness" nor a theoretical approach for cipher design. However, we can still aim to build a complex compression function such that it requires time to figure out the "non-ideal" properties, but by then we will have new technologies and new requirements and new designs.

- The new proposed hash function Hamsi$^{\oplus}$ can be described to fit the sponge construction except satisfying the requirement of the hermetic sponge strategy, that the underlying primitives should not have structural distinguishers. Hence, Hamsi$^{\oplus}$ defined as in the sponge construction can be used to output variable length messages; enabling to have one hash function for all output lengths. This is the main reason why we haven't specified 512 and 384 bit versions of Hamsi$^{\oplus}$. We leave the analysis of Hamsi$^{\oplus}$ in the context of the sponge construction as an open problem.

- In Chapter 5, we have presented the indifferentiability advantage of the hash function Hamsi$^{\oplus}$ from a random oracle truncated to a fixed output length. However, the bound we have obtained is not optimal, see Section 5.5 for a short discussion. We leave the improvement of this bound beyond the birthday bound as an open problem. We believe that the structure of the bound indicates the possibility for improvement. However, whether the

improvement can be achieved by a more careful computation of probability assumptions or by a completely different approach or by a combination of both, is left as an open problem. The recent result by Moody et al. [99] "shows that an n-bit iterative hash function can achieve both the rate 1 efficiency, and the indifferentiability security bound that is more than $n/2$ bits". It would be interesting to investigate whether it is possible to apply the techniques used in this paper to the indifferentiability proof of Hamsi$^{\oplus}$.

- Although it does not fit the contents of thesis, it would be a shame not to mention the analysis of hash functions that uses merely the operations addition, rotation and XOR (ARX), given the SHA-3 finalists that use them (BLAKE, Skein). The population of RFID protocols that is based on random and complex combination of those operations make this area even more challenging. We leave this problem to the patient and intelligent researcher.

"None of us knows anything,
not even whether we know or do not know,
nor do we know whether not knowing and knowing exist,
nor in general whether there is anything or not."

Metrodorus of Chios, Greek philosopher

Appendix A

Linear Codes Used for the Message Expansion of Hamsi

A.1 Construction of the Linear Code [128,16,70] over F_4

```
Magma V2.15-12
>
> SetVerbose("BestCode", true);
> SetPrintLevel("Minimal");
> F<w> := GF(4);
>
> C := BestKnownLinearCode(F, 130, 16);
Construction of a [ 130 , 16 , 72 ] Code:
[1]:  [4, 3, 2] Cyclic Linear Code over GF(2^2)
        Dual of the RepetitionCode of length 4
[2]:  [126, 13, 72] Quasicyclic of degree 2 Linear Code over GF(2^2)
        QuasiCyclicCode of length 126 with generating
        polynomials: w^2*x^62 + w*x^61 + w*x^60 + x^59 + w*x^58 +
        x^57 + w*x^56 + x^53 + w*x^52 + w^2*x^50 + x^47 + x^46 +
        w*x^45 + w^2*x^44 + w*x^43 + x^41 + x^40 + w*x^38 +
        w*x^37 + x^35 + w^2*x^34 + x^33 + x^31 + w*x^30 + w*x^29
        + x^28 + x^26 + x^25 + w^2*x^24 + x^23 + w*x^22 + x^21 +
        x^20 + x^18 + w^2*x^17 + w*x^16 + w*x^15 + w*x^13 + 1,
        x^62 + x^61 + x^60 + x^59 + w^2*x^58 + w^2*x^57 +
        w^2*x^54 + w^2*x^53 + w*x^50 + w^2*x^48 + w^2*x^47 + x^46
```

```
        + x^45 + w^2*x^44 + w^2*x^43 + w*x^42 + w*x^41 + w^2*x^39
        + x^38 + w*x^35 + x^34 + w*x^33 + w*x^32 + w*x^31 + x^28
        + w^2*x^26 + x^23 + x^22 + w^2*x^21 + w*x^19 + w^2*x^18 +
        x^17 + w*x^16 + x^11 + w*x^10 + w^2*x^8 + w^2*x^7 +
        w^2*x^6 + w*x^4 + w*x^3 + x^2 + w*x + w^2
[3]:  [126, 16, 70] Quasicyclic of degree 2 Linear Code over GF(2^2)
        QuasiCyclicCode of length 126 with generating
        polynomials: w^2*x^62 + x^61 + x^60 + x^59 + w^2*x^57 +
        w^2*x^56 + w*x^55 + x^54 + w^2*x^53 + w^2*x^52 + w^2*x^51
        + w*x^50 + x^49 + w*x^48 + x^46 + w^2*x^43 + w^2*x^41 +
        w*x^40 + w*x^39 + x^37 + w*x^36 + w*x^35 + w^2*x^34 +
        w*x^33 + w^2*x^31 + w*x^28 + w^2*x^25 + w^2*x^24 +
        w^2*x^23 + w^2*x^22 + w^2*x^19 + x^18 + x^17 + w*x^16 +
        1, w^2*x^62 + w^2*x^61 + w*x^60 + x^58 + x^57 + w^2*x^56
        + w*x^55 + x^53 + w*x^52 + w^2*x^51 + w*x^50 + x^48 +
        x^47 + w*x^46 + w^2*x^45 + w^2*x^44 + w^2*x^43 + w^2*x^42
        + w^2*x^41 + x^40 + w*x^39 + x^38 + x^36 + w^2*x^35 +
        w*x^34 + x^33 + w^2*x^31 + x^30 + w*x^29 + x^28 + w*x^26
        + x^25 + x^24 + w^2*x^22 + w*x^21 + w*x^20 + w^2*x^17 +
        w*x^16 + w*x^15 + w*x^14 + w^2*x^13 + w^2*x^12 + x^11 +
        w^2*x^10 + w^2*x^6 + x^3 + x^2 + x + 1
[4]:  [130, 16, 72] Linear Code over GF(2^2)
        ConstructionX using [3] [2] and [1]
> E := PunctureCode(C, {129..130});
> E;
[128, 16] Linear Code over GF(2^2)
```

A.2 Construction of the Linear Code [256,32,131] over F_4

```
Magma V2.15-12
>
> SetVerbose("BestCode", true);
> SetPrintLevel("Minimal");
> F<w> := GF(4);
>
> C1 := ShortenCode(ReedSolomonCode(63, 33), {12..31});
> C1;
[43, 11, 33] Linear Code over GF(2^6)
> C2 := BestKnownLinearCode(F, 6, 3);
Construction of a [ 6 , 3 , 4 ] Code:
```

```
[1]:  [6, 3, 4] Linear Code over GF(2^2)
        Extend the QRCode over GF( 4) of length 5
> C3 := ConcatenatedCode(C1, C2);
> C3;
[258, 33] Linear Code over GF(2^2)
> E := ShortenCode(PunctureCode(C3, 258), 33);
> E;
[256, 32] Linear Code over GF(2^2)
```

A.3 Generator Matrices

```
static const uint32_t gen[32][8] = {
  { 0x74951000, 0x5A2B467E, 0x88FD1D2B, 0x1EE68292,
    0xCBA90000, 0x90273769, 0xBBDCF407, 0xD0F4AF61 },
  { 0xCBA90000, 0x90273769, 0xBBDCF407, 0xD0F4AF61,
    0xBF3C1000, 0xCA0C7117, 0x3321E92C, 0xCE122DF3 },
  { 0xE92A2000, 0xB4578CFC, 0x11FA3A57, 0x3DC90524,
    0x97530000, 0x204F6ED3, 0x77B9E80F, 0xA1EC5EC1 },
  { 0x97530000, 0x204F6ED3, 0x77B9E80F, 0xA1EC5EC1,
    0x7E792000, 0x9418E22F, 0x6643D258, 0x9C255BE5 },
  { 0x121B4000, 0x5B17D9E8, 0x8DFACFAB, 0xCE36CC72,
    0xE6570000, 0x4BB33A25, 0x848598BA, 0x1041003E },
  { 0xE6570000, 0x4BB33A25, 0x848598BA, 0x1041003E,
    0xF44C4000, 0x10A4E3CD, 0x097F5711, 0xDE77CC4C },
  { 0xE4788000, 0x859673C1, 0xB5FB2452, 0x29CC5EDF,
    0x045F0000, 0x9C4A93C9, 0x62FC79D0, 0x731EBDC2 },
  { 0x045F0000, 0x9C4A93C9, 0x62FC79D0, 0x731EBDC2,
    0xE0278000, 0x19DCE008, 0xD7075D82, 0x5AD2E31D },
  { 0xB7A40100, 0x8A1F31D8, 0x8589D8AB, 0xE6C46464,
    0x734C0000, 0x956FA7D6, 0xA29D1297, 0x6EE56854 },
  { 0x734C0000, 0x956FA7D6, 0xA29D1297, 0x6EE56854,
    0xC4E80100, 0x1F70960E, 0x2714CA3C, 0x88210C30 },
  { 0xA7B80200, 0x1F128433, 0x60E5F9F2, 0x9E147576,
    0xEE260000, 0x124B683E, 0x80C2D68F, 0x3BF3AB2C },
  { 0xEE260000, 0x124B683E, 0x80C2D68F, 0x3BF3AB2C,
    0x499E0200, 0x0D59EC0D, 0xE0272F7D, 0xA5E7DE5A },
  { 0x8F3E0400, 0x0D9DC877, 0x6FC548E1, 0x898D2CD6,
    0x14BD0000, 0x2FBA37FF, 0x6A72E5BB, 0x247FEBE6 },
  { 0x14BD0000, 0x2FBA37FF, 0x6A72E5BB, 0x247FEBE6,
    0x9B830400, 0x2227FF88, 0x05B7AD5A, 0xADF2C730 },
  { 0xDE320800, 0x288350FE, 0x71852AC7, 0xA6BF9F96,
    0xE18B0000, 0x5459887D, 0xBF1283D3, 0x1B666A73 },
```

```
  { 0xE18B0000, 0x5459887D, 0xBF1283D3, 0x1B666A73,
    0x3FB90800, 0x7CDAD883, 0xCE97A914, 0xBDD9F5E5 },
  { 0x515C0010, 0x40F372FB, 0xFCE72602, 0x71575061,
    0x2E390000, 0x64DD6689, 0x3CD406FC, 0xB1F490BC },
  { 0x2E390000, 0x64DD6689, 0x3CD406FC, 0xB1F490BC,
    0x7F650010, 0x242E1472, 0xC03320FE, 0xC0A3C0DD },
  { 0xA2B80020, 0x81E7E5F6, 0xF9CE4C04, 0xE2AFA0C0,
    0x5C720000, 0xC9BACD12, 0x79A90DF9, 0x63E92178 },
  { 0x5C720000, 0xC9BACD12, 0x79A90DF9, 0x63E92178,
    0xFECA0020, 0x485D28E4, 0x806741FD, 0x814681B8 },
  { 0x4DCE0040, 0x3B5BEC7E, 0x36656BA8, 0x23633A05,
    0x78AB0000, 0xA0CD5A34, 0x5D5CA0F7, 0x727784CB },
  { 0x78AB0000, 0xA0CD5A34, 0x5D5CA0F7, 0x727784CB,
    0x35650040, 0x9B96B64A, 0x6B39CB5F, 0x5114BECE },
  { 0x5BD20080, 0x450F18EC, 0xC2C46C55, 0xF362B233,
    0x39A60000, 0x4AB753EB, 0xD14E094B, 0xB772B42B },
  { 0x39A60000, 0x4AB753EB, 0xD14E094B, 0xB772B42B,
    0x62740080, 0x0FB84B07, 0x138A651E, 0x44100618 },
  { 0xC04E0001, 0x33B9C010, 0xAE0EBB05, 0xB5A4C63B,
    0xC8F10000, 0x0B2DE782, 0x6BF648A4, 0x539CBDBF },
  { 0xC8F10000, 0x0B2DE782, 0x6BF648A4, 0x539CBDBF,
    0x08BF0001, 0x38942792, 0xC5F8F3A1, 0xE6387B84 },
  { 0x88230002, 0x5FE7A7B3, 0x99E585AA, 0x8D75F7F1,
    0x51AC0000, 0x25E30F14, 0x79E22A4C, 0x1298BD46 },
  { 0x51AC0000, 0x25E30F14, 0x79E22A4C, 0x1298BD46,
    0xD98F0002, 0x7A04A8A7, 0xE007AFE6, 0x9FED4AB7 },
  { 0xD0080004, 0x8C768F77, 0x9DC5B050, 0xAF4A29DA,
    0x6BA90000, 0x40EBF9AA, 0x98321C3D, 0x76ACC733 },
  { 0x6BA90000, 0x40EBF9AA, 0x98321C3D, 0x76ACC733,
    0xBBA10004, 0xCC9D76DD, 0x05F7AC6D, 0xD9E6EEE9 },
  { 0xA8AE0008, 0x2079397D, 0xFE739301, 0xB8A92831,
    0x171C0000, 0xB26E3344, 0x9E6A837E, 0x58F8485F },
  { 0x171C0000, 0xB26E3344, 0x9E6A837E, 0x58F8485F,
    0xBFB20008, 0x92170A39, 0x6019107F, 0xE051606E }
};

static const uint32_t gen512[64][16] = {
  { 0xEF0B0270, 0x3AFD0000, 0x5DAE0000, 0x69490000,
    0x9B0F3C06, 0x4405B5F9, 0x66140A51, 0x924F5D0A,
    0xC96B0030, 0xE7250000, 0x2F840000, 0x264F0000,
    0x08695BF9, 0x6DFCF137, 0x509F6984, 0x9E69AF68 },
  { 0xC96B0030, 0xE7250000, 0x2F840000, 0x264F0000,
    0x08695BF9, 0x6DFCF137, 0x509F6984, 0x9E69AF68,
    0x26600240, 0xDDD80000, 0x722A0000, 0x4F060000,
```

```
  0x936667FF, 0x29F944CE, 0x368B63D5, 0x0C26F262 },
{ 0x145A3C00, 0xB9E90000, 0x61270000, 0xF1610000,
  0xCE613D6C, 0xB0493D78, 0x47A96720, 0xE18E24C5,
  0x23671400, 0xC8B90000, 0xF4C70000, 0xFB750000,
  0x73CD2465, 0xF8A6A549, 0x02C40A3F, 0xDC24E61F },
{ 0x23671400, 0xC8B90000, 0xF4C70000, 0xFB750000,
  0x73CD2465, 0xF8A6A549, 0x02C40A3F, 0xDC24E61F,
  0x373D2800, 0x71500000, 0x95E00000, 0x0A140000,
  0xBDAC1909, 0x48EF9831, 0x456D6D1F, 0x3DAAC2DA },
{ 0x54285C00, 0xEAED0000, 0xC5D60000, 0xA1C50000,
  0xB3A26770, 0x94A5C4E1, 0x6BB0419D, 0x551B3782,
  0x9CBB1800, 0xB0D30000, 0x92510000, 0xED930000,
  0x593A4345, 0xE114D5F4, 0x430633DA, 0x78CACE29 },
{ 0x9CBB1800, 0xB0D30000, 0x92510000, 0xED930000,
  0x593A4345, 0xE114D5F4, 0x430633DA, 0x78CACE29,
  0xC8934400, 0x5A3E0000, 0x57870000, 0x4C560000,
  0xEA982435, 0x75B11115, 0x28B67247, 0x2DD1F9AB },
{ 0x29449C00, 0x64E70000, 0xF24B0000, 0xC2F30000,
  0x0EDE4E8F, 0x56C23745, 0xF3E04259, 0x8D0D9EC4,
  0x466D0C00, 0x08620000, 0xDD5D0000, 0xBADD0000,
  0x6A927942, 0x441F2B93, 0x218ACE6F, 0xBF2C0BE2 },
{ 0x466D0C00, 0x08620000, 0xDD5D0000, 0xBADD0000,
  0x6A927942, 0x441F2B93, 0x218ACE6F, 0xBF2C0BE2,
  0x6F299000, 0x6C850000, 0x2F160000, 0x782E0000,
  0x644C37CD, 0x12DD1CD6, 0xD26A8C36, 0x32219526 },
{ 0xF6800005, 0x3443C000, 0x24070000, 0x8F3D0000,
  0x21373BFB, 0x0AB8D5AE, 0xCDC58B19, 0xD795BA31,
  0xA67F0001, 0x71378000, 0x19FC0000, 0x96DB0000,
  0x3A8B6DFD, 0xEBCAAEF3, 0x2C6D478F, 0xAC8E6C88 },
{ 0xA67F0001, 0x71378000, 0x19FC0000, 0x96DB0000,
  0x3A8B6DFD, 0xEBCAAEF3, 0x2C6D478F, 0xAC8E6C88,
  0x50FF0004, 0x45744000, 0x3DFB0000, 0x19E60000,
  0x1BBC5606, 0xE1727B5D, 0xE1A8CC96, 0x7B1BD6B9 },
{ 0xF7750009, 0xCF3CC000, 0xC3D60000, 0x04920000,
  0x029519A9, 0xF8E836BA, 0x7A87F14E, 0x9E16981A,
  0xD46A0000, 0x8DC8C000, 0xA5AF0000, 0x4A290000,
  0xFC4E427A, 0xC9B4866C, 0x98369604, 0xF746C320 },
{ 0xD46A0000, 0x8DC8C000, 0xA5AF0000, 0x4A290000,
  0xFC4E427A, 0xC9B4866C, 0x98369604, 0xF746C320,
  0x231F0009, 0x42F40000, 0x66790000, 0x4EBB0000,
  0xFEDB5BD3, 0x315CB0D6, 0xE2B1674A, 0x69505B3A },
{ 0x774400F0, 0xF15A0000, 0xF5B20000, 0x34140000,
  0x89377E8C, 0x5A8BEC25, 0x0BC3CD1E, 0xCF3775CB,
  0xF46C0050, 0x96180000, 0x14A50000, 0x031F0000,
```

```
  0x42947EB8, 0x66BF7E19, 0x9CA470D2, 0x8A341574 },
{ 0xF46C0050, 0x96180000, 0x14A50000, 0x031F0000,
  0x42947EB8, 0x66BF7E19, 0x9CA470D2, 0x8A341574,
  0x832800A0, 0x67420000, 0xE1170000, 0x370B0000,
  0xCBA30034, 0x3C34923C, 0x9767BDCC, 0x450360BF },
{ 0xE8870170, 0x9D720000, 0x12DB0000, 0xD4220000,
  0xF2886B27, 0xA921E543, 0x4EF8B518, 0x618813B1,
  0xB4370060, 0x0C4C0000, 0x56C20000, 0x5CAE0000,
  0x94541F3F, 0x3B3EF825, 0x1B365F3D, 0xF3D45758 },
{ 0xB4370060, 0x0C4C0000, 0x56C20000, 0x5CAE0000,
  0x94541F3F, 0x3B3EF825, 0x1B365F3D, 0xF3D45758,
  0x5CB00110, 0x913E0000, 0x44190000, 0x888C0000,
  0x66DC7418, 0x921F1D66, 0x55CEEA25, 0x925C44E9 },
{ 0x0C720000, 0x49E50F00, 0x42790000, 0x5CEA0000,
  0x33AA301A, 0x15822514, 0x95A34B7B, 0xB44B0090,
  0xFE220000, 0xA7580500, 0x25D10000, 0xF7600000,
  0x893178DA, 0x1FD4F860, 0x4ED0A315, 0xA123FF9F },
{ 0xFE220000, 0xA7580500, 0x25D10000, 0xF7600000,
  0x893178DA, 0x1FD4F860, 0x4ED0A315, 0xA123FF9F,
  0xF2500000, 0xEEBD0A00, 0x67A80000, 0xAB8A0000,
  0xBA9B48C0, 0x0A56DD74, 0xDB73E86E, 0x1568FF0F },
{ 0x45180000, 0xA5B51700, 0xF96A0000, 0x3B480000,
  0x1ECC142C, 0x231395D6, 0x16BCA6B0, 0xDF33F4DF,
  0xB83D0000, 0x16710600, 0x379A0000, 0xF5B10000,
  0x228161AC, 0xAE48F145, 0x66241616, 0xC5C1EB3E },
{ 0xB83D0000, 0x16710600, 0x379A0000, 0xF5B10000,
  0x228161AC, 0xAE48F145, 0x66241616, 0xC5C1EB3E,
  0xFD250000, 0xB3C41100, 0xCEF00000, 0xCEF90000,
  0x3C4D7580, 0x8D5B6493, 0x7098B0A6, 0x1AF21FE1 },
{ 0x75A40000, 0xC28B2700, 0x94A40000, 0x90F50000,
  0xFB7857E0, 0x49CE0BAE, 0x1767C483, 0xAEDF667E,
  0xD1660000, 0x1BBC0300, 0x9EEC0000, 0xF6940000,
  0x03024527, 0xCF70FCF2, 0xB4431B17, 0x857F3C2B },
{ 0xD1660000, 0x1BBC0300, 0x9EEC0000, 0xF6940000,
  0x03024527, 0xCF70FCF2, 0xB4431B17, 0x857F3C2B,
  0xA4C20000, 0xD9372400, 0x0A480000, 0x66610000,
  0xF87A12C7, 0x86BEF75C, 0xA324DF94, 0x2BA05A55 },
{ 0x75C90003, 0x0E10C000, 0xD1200000, 0xBAEA0000,
  0x8BC42F3E, 0x8758B757, 0xBB28761D, 0x00B72E2B,
  0xEECF0001, 0x6F564000, 0xF33E0000, 0xA79E0000,
  0xBDB57219, 0xB711EBC5, 0x4A3B40BA, 0xFEABF254 },
{ 0xEECF0001, 0x6F564000, 0xF33E0000, 0xA79E0000,
  0xBDB57219, 0xB711EBC5, 0x4A3B40BA, 0xFEABF254,
  0x9B060002, 0x61468000, 0x221E0000, 0x1D740000,
```

```
    0x36715D27, 0x30495C92, 0xF11336A7, 0xFE1CDC7F },
  { 0x86790000, 0x3F390002, 0xE19AE000, 0x98560000,
    0x9565670E, 0x4E88C8EA, 0xD3DD4944, 0x161DDAB9,
    0x30B70000, 0xE5D00000, 0xF4F46000, 0x42C40000,
    0x63B83D6A, 0x78BA9460, 0x21AFA1EA, 0xB0A51834 },
  { 0x30B70000, 0xE5D00000, 0xF4F46000, 0x42C40000,
    0x63B83D6A, 0x78BA9460, 0x21AFA1EA, 0xB0A51834,
    0xB6CE0000, 0xDAE90002, 0x156E8000, 0xDA920000,
    0xF6DD5A64, 0x36325C8A, 0xF272E8AE, 0xA6B8C28D },
  { 0x14190000, 0x23CA003C, 0x50DF0000, 0x44B60000,
    0x1B6C67B0, 0x3CF3AC75, 0x61E610B0, 0xDBCADB80,
    0xE3430000, 0x3A4E0014, 0xF2C60000, 0xAA4E0000,
    0xDB1E42A6, 0x256BBE15, 0x123DB156, 0x3A4E99D7 },
  { 0xE3430000, 0x3A4E0014, 0xF2C60000, 0xAA4E0000,
    0xDB1E42A6, 0x256BBE15, 0x123DB156, 0x3A4E99D7,
    0xF75A0000, 0x19840028, 0xA2190000, 0xEEF80000,
    0xC0722516, 0x19981260, 0x73DBA1E6, 0xE1844257 },
  { 0x54500000, 0x0671005C, 0x25AE0000, 0x6A1E0000,
    0x2EA54EDF, 0x664E8512, 0xBFBA18C3, 0x7E715D17,
    0xBC8D0000, 0xFC3B0018, 0x19830000, 0xD10B0000,
    0xAE1878C4, 0x42A69856, 0x0012DA37, 0x2C3B504E },
  { 0xBC8D0000, 0xFC3B0018, 0x19830000, 0xD10B0000,
    0xAE1878C4, 0x42A69856, 0x0012DA37, 0x2C3B504E,
    0xE8DD0000, 0xFA4A0044, 0x3C2D0000, 0xBB150000,
    0x80BD361B, 0x24E81D44, 0xBFA8C2F4, 0x524A0D59 },
  { 0x69510000, 0xD4E1009C, 0xC3230000, 0xAC2F0000,
    0xE4950BAE, 0xCEA415DC, 0x87EC287C, 0xBCE1A3CE,
    0xC6730000, 0xAF8D000C, 0xA4C10000, 0x218D0000,
    0x23111587, 0x7913512F, 0x1D28AC88, 0x378DD173 },
  { 0xC6730000, 0xAF8D000C, 0xA4C10000, 0x218D0000,
    0x23111587, 0x7913512F, 0x1D28AC88, 0x378DD173,
    0xAF220000, 0x7B6C0090, 0x67E20000, 0x8DA20000,
    0xC7841E29, 0xB7B744F3, 0x9AC484F4, 0x8B6C72BD },
  { 0xCC140000, 0xA5630000, 0x5AB90780, 0x3B500000,
    0x4BD013FF, 0x879B3418, 0x694348C1, 0xCA5A87FE,
    0x819E0000, 0xEC570000, 0x66320280, 0x95F30000,
    0x5DA92802, 0x48F43CBC, 0xE65AA22D, 0x8E67B7FA },
  { 0x819E0000, 0xEC570000, 0x66320280, 0x95F30000,
    0x5DA92802, 0x48F43CBC, 0xE65AA22D, 0x8E67B7FA,
    0x4D8A0000, 0x49340000, 0x3C8B0500, 0xAEA30000,
    0x16793BFD, 0xCF6F08A4, 0x8F19EAEC, 0x443D3004 },
  { 0x78230000, 0x12FC0000, 0xA93A0B80, 0x90A50000,
    0x713E2879, 0x7EE98924, 0xF08CA062, 0x636F8BAB,
    0x02AF0000, 0xB7280000, 0xBA1C0300, 0x56980000,
```

```
    0xBA8D45D3, 0x8048C667, 0xA95C149A, 0xF4F6EA7B },
  { 0x02AF0000, 0xB7280000, 0xBA1C0300, 0x56980000,
    0xBA8D45D3, 0x8048C667, 0xA95C149A, 0xF4F6EA7B,
    0x7A8C0000, 0xA5D40000, 0x13260880, 0xC63D0000,
    0xCBB36DAA, 0xFEA14F43, 0x59D0B4F8, 0x979961D0 },
  { 0xAC480000, 0x1BA60000, 0x45FB1380, 0x03430000,
    0x5A85316A, 0x1FB250B6, 0xFE72C7FE, 0x91E478F6,
    0x1E4E0000, 0xDECF0000, 0x6DF80180, 0x77240000,
    0xEC47079E, 0xF4A0694E, 0xCDA31812, 0x98AA496E },
  { 0x1E4E0000, 0xDECF0000, 0x6DF80180, 0x77240000,
    0xEC47079E, 0xF4A0694E, 0xCDA31812, 0x98AA496E,
    0xB2060000, 0xC5690000, 0x28031200, 0x74670000,
    0xB6C236F4, 0xEB1239F8, 0x33D1DFEC, 0x094E3198 },
  { 0xAEC30000, 0x9C4F0001, 0x79D1E000, 0x2C150000,
    0x45CC75B3, 0x6650B736, 0xAB92F78F, 0xA312567B,
    0xDB250000, 0x09290000, 0x49AAC000, 0x81E10000,
    0xCAFE6B59, 0x42793431, 0x43566B76, 0xE86CBA2E },
  { 0xDB250000, 0x09290000, 0x49AAC000, 0x81E10000,
    0xCAFE6B59, 0x42793431, 0x43566B76, 0xE86CBA2E,
    0x75E60000, 0x95660001, 0x307B2000, 0xADF40000,
    0x8F321EEA, 0x24298307, 0xE8C49CF9, 0x4B7EEC55 },
  { 0x58430000, 0x807E0000, 0x78330001, 0xC66B3800,
    0xE7375CDC, 0x79AD3FDD, 0xAC73FE6F, 0x3A4479B1,
    0x1D5A0000, 0x2B720000, 0x488D0000, 0xAF611800,
    0x25CB2EC5, 0xC879BFD0, 0x81A20429, 0x1E7536A6 },
  { 0x1D5A0000, 0x2B720000, 0x488D0000, 0xAF611800,
    0x25CB2EC5, 0xC879BFD0, 0x81A20429, 0x1E7536A6,
    0x45190000, 0xAB0C0000, 0x30BE0001, 0x690A2000,
    0xC2FC7219, 0xB1D4800D, 0x2DD1FA46, 0x24314F17 },
  { 0xA53B0000, 0x14260000, 0x4E30001E, 0x7CAE0000,
    0x8F9E0DD5, 0x78DFAA3D, 0xF73168D8, 0x0B1B4946,
    0x07ED0000, 0xB2500000, 0x8774000A, 0x970D0000,
    0x437223AE, 0x48C76EA4, 0xF4786222, 0x9075B1CE },
  { 0x07ED0000, 0xB2500000, 0x8774000A, 0x970D0000,
    0x437223AE, 0x48C76EA4, 0xF4786222, 0x9075B1CE,
    0xA2D60000, 0xA6760000, 0xC9440014, 0xEBA30000,
    0xCCEC2E7B, 0x3018C499, 0x03490AFA, 0x9B6EF888 },
  { 0x88980000, 0x1F940000, 0x7FCF002E, 0xFB4E0000,
    0xF158079A, 0x61AE9167, 0xA895706C, 0xE6107494,
    0x0BC20000, 0xDB630000, 0x7E88000C, 0x15860000,
    0x91FD48F3, 0x7581BB43, 0xF460449E, 0xD8B61463 },
  { 0x0BC20000, 0xDB630000, 0x7E88000C, 0x15860000,
    0x91FD48F3, 0x7581BB43, 0xF460449E, 0xD8B61463,
    0x835A0000, 0xC4F70000, 0x01470022, 0xEEC80000,
```

```
    0x60A54F69, 0x142F2A24, 0x5CF534F2, 0x3EA660F7 },
  { 0x52500000, 0x29540000, 0x6A61004E, 0xF0FF0000,
    0x9A317EEC, 0x452341CE, 0xCF568FE5, 0x5303130F,
    0x538D0000, 0xA9FC0000, 0x9EF70006, 0x56FF0000,
    0x0AE4004E, 0x92C5CDF9, 0xA9444018, 0x7F975691 },
  { 0x538D0000, 0xA9FC0000, 0x9EF70006, 0x56FF0000,
    0x0AE4004E, 0x92C5CDF9, 0xA9444018, 0x7F975691,
    0x01DD0000, 0x80A80000, 0xF4960048, 0xA6000000,
    0x90D57EA2, 0xD7E68C37, 0x6612CFFD, 0x2C94459E },
  { 0xE6280000, 0x4C4B0000, 0xA8550000, 0xD3D002E0,
    0xD86130B8, 0x98A7B0DA, 0x289506B4, 0xD75A4897,
    0xF0C50000, 0x59230000, 0x45820000, 0xE18D00C0,
    0x3B6D0631, 0xC2ED5699, 0xCBE0FE1C, 0x56A7B19F },
  { 0xF0C50000, 0x59230000, 0x45820000, 0xE18D00C0,
    0x3B6D0631, 0xC2ED5699, 0xCBE0FE1C, 0x56A7B19F,
    0x16ED0000, 0x15680000, 0xEDD70000, 0x325D0220,
    0xE30C3689, 0x5A4AE643, 0xE375F8A8, 0x81FDF908 },
  { 0xB4310000, 0x77330000, 0xB15D0000, 0x7FD004E0,
    0x78A26138, 0xD116C35D, 0xD256D489, 0x4E6F74DE,
    0xE3060000, 0xBDC10000, 0x87130000, 0xBFF20060,
    0x2EBA0A1A, 0x8DB53751, 0x73C5AB06, 0x5BD61539 },
  { 0xE3060000, 0xBDC10000, 0x87130000, 0xBFF20060,
    0x2EBA0A1A, 0x8DB53751, 0x73C5AB06, 0x5BD61539,
    0x57370000, 0xCAF20000, 0x364E0000, 0xC0220480,
    0x56186B22, 0x5CA3F40C, 0xA1937F8F, 0x15B961E7 },
  { 0x02F20000, 0xA2810000, 0x873F0000, 0xE36C7800,
    0x1E1D74EF, 0x073D2BD6, 0xC4C23237, 0x7F32259E,
    0xBADD0000, 0x13AD0000, 0xB7E70000, 0xF7282800,
    0xDF45144D, 0x361AC33A, 0xEA5A8D14, 0x2A2C18F0 },
  { 0xBADD0000, 0x13AD0000, 0xB7E70000, 0xF7282800,
    0xDF45144D, 0x361AC33A, 0xEA5A8D14, 0x2A2C18F0,
    0xB82F0000, 0xB12C0000, 0x30D80000, 0x14445000,
    0xC15860A2, 0x3127E8EC, 0x2E98BF23, 0x551E3D6E },
  { 0x1E6C0000, 0xC4420000, 0x8A2E0000, 0xBCB6B800,
    0x2C4413B6, 0x8BFDD3DA, 0x6A0C1BC8, 0xB99DC2EB,
    0x92560000, 0x1EDA0000, 0xEA510000, 0xE8B13000,
    0xA93556A5, 0xEBFB6199, 0xB15C2254, 0x33C5244F },
  { 0x92560000, 0x1EDA0000, 0xEA510000, 0xE8B13000,
    0xA93556A5, 0xEBFB6199, 0xB15C2254, 0x33C5244F,
    0x8C3A0000, 0xDA980000, 0x607F0000, 0x54078800,
    0x85714513, 0x6006B243, 0xDB50399C, 0x8A58E6A4 },
  { 0x033D0000, 0x08B30000, 0xF33A0000, 0x3AC20007,
    0x51298A50, 0x6B6E661F, 0x0EA5CFE3, 0xE6DA7FFE,
    0xA8DA0000, 0x96BE0000, 0x5C1D0000, 0x07DA0002,
```

```
    0x7D669583, 0x1F98708A, 0xBB668808, 0xDA878000 },
  { 0xA8DA0000, 0x96BE0000, 0x5C1D0000, 0x07DA0002,
    0x7D669583, 0x1F98708A, 0xBB668808, 0xDA878000,
    0xABE70000, 0x9E0D0000, 0xAF270000, 0x3D180005,
    0x2C4F1FD3, 0x74F61695, 0xB5C347EB, 0x3C5DFFFE },
  { 0x01930000, 0xE7820000, 0xEDFB0000, 0xCF0C000B,
    0x8DD08D58, 0xBCA3B42E, 0x063661E1, 0x536F9E7B,
    0x92280000, 0xDC850000, 0x57FA0000, 0x56DC0003,
    0xBAE92316, 0x5AEFA30C, 0x90CEF752, 0x7B1675D7 },
  { 0x92280000, 0xDC850000, 0x57FA0000, 0x56DC0003,
    0xBAE92316, 0x5AEFA30C, 0x90CEF752, 0x7B1675D7,
    0x93BB0000, 0x3B070000, 0xBA010000, 0x99D00008,
    0x3739AE4E, 0xE64C1722, 0x96F896B3, 0x2879EBAC },
  { 0x5FA80000, 0x56030000, 0x43AE0000, 0x64F30013,
    0x257E86BF, 0x1311944E, 0x541E95BF, 0x8EA4DB69,
    0x00440000, 0x7F480000, 0xDA7C0000, 0x2A230001,
    0x3BADC9CC, 0xA9B69C87, 0x030A9E60, 0xBE0A679E },
  { 0x00440000, 0x7F480000, 0xDA7C0000, 0x2A230001,
    0x3BADC9CC, 0xA9B69C87, 0x030A9E60, 0xBE0A679E,
    0x5FEC0000, 0x294B0000, 0x99D20000, 0x4ED00012,
    0x1ED34F73, 0xBAA708C9, 0x57140BDF, 0x30AEBCF7 },
  { 0xEE930000, 0xD6070000, 0x92C10000, 0x2B9801E0,
    0x9451287C, 0x3B6CFB57, 0x45312374, 0x201F6A64,
    0x7B280000, 0x57420000, 0xA9E50000, 0x634300A0,
    0x9EDB442F, 0x6D9995BB, 0x27F83B03, 0xC7FF60F0 },
  { 0x7B280000, 0x57420000, 0xA9E50000, 0x634300A0,
    0x9EDB442F, 0x6D9995BB, 0x27F83B03, 0xC7FF60F0,
    0x95BB0000, 0x81450000, 0x3B240000, 0x48DB0140,
    0x0A8A6C53, 0x56F56EEC, 0x62C91877, 0xE7E00A94 },
};
```

"Science is facts; just as houses are made of stone,
so is science made of facts; but a pile of stones is not a house,
and a collection of facts is not necessarily science."

Jules Henri Poincaré, French mathematician

Bibliography

[1] From Wikipedia (View original Wikipedia Article) Last modified on 25 December 2010 at 19: 11.

[2] M. Ågren, M. Hell, T. Johansson, and W. Meier. Grain-128a: a new version of Grain-128 with optional authentication. *IJWMC*, 5(1):48–59, 2011.

[3] R. Anderson, E. Biham, and L. Knudsen. Serpent: A proposal for the advanced encryption standard. *NIST AES Proposal*, 1998.

[4] E. Andreeva, B. Mennink, and B. Preneel. Security properties of domain extenders for cryptographic hash functions. *Journal of Information Processing Systems 6 (4)*, pages 453–480, 2010.

[5] E. Andreeva, B. Mennink, and B. Preneel. The parazoa family: generalizing the sponge hash functions. Cryptology ePrint Archive: Report 2011/028, 2011. `http://eprint.iacr.org/`.

[6] E. Andreeva, B. Mennink, and B. Preneel. Security reductions of the second round SHA-3 candidates. In M. Burmester, G. Tsudik, S. S. Magliveras, and I. Ilic, editors, *ISC*, volume 6531 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2011.

[7] E. Andreeva, G. Neven, B. Preneel, and T. Shrimpton. Seven-property-preserving iterated hashing: ROX. In K. Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 130–146. Springer, 2007.

[8] Y. Arbitman, G. Dogon, V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFTX: A Proposal for the SHA-3 Standard. *Submission to NIST*, 2008.

[9] D. Augot, M. Finiasz, P. Gaborit, S. Manuel, and N. Sendrier. SHA-3 proposal: FSB. *Submission to NIST*, 2008.

[10] J. Aumasson. On the pseudorandomness of Hamsi. *NIST mailing list*, 2009.

[11] J. Aumasson, O. Dunkelman, S. Indesteege, and B. Preneel. Cryptanalysis of Dynamic SHA (2). In M. J. Jacobson, V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2009.

[12] J. Aumasson, E. Käsper, L. Knudsen, K. Matusiewicz, R. Ødegård, T. Peyrin, and M. Schläffer. Distinguishers for the compression function and output transformation of Hamsi-256. In R. Steinfeld and P. Hawkes, editors, *Information Security and Privacy - 15th Australasian Conference, ACISP 2010, Sydney, Australia, July 5-7, 2010. Proceedings*, volume 6168 of *Lecture Notes in Computer Science*, pages 87–103. Springer, 2010.

[13] J.-P. Aumasson and W. Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. *NIST mailing list*, 2009.

[14] B. Baldwin, A. Byrne, L. Lu, M. Hamilton, N. Hanley, M. O'Neill, and W. Marnane. FPGA Implementations of the Round Two SHA-3 Candidates. In *2010 International Conference on Field Programmable Logic and Applications*, pages 400–407. IEEE, 2010.

[15] M. Bellare and T. Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In X. Lai and K. Chen, editors, *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2006.

[16] M. Bellare and P. Rogaway. *Modern Cryptography*. Available from `http://cseweb.ucsd.edu/~mihir/cse207/`.

[17] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *CCS'93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5*, pages 62–73. ACM, 1993.

[18] K. Bentahar, D. Page, J. Silverman, M. Saarinen, and N. Smart. Lash. *NIST–Second Cryptographic Hash Workshop*, 2006.

[19] C. Berbain, H. Gilbert, and A. Maximov. Cryptanalysis of Grain. In M. J. B. Robshaw, editor, *Fast Software Encryption, FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 15–29. Springer-Verlag, 2006.

[20] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Radiogatun, a belt-and-mill hash function. In *Second Cryptographic Hash Workshop, Santa Barbara, August*, pages 24–25, 2006.

[21] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. In *ECRYPT hash workshop*, 2007.

[22] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, 4965:181–197, 2008.

[23] E. Biham, A. Biryukov, and A. Shamir. Miss in the Middle Attacks on IDEA and Khufu. In L. R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 1999.

[24] E. Biham and O. Dunkelman. A Framework for Iterative Hash Functions: HAIFA. In *second NIST hash workshop*, volume 4, page 138, 2006.

[25] E. Biham and A. Shamir. *Differential cryptanalysis of the Data Encryption Standard*, volume 28. Springer-Verlag, 1993.

[26] E. Biham and A. Shamir. Differential cryptanalysis of the full 16-round DES. In E. F. Brickell, editor, *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, 1993.

[27] A. Biryukov, C. De Canniére, A. Braeken, and B. Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In E. Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer, 2003.

[28] A. Biryukov and D. Wagner. Slide attacks. In L. R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.

[29] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002,*

Proceedings, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.

[30] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system I. The user language. Computational algebra and number theory (London 1993). *J. Symbolic Comput*, 24(3-4):235–265, 1997.

[31] C. Boura and A. Canteaut. Zero-sum distinguishers for iterated permutations and application to KECCAK-f and Hamsi-256. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2011.

[32] C. Boura, A. Canteaut, and C. D. Canniére. Higher-order differential properties of Keccak and Luffa. Cryptology ePrint Archive, Report 2010/589, 2010. `http://eprint.iacr.org/2010/589.pdf`.

[33] D. Brown, A. Antipa, M. Campagna, and R. Struik. ECOH: the Elliptic Curve Only Hash. *Submission to NIST*, 2008.

[34] Ç. Çalık and M. Turan. Message recovery and pseudo-preimage attacks on the compression function of Hamsi-256. In M. Abdalla and P. S. L. M. Barreto, editors, *Progress in Cryptology - LATINCRYPT 2010, First International Conference on Cryptology and Information Security in Latin America, Puebla, Mexico, August 8-11, 2010, Proceedings*, volume 6212 of *Lecture Notes in Computer Science*, pages 205–221. Springer, 2010.

[35] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited (preliminary version). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 209–218. ACM, 1998.

[36] R. Canetti, O. Goldreich, and S. Halevi. On the random-oracle methodology as applied to length-restricted signature schemes. In M. Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 40–57. Springer, 2004.

[37] F. Chabaud and A. Joux. Differential collisions in SHA-0. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.

[38] S. Contini, A. Lenstra, and R. Steinfeld. VSH, an efficient and provable collision-resistant hash function. In S. Vaudenay, editor, *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, volume 4004 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 2006.

[39] J. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to Construct a Hash Function. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005.

[40] Created by MinT, Dept. of Mathematics, University of Salzburg. Best known linear [256,32,d]-codes in base 4. `http://mint.sbg.ac.at/`.

[41] J. Daemen. Strategies based on linear and differential cryptanalysis. *Doct Dissertation Katholieke Universiteit Leuven*, 1995.

[42] J. Daemen and V. Rijmen. *The design of Rijndael: AES–the Advanced Encryption Standard*. Springer Verlag, 2002.

[43] J. Daemen and V. Rijmen. Plateau characteristics. *Information Security, IET*, 1(1):11–17, 2007.

[44] W. Dai. Official comment: LUX. NIST mailing list (local link), 2009.

[45] I. Damgård. Collision free hash functions and public key signature schemes. In D. Chaum and W. L. Price, editors, *Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings*, volume 304 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 1988.

[46] I. Damgård. A design principle for hash functions. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1990.

[47] C. De Cannière and Ö. Küçük. Some Ongoing Work on Hamsi. In *Early Symmetric Crypto (ESC) Seminar, 11-15 January 2010, Luxembourg.*

[48] C. De Cannière, Ö. Küçük, and B. Preneel. Analysis of Grain's initialization algorithm. In S. Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, volume 5023 of *Lecture Notes in Computer Science*, pages 276–289. Springer, 2008.

[49] R. Dean and A. Appel. Formal aspects of mobile code security. *Princeton University, Princeton, NJ*, 1999.

[50] B. Den Boer and A. Bosselaers. An attack on the last two rounds of MD4. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 194–203. Springer, 1992.

[51] I. Dinur and A. Shamir. An improved algebraic attack on Hamsi-256. In A. Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 88–106. Springer, 2011.

[52] D.J. Bernstein and T.Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems. `http://bench.cr.yp.to`.

[53] H. Dobbertin. Cryptanalysis of MD4. In D. Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 53–69. Springer, 1996.

[54] H. Dobbertin. RIPEMD with two-round compress function is not collision-free. *Journal of Cryptology*, 10(1):51–70, 1997.

[55] Y. Dodis, R. Oliveira, and K. Pietrzak. On the generic insecurity of the full domain hash. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 449–466. Springer, 2005.

[56] J. Evertse and E. Heyst. Which new RSA-signatures can be computed from certain given RSA-signatures? *Journal of Cryptology*, 5(1):41–52, 1992.

[57] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[58] S. Fluhrer. Collision Attack on the Waterfall Hash Function. Cryptology ePrint Archive, Report 2008/531, 2008. `http://eprint.iacr.org/`.

[59] T. Fuhr. Finding Second Preimages of Short Messages for Hamsi-256. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and*

Information Security, Singapore, December 5-9, 2010. Proceedings, volume 6477 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2010.

[60] K. Gaj, E. Homsirikamol, and M. Rogawski. Fair and comprehensive methodology for comparing hardware performance of fourteen round two SHA-3 candidates using FPGAs. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 264–278. Springer, 2010.

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

[62] M. Grassi. Bounds on the minimum distance of linear codes. `http://codetables.de/BKLC`.

[63] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont. Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations. In *The Second SHA-3 Candidate Conference*, August 2010.

[64] S. Halevi, W. Hall, and C. Jutla. The hash function Fugue. *Submission to NIST*, 2008.

[65] E. Heilman. Attacks Against Permute-Transform-Xor Compression Functions and Spectral Hash. Cryptology ePrint Archive, Report 2009/415, 2009. `http://eprint.iacr.org/`.

[66] M. Hell, T. Johansson, and W. Meier. Grain – A Stream Cipher for Constrained Environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/010, Apr. 2005. `http://www.ecrypt.eu.org/stream`.

[67] M. Hell, T. Johansson, and W. Meier. A Stream Cipher Proposal: Grain-128. eSTREAM, ECRYPT Stream Cipher Project, Apr. 2006. `http://www.ecrypt.eu.org/stream`.

[68] L. Henzen, P. Gendotti, P. Guillet, E. Pargaetzi, M. Zoller, and F. Gürkaynak. Developing a hardware evaluation method for SHA-3 candidates. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2010.

[69] E. Homsirikamol, M. Rogawski, and K. Gaj. Comparing hardware performance of fourteen round two SHA-3 candidates using FPGAs. Cryptology ePrint Archive, Report 2010/445, 2010. `http://eprint.iacr.org/`.

[70] S. Indesteege. Practical Collisions for Dynamic SHA. 2009. `http://homes.esat.kuleuven.be/~sindeste/dsha.html`.

[71] S. Indesteege, F. Mendel, B. Preneel, and M. Schläffer. Practical Collisions for SHAMATA-256. In M. J. J. Jr., V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2009.

[72] S. Indesteege and B. Preneel. Practical collisions for EnRUPT. *Journal of Cryptology*, 24(1):1–23, 2011.

[73] A. Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In M. K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

[74] J. Kelsey and T. Kohno. Herding hash functions and the nostradamus attack. In S. Vaudenay, editor, *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*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.

[75] J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005.

[76] D. Khovratovich and I. Nikolić. Rotational cryptanalysis of ARX. In S. Hong and T. Iwata, editors, *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *Lecture Notes in Computer Science*, pages 333–346. Springer, 2010.

[77] D. Khovratovich, I. Nikolić, and C. Rechberger. Rotational rebound attacks on reduced Skein. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2010.

[78] V. Klima. Huge Multicollisions and Multipreimages of Hash Functions BLENDER-n. Cryptology ePrint Archive, Report 2009/006, 2009. `http://eprint.iacr.org/2009/`.

[79] A. Klimov and A. Shamir. A new class of invertible mappings. In Burton S. Kaliski Jr., Çetin Kaya Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 470–483. Springer, 2003.

[80] A. Klimov and A. Shamir. New applications of T-functions in block ciphers and hash functions. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 18–31. Springer, 2005.

[81] L. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1995.

[82] L. Knudsen. DEAL-a 128-bit block cipher. *Complexity*, 258:2, 1998.

[83] L. Knudsen, C. Rechberger, and S. Thomsen. Grindahl-A family of hash functions. In A. Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2007.

[84] D. Knuth. The art of computer programming: Seminumerical algorithms, volume 2, 1981.

[85] Ö. Küçük. The hash function Hamsi. *Submission to NIST (updated)*, 2009.

[86] X. Lai and J. Massey. Hash functions based on block ciphers. In R. A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 1993.

[87] M. Lamberger and F. Mendel. Practical Collision and Preimage Attack on DCH-n. 2008. `http://ehash.iaik.tugraz.at/uploads/9/9b/Dch.pdf`.

[88] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[89] G. Leander and A. Poschmann. On the Classification of 4 Bit S-boxes. In C. Carlet and B. Sunar, editors, *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007.

[90] F. MacWilliams and N. J.A.Sloane. *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam, 1977.

[91] J. Martin. Essence: A family of cryptographic hashing algorithms. *Submission to NIST*, 2008.

[92] S. Matsuo, M. Knezevic, P. Schaumont, I. Verbauwhede, A. Satoh, K. Sakiyama, and K. Ota. How can we conduct fair and consistent hardware evaluation for SHA-3 candidate? In *The Second SHA-3 Candidate Conference by NIST*, 2010.

[93] U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In M. Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.

[94] P. Maxwell. Aww, p*sh! 2008. `http://www.allicient.co.uk/2008/11/05/aww-psh/`.

[95] F. Mendel, T. Nad, and M. Schläffer. Collision Attack on Boole. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 369–381, 2009.

[96] F. Mendel, C. Rechberger, M. Schläffer, and S. Thomsen. Rebound attacks on the reduced Grøstl hash function. In J. Pieprzyk, editor, *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings*, volume 5985 of *Lecture Notes in Computer Science*, pages 350–365. Springer, 2010.

[97] F. Mendel and M. Schläffer. On free-start collisions and collisions for TIB3. In P. Samarati, M. Yung, F. Martinelli, and C. A. Ardagna, editors, *Information Security, 12th International Conference, ISC 2009, Pisa, Italy, September 7-9, 2009. Proceedings*, volume 5735 of *Lecture Notes in Computer Science*, pages 95–106. Springer, 2009.

[98] R. Merkle. One way hash functions and DES. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989,*

Proceedings, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1990.

[99] D. Moody, S. Paul, and D. Smith-Tone. Indifferentiability security of the fast widepipe hash: Breaking the birthday barrier. Cryptology ePrint Archive, Report 2011/630, 2011. `http://eprint.iacr.org/`.

[100] N. Mouha. Collision for Khichidi-1. 2008. `http://ehash.iaik.tugraz.at/uploads/8/89/Khichidi-1.txt`.

[101] S. Murphy. The cryptanalysis of FEAL-4 with 20 chosen plaintexts. *Journal of Cryptology*, 2(3):145–154, 1990.

[102] National Institute of Standards and Technology. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. *Federal Register*, 72(212):62212-62220, Nov.2007.

[103] M. Naya-Plasencia, A. Röck, J. Aumasson, Y. Laigle-Chapuy, G. Leurent, W. Meier, and T. Peyrin. Cryptanalysis of ESSENCE. In S. Hong and T. Iwata, editors, *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2010.

[104] I. Nikolić. Near collisions for the compression function of Hamsi-256. *CRYPTO rump session*, 2009.

[105] G. Nivasch. Cycle detection using a stack. *Information Processing Letters*, 90(3):135–140, 2004.

[106] D. Osvik. Speeding up SERPENT. In *The Third AES Candidate Conference, printed by the National Institute of Standards and Technology, Gaithersburg, MD*, pages 317–329. Citeseer, 2000.

[107] B. Preneel. Analysis and design of cryptographic hash functions. *Doct Dissertation K.U. Leuven*, 1993.

[108] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In D. R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 1994.

[109] C. Rechberger. Cryptanalysis of hash functions. *Doct Dissertation Graz University of Technology, Graz*, 2009.

[110] M. Reyhanitabar, W. Susilo, and Y. Mu. Enhanced security notions for dedicated-key hash functions: definitions and relationships. In S. Hong and T. Iwata, editors, *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *Lecture Notes in Computer Science*, pages 192–211. Springer, 2010.

[111] V. Rijmen and E. Oswald. Update on SHA-1. In A. Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2005.

[112] T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In K. G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506. Springer, 2011.

[113] R. Rivest, B. Agre, D. Bailey, C. Crutchfield, Y. Dodis, K. Fleming, A. Khan, J. Krishnamurthy, Y. Lin, L. Reyzin, et al. The MD6 hash function A proposal to NIST for SHA-3. *Submission to NIST*, 2008.

[114] P. Rogaway. Formalizing human ignorance. In P. Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 2006, First International Conference on Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

[115] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In B. K. Roy and W. Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.

[116] P. Sarkar and P. J. Schellenberg. A parallel algorithm for extending cryptographic hash functions. In C. P. Rangan and C. Ding, editors, *Progress in Cryptology - INDOCRYPT 2001, Second International Conference on Cryptology in India, Chennai, India, December 16-20, 2001, Proceedings*, volume 2247 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 2001.

[117] A. Shamir and E. Biham. Differential cryptanalysis of DES-like cryptosystems. In A. Menezes and S. A. Vanstone, editors, *Advances in Cryptology*

- *CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1991.

[118] M. H. S.Khazaei and M.Kiaei. Distinguishing attack on Grain. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/071, June 2005. `http://www.ecrypt.eu.org/stream`.

[119] Søren S. Thomsen. Untangled. 2008. `http://www2.mat.dtu.dk/people/S.Thomsen/tangle/tangle-coll.pdf`.

[120] Tor E. Bjørstad. Collision for StreamHash. 2008. `http://ehash.iaik.tugraz.at/uploads/7/7b/Streamhash.txt`.

[121] M. S. Turan and E. Uyan. Practical Near-Collisions for Reduced Round Blake, Fugue, Hamsi and JH. In *Second SHA-3 Candidate Conference by NIST*, 2010.

[122] Turan M. S. and Perlner R. and Bassham L. E. and Burr W and Chang D. and Chang S. and Dworkin M. J. and Kelsey J. M. and Paul S. and Peralta R. Status report on the second round of the SHA-3 cryptographic hash algorithm competition. *NIST, Tech. Rep*, 2010.

[123] M. Ullrich, C. De Canniére, S. Indesteege, Ö. Küçük, N. Mouha, and B. Preneel. Finding Optimal Bitsliced Implementations of 4×4-bit S-boxes. In *Symmetric Key Encryption Workshop*, 2011.

[124] J. Van Lint. *Introduction to coding theory*, volume 86. Springer Verlag, 1999.

[125] P. van Oorschot and M. Wiener. Improving implementable meet-in-the-middle attacks by orders of magnitude. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 229–236. Springer, 1996.

[126] P. van Oorschot and M. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[127] D. Wagner. The boomerang attack. In L. R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

[128] M. Wang, X. Wang, K. Jia, and W. Wang. New Pseudo-Near-Collision Attack on Reduced-Round of Hamsi-256. Cryptology ePrint Archive, Report 2009/484, 2009. `http://eprint.iacr.org/`.

[129] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.

[130] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[131] D. A. Wilson. Constructing Second Preimages in the WaMM Hash Algorithm. 2008. `http://web.mit.edu/dwilson/www/hash/wamm.html`.

[132] H. Yoshida, D. Watanabe, K. Okeya, J. Kitahara, H. Wu, Ö. Küçük, and B. Preneel. MAME: A compression function with reduced hardware requirements. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 148–165. Springer, 2007.

[133] G. Yuval. How to swindle Rabin. *Cryptologia*, 3(3):187–191, 1979.

[134] L. Zhimin, W. Licheng, L. Daofeng, and Y. Y. Collision Attacks on NaSHA-384/512. Cryptology ePrint Archive, Report 2009/026, 2009. `http://eprint.iacr.org/`.

En güzel deniz:
henüz gidilmemiş olanıdır...
En güzel çocuk: henüz büyümedi.
En güzel günlerimiz:
henüz yaşamadıklarımız.
Ve sana söylemek istediğim en güzel söz:
henüz söylememiş olduğum sözdür ...
Nazım Hikmet Ran - 1946

Curriculum Vitae

Özgül Küçük was born in Yalova, Türkiye. She completed her five years of primary education in five different schools in four cities. After that she started at the Gazi Anatolian High School for her secondary and high school education. She received the degree of Mathematician from the Middle East Technical University (METU) in 1995. She obtained a Master of Science degree from the Department of Mathematics of Bilkent University in 1997. Her thesis was about a classification problem of Enrique Surfaces on algebraic topology. She worked as a researcher at TUBITAK-BILGEM on cryptography between 1997 and 2003. She joined the research group COSIC (COmputer Security and Industrial Cryptography) at the Department of Electrical Engineering (ESAT) of the KU Leuven in 2005. She started her PhD in July 2007 in the same group. In 2010, from September to December she visited the Security Technologies Group of Sony corporation in Tokyo, Japan.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty Engineering
Department of Electrical Engineering (ESAT)
Computer Security and Industrial Cryptography (COSIC)
Kasteelpark Arenberg 10-2446, B-3001 Heverlee, Belgium

KATHOLIEKE UNIVERSITEIT LEUVEN

ASSOCIATIE K.U.LEUVEN