ANALYSIS OF THREE BLOCK CIPHER BASED HASH FUNCTIONS: WHIRLPOOL, GRØSTL AND GRINDAHL

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

RITA ISMAILOVA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHYLOSOPHY
IN
CRYPTOGRAPHY

SEPTEMBER 2012

Approval of the thesis:

ANALYSIS OF THREE BLOCK CIPHER BASED HASH FUNCTIONS: WHIRLPOOL, GRØSTL AND GRINDAHL

submitted by **RITA ISMAILOVA** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Department of Cryptography**, **Middle East Technical University** by,

Prof. Dr. Bülent Karasözen _____
Director, Graduate School of **Applied Mathematics**

Prof. Dr. Ferruh Özbudak _____
Head of Department, **Cryptography**

Assoc. Prof. Dr. Melek Diker Yücel _____
Supervisor, **Department of Electrical and Electronics Engineering**

Examining Committee Members:

Prof. Dr. Ersan Akyıldız _____
Department of Mathematics, METU

Assoc. Prof. Dr. Melek Diker Yücel _____
Department of Electrical and Electronics Engineering, METU

Assoc. Prof. Dr. Ali Doğanaksoy _____
Department of Mathematics, METU

Assist. Prof. Dr. Zülfükar Saygı _____
Department of Mathematics, TOBB ETU

Dr. Hamdi Murat Yıldırım _____
Department of Computer Technology and Information Systems,
Bilkent University

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: RITA ISMAILOVA

Signature :

ABSTRACT

ANALYSIS OF THREE BLOCK CIPHER BASED HASH FUNCTIONS: WHIRLPOOL, GRØSTL AND GRINDAHL

Ismailova, Rita
Ph.D., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Melek Diker Yücel

September 2012, 101 pages

The subject of this thesis is the study of cryptographic hash functions, which utilize block ciphers as underlying chain functions. It is mainly concerned with the analysis of the three hash algorithms, the Whirlpool, Grøstl and Grindahl. All these hash functions have underlying block ciphers that are modified versions of the Advance Encryption Standard and we investigate the behavior of these block ciphers under the integral attack.

Statistical tests, such as the avalanche test and the collision test, are the regular tools for examining the hash function security. In this work, we inspect the statistical behavior the three hash functions and search for collisions. Although it is very difficult to obtain collisions for the actual algorithms, we find some collisions under slight modifications of the original constructions. The ease or difficulty of finding a collision for a modified version also shows the respective importance of the specific hash function branch, missing in the modified version.

Keywords: Iterated Hash functions, Integral attack, Whirlpool, Grøstl, Grindahl

ÖZ

BLOK ŞİFRE TABANLI ÖZET FONKSİYONLAR ANALİZİ: WHIRLPOOL, GRØSTL VE GRINDAHL

Ismailova, Rita
Doktora, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Melek Diker Yücel

Eylül 2012, 101 Sayfa

Bu tez, zincirleme yapısının her zincirinde blok şifreleri kullanan kriptografik özet fonksiyonlar hakkındadır ve özünde üç özet fonksiyonuyla, Whirlpool, Grøstl ve Grindahl'un analizi ile ilgilidir. Bu üç özet fonksiyon, blok şifre olarak (Gelişmiş Şifreleme Standardı) AES'e benzeyen şifreler kullanır; ve tezde bu blok şifrelerin integral atağına karşı davranışları incelenmektedir.

Çığ testi ve çarpışma testi gibi istatistiksel testler, özet fonksiyonlarının güvenlik incelemesinde kullanılan olağan tekniklerdir. Bu çalışmada üç özet fonksiyonunun istatistiksel özellikleri incelenmiş ve çarpışmalar aranmıştır. Algoritmaların aslı için çarpışma bulmak çok zor olsa da, yapılarında ufak değişiklikler oluşturularak bazı çarpışmalar bulunmuştur. Değiştirilmiş herhangi bir yapı için çarpışma bulunmasındaki kolaylık veya zorluk, o yapıyı oluşturmak için özet fonksiyonun ana yapısından çıkarılan kolun önem derecesinin de bir göstergesidir.

Anahtar Kelimeler: Yinelemeli özet fonksiyonlar, İntegral atağı, Whirlpool, Grøstl, Grindahl

To my mother

ACKNOWLEDGMENTS

First of all I would like to thank my supervisor, Assoc. Prof. Dr. Melek Diker Yücel, for her guidance, termless support and encouragement throughout my thesis work, and in my life. Without her, this work would not have been completed.

I am deeply grateful to my family for their endless support, understanding and patience.

Also, I would like to thank my friends for their invaluable support and assistance both in my thesis work and in my life.

TABLE OF CONTENTS

ABSTRACT iv
ÖZ v
ACKNOWLEDGMENTS vii
TABLE OF CONTENTS viii
LIST OF NOTATIONS x
LIST OF ABBREVIATIONS xi
LIST OF TABLES xii
LIST OF FIGURES xiii
CHAPTERS:
1. INTRODUCTION 1
1.1. Cryptographic Hash Functions 1
1.2. Hash Function Design Principles 3
1.2.1. Iterated Hash Functions 3
1.2.2. Security of Iterated Hash Functions 6
1.3. Contribution of the Thesis 7
2. AES BASED HASH FUNCTIONS AND ATTACKS ON THE ALGORITHMS 10
2.1. Iterated Hash Functions 11
2.1.1. Three Constructions for Iterated Hash Functions 11
2.1.2. Whirlpool Hash Function 13
2.1.3. Grøstl Hash Function 15
2.1.4. Grindahl Hash Function 17
2.2. Comparison of the Block Ciphers of the Three Hash Functions with the AES . 18
2.3. Hash Function Cryptanalysis 19
2.3.1. Brief History of Hash Function Proposals 20
2.3.2. Chronology of Attacks on Hash Functions 21
2.3.3. Description of Some Well Known Attacks 26
3. INTEGRAL STRUCTURES FOR UNDERLYING ENCRYPTION ALGORITHM OF HASH FUNCTIONS 32
3.1. Definitions 33
3.2. Whirlpool 35
3.3. Grøstl 39
3.4. Grindahl 42
3.5. Role of Cipher Operations in the Integral Attack for Whirlpool 45
3.6. Conclusion 48

4. ANALYSIS OF HASH ALGORITHMS CONSTRUCTIONS 50
4.1. Definitions 51
4.2. Collisions for Modified Forms of Whirlpool Hash Function.....53
4.2.1. Schemes Under Backward Analysis 53
4.3. Collisions for Modified Forms of Grøstl Hash Function 58
4.4. Collisions for Modified Forms of Grindahl Hash Function 61
4.5. Conclusion 63
5. CONCLUSION.....66
REFERENCES 69
APPENDICES.....78
A. STATISTICAL ANALYSIS OF BLOCK CIPHER BASED HASH FUNCTIONS 78
A.1. Statistical Analysis of Weights and Distances 78
A.2. Statistical Analysis of Correlation 82
A.2.1 Correlation Between Message and Message Digests of the Hash Functions 82
A.2.2 Correlation Between Round Outputs of the W Cipher.....84
A.2.3 Correlation Between Round Outputs of Grøstl Permutations P and Q 87
A.3. Whirlpool with Different Chaining Schemes with Original S-Box and AES S-Box 90
A.3.1 Comparison with the S-Boxes of Serpent and AES 90
A.3.2 Statistical Analysis of Weights.....91
A.3.3 Statistical Analysis of Correlations between Message and Message Digests of the Whirlpool Hash Function 94
A.4. Conclusion 96
B. 16-BIT VERSION OF WHIRLPOOL HASH FUNCTION 97
B.1. Implementation.....97
B.2. Test of Collisions 98
B.3. Analysis of Whirlpool Hash Function in Backward Direction..... 100

LIST OF NOTATIONS

h	Hash function
f	Chain function
X	Message
Y	$= h(X)$ – Message digest
K	Key
N	Length of original message
n	Length of Message digest
b	Length of message blocks
t	Number of message blocks
A	Input state of a cipher
B	Output state of a cipher
γ	Substitute Bytes operation
π	Shift Columns (Rows) operation
θ	Mix Rows (Columns) operation
σ	Add Round Key operation
κ	Add Round Constant operation
δ	Vector showing the number of positions to be shifted at each row
Ω	Truncation function
$pad(X)$	Padding function

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
CPT	Concatenate-Permute-Truncate
CRF	Collision Resistant Function
CRHF	Collision Resistant Hash Function
CTFP	Chosen Target Forced Prefix
FF	Feed-Forward (or Fed-Forward)
IV	Initial Value (or Vector)
MAC	Message Authentication Code
MDC	Modification Detection Code
MDS	Maximum Distance Separable
OWF	One-Way Function
OWHF	One-Way Hash Function
SP	Substitution Permutation

LIST OF TABLES

Table 2-1 Comparison of block cipher W of Whirlpool, P and Q of Grøstl, P of Grindahl and AES 19
Table 2-2 Standard hash algorithm proposals 21
Table 2-3 Attacks on hash functions 24
Table 3-1 The number of occurrences of frequencies, f_{jk} in W cipher 37
Table 3-2 Active bytes that yield the elements of frequency 8 at the same position of the 3rd round output 39
Table 3-3 Number of occurrences of frequencies $\mathbf{fjk\beta}$ of field elements $\boldsymbol{\beta}$ at the round outputs of P and $\boldsymbol{Q}$ ciphers of Grøstl-512 41
Table 3-4 Average number of occurrences of frequencies $\mathbf{fjk\beta}$ for fixed input-output byte positions 41
Table 3-5 Active bytes that yield the elements of frequency 8 at the same position of the 3rd round output of the P cipher 42
Table 3-6 Number of occurrences of frequencies, $\mathbf{fjk}$ in 3 chains of Grindahl hash function 44
Table 3-7 Number of occurrences of frequencies, $\mathbf{fjk}$ in 3 chains of Grindahl hash function 44
Table 3-8 The maximal frequencies with which an output value occurs when passive bytes are different 45
Table 3-9 Active bytes that yield the elements of frequency 8 at the same position of the 3rd round output when the role of key and message is switched 46
Table 4-1 Attacks on 64 different schemes (reproduced from [79]) 52
Table A-1 Size and element weights of the input data sets used for the statistical analysis 79
Table A-2 Statistics of correlation for the Whirlpool hash function 83
Table A-3 Statistics of intermediate values normalized correlation 84
Table A-4 Comparison of the Whirlpool block cipher W and Serpent's s-boxes 91
Table A-5 Comparison between the s-boxes of the Whirlpool block cipher W and the AES 91
Table A-6 Statistics of weights for different schemes 92
Table A-7 Statistics of weights for different schemes with AES's s-box 92
Table A-8 Statistics of normalized correlations between message and message digest for different schemes 94
Table A-9 Statistics for normalized correlation between message and message digest for different schemes with AES's s-box 94
Table B-1 The number of collisions 99

LIST OF FIGURES

Figure 1-1 Classification of cryptographic hash functions 3
Figure 2-1 Structure of the Whirlpool hash function 14
Figure 2-2 Whirlpool cipher W 15
Figure 2-3 Structure of the Grøstl hash function 16
Figure 2-4 Structure of the Grindahl-512 hash function 17
Figure 2-5 Chronology and interrelation of attacks on hash functions 25
Figure 2-6 A schematic view of the rebound attack 26
Figure 2-7 A schematic view of the attack on 4 rounds of the Whirlpool with round key inputs and feed-forward (reproduced from [63]). 28
Figure 2-8 The inbound phase of the distinguishing attack 28
Figure 2-9 Systematic view of the Super-Sbox attack 30
Figure 3-1 Frequencies of the elements in the outputs of round 3 at position k=0 as α=0,…,255 35
Figure 3-2 The number of occurrences of frequencies, $\boldsymbol{fjk}$ (notice that for $\boldsymbol{fjk} > \mathbf{5}$, vertical resolution is not sufficient for the demonstration of $\boldsymbol{Nf}$) 38
Figure 3-3 The number of occurrences of frequencies for $\mathbf{fjk} > \mathbf{6}$. 38
Figure 3-4 Frequencies of the elements in the P and Q ciphers outputs of round 3 at position k=0 as α=0,…,255 40
Figure 4-1. Scheme 1 for Merkle-Damgård construction 54
Figure 4-2 Scheme 2 for Merkle-Damgård construction 55
Figure 4-3 Scheme 3 for Merkle-Damgård construction 57
Figure 4-4. The chain function *f'* of the Grøstl hash algorithm. 58
Figure 4-5 Scheme 1 for Wide-Pipe construction 59
Figure 4-6 Scheme 2 for Wide-Pipe construction 61
Figure 4-7 One chain of the Grindahl hash function 62
Figure 4-8 Positions of the inserted 8-byte block after ShiftRow 63
Figure A-1 Weight histogram of the Whirlpool hash function for 512 weight-1 inputs. 80
Figure A-2 Weight histogram of the Whirlpool hash function for 93350 random inputs. 80
Figure A-3 Maximal, minimal and average weights of message digests for the Whirlpool hash function 81
Figure A-4 Distances between the hashes of 512 weight1 vectors and the all-zero vector for the Whirlpool hash function. 81
Figure A-5 Average message digest weights for the Whirlpool, Grøstl and Grindahl hash functions 82
Figure A-6 Normalized correlation magnitudes (average and maximum values) between messages and message digests 83

Figure A-7 Maximal normalized correlations between intermediate values 85
Figure A-8 Normalized correlation between successive round outputs over 512 inputs of weight 1 for all-0 key.....85
Figure A-9 Normalized correlation between successive round outputs over 512 inputs of weight 1 for all-1 key.....86
Figure A-10 Normalized correlation between successive round outputs over 512 inputs of weight 1 for a random key 86
Figure A-11 Normalized correlation between inputs and round-2 outputs over 512 inputs of weight 1 different keys 87
Figure A-12 Normalized correlation between successive round outputs of permutation P over 512 random inputs.....88
Figure A-13 Normalized correlation between successive round outputs of permutation Q over 512 random inputs 88
Figure A-14 Normalized correlation between round outputs of permutations P and Q over 512 random inputs 89
Figure A-15 Normalized correlation between input and round outputs of permutation P over 512 random inputs.....89
Figure A-16 Normalized correlation between input and round outputs of permutation Q over 512 random inputs 89
Figure A-17 Frequency of weights of MD's for the different schemes with the original s-box 93
Figure A-18 Frequency of weights of MD's for the different schemes with s-box of AES.....93
Figure A-19 Maximal correlation magnitudes between message and message digests 95
Figure A-20 Average correlation magnitudes between message and message digests 95
Figure B-1 S-box used in the design 98
Figure B-2. Diagonal structure of an input 99
Figure B-3 Output of W cipher for all zero input 100
Figure B-4 Backward search of the second preimage 100

CHAPTER 1

INTRODUCTION

In the modern world, where information is mainly stored on computers, it is essential to guarantee data integrity. Hash functions have many applications in the field of information security, in particular in digital signatures, message authentication codes (MAC) and other methods of information authentication. Besides, hash algorithms can be used for fingerprinting, to detect duplicate data, to uniquely identify files and checksums to discover accidental corruption of data. In digital signature applications, hash functions can be utilized to protect data from intentional alteration. Hash algorithms are many-to-one functions, so that finding the message from its hash (or digest) is not possible. Cryptographic hash functions are required to have two main properties, namely, they must be one-way, and collision resistant. Breaking a hash function means that one or both of those properties are not true.

Starting from 2004, substantial advances have been made in the cryptanalysis of hash functions [40]. In 2005 a full version of SHA-1 has been reported broken in [92]. These achievements in cryptanalysis of hash functions stimulate cryptographic community to pay more attention to the analysis of the algorithms.

One of the approaches for assessing hash function security are statistical tests, such as the avalanche test [35], [64] and the collision test [64], [88], [90].

1.1. Cryptographic Hash Functions

Hash functions, also known as message digests, are important cryptographic primitives. The term hash function originates historically from computer science, where it denotes a function that compresses a string of arbitrary input to a string of fixed length. The name hash function has also been widely adopted for cryptographic hash functions or cryptographically strong compression functions. Cryptographic hash functions are algorithms h mapping bit strings of arbitrary finite length to strings of fixed length, say n bits. For a domain D and range R with

$$h: D \to R \quad \text{and} \quad |D| > |R|,$$

the hash function is *many-to-one*; i.e., the pairs of inputs with identical output are unavoidable, which are called *collisions*. Indeed, restricting h to a domain of N-bit inputs $N > n$, if h were "random" in the sense that all outputs were essentially equiprobable, then about 2^{N-n} inputs would map to each output, and two randomly chosen inputs would yield the same output with probability 2^{-n} .

A small change in the input value can cause a major bit shift on the entire output string. A shift or change of 1 bit in the input message will prompt a shift of about half of the total bits in the resulting hashcode. This is called the *avalanche effect*.

There are two main types of cryptographic hash functions, named Modification Detection Codes (MDC) and Message Authentication Codes (MAC). The difference between these two types is in the number of inputs [76]. Hash functions take as input a message to be hashed and some initial vector (IV). The IV can be fixed to some predefined value (in MDC), or can serve as a key to hash algorithm (in MAC). Therefore, in the literature MDC and MAC may be classified as unkeyed and keyed hash functions respectively:

- ***Modification detection codes*** are used to check if the message has been altered (data integrity), they are a subclass of unkeyed hash functions.
- ***Message authentication codes*** are used to prove the data origin (data authentication), and they involve a secret key. Classification of cryptographic hash functions done in [78] is given in Figure 1-1.

The definition of a *one-way function* (OWF) was given first in the seminal paper by Diffie and Hellman [31]. R. Merkle later defined a ***one-way hash function*** (OWHF) as a function h satisfying the following conditions [66]:

- The description of h must be publicly known and should not require any secret information for its operation (extension of Kerckhoffs's principle).
- The argument X can be of arbitrary length and the result $h(X)$ has a fixed length of n bits.
- Given h and X, the computation of $h(X)$ must be "easy".
- The hash function must be one-way in the sense that given a Y in the image of h, it is "hard" to find a message X such that $h(X) = Y$ and given X and $h(X)$ it is "hard" to find a message $X' \neq X$ such that $h(X') \neq h(X)$.

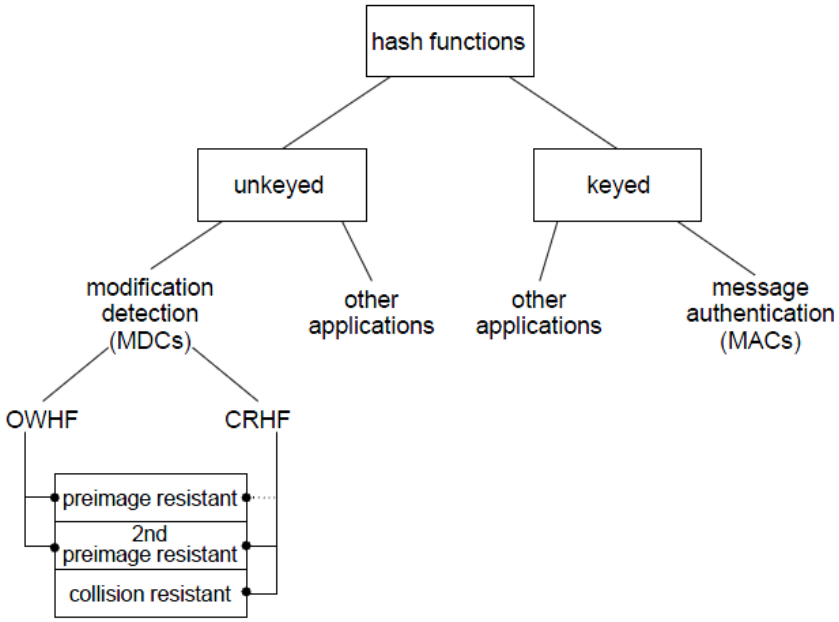

Figure 1-1 Classification of cryptographic hash functions

This definition of OWFH differs from the *one-way function* (OWF) of [31] only by condition 2, i.e., OWHF operates with messages of arbitrary length, while OWF can take input messages of some predefined, fixed size.

Security of a hash function means a high level of collision resistance. The first definition of collision resistance was given by Damgård [25]. Here we provide an informal definition given by B. Preneel [77]. A ***collision resistant hash function*** (CRHF) is a one-way hash function (OWHF) satisfying the following additional condition:

5. The hash function must be collision resistant: this means that it is "hard" to find two distinct messages that hash to the same result.

1.2. Hash Function Design Principles

1.2.1. Iterated Hash Functions

Almost all known hash functions are based on some internal function that processes each message block of a fixed size in a similar way. Such hash functions, for which message blocks are subjected to the same function successively, are called "iterated" [54].

Iterated hash functions use the so-called linear hashing technique, which has *t* chains. The linear hash algorithm performs hashing by splitting a message in a deterministic order, i.e., at the first step it prepares the message for hashing by dividing it into blocks. The splits are performed in

linear order (chain 0 first, then chain 1, then 2, and so on), and a new split is performed when any chain overflows, i.e., in linear hashing, the number of chains increases linearly and is exactly as large as needed.

According to [28], the operations of cryptographic hash algorithms can be divided into four distinct parts:

- Preprocessing stage
- Padding rule
- Initial value (IV)
- Compression function

The ***preprocessing stage*** is optional and introduces additional redundancy in order to increase the strength of the hash function. It can consist of constant bits, repetition of some bits of the message or can be a more complex procedure. Drawback of adding redundancy is the deceleration of the hash function.

The information is divided into t b-bit blocks x_1 through x_t. If the total number of bits in a message is no multiple of the block length b, a ***padding procedure*** has to be specified to make it a multiple. Also, in the padding procedure, the Merkle-Damgård strengthening rule should be applied, which means that in padded bits the length of the original message should be encoded [26], [68].

Also, every specification of a hash function should fix an ***initial value*** IV (or a small set of initial values), together with a motivation for the choice. If IV is generated pseudo-randomly, the algorithm should be described. If IV is not specified, it should be hard to produce collisions for any IV. In that case it is clearly necessary to add the length of the message at the end, in order to avoid trivial attacks, like omitting one or more blocks at the beginning of the message.

The most important part of a hash function is the function f used in each chain. In this work, we are concentrated on three different ***constructions of hash functions***; the first of which, **Merkle-Damgård construction**, can then be defined as follows:

$$H_0 = \text{IV}$$
$$H_i = f(x_i, H_{i-1}) \oplus FF\,, \qquad i = 1,2,\dots,t$$
$$h(X) = H_t$$

Here H_i is the n-bit intermediate variable (or chaining variable), IV is the Initial Value, x_i is the b-bit block obtained by segmenting the input message X, and t is the number of chains. The

abbreviation *FF* denotes a fed-forward combination that may contain the output H_{i-1} of the previous chain, or the message block x_i, or both.

Wide-Pipe construction differs from the above description by the use of a second function f" after the last chain:

$$H_0 = \text{IV}$$
$$H_i = f'(x_i, H_{i-1}) \oplus FF\ , \qquad i = 1,2, \dots, t$$
$$h(X) = f"(H_t)$$

The third construction that we use, **Concatenate-Permute-Truncate construction** (CPT), is quite similar to the Wide-Pipe construction, except for the lack of the *FF* path. Moreover, f' function is chosen as a Concatenate-Permute-Truncate operation until the last chain, where truncation is discarded.

The relation between the bit lengths of the hash $h(X)$ and the message block x_i differs for these three constructions. The size of the hash output $h(X)$

- equals the message block length for the Merkle-Damgård,
- equals one half of the message block length for the Wide-Pipe, and it
- is larger than the message block length for the Concatenate-Permute-Truncate (CPT) constructions.

For each of the above constructions, we have chosen and analyzed a specific hash function

- the Whirlpool as an example for the Merkle-Damgård construction,
- 512-bit version of Grøstl as an example for the Wide-Pipe construction, and
- 512-bit version of Grindahl as an example for the CPT construction.

All these hash algorithms utilize AES-based block ciphers; where Whirlpool and Grindahl-512 use a single block cipher in each chain, and Grøstl uses two of them. These ciphers are named as *W* for Whirlpool, *P* for Grindahl and *P* & *Q* for Grøstl.

Since Whirlpool has an invertible cipher as its internal function that uses a key scheduling algorithm, it is stated that more testing and research are needed to confirm its security [6]. As regards Grøstl [38], which is one of the five finalists of the SHA-3 competition; since the algorithm construction is highly complex and the underlying block ciphers use 14 rounds each, the cryptanalysis seems to be hard. Andreeva et al. show that Grøstl is indifferentiable from a random oracle, under the assumption that the underlying permutations *P* & *Q* are ideal [2]. However, the speed of hashing under this design decreases significantly. Unlike Grøstl, the design principle of Grindahl hash function [51] is light and the hashing speed is high. One may

conjecture that the security properties of the algorithm are therefore reduced; given that the 256 bit version of the hash function was reported to be broken [75], and it is shown [52] that the organization of internal states can drastically reduce the complexity of collision search.

1.2.2. Security of Iterated Hash Functions

The underlying block cipher of a hash algorithm has two inputs (the plaintext and the key) and one output. These two inputs can be chosen as one of the four variables $\{H_i,\ x_i$ and $x_i \oplus H_{i-1}$, or V$\}$: where x_i is a message block, H_{i-1} is the chaining variable (output of the previous chain that connects two chains), and V is some constant vector.

Hash algorithm constructions may also have an additional feed-forward path as mentioned in Section 1.2.1. Along with the choice of the underlying chain function, the choice of the feed-forward path, which determines the ***chaining mode*** of the hash function, has great influence on the security of the hash function. The inputs of the chaining function and the feed-forward path determines the ***scheme*** of the hash function.

Since the two inputs of the cipher and the feed-forward path can all be chosen from the set of four variables $\{H_i,\ x_i$ and $x_i \oplus H_{i-1}$, or some fixed value V$\}$, one can talk about 64 possible schemes for the iterated hash functions [79], although some of these 64 schemes are trivial and not useful.

Both Merkle-Damgård and Wide-Pipe constructions can be used in one of the nontrivial schemes among the 64 possible schemes [79]. One of the meaningful, but yet, not safe schemes is the Rabin scheme, where the message x_i is the plaintext and the chaining variable H_{i-1} is the key of the underlying cipher; and feed-forwarding is omitted (or, it can be considered as XOR'ing with the all zero constant vector). Although hash functions, built using this scheme are fast, they cannot be considered secure. In [14] it is shown that among the 64 schemes, only 12 can be considered secure.

Some of the schemes are named after the designers, who proposed the specific scheme. In addition to the Rabin scheme mentioned above, there are other schemes called Miyaguchi-Preneel scheme [77], Matyas-Meyer-Oseas scheme [59], and Davies-Meyer scheme [97], [29].

The choice of the scheme for a hash function should be done taking into account the construction of the underlying block cipher and it should be resistant to the following general attacks:

- direct attack,
- permutation attack,

- forward attack,
- backward attack
- fixed point attack.

Some of the attacks listed above use weaknesses of the schemes, while others exploit the weaknesses of the underlying block ciphers. For example, the backward attack is based on the fact that the underlying block ciphers are invertible, and the security of the scheme relies on the strength of the chaining mode. The fixed point attack is an attack that uses so called fixed points of the algorithm; which occur when the output at some point of the chain with the given chaining variable and the message block, is equal to the output at some other point with other inputs. If the feed-forward path is not chosen carefully, the hash function can become vulnerable to this class of attacks.

Although all three hash functions analyzed in this work are built using iterated principle; only two of them, Whirlpool and Grøstl, are to be described in terms of the schemes they exploit. The CPT construction does not use any feed-forward path, but concatenate and truncate operations, since the XOR operation, used in feed-forwarding is not always secure (however, the construction still can be considered as the Rabin scheme).

The other type of attacks exploit the weaknesses of the underlying chain function of the algorithm. The integral cryptanalysis can be an example of this type of attacks. In [50], nonrandom properties of the substitution-permutation (SP) networks was shown, which can be used to attack the Square cipher. The name of the integral cryptanalysis came from the idea of the attack, where the propagation of sums (or integrals) of specific collection of the plaintexts are considered. The integrals have some interesting properties, and it was shown that after 3 rounds of the encryption one could predict the values of these sums. Using this knowledge, the 4-round integral cryptanalysis for Square, Rijndael and W ciphers can be briefly described as follows: to attack 4 rounds of the cipher one predicts the value of a single key byte at a time; and counting backward, checks, if the 3^{rd} round outputs corresponding to 256 chosen plaintexts sum up to zero.

1.3. Contribution of the Thesis

In this thesis, we try to find collisions for block cipher based hash functions, Whirlpool, Grøstl and Grindahl, by means of known attacks. Our choice of these hash functions is due to the internal functions used in each chain, which are all modified versions of the AES. All three hash algorithms utilize an iterated principle; however, they are built using dissimilar constructions; the

Merkle-Damgård construction of Whirlpool, the Wide-Pipe construction of Grøstl, and the Concatenate-Permute-Truncate (CPT) construction of Grindahl.

We compute the integral structures for the underlying block ciphers *W* for Whirlpool, *P* for Grindahl, *P* & *Q* for Grøstl; and perform backward cryptanalysis of the overall constructions. In order to find the integral structures to be used for the integral attack, a set of experiments is carried out and patterns of cipher round outputs are investigated for the chosen plaintexts, which have all passive bytes except a single active byte that spans the whole field. At round outputs, the frequencies, with which elements occur at a particular position, and the number of high frequencies in the collection of all possible sets of integrals are studied. The ways to construct the integral structures for the selected algorithms are carried out, as well as the occurrence rates of each frequency, which are tabulated for each hash function. Also, the effects of core operations on the states of the ciphers are examined to determine the operations leading to the systematic integral characteristics.

In hash function constructions, an adversary has the full knowledge of initial vectors and round constants used as cipher keys, at least for the first chain of a given hash function; since the initial vectors, IV, of the algorithms are fixed and not secret. Also, since the block ciphers, which are invertible by definition, are used as underlying chain functions, one can always proceed backward, if the schemes (determined by the inputs of the chaining function and the feed-forward path) are not chosen carefully. In order to examine the collision resistances of Merkle-Damgård, Wide-Pipe and CPT constructions, we try to find collisions by backward analysis. In these analyses we mainly make use of the fixed points in the pre-last chain of the hash functions, and proceed in backward direction, aiming to obtain the IV, defined by the algorithm design.

We study three schemes of the Merkle-Damgård construction on the example of Whirlpool hash function. Three different schemes with different choices of cipher inputs and feed-forward paths are considered, for which we find some collisions and some pseudo-collisions (called ‘pseudo’ since they correspond to different IV’s). Collisions that we obtain show that the chosen three schemes are not secure; however, they don’t necessarily indicate a vulnerability for the original Whirlpool, whose feed-forward path includes modulo 2 sum of the input and output of the previous chain. The collisions that we find simply highlight the importance of the absent feed-back branches of the chosen three schemes; with respect to the original (Miyaguchi-Preneel) scheme of Whirlpool.

As for the Wide-Pipe construction, we attain fixed points on the modified version of Grøstl; obtain pseudo-collisions with the respective IV’s and show the impossibility of building the backward attack on the original version of the algorithm. The results of this part show that some

schemes under Wide-Pipe construction are more secure than the same schemes under Merkle-Damgård construction.

Along with the Concatenate Permute Truncate (CPT) construction, the Grindahl hash function uses Merkle-Damgård strengthening. The backward analysis on the original version of Grindahl and the role of Merkle-Damgård strengthening in this construction are shown. As a result of the backward analysis, we obtain pseudo-collisions for the Grindahl hash function.

The thesis is organized in the following way:

Chapter 2 starts by the general description of the three different constructions for the iterated hash functions based on block ciphers: Merkle-Damgård, Wide-Pipe and Concatenate Permute Truncate (CPT). Then, the structures of the chosen algorithms, namely, Whirlpool, Grøstl and Grindahl that utilize these constructions, are presented. A brief chronology of the attacks on hashing algorithms, and the interrelation among proposed attacks is discussed.

In Chapter 3, round output integrals for the block ciphers of Whirlpool and 512 bit versions of Grøstl and Grindahl are constructed, frequencies of occurrence of field elements are counted, and the applicability of the integral attack on these ciphers is discussed. Also, the effects of core operations on the cipher states are investigated.

Chapter 4 is devoted to the search for collisions. It starts with the review of a seminal paper [79] on the evaluation of attacks on 64 schemes. We then present our 3-round, 3-chain collisions or pseudo-collisions for the Merkle-Damgård, Wide-Pipe constructions and 3-chain pseudo-collisions for the Concatenate Permute Truncate construction.

Chapter 5 summarizes our conclusions and gives directions for future work.

CHAPTER 2

AES BASED HASH FUNCTIONS AND ATTACKS ON THE ALGORITHMS

Block ciphers are ideal candidates for underlying chain functions in iterative hash algorithms. However, even a cryptographically strong block cipher may exhibit weaknesses that may not be significant unless the cipher is used in a hash algorithm. Since the AES (Advanced Encryption Standard) has been carefully analyzed and the cryptographic world is closely familiar with the underlying cipher, Rijndael; it is not surprising that modified versions of Rijndael are widely used as chain functions in the design of hash algorithms. The modifications are mostly aimed at increasing the input size, since a larger block size is desirable for reasons of both efficiency and security of hashing.

In this chapter, we are going to give the general description of AES based hash functions, which are analyzed in the thesis. The choice of the following three algorithms is made due to their dissimilar constructions. As an example for Merkle-Damgård construction, we have chosen the Whirlpool hash algorithm, secondly, Grøstl hash function is a wide-pipe construction where the size of the internal state is twice the size of the output and thirdly, Grindahl family of hash functions exploits the so called *Concatenate-Permute-Truncate* (CPT) design [51]. Below, we first summarize the three different constructions in Section 2.1 and then continue with the explanations of the Whirlpool, Grøstl and Grindahl hash functions in sections 2.1.2, 2.1.3 and 2.1.4 respectively. In section 2.2, we compare the AES-based block ciphers used in these three hash functions with the AES. Finally, we give the history of hash function proposals, as well as the brief chronology of attacks on hashing algorithms, and we present the interrelation of proposed attacks in Section 2.3.

2.1. Iterated Hash Functions

2.1.1. Three Constructions for Iterated Hash Functions

In hash functions that use the iterative principle of hashing, the internal chain function *f* is the most important part. However, iterated hash functions differ not only by the design of the internal function *f*, but also by the overall construction of the algorithm. In this work, we consider three different constructions corresponding to three hash functions.

The Merkle-Damgård structure ([26], [69]) was the one of the first proposals for constructing hash functions. The Wide-Pipe construction [58] is similar to Merkle-Damgård, but the message is processed through two internal functions. In the Concatenate-Permute-Truncate (CPT) construction [51], the main idea is to use concatenate-truncate operations. Also, the relation between the bit lengths of the hash $h(X)$ and the message block x_i differs for these three constructions. The size of each message block x_i

- equals the length of the hash $h(X)$ for the Merkle-Damgård,
- equals twice the length of the hash for the Wide-Pipe, and it
- is smaller than the length of the hash for the Concatenate-Permute-Truncate (CPT) constructions.

Let us describe these design principles in more details.

Merkle-Damgård Construction

Ivan Damgård [26] and Ralph Merkle [69] in two independent works published in Crypto'89 showed that if the IV of a hash algorithm is fixed, and padding rule with encoded message length is added to the end of the message; then, if one-way chain function *f* is collision resistant, so is the hash function constructed using it. Lai and Massey called it the ***Merkle-Damgård strengthening*** [54] and the design approach is commonly called Merkle-Damgård design.

In general, given a message $X = \{0,1\}^*$, one computes the message digest as follows:

INPUT: a message $X = \{0,1\}^*$, OUTPUT: the message digest $h(X) = \{0,1\}^n$

Expand a message: $X = (x_1, \dots, x_t)$;

MD strengthening: last block x_i takes the length $|X|$ in bits;

Set $H_0 = IV$;

for ($i = 1$; $i \leq t$; i++)

{

$H_i = f(x_i, H_{i-1}) \oplus FF$ where $f: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$

}

Set $h(X) = H_t$

Here, *FF* is a fed-forward bit sequence, which may contain x_i, H_{i-1}, or both. Unfortunately, this construction has several undesirable properties:

- Finding a second preimage for long messages is always much more efficient than brute force attack.
- If a collision is found, multi-collisions (many messages mapping to the same hash value) can be found with only a little more effort.
- "Herding attack" [45], also known as the chosen-target-forced-prefix (CTFP) attack, shows that an attacker can put together some prefix as part of the original preimage corresponding to the committed value $h(X)$.
- "Extension attack", also called "message extension" or "padding" attack, is based on the fact, that given the hash $h(X)$ of an unknown input X, and length(X), one is able to compute $h(X||\text{pad}(X)||X')$ for any X', where pad is the padding function of the hash. Since the $h(X)$ and length(X) are known, the new blocks can be added and new padding can be attached. That is, it is possible to find hashes of inputs related to X even though X remains unknown.

Wide-Pipe Construction

In 2005, Lucks proposed the idea to preserve the internal state of the hash function twice as big as the hash, so that finding a collision on the internal state by means of brute-force attack is unacceptable - Wide-Pipe Hash [58]. Using this fact, Lucks showed that the second preimage attack may not be faster than exhaustive search. The main disadvantage of this method is its large memory requirement. Nevertheless, this strategy is very successful, and it has been used by many hash functions, which proceeded the second round of the SHA-3 competition, namely Blue Midnight Wish, [7], Fugue, Grøstl [38], JH, SIMD and Skein.

The idea of Wide-Pipe construction can be described as follows:

INPUT: a message $X = \{0,1\}^*$, OUTPUT: the message digest $h(X) = \{0,1\}^n$

Expand a message: $X = (x_1, \dots, x_t)$;

Set $H_0 = IV$;

for (i = 1; $i \leq t$; i++)

{

$H_i = f'(x_i, H_{i-1}) \oplus FF \quad where\ f'\colon \{0,1\}^w \times \{0,1\}^m \rightarrow \{0,1\}^w$, $w>m$

}

Set $h(X) = f''(H_t)$ where $f''\colon \{0,1\}^w \rightarrow \{0,1\}^n$

Concatenate-Permute-Truncate (CPT) Construction

This construction was first named in [51] by referring to the proposal of Snefru hash function [69]. The original idea that has been developed in the Snefru proposal is to use a different design method for the chain function, which is not based on traditional adaptation of a block cipher. The principle behind CPT construction can be described as follows:

Expand a message: $X = (x_1, \dots, x_t)$;

Set $H_0 = IV$;

for (i = 1; $i \leq t$; i++)

{

$S_i = x_i \parallel H_{i-1}, \qquad where\ S_i\colon \{0,1\}^w \times \{0,1\}^m \rightarrow \{0,1\}^{w+m}$

$\hat{s}_i = P(S_i), \qquad where\ \hat{s}_i\colon \{0,1\}^{w+m} \rightarrow \{0,1\}^{w+m}$

$H_i = trunc(\hat{s}_i), \qquad where\ H_i\colon \{0,1\}^{w+m} \rightarrow \{0,1\}^m$

}

Set $h(X) = f(H_t), where\ f\colon \{0,1\}^m \rightarrow \{0,1\}^n$

where $P(S_i)$ is a permutation, and truncation is discarding all but t least significant bits of a string [51]. Unlike the previous constructions, where the message (itself, or after being XORed with the previous chain output) plays the role of plaintext in the underlying cipher; in CPT construction, the message is inserted to the chain function by concatenating the input block to a truncated internal state. There have been early attacks against Snefru [12] improved in [9] as well as for Grindhal [75].

2.1.2. Whirlpool Hash Function

Designed by Vincent Rijmen and Paulo S.L.M.Barreto, the Whirlpool hash function was endorsed by NESSIE project [6]. Whirlpool is an iterated hash function, based on Miyaguchi-

Preneel scheme. As an underlying function, it uses a modified version of AES, the symmetric key block cipher W.

Whirlpool (Figure 2-1) creates a 512 bit message digest from a message of length divisible by 512 and less than 2^{256}. Padding bits are arranged as a single 1 followed by necessary number of 0's. The length of the original message is enclosed in the last 256 bits of a padded message.

The initial value of the hash, H_0 is set to all 0's, and it becomes the cipher key for the first encryption by the W cipher. Being XORed with the previous cipher key and plaintext, ciphertext becomes the cipher key for the next block. The message digest is the 512-bit output of the last block: $H_i = W(H_{i-1}, x_i) \oplus x_i \oplus H_{i-1}$.

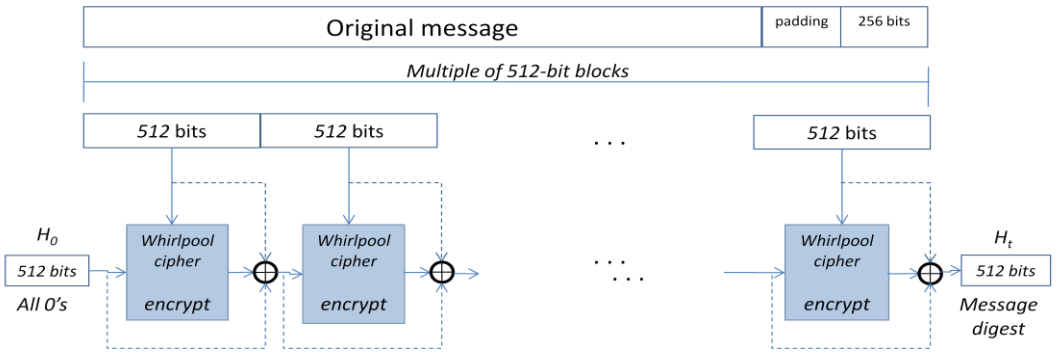

Figure 2-1 Structure of the Whirlpool hash function

W is a round cipher with 10 rounds (Figure 2-2). Block size and key sizes are both 512 bits (64 bytes). Encryption in W is similar to that of AES; i.e., each round contains the SubBytes, ShiftColumn, MixRow and AddRoundKey operations. If the state of the cipher is viewed as an 8×8 matrix of 64 bytes, an input state *A* is converted to the output state *B*.

SubBytes transformation, γ, provides the confusion effect and it is performed on a single byte at a time:

$$\gamma(A) = B \Leftrightarrow b_{ij} = S[a_{ij}],\ 0 \le i, j \le 7$$

The entries of the s-box, *S*, can also be calculated in $\mathbb{F}_{2^4}$ generated by the irreducible polynomial $x^4 + x + 1$.

ShiftColumn, π, and MixRow, θ, transformations are some permutations to provide diffusion. ShiftColumn is similar to the ShiftRow transformation in AES:

$$\pi(A) = B \Leftrightarrow b_{ij} = a_{(i-j) mod 8,\ j},\ 0 \le i, j \le 7$$

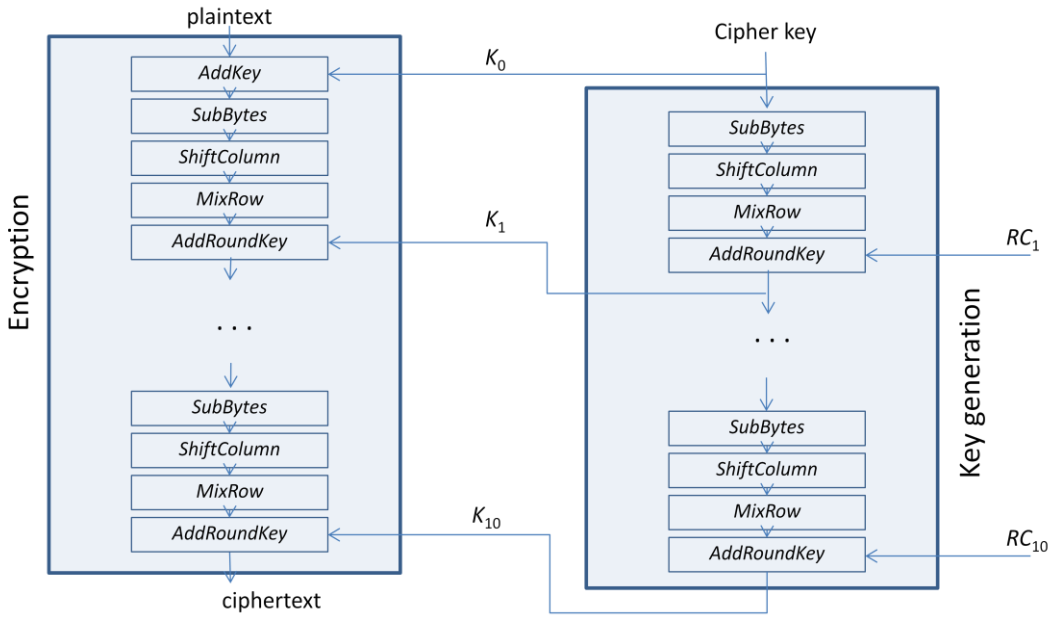

Figure 2-2 Whirlpool cipher W

For the MixRow transformation defined by the circulant MDS matrix *C*, Whirlpool uses the polynomial $x^8 + x^4 + x^3 + x^2 + 1$, (i.e., 0x11D) as the modulus.

$$\theta(A) = B \iff B = A \cdot C$$

AddRoundKey transformation σ is performed byte by byte. Instead of using a new key expansion algorithm, W employs a copy of the encryption algorithm with round constants to generate the round keys (Figure 2-2). Each round constant RC is an 8×8 matrix, where only the first row has nonzero entries.

2.1.3. Grøstl Hash Function

Grøstl is proposed as a candidate to a SHA-3 competition [38]. It is a byte-oriented iterated hash function, which is based on components of the AES and uses an SP-network. Grøstl's underlying function is built from two fixed, large permutations.

Grøstl is a so-called wide-pipe construction where the size of the internal state is twice as large as the size of the output.

The padding of an N-bit message in Grøstl starts by appending the bit 1, then $w = -N - 65 \bmod 1024$ bits of 0 are added. The last 64 bits are for the binary representation of the number $(N + w + 65)/1024$, which represents the number of message blocks in a padded message.

To be hashed, a message is processed block by block through two permutations, Q and P:

$$H_i = P(H_{i-1} \oplus x_i) \oplus Q(x_i) \oplus H_{i-1}$$

where x_i is a message block and H_i is a chaining variable (Figure 2-3).

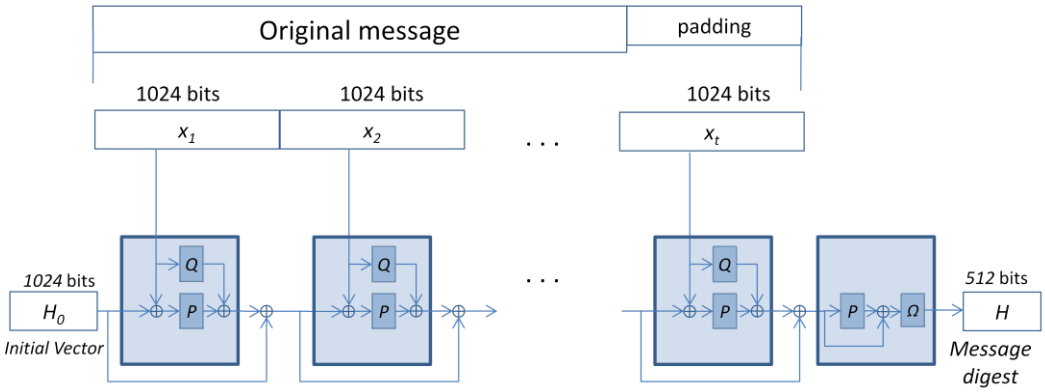

Figure 2-3 Structure of the Grøstl hash function

Initial vectors are predefined for each version of Grøstl, and for 512 bit adaptation it is equal to 00 … 00 02 00. Functions Q and P use the same structure as Rijndael, but they use round constants instead of keys. Q and P only differ in the description of round constants. The states of Q and P are much bigger than those of AES, so some operations are redefined.

Message blocks are mapped to a state matrix in the similar way as in Rijndael. The state matrix is 8×16. Number of rounds in both Q and P are equal to 14. Each round consists of four transformations, namely, MixBytes, ShiftBytes, SubByte, and AddConstant operations:

$$P(A) = \theta(A) \circ \pi(A) \circ \gamma(A) \circ \kappa(A).$$

No key scheduling is used in the algorithm but a constant matrix.

MixBytes operation, θ, is defined by left-multiplication with the circulant MDS matrix C via irreducible polynomial $x^8 + x^4 + x^3 + x + 1$, (i.e., 0x11B) as the modulus.

ShiftBytes moves all bytes by predefined number of positions. Let δ be a vector showing the number of positions to be shifted at each row. For Grøstl-512 this vector is

$$\delta = [0, 1, 2, 3, 4, 5, 6, 11]$$

SubBytes transformation, γ, uses the same S-box as in Rijndael and operates one byte at a time.

AddRoundConstant, κ, is simply defined by XORing a constant matrix D to a state matrix, i.e., $\kappa(A) = A \oplus D$. This is the operation, in which permutations P and Q differ, i.e., they have dissimilar round constants. For P the constant in round i has the value $d_{0,0} = i$, and for Q it is $d_{7,0} = i \oplus \text{FF}$, while other elements of constant matrices are equal to 0.

The output transformation Ω is the truncation of 512 bits of $P(H_t) \oplus H_t$.

2.1.4. Grindahl Hash Function

While Whirlpool hash function is based on Merkle-Damgård construction, Grindahl family of hash functions exploits the *Concatenate-Permute-Truncate* (CPT) design [51]. In this section we give a description of the 512-bit version of the algorithm.

Grindahl-512 creates a 512 bit message digest from a message of length at most $64(2^{64}-1)$ bits. This is due to the padding rule of the hash function, which is arranged as a single 1 followed by necessary number of 0's and last 64 bits enclose the number of message blocks (of 64 bits) in a padded message.

The initial state of the hash, s_0 is set to all zero string of length equal to 96 bytes, and being concatenated with the message block of 8 bytes, it becomes an input to the permutation function $P(S_i)$. The output of the function is then truncated, so that the state becomes 96 bytes again. For the last iteration, the truncation is omitted. Finally, eight blank rounds are applied, and the output of the hash function is truncation of last 512 bits, i.e., $H = trunc_{512}(\hat{s})$ (Figure 2-4)

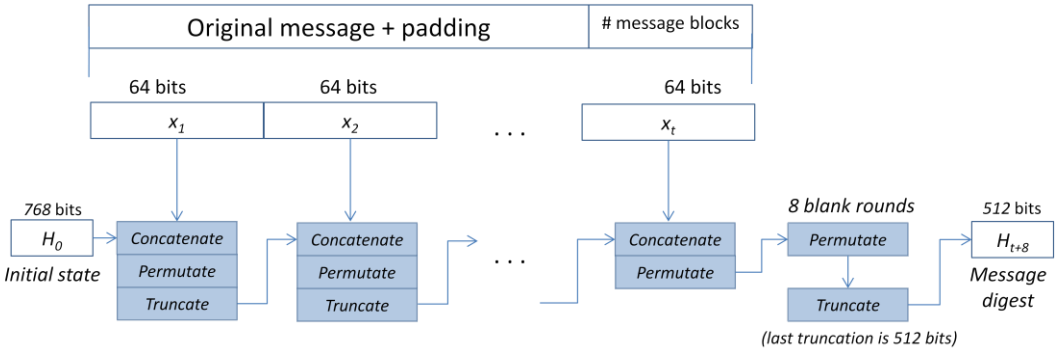

Figure 2-4 Structure of the Grindahl-512 hash function

The permutation $P(S_i)$ is defined as $P(S_i) = \theta(A) \circ \pi(A) \circ \gamma(A) \circ \kappa(A)$.

Again, we consider the extended state of the permutation function as an 8×13 matrix of 104 bytes, an input state *A* is converted to the output state *B*.

MixColumns. This transformation is defined as in the Rijndael specifications, but since the state is 8×13 matrix, the MDS matrix is redefined.

ShiftRows. This transformation cyclically shifts bytes a number of positions along each row. The vector δ for Grindahl-512 is

$$\delta = [1, 2, 3, 4, 5, 6, 7, 8]$$

SubBytes. The non-linear part of the permutation, exactly defined as the SubBytes function of Rijndael.

AddConstant. Instead of AddRoundKey transformation, Grindahl also uses a constant. This function is simply defined by XORing a constant matrix *D* to a state matrix, i.e., $\kappa(A) = A \oplus D$, and element $d_{7,12} = 01$, where 01 is the byte-wise hexadecimal value of 1.

2.2. Comparison of the Block Ciphers of the Three Hash Functions with the AES

Each hash function described in this chapter uses a cipher based on the AES as its chain function, which is proved to be resistant to attacks and expected to be a strong base for hash functions. However, block cipher based hash functions can be subject to attacks applicable to block ciphers [78]. For example, although there are some important differences between differential attacks on block ciphers and differential attacks on hash functions, basically the same techniques and reasoning apply. Both attacks require that a differential characteristic is found that has a sufficiently large probability.

It was shown in [24] that there are some nonrandom properties of the AES under integral cryptanalysis. As all three hash functions use some version of Rijndael as an underlying chain function, they may inherit this weakness. We examine some attacks on the underlying ciphers in Chapter 3.

Since Whirlpool has not been extensively studied or tested and uses an invertible cipher as its underlying chain function, more testing and research are needed to confirm its security [6].

Table 2-1 Comparison of block cipher W of Whirlpool, P and Q of Grøstl, P of Grindahl and AES

	W of Whirlpool	**P and Q of Grøstl**	**P of Grindahl**	**AES**
Construction principle	Merkle-Damgård	Wide-Pipe	Concatenate-Permute-Truncate	-
Block size (bits)	512	1024	832	128
Key size (bits)	512	-	-	128, 192, or 256
State matrix	8×8	8×16	8×13	4×4
Matrix orientation	Input is mapped row-wise	Input is mapped column-wise	Input is mapped column-wise	Input is mapped column-wise
Number of rounds	10	14	1	10, 12, or 14
Key expansion	W round function	-	-	dedicated expansion algorithm
Origin of round constants	Successive entries of the S-box	For P $d_{0,0} = i$, for Q $d_{7,0} = i \oplus$FF,	$d_{7,12} = 01$, Other entries are 0	elements 2^i of GF(2^8)
GF(2^8) polynomial	$x^8 + x^4 + x^3 + x^2 + 1$ (011D)	$x^8 + x^4 + x^3 + x + 1$ (011B)	$x^8 + x^4 + x^3 + x + 1$ (011B)	$x^8 + x^4 + x^3 + x + 1$ (011B)
Origin of S-box	recursive structure	multiplicative inverse in GF(2^8) plus affine transformation	multiplicative inverse in GF(2^8) plus affine transformation	multiplicative inverse in GF(2^8) plus affine transformation
Diffusion layer	right multiplication by 8x8 circulant MDS matrix (1,1,4,1,8,5,2,9) - mix rows	left multiplication by 8x8 circulant MDS matrix (2,2,3,4,5,3,5,7) - mix columns	left multiplication by 8x8 circulant MDS matrix (2,c,6,8,1,4,1,1) - mix columns	left multiplication by 4x4 circulant MDS matrix (2, 3, 1, 1) - mix columns
Permutation	shift columns, shift vector is [0,1,2,3,4,5,6,7]	shift rows shift vector is [0,1,2,3,4,5,6,11]	shift rows shift vector is [1,2,3,4,5,6,7,8]	shift rows shift vector is [0,1,2,3]

As regards Grøstl, it is one of the five finalists of the SHA-3 competition. The best known attack on Grøstl uses truncated differences and it is applied to 3 rounds of the algorithm [87]. Since the

algorithm construction is highly complex and the underlying block ciphers use 14 rounds each, it makes cryptanalysis hard. However, the speed of hashing under this design decreases significantly. In Chapter 4 we show the possibility of a fixed point attack to Wide-Pipe construction with semi-free IV.

Unlike Grøstl, the design principle of Grindahl [51] hash function is light and hashing speed is high. The 256 version of Grindahl hash function has already been reported broken by means of truncated differences in [75].

In Table 2-1, we present a comparison of the Rijndael cipher with the underlying ciphers of the considered hash algorithms.

2.3. Hash Function Cryptanalysis

2.3.1. Brief History of Hash Function Proposals

Primary hash functions were intended to be used in password protection schemes, and, in fact, they did not compress. One of the first practical hash algorithms [80] was proposed in 1974 and was based on polynomials over finite fields. In the same journal, Evans, Kantrowitz and Weiss described a hashing method based on a block cipher [34].

The role of hash functions increased with the development of public key cryptosystems, which provided a method for obtaining digital signatures. Rabin introduced the idea of applying a hash function to the message before signing, for increased performance and security [81]. Later, some constructions that utilize block ciphers as underlying functions were described [59], which are still in use today. Based on this work, IBM developed the MDC-2 construction [71], which produces a 2n-bit hash function using an n-bit block cipher.

The first dedicated publicly known hash algorithm was proposed in 1988 by Rivest, and was called MD2 in [43], [56].

The works of Damgård [26] and Merkle [69] independently described construction methods that strengthen hash functions. Based on the work of Merkle and Damgård, another design, an MD4 algorithm by Rivest [83], [84], [86] was proposed. Due to weaknesses found in the MD4, it was superseded by MD5 [85].

Table 2-2 Standard hash algorithm proposals

Hash functions	**Authors**
MDC-2	Meyer, Schilling, 1988
MD-2	Rivest, 1988
MD-4	Rivest, 1990
MD-5	Rivest, 1992
SHA-0	NIST, 1993
SHA-1	NIST, 1995
RIPEMD	H. Dobbertin, A. Bosselaers and B. Preneel, 1996
Whirlpool	V. Rijmen and P.S.L.M.Barreto. 2000
SHA3 competition finalists • BLAKE • Grøstl • JH • Keccak • Skein	2008 Jean-Philippe Aumasson Lars R. Knudsen Hongjun Wu The Keccak Team Bruce Schneier

In 1993, the U.S. National Institute of Standards and Technology (NIST) developed the Secure Hash Standard (SHA) [72] on the basis of MD4 and MD5. Two years later, NIST revised the hash standard [73]. The new algorithm was called SHA-1, and the first version is often named SHA-0. MD5 and SHA-1 were very popular hash functions, and they are still in widespread use.

The year 2005 was a breaking point in the cryptanalysis of hash functions, and it appeared that with the growth of computational power, the existing hash standards were not secure. Therefore NIST announced a competition for a new secure hash algorithm standard.

2.3.2. Chronology of Attacks on Hash Functions

Attacks on hash functions are aimed at finding collisions. The most consequential way to construct collisions is the use of slight modifications of classical differential cryptanalysis on block ciphers together with some specific methods.

The first weaknesses in MD4 were published in **1991** [15], where the possibility to construct a differential pattern for the last two rounds of the hash function was shown. MD5 was partly cryptanalysed in 1993 [16]. [8] first introduced a differential cryptanalysis with respect to modular addition and the attack was applied to MD5. The first collision in MD4 was found by Dobbertin in **1996** [32], [33] by describing the hash function as a large system of nonlinear

equations. He developed a method of applying constraints on the system such that it became possible to solve.

In **1998** Chabaud and Joux proposed a method, which exploit difference patterns in cryptanalysis of SHA-0 [20]. However, the runtime of the algorithm was too high to be carried out in practice. In **1999** Dean [30] showed in his Ph.D. thesis that fixed points of the compression function can be transformed into a long message second preimage attack on the Merkle-Damgård functions.

In **2002** Knudsen described an ***integral distinguisher*** for AES-like permutations [49].

The substantial progress in the cryptanalysis of hash functions started in **2004** with the improvement of Chabaud-Joux attack by Biham and Chen. They presented a new cryptanalytic method, in which they look at "the neutral bit" in the difference propagation. This method was called the ***neutral bit technique*** [10] and first was applied to find near-collisions of SHA-0. The same year Joux, Carribault, Jalby and Lemuet generalized Chabaud-Joux attack for iterated hash functions and applied the neutral bit technique to the full SHA-0 [41]. They obtained a practical collision by combining 4 differential patterns (colliding messages were 4 blocks each). All these attacks use XOR difference patterns. Also, the multi-collision attack of Joux [40] was developed using the technique combining neutral bit technique and Chabaud-Joux attack. The method was called a ***multi-block technique***, and it exploits *modular differences*, that is, differences with respect to integer addition usually modulo 2^n, rather than XOR-difference.

- Chabaud-Joux attack
- The neutral bit technique [10]
- Joux, Carribault, Jalby and Lemuet attack

In **2005** [91] applied a multi-block technique to MD4 and RIPEMD, using chosen message preimage attack, where they describe a method to derive a set of the sufficient conditions on the chaining values to ensure the differential path to hold in iterated hash functions. Then, the first collision attack on the full MD5 hash function was described [93]. The attack is based on three-step approach: find a proper differential path, obtain a set of sufficient conditions for the differential path to hold, and modify the message words to satisfy these conditions in the first round, thus, increase the probability of collisions. This method was called ***Message Modification Technique***. A practical collision attack on SHA-0 was published later the same year [94], and so was the first collision attack on SHA-1 [92], [95], [98].

Besides, improved collision attacks on MD4 and SHA-0 and for reduced round SHA-1 have been proposed [11], using the original multi-block technique. At the same time, Rijmen and Oswald reported an attack based on improvement of Chabaud-Joux cryptanalysis technique [82].

Preimage-style attacks on Merkle-Damgård strengthening by Kelsey et al. [46] was introduced the same year of 2005 and allowed to construct multi-collisions more efficient than the Joux attack. The technique used by Kelsey applied the multi-collision ideas of Joux to Dean's attack, and eliminated the need for finding fixed points in the compression function by *expandable messages* - patterns for producing messages of varying length, which all collide on the intermediate hash result immediately after processing the message. The method was further modified in [45] in the "Nostradamus attack" or "Herding attack" which makes it possible to commit to a hash value *h* and then to find a message that hashes to *h* with any desired prefix. For their attack, they have introduced the "diamond" structure which is reminiscent of a binary tree.

Further, in **2006**, the message modification method by Wang *et al*, especially its application to SHA-1, has then been studied by several authors, e.g. in [61], [19]. In March 2006, Black and Cochran combined the existing improvements on the Wang *et al.*'s attack such that colliding two-block message pairs could be generated in an average of 11 minutes on commodity PC hardware [13]. Vastimil Klima released a paper in which he introduced a new technique he called "***tunneling***" which reduced the search time necessary to find an MD5 collision to about 31 seconds on commodity hardware [48].

2007 came up with new modifications of Wang's attack in [18], [62], and a boomerang attack by Joux and Peyrin [42], which is based again on neutral bits tool and the tunneling technique.

In FSE 2007, a potential attack method was pointed out by an anonymous reviewer: the attacker does not look at the actual values of differences inserted in the bytes of the internal state, but only checks if there is a difference or not. This approach was called ***truncated-differences***.

The improvement on herding attack was presented in **2008** by [2], which can be considered as more flexible technique to build expandable messages, by choosing a prefix of the appropriate length and connecting it to the collision tree in the original Nostradamus attack.

As a response to all these attacks, and in particular the attacks on MD5 and SHA-1, NIST updated its suite of secure hash functions with the so-called SHA-2 family [74], but these hash functions were developed on the same principles as MD4, MD5, and SHA-1. It was then decided to initiate a public competition to develop a new set of hash functions, to be named SHA-3, to expand the existing Secure Hash Standard [74].

2009 started with new cryptanalytic methods on SHA-3 candidates. At FSE 2009, Mendel et al. proposed a new technique – the ***rebound attack***, for the analysis of hash functions [63]. The idea of the attack is based on truncated differences. The attack is divided into inbound and outbound phases. In the inbound phase, degrees of freedom are used, such that in the outbound phase

several rounds can be bypassed in both forward and backward directions. The attack was applied on round-reduced Whirlpool for up to 7.5 and Grøstl for up 6 rounds. Later this year a ***distinguishing attack*** was proposed on the full underlying block cipher of Whirlpool by improving upon the rebound attack and integral attack in several ways [55].

The same year, the complexity of collision search on SHA-1 was reduced to 2^{52} [60].

In **2010**, further improvement of the rebound attacks for AES-like permutations was presented [37]. The idea is to view two consecutive rounds of an AES-like permutation as the application of a so-called Super-Sbox (***Super-Sbox attack***). The same year, improved single key attack on AES was proposed, based on a new attack, ***biclique attack***, put forward in [53]. The attack was also modified for block ciphers and was used in the attack on AES [17].

Also, [1] generalize the herding attack by Kelsey and Kohno in **2011**.

The sketch of the main attacks is given in Table 2-3. Besides, our contribution as a diagram of the chronology and interrelations among the known attacks on hash algorithms is depicted in Figure 2-5.

Table 2-3 Attacks on hash functions

Hash function	**Author**	**Type**	**Complexity**	**Year**
MD4	Dobbertin	Collision	2^{22}	1996
	Wang et. Al	Collision	2^{8}	2005
MD5	dan Boer & Bosselaers	Pseudo-collision	2^{16}	1993
	Dobbertin	Free-start	2^{34}	1996
	Wang et. Al	Collision	2^{39}	2005
SHA-0	Chabaud & Joux	Collision (differential patterns)	2^{61} (theory)	1998
	Biham & Chen	Near-collision (neutral bit technique)	2^{40}	2004
	Biham et. al	Collision	2^{51}	2005
	Wang et. Al	Collision	2^{39}	2005
SHA-1	Biham et. Al	Collision (40 rounds)		2005
	Biham et. al	Collision (58 rounds)	2^{75} (theory)	2005
	Wang et. al	Collision (58 rounds)	2^{33}	2005
	Wang et. Al	Collision	2^{63} (theory)	2005
RIPEMD	Wang et. Al	Collision		2005

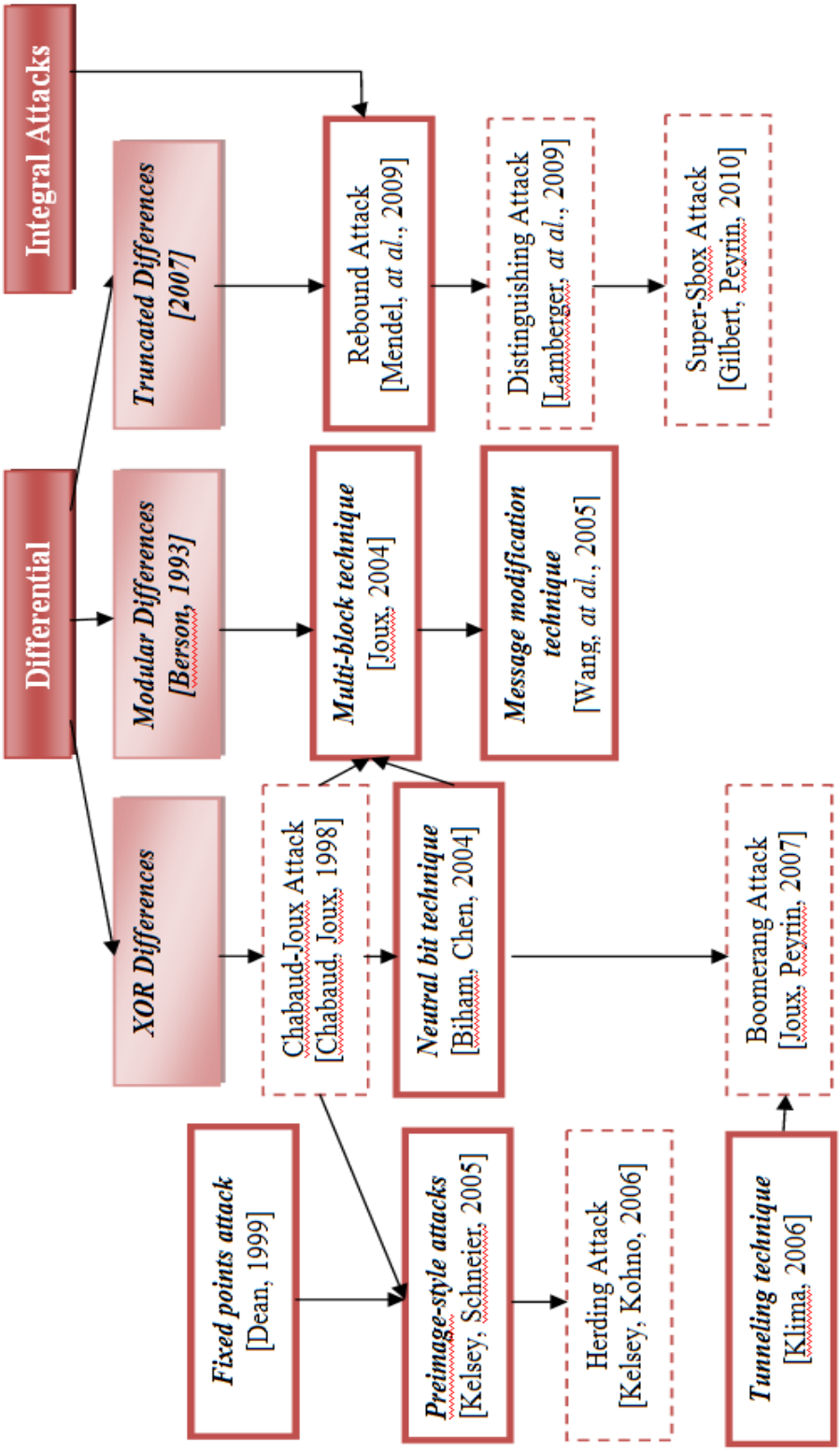

Figure 2-5 Chronology and interrelation of attacks on hash functions

2.3.3. Description of Some Well Known Attacks

General Description of the Integral Attack

Designers of the dedicated attack to Square cipher used the idea of integral cryptanalysis, without naming it [24]. Later, Knudsen called this technique "integral cryptanalysis" for the first time [50]. The technique considers a collection of 256 texts, each having 64 bytes, where 63 bytes are passive, and one byte spans all different 256 values of $\mathbb{F}_{2^8}$. It is discussed in [50], that after two rounds of encryption, the texts take all possible 256 values in each of the 64 bytes; and after three rounds of encryption, the sum of the 256 bytes in each position is zero. It is also shown that this weakness can be used to attack four rounds of Rijndael (or Square, Crypton ciphers) with small complexity. The attack with one active byte was called the 1st order integral cryptanalysis, and it was shown that it can be extended to 6 rounds by means of higher order integrals, where more than one byte span the field.

According to [49], W cipher also reveals the same behavior. So, at each output position, after the second round of W, each element of the field $\mathbb{F}_{2^8}$ occurs once over the collection of 256 input texts. Similarly, the sum of all 256 third round output texts will give zeros at each of 64 positions.

The attack is extended in [49] to five rounds by using the sum of bytes in the same position at the third round output, by means of the r^{th} *order integral* (i.e., there are r active bytes in the extended attack). They guess one key byte in the fifth round and four bytes in the fourth round at a time. Also, it is shown that the attack can be further extended to six rounds using the same sums.

General Description of the Rebound Attack

In the rebound attack, the internal cipher of the hash function is considered [63]. The cipher is regarded as the combination of three sub-ciphers, an inner part, as well as forward and backward paths. The result of the attack is the fixed point, obtained within one chain of the hash function.

Thus, the attack consist of two outbound and one inbound phase.

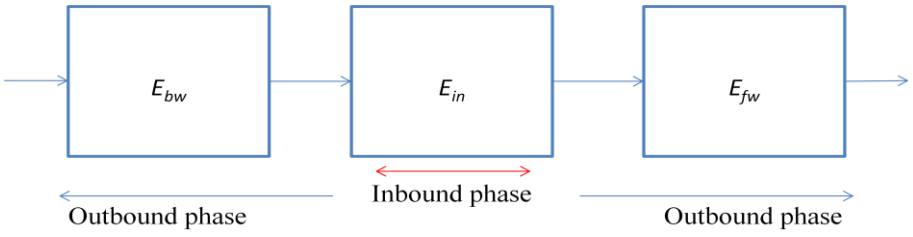

Figure 2-6 A schematic view of the rebound attack

For example, a single round of the W block cipher used in Whirlpool consists of the operations $AK \circ MR \circ SC \circ SB$, where *AK* denotes Add Round Key, *MR* – Mix Row, *SC* – Shift Column, and *SB* – SubBytes operations. Four successive rounds of *W* can then be expressed in terms of sub-ciphers as

$$W = W_{fw} \circ W_{in} \circ W_{bw}$$

where

$$W_{fw} = AK \circ MR \circ SC \circ SB \circ AK$$

$$W_{in} = MR \circ SC \circ SB \circ AK \circ MR$$

$$W_{bw} = SC \circ SB \circ AK \circ MR \circ SC \circ SB$$

In the attack, so called truncated differences are used, where the actual value of the difference is not considered and bytes are either active (i.e., the difference is not zero), or passive (meaning, the bytes cancel each other).The rebound attack consists of four steps:

Inbound phase

Step 1: the attack starts with 8-byte truncated differences of the diagonal form at the MixRows layer of round 2 and 3, and propagates forward and backward to the S-box layer of round 3.

Step 2: connect the input and output of the S-boxes of round 3 to form the three middle states $8 \rightarrow 64 \rightarrow 8$ of the trail, that is, we have 8 truncated after and before MixRows operation at round 2 and 3, respectively, and when propagated forward and backward, at the SubBytes layer of round 3 we have 64 truncated differences.

Outbound phase

Step 3: then one should extend the trail in both forward and backward directions to obtain the trail $1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 1$ through MixRows in a probabilistic way. These truncated differences are achieved due to structure of the operation ShiftColumn and MixRows.

Step 4: link the beginning and the end of the trail using the feed-forward of the hash function (which is XOR of the output of the previous and present chains).

If the differences in the first and last step are identical, they cancel each other through the feed-forward. The result is a collision of the round-reduced compression function of Whirlpool [63].

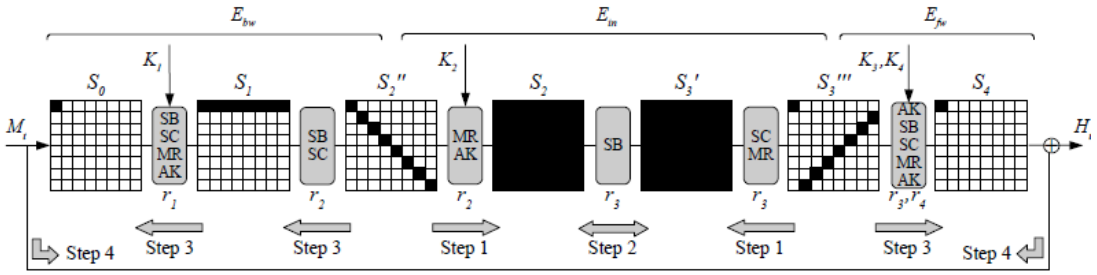

Figure 2-7 A schematic view of the attack on 4 rounds of the Whirlpool with round key inputs and feed-forward (reproduced from [63]).

General Description of the Distinguishing Attack

Lamberger *et al.* proposed improvements of the rebound attack in [65]. The new cryptanalysis was given the name of the distingushing attack, and it extends the rebound attack to 5.5 rounds in inbound phase and 7.5 rounds in outbound phase.

The main idea of the distinguishing cryptanalysis is to use two inbound phases instead of one, used in the rebound attack. The inbound phase of the rebound attack is considered as the meet-in-the-middle step, and it is applied at the rounds 1-2 and 4-5 of the hash function's underlying cipher.

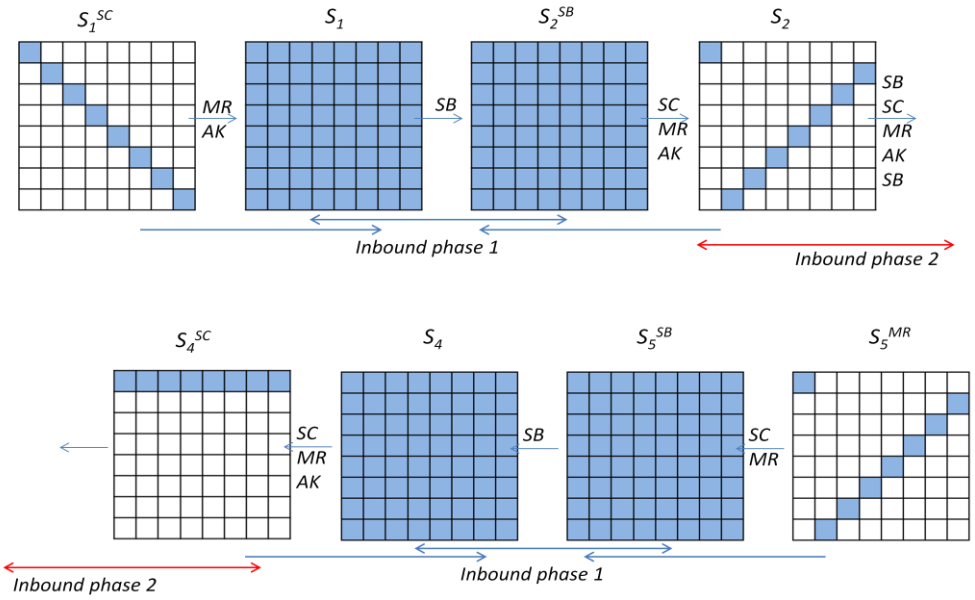

Figure 2-8 The inbound phase of the distinguishing attack

There are 8 active bytes at the beginning and end of the first part of the inbound phase, which are turned into 64 active bytes at the middle. The idea to match them is the same as in the rebound

attack. The second part of the inbound phase takes place between rounds 2 and 4. It is done by choosing the appropriate subkeys in the AddRoundKey operation. The authors suggest solving the following equation in order to connect 8 active bytes at round 4 output state S_4 of the cipher, defines by the first part of the inbound phase:

$$MR\left(SC\left(SB\left(MR\left(SC\left(SB\left(MR\left(SC(S_2^{SB})\right)\oplus K_2\right)\right)\right)\oplus K_3\right)\right)\right)\oplus K_4 = S_4$$

$$K_3 = MR\left(SC\left(SB(K_2)\right)\right)\oplus C_3$$

$$K_4 = MR\left(SC\left(SB(K_3)\right)\right)\oplus C_4$$

where C_i are round constants, used in key scheduling algorithm of the W cipher and S_2^{SB} stands for the state of the cipher at the SubBytes operation of round 2. For the choice of S_2^{SB} there are 2^{64} candidates, and so is for S_4. Besides, the subkeys K_2, K_3 and K_4 can be any of 2^{512} vectors, thus, authors expect to find 2^{64} solutions of the equation above.

The outbound phase of the attack is the same as in the rebound cryptanalysis, i.e., it obtains the trail $1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 1$ through MixRows in a probabilistic way by extending the trail in both forward and backward directions. Then one links the beginning and the end of the trail using the feed-forward of the hash function.

General Description of the Super-Sbox Attack

The further improvement of the rebound cryptanalysis was given in [37] and called the Super-Sbox attack. The Super-Sbox attack is based on the work of Daemen and Rijmen [21], [22] and [23], where they present the super s-box view of the two rounds of AES. In the internal state of the cipher, two rounds of the encryption are:

$$MC \circ SR \circ SB \circ AK \circ MC \circ SR \circ SB \circ AK\ .$$

The attack exploits the fact that when dealing with truncated differences, only the MixColumns operation does not behave deterministically, since the AddKey and SubBytes operations do not impact the value of differences and ShiftRows operation just shifts arrays of these differences.

Here, we give the notation, adopted for the W cipher of the Whirlpool hash function, where the operation MC (MixColumns) is rewritten as MR (MixRows) and SC instead of SR (which stand

for ShiftColumns and ShiftRows, respectively). Also, the places are interchanged in the design of the W cipher, so, two rounds are rewritten in the following way:

$$SB \circ SC \circ MR \circ AK \circ SB \circ SC \circ MR \circ AK$$

If one interchanged the place of and, this will not affect the encryption process, so, it is possible to rewrite rounds in the following way:

$$SC \circ SB \circ MR \circ AK \circ SB \circ SC \circ MR \circ AK$$

The middle part

$$Super\_SB = SB \circ MR \circ AK \circ SB$$

can be viewed as a row-wise application of the Super-sbox. Thus, the considered two rounds become

$$SC \circ Super\_SB \circ SC \circ MR \circ AK$$

The attack consists of two parts, which are called “controlled rounds” and, accordingly, “uncontrolled rounds”.

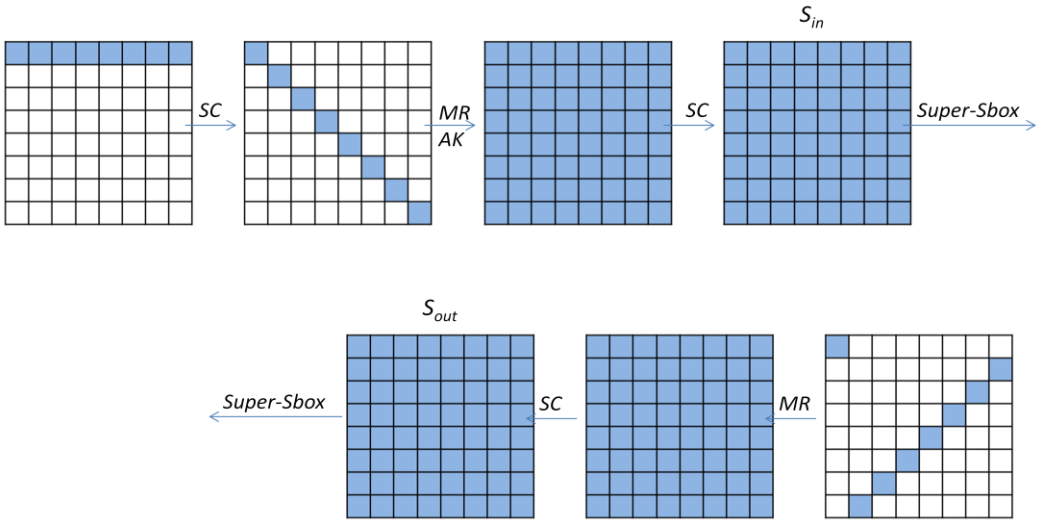

Figure 2-9 Systematic view of the Super-Sbox attack

Cryptanalysis starts with 8 bytes truncated difference, which spread to the whole cipher state after one round. Since the ShiftColumns does not affect the truncated differences, after this operation one has specified difference mask $\Delta = \Delta_1, \ldots, \Delta_r$ for r tuples input of the Super-sbox layer, where r is the number of columns. Then, for all the 2^{rc-1} pairs of input differences Δ_i one

should precompute Super-sboxes forward starting from the S_{in} position. Here, c is the is the number of bits in the truncated difference bytes (which is usually 8). The output differences are stored in the tables T_{i}. Applying the same reasoning, at the end of round 4 one goes backward and pre-computes the difference masks $\Delta' = \Delta'_1, \ldots, \Delta'_r$ for r tuples input of the Super-sbox layer backward at S_{out} and stores the differences in the table T'_{i}. If all the differences present in T_{i} are also present in T'_{i}, one can enumerate input difference Δ at S_{in} leading to an output difference masks Δ' at S_{out}.

The uncontrolled rounds path is fulfilled probabilistically, since the operations (mostly SubBytes and MixRows) followed after the controlled rounds do not allow to manage the cipher behavior. Author say that the path in uncontrolled rounds can be fulfilled with propability about $2^{-c(r-1)}$ at round 1 and round 5.

CHAPTER 3

INTEGRAL STRUCTURES FOR UNDERLYING ENCRYPTION ALGORITHM OF HASH FUNCTIONS

Despite the fact that AES has passed many cryptographic tests, in [50], it has been observed that ciphers exploiting substitution permutation networks (SPN's) demonstrate some nonrandom properties, which make them vulnerable to integral cryptanalysis. The hash functions Whirlpool [6] and ECHO [7] that are based on AES, reveal similar nonrandom behavior. Integral cryptanalysis is adapted to Whirlpool's underlying cipher W in [49], where it is observed that the cipher with number of rounds reduced from ten to six possesses systematic characteristics. Also, the same behavior was observed in the 256 bit version of the SHA-3 candidate Grøstl [38].

In this chapter, we construct integrals for AES based hash functions, namely, Whirlpool, Grøstl and Grindahl. For the special plaintexts having all passive bytes except a single active byte, whose value spans the whole field, a set of experiments is carried out and patterns of round outputs of the ciphers used in Whirlpool and in the 512 bit versions of the Grøstl and Grindahl hash functions, are investigated. First, at each round output, we study the frequency, with which elements occur at a particular position in the collection of 256 plaintexts with one active byte (1^{st} order integrals). In the second set of experiments, we count the number of high frequencies in the collection of all possible sets of 1^{st} order integrals. The difficulty of constructing integral structures on the 512 bit versions of the Grøstl and Grindahl arises from the fact that both algorithms utilize non-square state matrices, while the integral attack is applied on square state matrices, such as the state matrix of W cipher of Whirlpool. In sections 3.2, 3.3 and 3.4 we give the analysis of constructed integrals for the chosen hash algorithms; calculate the occurrence rate of each frequency and summarize the results in tables. In Section 3.5 we examine the effect of core operations on the states of the ciphers, in order to determine the operations leading to the systematic integral characteristics. Section 3.6 summarizes the conclusions related to this chapter.

3.1. Definitions

The *integral cryptanalysis* is proposed by Knudsen as a dedicated attack to Square cipher [24] and considers the propagation of sums of (many) values which are called the *integrals*. This section encloses definitions and description of the 1st order integrals adapted in [49] to Whirlpool hash function. The integrals are designed for the underlying block cipher W of the Whirlpool algorithm. We construct the same structures for 3 rounds of Grøstl and for 3 chains of Grindahl hash functions. We propose the notation below to simplify understanding of the integrals.

The definitions given in this section are adapted to the 512 bit state cipher (8×8 bytes) of the Whirlpool hash algorithm's W cipher. We start by redefining the double indices of each byte in the input states *A* and output states *B* of the cipher W to single indices

$$A = \begin{bmatrix} a_0 & \cdots & a_7 \\ \vdots & \ddots & \vdots \\ a_{57} & \cdots & a_{63} \end{bmatrix} \rightarrow \begin{bmatrix} b_0 & \cdots & b_7 \\ \vdots & \ddots & \vdots \\ b_{57} & \cdots & b_{63} \end{bmatrix} = B,$$

such that all input and output states can be expressed as 1×64 vectors.

We consider a collection of 256 texts, each having 64 bytes, where a byte is an element of $\mathbb{F}_{2^8}$. Each text defined in [24] differs from others in one byte and has equal values in all other bytes. 63 bytes with similar values are called *passive bytes*, and the byte, which spans all different values of $\mathbb{F}_{2^8}$ in the collection of 256 texts, is an *active byte*.

We define α_j as the specific 64 byte input of the W cipher, which has an active byte of value $alpha = \alpha \in \mathbb{F}_{2^8}$ at position *j*, and 0's at all other positions,

$$\alpha_j = (0, \dots, 0,\ \alpha,\ 0, \dots, 0)_{1\times 64}.$$

Since each byte of the output state denoted by b_k depends on the whole input state $\alpha_j = (0, \dots, 0, \alpha, 0, \dots, 0)$, b_k can also be expressed using the notation $b_k = b_k(\alpha_j)$, whenever needed.

For example, if the active byte is at position 0, we have

$$\alpha_0 = [\alpha \quad 0 \quad \cdots \quad 0] \rightarrow [b_0(\alpha_0) \quad \cdots \quad b_{63}(\alpha_0)] = B(\alpha_0).$$

The integral attack considers a collection of 256 plaintexts, which span all byte values $\alpha \in \mathbb{F}_{2^8}$ at position j and zero values at the remaining 63 positions. We collect the outputs of these 256 plaintexts in rows of a 256×64 matrix C_j:

$$C_j = \begin{bmatrix} b_0(0_j) & \cdots & b_{63}(0_j) \\ \vdots & \ddots & \vdots \\ b_0(255_j) & \cdots & b_{63}(255_j) \end{bmatrix} = \begin{bmatrix} B(0_j) \\ \vdots \\ B(255_j) \end{bmatrix}$$

The sum of bytes in a particular position is called an *integral* in [50]. In our notation it corresponds to the column sum $\sum_{\alpha \in \mathbb{F}_{2^8}} b_k(\alpha_j)$. An integral is named with respect to the position j of its active input byte as the j^{th} *integral*. Since one can choose 64 different positions $j = 0, \ldots, 63$ for the active input byte, and for each j, columns of C_j define 64 integrals at output locations $k = 0, \ldots, 63$, there are 64×64 integrals in total.

Let $C_{(j)}^{(k)}$ denote column vectors of C_j,

$$\begin{bmatrix} b_0(0_j) & \cdots & b_{63}(0_j) \\ \vdots & \ddots & \vdots \\ b_0(255_j) & \cdots & b_{63}(255_j) \end{bmatrix} = \left[C_{(j)}^{(0)} \; C_{(j)}^{(1)} \ldots C_{(j)}^{(63)} \right].$$

So, according to this notation, we can define the integrals in the following way: at each output position, after the second round of W, each element of the field $\mathbb{F}_{2^8}$ occurs once over the collection of 256 input texts. Similarly, the sum of all 256 third round output texts will give zeros at each of 64 positions; i.e., applying our notation to Knudsen's results [49] for W,

a) Round 2 outputs $b_k(\alpha_j)$ satisfy

$$\{b_k(\alpha_j) : \alpha_j \in \mathbb{F}_{2^8}\} = \mathbb{F}_{2^8},$$

i.e., each element of $\mathbb{F}_{2^8}$ appears once in the set $C_{(j)}^{(k)}$.

b) Round 3 outputs $b_k(\alpha_j)$ satisfy

$$\sum_{\alpha_j \in \mathbb{F}_{2^8}} b_k(\alpha_j) = 0,$$

i.e., although elements of $\mathbb{F}_{2^8}$ can be repeated and (a) is not satisfied, the sum of elements in the column $C_{(j)}^{(k)} = \{b_k(\alpha_j) : \alpha_j \in \mathbb{F}_{2^8}\}$ is still equal to 0.

Using this nonrandom property, the 4-round integral cryptanalysis for Square, Rijndael and W ciphers can be described as follows: since the integrals are known after 3 rounds of encryption, one can attack 4 rounds of the cipher by predicting the value of a single key byte at a time, going backward from the 4^{th} round, and checking if all of the 256 outputs after round 3 sum up to zero.

3.2. Whirlpool

In this section, we give round statistics of W cipher under the integrals, by analyzing the frequencies, with which elements occur in the round outputs of W cipher. At each round output, we study the frequency, with which elements occur at a particular position in the collection of 256 plaintexts. The aim is to detect a pattern, if any, in the statistical behavior of outputs.

We design a set of experiments in which the j'th position of the input state is activated. Over the 256 input texts with an active byte at the j'th position, we count the frequency of occurrence of $\mathbb{F}_{2^8}$ elements at each output position k. We define $f_{jk}(\beta)$ as the frequency, with which an element $\beta \in \mathbb{F}_{2^8}$ occurs in the vector $C_{(j)}^{(k)}$. Since the experiment is over the 256 input texts, the frequency $f_{jk}(\beta)$ varies in the interval, $0 \leq f_{jk}(\beta) \leq 256$.

The data used for this analysis is the round outputs of W, when the passive input bytes are 0's, i.e.,

$$\alpha_j = (0, \dots, 0,\ \alpha,\ 0, \dots, 0),$$

where $\alpha \in \mathbb{F}_{2^8}$. We first fix the position of the active byte j and count the frequencies $f_{jk}(\beta)$ with which an element β occurs at position k as α spans the field $\mathbb{F}_{2^8}$; that is, we count the number of β's in the set $C_{(j)}^{(k)}$. Then we repeat the same procedure for all $j = 0, \dots, 63$, changing the active byte position.

In Figure 3-1, we illustrate the frequencies $f_{00}(\beta)$ with which elements $\beta \in \mathbb{F}_{2^8}$ occur in $C_{(0)}^{(0)}$ at the third round outputs.

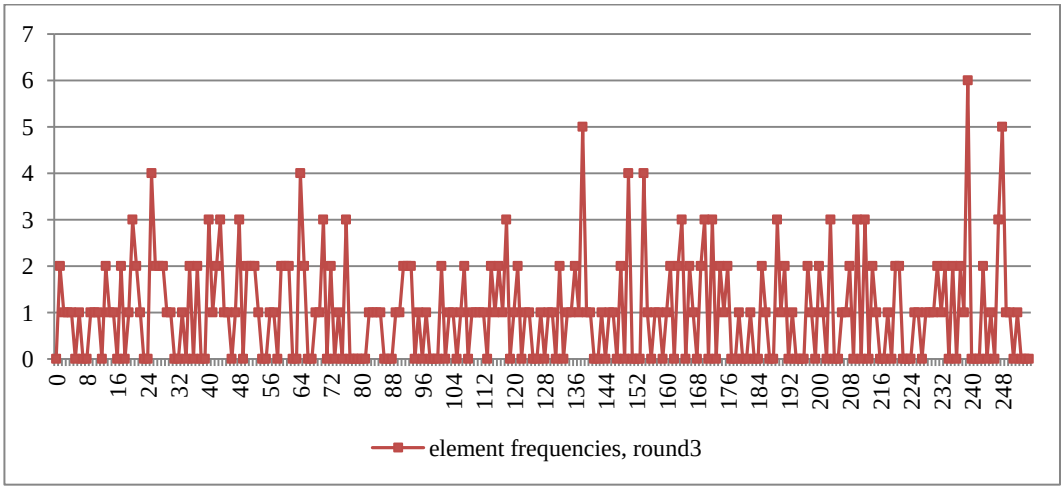

Figure 3-1 Frequencies of the elements in the outputs of round 3 at position k=0 as α=0,…,255

Figure 3-1 shows that in the set $C_{(0)}^{(0)}$ 83 elements occur once, 43 elements occur twice, and 15, 4, 2, 1 elements occur with frequencies 3, 4, 5, 6 respectively. The remaining 108 elements of the field do not appear in the set. The observation of the remaining positions $C_{(j)}^{(k)}$ for $k = 1, \dots, 63$ of round 3 outputs gives results in approximately the same range. Although elements of $\mathbb{F}_{2^8}$ appearing in the set do not create any obvious pattern, their sum $\sum_{\alpha_j \in \mathbb{F}_{2^8}} b_k(\alpha_j)$ still equals 0 at the third round output for all *k*.

Our next aim is to examine if any pattern, which would lead to a weakness, exists. Therefore, we count the number $N(f)$ of high frequencies in $C_{(j)}^{(k)}$ considering the collection of all sets, $k = 0, \dots, 63,$ and $j = 0, \dots, 63$.

Algorithm 1 Compute frequencies with which an element occurs in the given set

INPUT: Round outputs $B_{(j)}^{(k)} = \left(b_k(0_j), b_k(1_j), \dots, b_k(255_j)\right)$ of W cipher

OUTPUT: Frequencies f_{jk} with which an element β occurs at position *k* as α spans the field $\mathbb{F}_{2^8}$.

```
for j = 0,…,63                                  // run through all 64 integral structures
{
        for k = 0,…,63                          // run for all 64 bytes of the Whirlpool's state
        {
                for alfa = 0,…,255              // run for all 256 texts
                        if b_k(alfa) = beta  ⟹ f(beta)++
        }
}
```

The output of the Algorithm 1 is given in Table 3-1 and shows the number of frequencies $f_{jk}(\beta) = f$ over 64 possible active input positions, 256 possible active byte values and 64 output positions. Hence, the total number of elements examined in the analysis is over (256 field elements) × (64 active byte positions) × (64 output byte positions) =1048576=2^{20}, i.e., $0 \leq N(f) \leq 256 \times 64 \times 64$. The algorithm counts the frequencies $f_{jk}(\beta)$ in each vector $C_{(j)}^{(k)}$, and outputs the total number $N(f)$ of the frequencies at a specific value $f = f_{jk}(\beta)$, without making any distinction among different values of β, over all possible 64×64 vectors for $k = 0, \dots, 63$, $j = 0, \dots, 63$.

Since each element $\beta \in \mathbb{F}_{2^8}$ occurs once at the output of round 2, $f_{jk} = 1$ in each $C_{(j)}^{(k)}$ vector; hence, the second column of Table 3-1 shows the total number 1048576 of elements within the data set. Frequencies of the third round outputs show that the elements are now repeated, and

there are 8 elements (that may be different or the same) occurring 8 times and 1 element occurring 9 times.

Table 3-1 The number of occurrences of frequencies, f_{jk} in W cipher

$f_{jk}(l)$	round2	round3	round4	round5	round6	round7	round8	round9
0	0	384651	385236	390973	385309	391091	385215	384912
1	1048576	387138	386173	392632	385935	392550	385970	386775
2		193208	193225	196295	193585	196298	193524	193040
3		63785	64261	65062	64004	64877	64356	64090
4		16049	15948	16209	15971	16266	15766	16029
5		3087	3119	3136	3152	3260	3135	3129
6		580	526	550	538	528	525	529
7		69	80	90	70	71	79	61
8		8	6	12	11	17	6	10
9		1	1	0	1	2		1
10			1	1				

From the histogram in Figure 3-2, we can see that the probability that a certain value occurs once or never in the vector $C_{(j)}^{(k)}$ is much higher than it occurs 10 times. This is true for all 10 rounds of the W cipher. For the collection of 256 plaintexts with all passive bytes being 0, we can see some values of $b_k(\alpha_j)$ appearing in the vector 10 times at the outputs of round 4 and round 5. For instance, at the output of round 4, this most frequent element is $b_k(\alpha_j) = \beta = 62$, and it can be seen in the vector $C_{(3)}^{(49)}$. That is, in the collection of 256 plaintexts with an active input byte at position j=3, we have found ten different values of the active byte (namely, $\alpha = 62, 77, 94, 108, 114, 128, 173, 195, 222\ and\ 245$) that yield a 4th round output byte value $\beta = 62$ at the output position, k=49.

Figure 3-3 is sketched only for the purpose of demonstrating the invisible part of Figure 3-2, by using a different vertical scale.

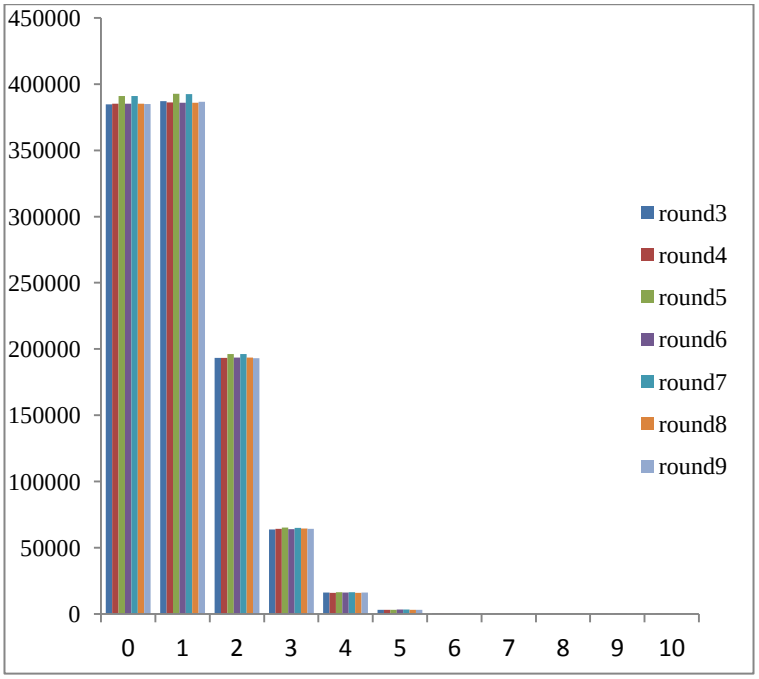

Figure 3-2 The number of occurrences of frequencies, f_{jk} (notice that for $f_{jk} > \mathbf{5}$, vertical resolution is not sufficient for the demonstration of $N(f)$)

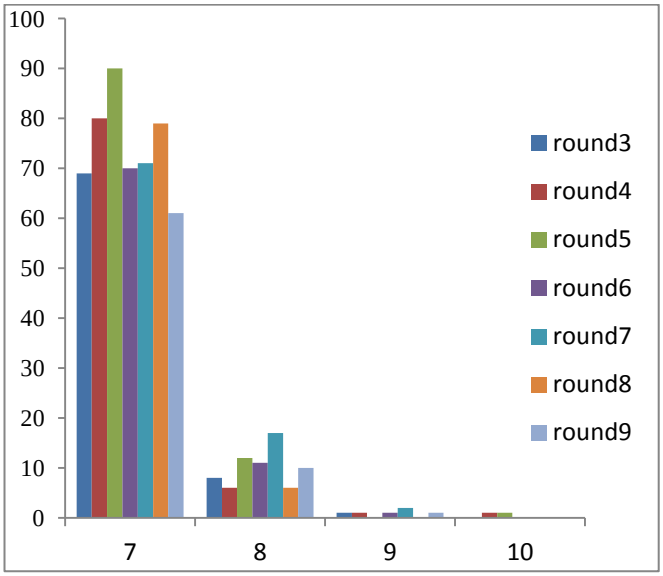

Figure 3-3 The number of occurrences of frequencies for $\mathbf{f_{jk} > 6}$.

Next, we look at the positions, where elements β appear with high frequencies. The aim of this analysis is to see the diffusion flow in the integral structures, and to see if the integrals can be constructed for a smaller collection of plaintexts we found all the elements in integrals with high frequencies, so, we find matches of positions, where the frequency of certain value is high, which would tell us if there is a pattern that could be used in building more powerfull integral structures

of the 1st order, i.e., we would like to see if the integrals can be constructed for a smaller collection of plaintexts. So, we are looking for such $b_k(\alpha_j)$'s, where $f_{kj} = f_{kj'}$ for some $j \neq j'$. Such $b_k(\alpha_j)$'s are found for $j = 28$ and $j' = 35$ for $k = 14$, and $f_{kj} = f_{kj'} = 8$. In Table 3-2 prepared for the matching position *k*=14 at the third round output, input positions *j* and values of active input bytes at these positions are given.

Table 3-2 Active bytes that yield the elements of frequency 8 at the same position of the 3rd round output

Output byte position *k*	Active input byte position *j*	Value of the output byte (β)	Values of the active input byte (α)
14	28	35	18, 30, 34, 104, 119, 157, 159, 190
14	35	27	3, 43, 78, 87, 117, 119, 153, 251

Further analysis employing $\mathbb{F}_{2^8}$ tools is needed to find the relation among these values.

3.3. Grøstl

Since the *P* and *Q* ciphers of Grøstl are also derived from the AES, their behavior is similar to W of the Whirlpool hash function. In the Grøstl hash function, the message blocks are mapped column wise, to the 8×16 bytes, non-square state matrices of the ciphers *P* and *Q* as in Rijndael. Since the states occupy 128 bytes, the matrix C_j defined in Section 3.1 is 128×256 bytes.

The data that we use for the frequency analysis in case of Grøstl hash function is the round outputs of *P* and *Q* ciphers. Plaintexts are $\alpha_j = (0, \dots, 0, \ \alpha, \ 0, \dots, 0)$, where α spans $\mathbb{F}_{2^8}$, and $0 \leq j \leq 127$ due to the state matrix size, thus there are 128 C_j matrices of size 128×256. We first fix the position of the active byte *j* and count the frequencies $f_{jk}(\beta)$ with which an element β occurs at position *k* as α spans the field $\mathbb{F}_{2^8}$; that is, we count the number $f_{jk}(\beta)$ of β's in the set $C_{(j)}^{(k)}$. Then we compute the corresponding integral and repeat the same procedure for $0 \leq j, k \leq 127$. So, frequencies are counted over (256 field elements) × (128 active byte positions) × (128 output byte positions) $=2^{22}$ cases; whereas 2^{14} integrals are found.

In Figure 3-4, it is seen that although round 3 output bytes at each position sum up to zero, elements of $C_{(j)}^{(k)}$ after round 2 do not span the elements of $\mathbb{F}_{2^8}$ in all positions. Such behavior of the matrix C_j is due to the non-square structure of the state matrices, which is 8×16 for both *Q* and *P*: the MixBytes operation affects 8 bytes of the state matrix at each round, which is followed by the ShiftBytes operation spreading these bytes into 8 columns, thus, 8 other columns remain unmodified.

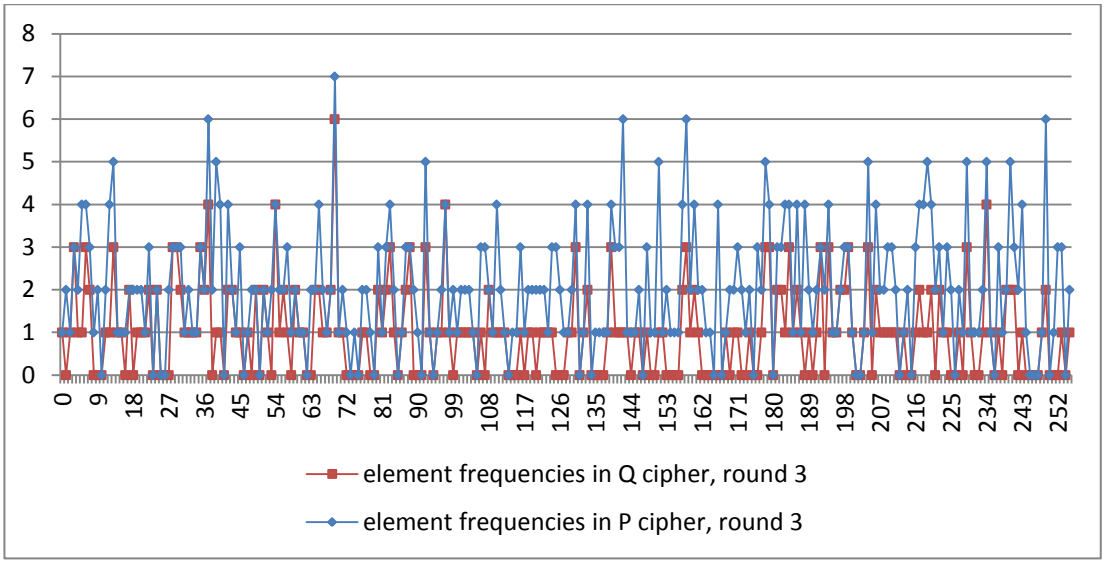

Figure 3-4 Frequencies of the elements in the P and Q ciphers outputs of round 3 at position k=0 as α=0,…,255

The maximal occurrence of elements in the set $C_{(0)}^{(0)}$ is 6 for the Q cipher and 7 for the P cipher. The range stays the same in the remaining positions $C_{(j)}^{(k)}$ for $k = 1, \dots, 127$ of round 3 outputs. Again, elements of $\mathbb{F}_{2^8}$ appearing in the set do not create an obvious pattern, yet, their sum $\sum_{\alpha_j \in \mathbb{F}_{2^8}} b_k(\alpha_j)$ equals 0 at the third round output for all k and thus, allowing 4-round integral cryptanalysis to be applied.

Our next aim is to examine if any pattern, which would lead to a weakness, exists. Therefore, we count the number $N(f)$ of high frequencies in $C_{(j)}^{(k)}$ considering the collection of all sets, $k = 0, \dots, 127,\ j = 0, \dots, 127$. In Table 3-3, it is seen that at the output of round 2 for both ciphers, out of 2^{22}, there are 2088960 cases that a field element is not observed, 2097152 cases that an element appears once and 8192 cases that an element occurs 256 times. Last row shows that the round 3 integrals at each output position are equal to zero, but round 2 integrals are not.

It may be easier to understand what Table 3-3 means, by normalizing the given frequencies with possible number of 128×128 input-output byte positions as in Table 3-4.

Table 3-4 shows that for fixed input-output positions, the k'th output byte of round 2 (i.e., $C_{(j)}^{(k)}$ elements corresponding to 256 different plaintexts with active byte at position j); either spans all field elements (hence their frequencies are 1), or it remains constant (so the corresponding field element has frequency 256), whereas other elements never show up (hence their frequencies are 0). So, in half of the 2^{22} cases, all field elements occur once as the value of the active byte spans $\mathbb{F}_{2^8}$; and in the other half, the output byte always takes the same value and no other field element can be observed.

Table 3-3 Number of occurrences of frequencies $f_{jk}(\beta)$ of field elements β at the round outputs of *P* and *Q* ciphers of Grøstl-512

$f_{jk}(l)$	Occurrence rate of $f_{jk}(\beta)$			
	Round 2		Round 3	
	Q	*P*	*Q*	*P*
0	2088960	2088960	1541399	1541845
1	2097152	2097152	1543429	1543554
2	0	0	774137	773609
3	0	0	256687	256072
4	0	0	63553	63835
5	0	0	12652	12713
6	0	0	2109	2365
7	0	0	297	268
8	0	0	35	42
9	0	0	5	1
10	0	0	1	0
…	…	…	…	…
256	8192	8192	0	0
All 2^{14} integrals	Sum up to 0	Sum up to 0	Sum up to 0	Sum up to 0

Table 3-4 Average number of occurrences of frequencies $f_{jk}(\beta)$ for fixed input-output byte positions

$f_{jk}(l)$	Round 2		Round 3	
	Q	*P*	*Q*	*P*
0	127.5	127.5	94.08	94.11
1	128	128	94.20	94.21
2	0	0	47.25	47.22
3	0	0	15.67	15.63
4	0	0	3.88	3.90
5	0	0	0.77	0.78
6	0	0	0.13	0.14
7	0	0	0.02	0.02
8	0	0	0	0
9	0	0	0	0
…	…	…	…	…
256	0.5	0.5	0	0
Sum of average occurrences	256	256	256	256

So, the elements of $C_{(j)}^{(k)}$ after round 2 (or 3) do not span the elements of $\mathbb{F}_{2^8}$ in all positions (if they did, only the frequency 1 would occur 256 times in

Table 3-4). This is due to the non-square structure of the 8×16 state matrices of P and Q: MixColumn operation affects only the 8 state bytes (in the column of the j^{th} active byte) at each round. This is followed by the ShiftRow operation spreading these bytes into 8 columns, thus, the remaining 8 columns would be unmodified. In [39] the explanation of such frequency behavior is given for the W cipher of the Whirlpool hash function.

Frequencies of the 3rd round outputs in Table 3-3 show that the elements are now repeated more than once. There are five elements (that may be different or the same) occurring 9 times in *P*; and one element, occurring 10 (or 9) times, in *P* (or Q). The maximal occurrence of elements in the set $C_{(0)}^{(0)}$ is 7 for the *P* cipher and 6 for the Q cipher. The range stays the same in the remaining positions $C_{(j)}^{(k)}$ for $k = 1, \dots, 127$ of round 3 outputs. Elements of $\mathbb{F}_{2^8}$ appearing in the set do not create an obvious pattern, yet, the integrals (i.e., sum $\sum_{\alpha_j \in \mathbb{F}_{2^8}} b_k(\alpha_j)$) are equal to 0 at the third round output for all *k* and *j*; thus, allowing integral cryptanalysis to be applied.

Next, we search for output byte positions, where the field elements appear with high frequencies. Whenever the frequencies of field elements at a certain output position *k* are high, it may be possible to find highly probable input state differences yielding 0 output difference at this specific *k*. In Table 3-5, the active input bytes yielding the same output at position *k*=34 of the third round output of *P*, are given with corresponding active byte positions. Notice that multiple active byte values $[\alpha, \alpha']$ = [135 $(10000111)_2$, 144 $(10010000)_2$] or [168 $(10101000)_2$, 191 $(10111111)_2$] yield the same input difference of 23 $(00010111)_2$, that enhances the differential probability that the output difference $\{b_{34}(\alpha_{18}) - b_{34}(\alpha'_{18})\}$ is equal to 0.

Table 3-5 Active bytes that yield the elements of frequency 8 at the same position of the 3rd round output of the P cipher

Output byte position *k*	Active input byte position *j*	Value of the output byte (β)	Values of the active input byte (α)
34	18	112	30, 135, 144, 149, 155, 168, 181, 191
34	35	1	1, 8, 30, 75, 96, 189, 192, 241

3.4. Grindahl

The difficulty in building the integral structures for the Grindahl hash function lies in the construction of the algorithm, since there is only one round of the cipher at each iteration, and

after each iteration the message blocks are overwritten in the first column of the state matrix. Also, as in the case of Grøstl hash function, the non-square state of the *P* cipher prevents the occurrence of the field elements regularly with frequency 1, after chain 2.

In Grindahl, the message blocks are mapped to the 8×13 state matrix of the *P* cipher. There is one round of *P* at each iteration, consisting of the four transformations (MixColumn, ShiftRow, SubByte, and AddConstant).

Integral cryptanalysis looks at the outputs of rounds 2 and 3, and uses feed-forward of the cipher to achieve the fixed points, but in Grindahl the permutation *P* has one round only, and the first column of the state is updated by the new message block, which has not been affected by the operations applied before. Also, there is no feed-forwarding in the CPT constructing. It could be possible to avoid unmodified bytes from having an effect on the integrals, if it were possible to shift the new message bytes to the last 5 columns of the state which were not previously affected by the ShiftRow operation. However, all injected new message bytes are spread to the 8 columns of the state matrix affected by the active bytes; therefore, the chance of building integral structures is decreased.

The second way to be able to implement the integral structures, is to choose the second and third 8-byte blocks of messages to follow the pattern of the outputs of each iteration. That is, the injected new message bytes are chosen exactly the same as the first 8 bytes of the chain outputs. Notice that this kind of message injection is equivalent to using the *P* cipher for 3 rounds, since the same message blocks are first truncated and then concatenated.

In Table 3-6, the occurrence rates of frequencies $f_{jk}(\beta)$ are calculated for such a situation, over (256 field elements) × (8 active byte positions) × (8×13 output byte positions) = 212992 cases; and (8×8×13=) 832 integrals are found for each input-output position. The integrals found for chain 1 and chain 2, follow two patterns; either all field elements occur once ($f_{jk}(\beta)$= 1 for all $\in \mathbb{F}_{2^8}$), or a single field element occurs 256 times and remaining elements never occur ($f_{jk}(\beta)$= 256 for one element and $f_{jk}(\beta) = 0$ for 255 elements). Both of these patterns yield zero integrals. To be precise, pattern 1 occurs in the outputs of chain 1 in (256 field elements) × (8 active byte positions) × (8 bytes affected by MixColumn and ShiftRow) = 16384 cases; whereas pattern 2 is separated as (8 active byte positions) × (7 × 13 unmodified bytes) = 768 cases for $f_{jk}(\beta)$= 256, and 768 × 255 = 195840 cases for $f_{jk}(\beta) = 0$. Since the single column affected by active byte at chain 1 is spread to 8 columns, pattern 1 occurs in chain 2 outputs in 16384 × 8 = 131072 cases. Although in outputs of chain 3, there is no obvious pattern; our experimental results with the plaintexts chosen as $\alpha_j = (0, \dots, 0,\ \alpha,\ 0, \dots, 0)$, show that all 832 integrals sum up to 0.

Table 3-6 Number of occurrences of frequencies, f_{jk} in 3 chains of Grindahl hash function

$f_{jk}(\beta)$	Occurrence rate of $f_{jk}(\beta)$		
	Chain 1	Chain 2	Chain 3
0	195840	81600	78303
1	16384	131072	78393
2	0	0	39226
3	0	0	13076
4	0	0	3202
5	0	0	655
6	0	0	123
7	0	0	14
…	…	…	…
256	768	320	0
All 832 integrals	Sum up to 0	Sum up to 0	Sum up to 0

Table 3-7 shows that when the input-output positions are fixed, the k^{th} output byte of chain 2 spans all field elements (hence their frequencies are 1) in more than half cases (to be precise, in 64 out of 104 output byte positions). For the remaining cases, the byte value does not change (so, the corresponding field element has the frequency 256), so other field elements do not appear (thus their frequencies are 0). So, for Grindahl, the same patterns seen in the analysis of Grøstl hash function are observed. The elements of $C_{(j)}^{(k)}$ after chain 2 follow the integral cryptanalysis scenario in 64 positions out of 104. After chain 3, bytes do not span the elements of $\mathbb{F}_{2^8}$, yet, integrals $\sum_{\alpha \in \mathbb{F}_{2^8}} b_k(\alpha_j)$ are equal to 0 at all output positions.

Table 3-7 Number of occurrences of frequencies, f_{jk} in 3 chains of Grindahl hash function

$f_{jk}(\beta)$	Chain 1	Chain 2	Chain 3
0	235.38	98.08	94.11
1	19.69	157.54	94.22
2	0	0	47.15
3	0	0	15.72
4	0	0	3.85
5	0	0	0.79
6	0	0	0.15
7	0	0	0.02
…	…	…	…
256	0.92	0.38	0.00
Sum of av. occurrences	256	256	256

The frequencies of 3 chain outputs show the same behavior as in the P and Q ciphers of the Grøstl hash function and are due to non-square structure of the state matrix. The results of the analysis show that Grindahl reveal non-random behaior, nevertheless, since blocks of the message are chosen so that they follow the pattern defined by the outputs of the previous chains, further investigations are not expected to give any valuable results.

3.5. Role of Cipher Operations in the Integral Attack for Whirlpool

In this section, we examine the effect of core operations of underlying AES based functions on the states of the cipher. Our aim is to understand the structure of the nonrandom behavior and investigate which of the operations lead to the systematic integral characteristics mentioned above. Experiments are accomplished on the W cipher of the Whirlpool hash function.

Role of the Passive Byte

In the tests above the value of the passive bytes are equal to zero. Here, we change the value of the passive bytes. In Table 3-8 we can see the results obtained when the passive bytes are 17's and 255's. As in the case of all zero passive bytes, round 2 outputs run all over the field $\mathbb{F}_{2^8}$. The frequencies at round 3 up to round 9 outputs remain in the same range as in the case of zero passive bytes.

Table 3-8 The maximal frequencies with which an output value occurs when passive bytes are different

Value of passive byte	Round							
	2	3	4	5	6	7	8	9
0	1	9	10	10	9	9	8	9
17	1	9	8	9	9	10	9	8
255	1	10	9	9	10	8	8	9

Role of the Chaining Mode

Whirlpool hash function uses Miyagichi-Preneel scheme, where a message is divided into blocks of length 512 bits and encrypted using all zero initial value as a key at the first chain. Being XORed with the previous cipher key and plaintext, ciphertext becomes the cipher key for the next block.

Since the key expansion is very similar to encryption, except for AddRoundKey operation, we repeat what we have done in case of plaintexts for different keys to examine their effect on the integrals. Hence, we change the original scheme of Whirlpool hash function to Matyas-Meyer-Oseas scheme, i.e., switch the roles of key and message in the W cipher. Thus, the collection of 256 texts becomes a key used to encrypt a single all zero plaintext.

The resulting integrals show the same effect, i.e., as the active input byte α runs all over the field $\mathbb{F}_{2^8}$, each byte of the second round output also runs over the field $\mathbb{F}_{2^8}$, and the integral at each position of round 3 output equals zero. This can be explained by the structure of the key expansion algorithm of the W cipher, which is the copy of the encryption algorithm itself with the only difference at AddRoundKey operation, where constant keys derived from the s-box of the cipher are used.

Besides, we detect the similar frequency behavior with the original case at round 3 outputs. Matches of positions k, where the frequency of a certain value is equal to 8, are found, i.e., $b_k(\alpha_j)$'s, where $f_{kj} = f_{kj'}$ for some $j \neq j'$. Such $b_k(\alpha_j)$'s are found for $j = 13$ and $j' = 63$ for $k = 40$. Results are given in Table 3-9.

Table 3-9 Active bytes that yield the elements of frequency 8 at the same position of the 3rd round output when the role of key and message is switched

Output byte position k	Active input byte position j	Value of the output byte (β)	Values of the active input byte (α)
40	13	215	41, 50, 53, 67, 97, 101, 162, 243
40	63	37	95, 128, 149, 184, 202, 224, 234, 254

Role of the SubByte Operation

At this step, we study the effect of the s-box on the integral structures. We replace the original s-box with that of AES. The result demonstrates that although the obtained values at given positions are different, the structure observed in the case of original s-box of W still holds. Similar situation can be monitored when we skip SubByte operation. It can be explained by the bijective structure of s-boxes. The mapping is one-to-one, which allows the structure of the input to be kept unchanged. The relationship between values obtained in these three cases considering the role of SubByte operation is of interest as the next step of the analysis.

Role of the MixRow Operation

MixRow layer of W uses the MDS matrix C that provides optimal diffusion. Input state of the MixRow is updated by

$$\theta(A) = B \iff B = A \cdot C,$$

thus, every output byte after θ transformation is a linear combination of eight input bytes. So, the operation transforms the row with the single active byte into a row of all active bytes.

When this operation is eliminated, we have a single position at the second round output, where all $\mathbb{F}_{2^8}$ elements are spanned. This position is the one, where the active byte of the plaintext is brought by ShiftColumn operation. For example, when the active byte of input is at position $j = 63$, we observe the value running all over the field $\mathbb{F}_{2^8}$ at position $k = 47$ after round 2, and $k = 39$ after round 3. The values at all other positions are the same in the collection of 256 output texts and their values are defined by other operations of W.

Role of the ShiftColumn Operation

The permutation layer makes each column of the state shift downwards circularly, except the first column. The shift operation moves a single byte from one row to another. Since MixRow operation results in a whole row containing active bytes, ShiftColumn operation distributes these bytes to all columns. After the ShiftColumn operation of the second round, we have one active byte at each column of the cipher state. MixRow operation of the second round converts each row with a single active byte into a complete row of active bytes. The result of the test shows that ShiftColumn operation has a great effect on the systematic integral characteristics. If the operation is eliminated, round 2 output values of vector $C_{(j)}^{(k)}$ do not span the field $\mathbb{F}_{2^8}$. Moreover, for all k's, which are not at the same row with the active byte of the input, vectors $C_{(j)}^{(k)}$ consist of all equal elements.

Role of the AddRoundKey Operation

As described in section 2.1.2, 10 rounds of W cipher consist of 4 core operations used 10 times. At the beginning, a message is XORed with a cipher key, then the round operations start.

Instead of using all zero vector as the cipher key, we change the initial key vector in order to obtain plaintexts, which do not have the same bytes at all passive positions, but still we do have one byte running all over the field $\mathbb{F}_{2^8}$ as an active one. We observe that systematic integral characteristics of the second and third round outputs remain the same. Although the key used in the study is totally random, it does not hamper the integral structures because of key expansion algorithm, used in W. The key and plaintexts have only one byte difference as they enter cipher rounds. They pass through similar operations with the only difference at AddRoundKey transformation σ, where the key is XORed with round constant, which consists of all zero entries except for the first row. As the result of previous operations of round 1, one has active bytes in the first row at each of 256 texts, and, after being XORed with the key, the texts in the collection still have a complete row of active bytes (and 0's at all other positions); so, conditions for integral cryptanalysis hold, i.e., if

$$C_{(j)}^{(0)} = \begin{bmatrix} b_0(0_j) \\ \vdots \\ b_0(255_j) \end{bmatrix}$$

is a vector of active bytes, then

$$\sigma[K_0]\left(C_{(j)}^{(0)}\right) = \begin{bmatrix} b_0(0_j) \\ \vdots \\ b_0(255_j) \end{bmatrix} \oplus K_0$$

again gives a vector of active bytes. Here K_0 is a key byte at position 0.

As the next step, we eliminate AddKey operation before rounds. When the key is random, we can observe the situation above with the only difference that round 1 outputs may not have 0's at rows with non-active bytes. In the case of all zero key, elimination of AddKey operation does not make sense since the cipher key is simply XORed with a message.

After that, we try to exclude the AddRoundKey operation, which also does not affect the scenario; i.e., the integral cryptanalysis is still applicable.

3.6. Conclusion

In this chapter we considered 1st order integral structures in 3 rounds of Whirlpool and Grøstl hash functions. Since the Grindahl hash function consists of only one-round P permutation, we have built the integral structures for 3 chains of the hash function, considering them as 3 rounds with messages specified for the cryptanalysis. In the integral structures defined by the integral cryptanalysis, we have investigated patterns at the round outputs of the underlying block ciphers. The occurrence rate of each frequency have been found and tabulated. Also, we have found the values and positions of active bytes yielding high-frequency byte values at the output. Although our results do not testify any weakness of ciphers used in the considered hash algorithms, it is quite possible that they can be adapted to other cryptanalytical techniques, such as the rebound attack on W [63]. We have then given the explanation of how the non-square states of the ciphers used in Grostl-512 and Grindahl change the scenario of forming the integral structures for these ciphers; but nevertheless, we have observed that construction of integrals to be used in an integral attack is still possible (i.e., the output byte values corresponding to 256 inputs with 256 different field elements at a specific active byte position still sum up to zero).

More specifically, for Grostl-512, we have seen that although round 3 output bytes at each position sum up to zero, elements of $C_{(j)}^{(k)}$ after round 2 do not span the elements of $\mathbb{F}_{2^8}$ in all positions. Such behavior of the matrix C_j is due to the non-square structure of the state matrices,

which is 8×16 for both Q and P: the MixBytes operation affects 8 bytes of the state matrix at each round, which is followed by the ShiftBytes operation spreading these bytes into 8 columns, thus, 8 other columns remain unmodified.

For Grindahl, the integrals found for chain 1 and chain 2, follow two patterns; either all field elements occur once ($f_{jk}(\beta)$= 1 for all $\in \mathbb{F}_{2^8}$), or a single field element occurs 256 times and remaining elements never occur ($f_{jk}(\beta)$= 256 for one element and $f_{jk}(\beta) = 0$ for 255 elements). Both of these patterns yield zero integrals.

In addition, we try to inquire into a question of the cipher operations' individual effects on integral structures. Since the roles of operations are carefully examined, we believe that the obtained systematic integral characteristics can be used to enhance the integral attack.

CHAPTER 4

ANALYSIS OF HASH ALGORITHMS CONSTRUCTIONS

In this chapter, we try to find some collisions for three chains of the Merkle-Damgård, Wide-Pipe and Concatenate-Permute-Truncate constructions. First, we consider the Whirlpool algorithm in three different Merkle-Damgård schemes, corresponding to different choices of the feed-forward path, that we call Scheme 1 (with no feed-forward path), Scheme 2 (with the feed-forward path $FF = H_{i-1}$) and Scheme 3 (with $FF = x_i$, and by swapping the roles of x_i and H_{i-1}, where x_i is the key and H_{i-1} is the plaintext of the W cipher). All these schemes yield modified forms of the original Whirlpool algorithm that uses $FF = x_i + H_{i-1}$ of the Miyaguchi-Preneel scheme.

Secondly, we consider the example of Grøstl for the Wide-Pipe construction and apply the same schemes, Scheme 1 and Scheme 2. Although Scheme 1 is a modified form of Grøstl, Scheme 2 can be considered as the original Grøstl hash function.

With regard to the Concatenate-Permute-Truncate construction, we attempt to find a collision for three chains of Grindahl only in Scheme 1, which is the original scheme for Grindahl.

In Merkle-Damgård Construction an adversary has a full knowledge of a cipher key, at least for the first chain of a hash function, since the initial vectors (IV) of the algorithms are fixed and not secret. In Section 4.1, we examine Scheme 1, Scheme 2 and Scheme 3 for Merkle-Damgård construction, using three rounds of the cipher W and the inverse cipher W^{-1} of Whirlpool hash function, as an example.

In Section 4.2, for the Wide-Pipe construction, we try to find fixed points on modified version of Grøstl (Scheme 1) and show impossibility of building the backward attack on the original version of the algorithm (Scheme 2) using three rounds of the ciphers *P* & *Q*.

As for the Concatenate Permute Truncate (CPT) construction; the backward analysis on the original version of the Grindahl (Scheme 1) and the role of Merkle-Damgård strengthening in this construction are discussed in Section 4.3. Section 4.4 concludes the chapter.

4.1. Definitions

As mentioned previously, the Merkle-Damgård hash function *h* with chain (or round) function *f* can be defined as follows:

$$H_0 = \text{IV}$$
$$H_\text{i} = f(x_i, H_{i-1}) \oplus FF\ , \qquad i = 1,2,\dots,t$$
$$h(X) = H_t$$

Here H_i are the *n*-bit intermediate variables (or chaining variables), x_i are the *n*-bit message blocks, the final hash is denoted by $h(X)$ and IV is the Initial Value. This also corresponds to the description of a finite state machine with initial state IV, input x_i and the next state function *f*. Also, a feed-forward operation can be defined in a chaining mode.

Thereby, there are two inputs of the chain function. These two inputs can be one of the followings: the message block x_i, the chaining variable H_{i-1}, XOR of the message block and the chaining variable $x_i \oplus H_{i-1}$ or some constant vector V. The two inputs of the chaining function and the feed-forward path can all be chosen from the 4-element set defined above and this choice determines the scheme of the hash function. Both Merkle-Damgård and Wide-Pipe constructions can be used in any of 4^3=64 possible schemes [79]. However, within these 64 schemes, some are trivial and, thus, not useful. One of the meaningful, but yet, not safe schemes is the Rabin scheme, where the message is a plaintext of the underlying cipher, the chaining variable is a key, and feed-forwarding is omitted (or, it can be considered as XORing with the all zero constant vector). Although hash functions that are built using this scheme are fast, they cannot be considered secure.

In Table 4-1 that we reproduce from [79], the attacks that can be applied to a particular scheme are listed, where D stands for a direct attack, P is a permutation attack, F is a forward attack, B is a backward attack and FP is a fixed point attack. The superscripts are used to count the number of a particular scheme and accordingly name it, the symbol "√" means that the scheme can be considered secure, while "-" means that the scheme is trivially weak or not applicable. Only four schemes indicated by check marks (schemes 3,10, 30 and 38) are considered to be safe, others are marked with different attacks D, P, F, B or FP; however, no proof or prescribed allegations were given about the applicability of these attacks. Later, it was proven in a black-box model that in addition to the four schemes indicated as secure in Table 4-1, there are eight more secure

schemes [14]. So, one can say that 12 schemes can be considered to be secure among these 64 schemes.

Table 4-1 Attacks on 64 different schemes (reproduced from [79])

Choice of feed-forward path	**Choice of the key**	**Choice of plaintext**			
		x_i	H_{i-1}	$x_i \oplus H_{i-1}$	V
V	x_i	-	B[13]	B[25]	-
	H_{i-1}	D[1]	-	D[26]	-
	$x_i \oplus H_{i-1}$	B[2]	B[14]	F[27]	F[41]
	V	-	-	D[28]	-
x_i	x_i	-	B[15]	B[29]	-
	H_{i-1}	√[3]	D[16]	√[30]	D[42]
	$x_i \oplus H_{i-1}$	FP[4]	FP[17]	B[31]	B[43]
	V	-	D[18]	B[32]	-
H_{i-1}	x_i	P[5]	FP[19]	FP[33]	P[44]
	H_{i-1}	D[6]	-	D[34]	-
	$x_i \oplus H_{i-1}$	FP[7]	FP[20]	B[35]	B[45]
	V	D[8]	-	D[36]	-
$x_i \oplus H_{i-1}$	x_i	P[9]	FP[21]	FP[37]	P[46]
	H_{i-1}	√[10]	D[22]	√[38]	D[47]
	$x_i \oplus H_{i-1}$	B[11]	B[23]	F[39]	F[48]
	V	P[12]	D[24]	F[40]	D[49]

Some of the schemes in Table 4-1 are named as:

c) (1) Rabin Scheme, where x_i is the plaintext of the underlying cipher and H_{i-1} is a key, with no feed-forward path

d) (3) Matyas-Meyer-Oseas Scheme, where x_i is the plaintext of the underlying cipher and H_{i-1} is a key, the feed-forward path is H_{i-1}

e) (10) Miyaguchi-Preneel Scheme (that Whirlpool uses), where x_i is the plaintext of the underlying cipher and H_{i-1} is a key, the feed-forward path is $x_i \oplus H_{i-1}$

f) (19) Davies-Meyer Scheme, where H_{i-1} is the plaintext of the underlying cipher and x_i is a key, the feed-forward path is $x_i \oplus H_{i-1}$

Weaknesses of the schemes or of the underlying chain function can lead to certain types of attacks. Backward attacks are based on the fact that the underlying block ciphers are reversible, and the security scheme is mostly based on the strength of chaining mode. The fixed point attack

is an attack that uses the so called fixed points of the algorithm, i.e., when the output at some point of the chain, with the given chaining variable and the message block, is equal to the output at some other point with another input, i.e.,

$$H_i = H_j \text{ for } i \neq j$$

or, if we speak of fixed point within one chain,

$$H_{i-1} = f(x_i, H_{i-1}) \oplus FF$$

For the hash function to be resistant to this class of attacks, the feed-forward path, as well as the inputs of the underlying block ciphers, should be chosen carefully.

When the feed-forward path *FF* is defined, we denote the output of a function *f* by y_i. Thus, the general model of a hash function can be descibed as

$$\begin{aligned} &H_0 = y_0 = \text{IV} \\ &y_\text{i} = f(x_i, H_{i-1}), \quad i = 1,2,\dots,t \\ &H_\text{i} = y_\text{i} \oplus FF, \\ &h(X) = H_t \end{aligned}$$

Although all three hash functions, analyzed in this work, are built using the iterated principle, only two of them, Whirlpool and Grøstl, can be described in terms of the schemes they exploit. The CPT construction does not use any feed-forward path, since the XOR operation used in feed-forwarding is conjectured to be not always secure.

4.2. Backward Attack on Modified Forms of Whirlpool Hash Function

4.2.1. Schemes Under Backward Analysis

In Merkle-Damgård Construction an adversary has a full knowledge of a cipher key, at least for the first chain of a hash function, since the IV of the algorithms are fixed and not secret.

We examine some of the schemes, using the W cipher of Whirlpool hash function, as an example. As all operations of the W cipher are invertible, a message processed with the same key gives the same output, i.e. the mapping is 1:1, which allows the use of the inverse cipher W^{-1}.

Below we will consider three different modes corresponding to different choice of feed-forward path, that we call Scheme 1 (with no feed-forward path), Scheme 2 (with the feed-forward path

$FF = H_{i-1}$) and Scheme 3 (with $FF = x_i$, and by using x_i as the key and H_{i-1} as the plaintext of the W cipher).

Backward Analysis of Scheme 1

Again, we denote the 512 bit input of the W cipher by x_i, where *i* represents the chain number of the Whirlpool hash function, and the output of the W cipher by y_i.

Our aim is to find a second preimage for a given hash value. We consider 3 rounds of the W cipher. First, we calculate a message digest for some fixed message. Let this message be all zero 1024 bit string. That means that we deal with three chains of the Whirlpool hash functions, first two chains being the message itself and the third is padding along with 256 bits showing the length of the original message (Figure 4-1).

The difficulty of calculating the reverse hash function at this point is the XOR operation before feeding the cipher of each chain to the next one as a key. We skip this operation. Let y_2 be our message digest. We use W^{-1} (y_2, y_1) cipher to calculate the block x'_2 of the message X'. The key y_1 (i.e. output of the first chain) is chosen arbitrarily. Now using the ciphertext y_1 and all zero initial vector of the Whirlpool cipher, we calculate a message x'_1. The message $x' = (x'_1||x'_2)$ gives the same message digest as the message *x* (Figure 4-1).

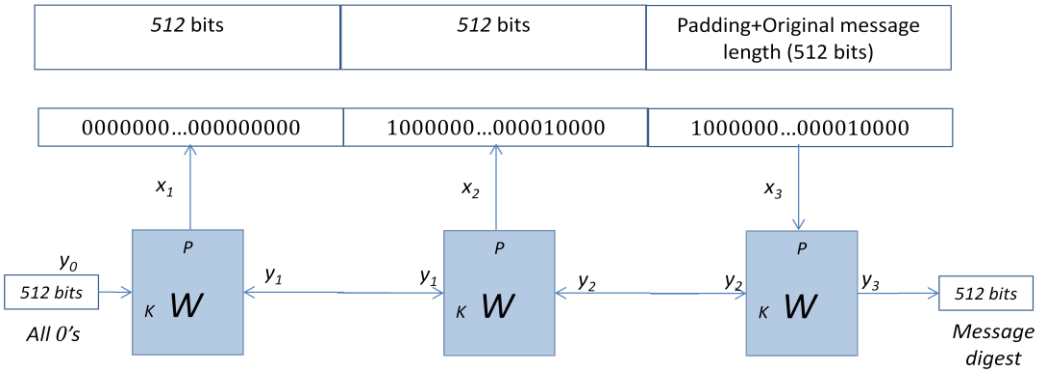

Figure 4-1. Scheme 1 for Merkle-Damgård construction

The scheme analyzed in this section is known as the Rabin scheme. Thus, we obtain a collision for the Whirlpool hash algorithm used in Rabin scheme. However, when we calculate the hash value for the message X', it differs from the one of the message *X* due to the padding rule applied to X' (i.e. X' proceeds through 3 chains of Whirlpool hash function); i.e., the scheme can be cyrptanalyzed by omitting one of the Merkle-Damgård strengthening rules and the resulting collision is a pseudo- collision.

Backward Analysis of Scheme 2

We start with the backward analysis of the scheme given in Figure 4-2. In this scheme, we only consider three chains, which are sufficient to construct multi-collisions. First, we run the hash function in the forward direction and calculate a hash value for some fixed message. Let us denote the output of the forward calculated message digest as $h(X_1)$, where X_1 is the message.

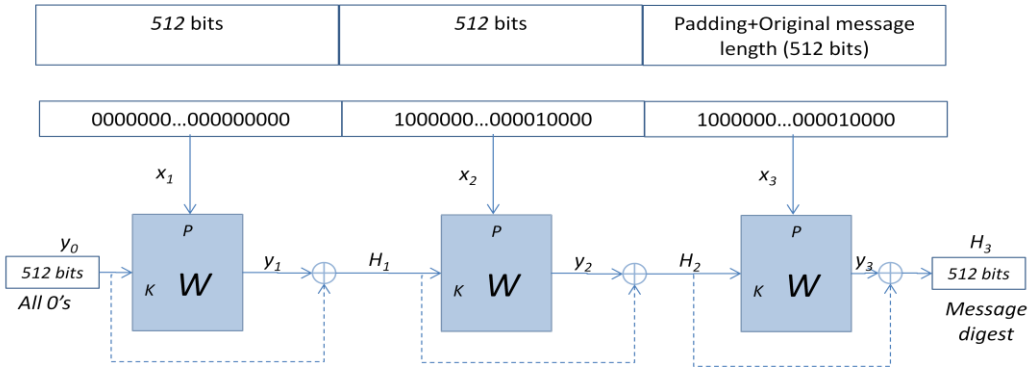

Figure 4-2 Scheme 2 for Merkle-Damgård construction

In order the hash function to be secure, Merkle and Damgård proposed that messages be padded with a padding that encodes the length of the original message. This is called *length padding* or *Merkle–Damgård strengthening*. Whirlpool hash function uses this strengthening along with a fixed initial vector. To construct a collision, we fix the input to the last chain, coming from the previous chain of the hash function. By fixing the value of H_2, we maintain the last chain of the hash algorithm, thus, the padding rule is not violated. Then we propagate in the backward direction, starting from H_2. We can assign the value of H_1 arbitrarily and calculate $y_2 = H_2 \oplus H_1$. Now, at the second chain, we have both inputs for W^{-1}, so we can calculate the value x_2 of the message. For the first chain we use the all zero vector as a cipher key (i.e., the fixed IV proposed by the design of the Whirlpool hash algorithm) and the value $y_1 = H_1 \oplus$ IV as a ciphertext. Since the value of H_1 can be any of 2^{512} vectors of length 512, we can construct a large number of multi-collisions. Below, the messages with the same hash value after the third chain are given:

$X_1 =$ 0000000000000000000000000000000000000000000000000000000000000000

$X_2 =$CDDDFCD58AB8F5DC72DAC1AB49B5A6A2C4955A92BBEC1133429C496D8488DEC597
D1771EEC4C489AF91CC190767A117EC7E261491C4EABA35CE40BE59E7

$X_3 =$EDCC3473FDBEF8F8DBD8649F3CF3F4ABD8A5ADE3375025EADE13318EA2C340DD8D5
5482561CAF2BAC5FC1CC6E3CFD9D016B52DABA25E85DB77BAE06972FC

X_4 =BC89D36E7583EFF1FDB6FB3FAE2FFB63A77FB1FEF1C4BF56CB1F8B8B41A3F5EC54C2A
18AC653D87FDCD3D432D64C66FA8A3BE9C0B73D649E7CE06A1A

X_5 =452A37A99E13BC4F8D28425816AFE7885542DE3F31E740A91FFCDB6AA139B2EFCC9F
D94F38795E0386FF0381C82876B2AFC61C69C64738DCB61437FD6FB4BC

X_6 =27C42D71DBE55C8BFECB96965113335C969C86A5D2A4CEF70428AAA249EC04B982D
693D507BB918A31CAE49390FAD81CAB991510D761FBACA28E2E4CC57CA

The message digest $h(X)$ is

BCA02AAC1C92FA4AEB68EFA4B863510C52E988F12534C74FA6E883D27B9959C8F212D4CB
2CB915D0E59E7EC84F3E7125BF842DCF2D7ADA1EB3D687329CB

Thus, we construct the multi-collision for scheme 2 and obtained 6 messages having the same hash output. The Merkle-Damgård strengthening is not violated and the IV remains the same as in the original design of the Whirlpool hash algorithm; hence, we find non-pseudo collisions. The scheme cryptanalyzed in this subsection is a version having a feed-forward path, H_{i-1}. In the original design of the Whirlpool hash function, this path is $H_{i-1} \oplus x_i$. In the backward analysis of the original version, the value of the plaintext x_i is unknown and in the backward analysis it is calculated employing the values of H_{i-1} (used as a key) and the ciphertext y_i. This makes finding collisions with the original design very hard.

Backward Analysis of Scheme 3

Our next analysis is of the scheme given in a Figure 4-3. In this scheme, we consider again three chains. In this scheme, the feed-forward path uses the message blocks instead of the output of the previous chain, and the role of the plaintext and the cipher key are interchanged. Since the key scheduling in a W cipher is independent on the encryption process, we can use a backward analysis in the Scheme 3.

The method of finding collisions is the same as in the previous section, i.e., we fix the value H_2, the output of the second chain of the hash function. Then we go backward starting from the value H_2. The value of the second chain message x_2 we choose arbitrarily. After XORing the message block x_2 with H_2, we run $W^{-1}(y_2, x_2)$ and obtain the input for the first chain.

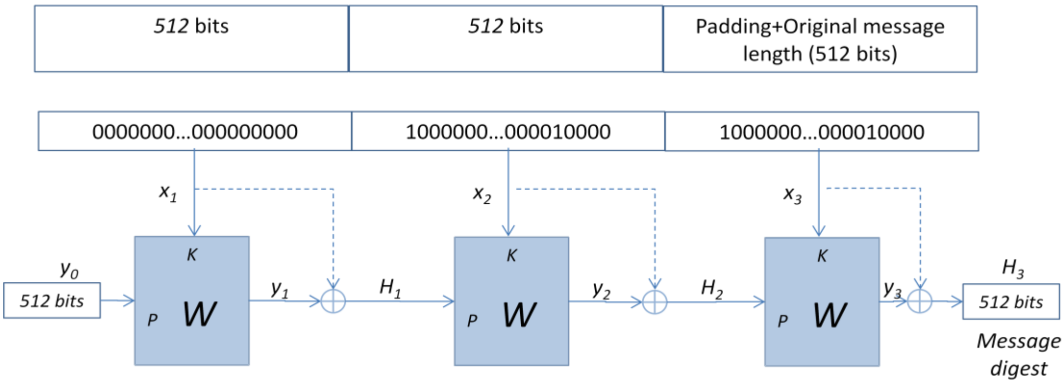

Figure 4-3 Scheme 3 for Merkle-Damgård construction

The problem here is that the IV of the first chain is the output of $W^{-1}(y_1, x_1)$; i.e., we cannot predict its value a priori, so, finding collisions for the fixed values of IV is hard. Nevertheless, for the scheme 3, we obtain a set of multi-collisions for different messages with corresponding IV's. The colliding messages with the respective IV's are given below:

X_1 =00000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000

X_2 =8000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000

with IV's, respectively:

IV_1 =0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000

IV_2 =A8B819A5B44FF071A4F3132691C13FEA03B145A2C16231EC54D62268D886E07BCB
E2B2248174F983444B1A5D0841A0CE06642ECA4B34B64868F6F478CE04790A

Give the same message digest $h(X)$ =:

12DADA46D8E886AD2FB80C82D72310599E5920F29CE29A574B7F0F490FA6F62CD6687
C70ABCDA9DDF75083B90C2746D7470D2F6A22C8E5DDF3B47A3CB0D754A5

4.3. Backward Attack on Modified Forms of Grøstl Hash Function

Wide-Pipe construction uses two different chain functions. In the case of Grøstl hash function, a message is processed through two ciphers, *P* and *Q*, in parallel, and the last chain utilizes only the cipher P together with truncation. Within the chains, the message block x_i is encrypted by the *Q* cipher, while the *P* cipher encrypts $h(x_{i-1}) \oplus x_i$. The chain output is defined as $\{Q(x_i) \oplus P(h(x_{i-1}) \oplus x_i)\} \oplus h(x_{i-1})$

Backward Analysis of Scheme 1

To construct a fixed point in the Wide-Pipe construction, i.e., to obtain $h(x_{i-1}) = h(x_i)$, we need to fix value of $h(x_i)$. If we try to find the output of the chain $h(x_i) = Q(x_i) \oplus P\big(x_i \oplus h(x_{i-1})\big) \oplus h(x_{i-1})$, we need $Q(x_i) = P\big(x_i \oplus h(x_{i-1})\big)$. Both *P* and *Q* do not use cipher keys but they use constants, which we do not have control over. Besides, the inputs of the ciphers *P* and *Q*, on their turn, both depend on the input message x_i, over which we do not have any control. That is, for *P* and *Q* to be equal, we need to find an input of the *P* cipher $x_i \oplus h(x_{i-1})$ leading us to some predefined output, which can be considered as breaking the 14 round *P* cipher. Thus, finding fixed points for Grøstl hash function with keeping the algorithm's original IV seems to be equivalent to the breaking the underlying block cipher.

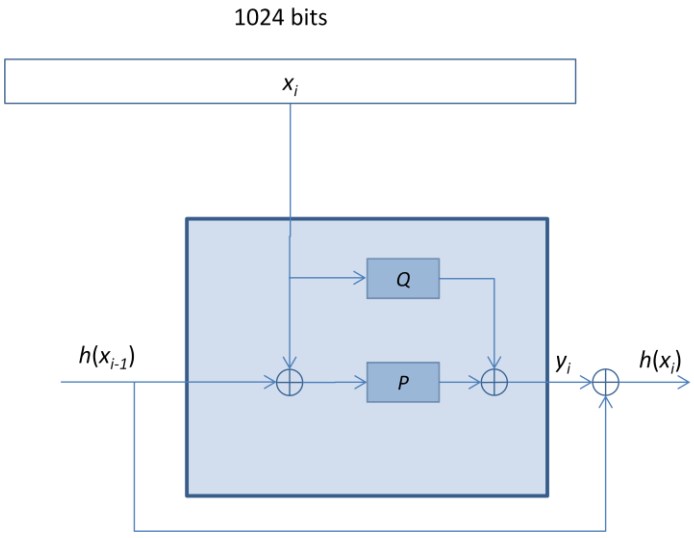

Figure 4-4. The chain function *f* of the Grøstl hash algorithm.

Let us consider the 3 chain message, including the padding. In the analysis, we use 3-round encryption of *P* and *Q*, although the analysis can be applied to 14 rounds as well. If we omit the XOR operation of the feed-forwarding, the same issue occurs. The output of the chain $h(x_i)$ would become $h(x_i) = Q(x_i) \oplus P\big(x_i \oplus h(x_{i-1})\big)$.

We start the backward analysis at the second chain output point $h(x_2)$. First, we calculate the hash value of the message X. Then, we go backward from the value of y_2. The inputs of the inverse ciphers P and Q take a single input, and since there is no key scheduling, the same input y_2 would give us the same output $x'_2 = x_2$. However, there is an XOR operation, which defines the inputs of the inverse ciphers P^{-1} and Q^{-1}. Let us denote the inputs of the P^{-1} and Q^{-1} by v_i and u_i, respectively, thus, $y_i = v_i \oplus u_i$. We are going to make use of this XOR operation defined within the chain function and obtain two different ciphertexts v'_i and u'_i such that $y_i = v'_i \oplus u'_i$. By calculating the inverse ciphers, we obtain two different messages X and X', yielding to the same hash value H, with two IV's. So, again we obtain pseudo-collisions.

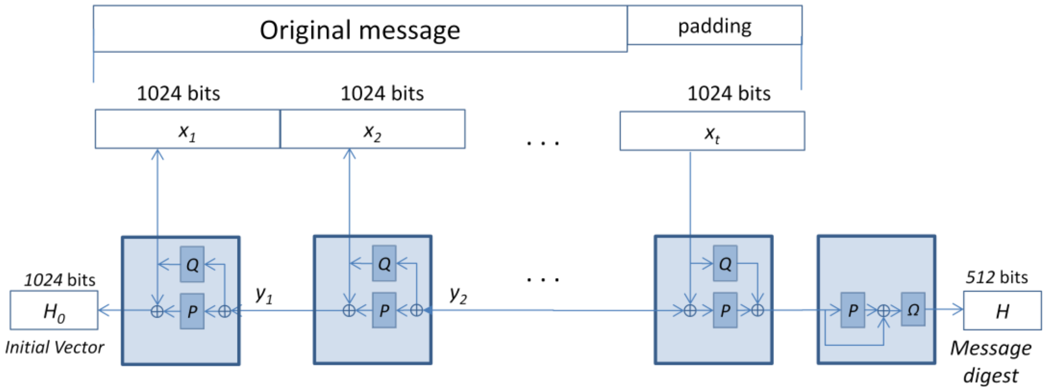

Figure 4-5 Scheme 1 for Wide-Pipe construction

The colliding messages with the respective IV's are given below:

X_1 =00000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000

X_2 =EC4D0B6B7109159E8ABDE1275D25A67C9E08B9F43381DEC9E08B9F43381DE23884
16E145A6962388416E145A6960B1E53D3894330B1E53D389433DD806A81BD8B4454DD80
6A81BD8B4454223266D4C47A1F7223266D4C47A1F75688E97850EBB4E05688E97850EBB
4E0EC48B1C351BA67F2EC48B1C351BA67F2ECE043BD47B4F2E8E043BD47B4F2C988E5
814CEB6759C988E5814CEB676723B16A6D50BA91DE23B16A6D50BA5ADE080678511019
608067851D2196DD32E9C3B738A63DD32E9C37538A632288B1B6434543542288B1124345
43545648D09FE18944F75648DE9FE18944F7EC48B16D38BA1E0ECAB8B16D38BA1E0EA0
3B145A2C16231EC54D62268

with IV's, respectively:

IV_1 =0000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000

IV_2 =67F714C5C911F14CC1CB15B9A6A7DC551EA8F8237FB3D698A0E7632CE3E830EC1
3ABA7B7DCFDCD495CB783F5875D9FDBA01BFF6CEC8B5B7C2B82DEA4D5798CCBCE5
5C26181FCEF67894A7AC47343AA4F4454E4C642D3BAC998F96E6AFAE2DAD2B874B3C
2CABAFA1867A26BDB480AFF53C39B6A7829457503A8AE62B9AB8A5FEF

Give the same message digest $h(X)$ =:

98A0E7632CE3E83098A0E7632CE3E830495CB783F5875D9495CB783F5875D95B7C2B82D
EA4D5795B7C2B82DEA4D579CEF67894A7AC47CEF67894A7AC472D3BAC

Backward Analysis of Scheme 2

To construct a collision on the original scheme, which we denote as Scheme 2 for Wide-Pipe construction, we again fix the input to the last chain of the chain function f', coming from the previous chain of the hash function, which contains the bits, defined by padding rule. Thus, by fixing the value of H_2, we maintain the last chain of the hash algorithm, thus, the padding rule is not violated. Also, by this we prevent the changes in the function f'' of the Grøstl. Then we propagate in the backward direction, starting from H_2. We can assign the value of H_1 arbitrarily and calculate y'_2 in two ways: either $y'_2 = H_2 \oplus H'_1$ or, by setting $H'_1 = 0$, choose $y'_2 = v'_2 \oplus u'_1$.

The steps of the inverse calculation are:

$$y'_2 = v'_2 \oplus u'_2$$

$$x'_2 = Q^{-1}(u'_2)$$

$$H'_1 = P^{-1}(v'_2) \oplus Q^{-1}(u'_2)$$

But the value of H'_1 depends on the next chain of the hash function:

$$H'_1 = y'_2 \oplus H_2$$

Although it seems that there is more freedom we could use in the analysis due to feed-forward, unlike in Whirlpool, we cannot use feed-forward in the backward analysis of the Grøstl. The value of H'_1 should be defined a priori, therefore, we cannot achieve the predefined value at the output of the *P*. Thus, finding fixed points for Grøstl hash function while keeping the algorithm's original IV seems to be equivalent to the breaking the underlying block cipher *P*.

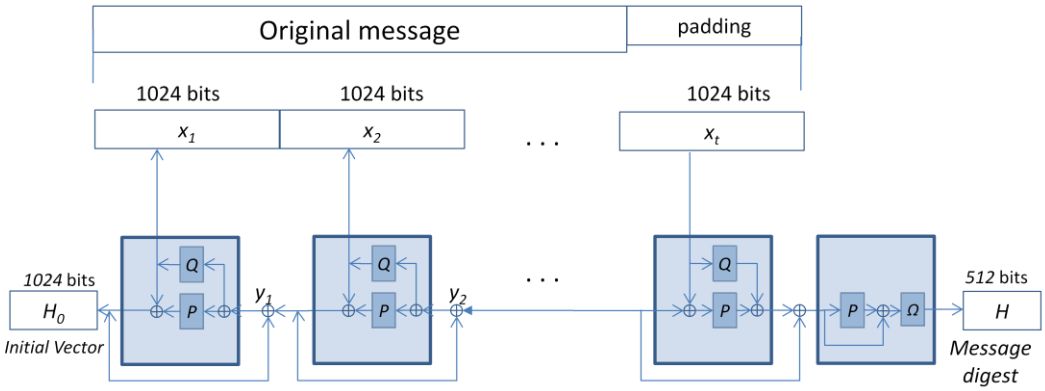

Figure 4-6 Scheme 2 for Wide-Pipe construction

Notice that Scheme 2 is the same scheme for the Wide-Pipe and Merkle-Damgård constructions, both take the plaintext input of the underlying chain function as the incoming message block and use a feed-forward path $FF = H_{i-1}$. Interestingly, finding collisions in Scheme 2 is not possible for Grøstl, while we are able to locate colliding messages with the original IV for the Whirlpool hash algorithm. Thus, this scheme, used in the Wipe-Pipe appears to be more secure.

The other way to look at the inputs of the underlying chain function is taking the message blocks as the key of the cipher, and the chaining variable as the message. In [5], there was given an new observation of the chain function of Grøstl as a cipher with *P* permutation being the encryption algorithm and *Q* being the key scheduling, which turns Scheme 2 into Davies-Meyer scheme, which is proven to be secure.

4.4. Backward Attack on Modified Forms of Grindahl Hash Function

Along with Concatenate Permute Truncate (CPT) construction, the Grindahl hash function uses Merkle-Damgård strengthening. In the backward analysis, the construction of the hash function allows inserting part of the message block in the truncation step, giving the cryptanalyzer the freedom of controlling the inputs of the function.

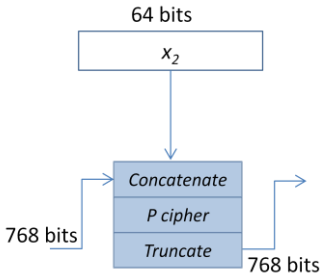

Figure 4-7 One chain of the Grindahl hash function

Truncation operation of the Grindahl cuts off the first 64 bits of the *P* permutation output of the algorithm. In the backward analysis, we are going to use this 64 bits (8 bytes) to construct collisions. First, we calculate the message digest for the given message X. As in the analysis of the Merkle-Damgård and Wide-Pipe constructions, we start the backward assembly of the colliding message by fixing the value H_2 of the second chain output. Thus, we preserve the padding rule and also the blank rounds of the Grindahl, where we do not have control over the hashing process. Remember, that the chain function of Grindahl is defined as follows:

$$S_i = x_i \parallel H_{i-1}, \qquad where\ S_i: \{0,1\}^w \times \{0,1\}^m \rightarrow \{0,1\}^{w+m}$$
$$\hat{s}_i = P(S_i), \qquad where\ \hat{s}_i: \{0,1\}^{w+m} \rightarrow \{0,1\}^{w+m}$$
$$H_i = trunc(\hat{s}_i), \qquad where\ H_i: \{0,1\}^{w+m} \rightarrow \{0,1\}^m$$

Before and after the P^{-1} inverse permutation, there is the truncation and concatenation, whose places should be exchanged in the backward analysis. The truncated 8 byte block of the P permutation is $\hat{s}_i$. So, first, we truncate the input of the P^{-1} inverse permutation with the 8 byte block $\hat{s}'_2$:

$$S'_2 = \hat{s}'_2 \parallel H_2,$$
$$\widehat{s'}_2 = P(S'_2),$$
$$H'_1 = trunc(x'_2),$$

Thus, with a different concatenated 8-byte block, we obtain another message X', hashing to the same value *H*. Messages such as:

X_1 =0101010101010101

X_2 =0A61BA11EDB55B68

with IV's, respectively:

IV_1 =0000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000

IV_2 =44DD1DC940A4014D3C77B4CD2E54BCF83DD7DB392904A4F987BDF8FE4D3B4FC
A30180ED5FEB47F1E49CA3D43534FDC8F22D584B510766F3560F6D451C69300670CECE
A6FB9C519C2122C660B96A1E2D92CB92892B8845EC6D980865391AF3912

Give the same message digest:

$h(X)$ =17C1A65FD87ED42BDADA46D8E886AD2FB80C82D72310599E5920F29CE29A574
B7F0F490FA6F62CD6687C70ABCDA9DDF75083B90C2746D7470D2F6A22C8E5DDF3

Besides the pseudo-collisions obtained above, since the P permutation of the Grindahl hash function consists of only one round of the AES modification, we also try to obtain collisions with the same IV's. However, if we take a close look at the permutation structure, we can see that the bytes we truncate before processing the state by P cipher, are located at the first row. MixColumn operation leaves the bytes on the same row, but the picture changes after the ShiftRows operation, and the inserted bytes now are on 8 columns of the cipher state matrix, and none of them is a part of the first row, which is truncated to become the message block x'_2. Indeed, all the changes brought by $\hat{s}'_2 \neq \hat{s}_2$ are in the IV's.

99	124	99	99	99	99	99	99	99	99	99	99	99
99	99	124	99	99	99	99	99	99	99	99	99	99
99	99	99	124	99	99	99	99	99	99	99	99	99
99	99	99	99	124	99	99	99	99	99	99	99	99
99	99	99	99	99	124	99	99	99	99	99	99	99
99	99	99	99	99	99	124	99	99	99	99	99	99
99	99	99	99	99	99	99	124	99	99	99	99	99
99	99	99	99	99	99	99	124	124	99	99	99	99

Figure 4-8 Positions of the inserted 8-byte block after ShiftRow

4.5. Conclusion

For the hash functions constructed using iterated structure, the alteration of the messages due to fixed points can become a major threat to the security of the algorithms. In Whirlpool, Grøstl and Grindahl, the underlying block ciphers are invertible, and this makes them more vulnerable to the attack, since mapping is 1:1. In this section, three chains of the modified forms of Whirlpool, Grøstl and Grindahl are cryptanalyzed in the backward direction, exploiting this vulnerability.

Since the given examples of the cryptanalysis are conducted on the modified versions, found collisions or pseudo-collisions cannot be considered as a real threat to the original versions of the hash functions. Rather, they should be understood as indicators of the importance of the missing feed-forward paths as compared to those of the more secure schemes.

Backward Analysis of Whirlpool

We consider the Whirlpool algorithm in three different Merkle-Damgård construction modes, corresponding to different choices of the feed-forward path, that we call Scheme 1 (with no feed-forward path), Scheme 2 (with the feed-forward path $FF = H_{i-1}$) and Scheme 3 (with $FF = x_i$, and by using x_i as the key and H_{i-1} as the plaintext of the W cipher).

- We obtain a pseudo-collision for the version of Whirlpool hash algorithm used in Scheme 1, where the feed-forward paths are removed. We call it a a pseudo-collision since Scheme 1 omits one of the Merkle-Damgård strengthening rules, which requires the IV of the algorithm to be fixed (in case of the Whirlpool hash function, IV is chosen as the all zero vector).
- We construct multi-collisions for Scheme 2 and give 6 messages having the same hash output as an example of multi-collisions. The Merkle-Damgård strengthening is not violated and the IV remains the same as in the original design of the Whirlpool hash algorithm.
- For Scheme 3, we obtain a set of a pseudo-multi-collisions for different messages with corresponding IV's.

Backward Analysis of Grøstl

We consider the Grøstl algorithm in two different Wide-Pipe construction modes, corresponding to different choices of the feed-forward path, that we call Scheme 1 (with no feed-forward path), and Scheme 2 (with the feed-forward path $FF = H_{i-1}$) as in the case of Whirlpool.

- We obtain two different ciphertexts v'_i and u'_i such that $y_i = v'_i \oplus u'_i$. By calculating the inverse ciphers, we obtain two different messages X and X', yielding to the same hash value H. Since two IV's are found, we obtain a pseudo-collison.
- For Scheme 2, we show that obtaining a collision by using backward analysis can be equivalent to breaking the underlying block cipher P. Notice that the chaining mode used in the original version of the Grøstl hash function can be considered as the Matyas-Meyer-Oseas scheme, where the feed-forward sequence is the message block (with $FF = x_i$), or it can also be considered as our Scheme 2 (with the feed-forward path $FF = H_{i-1}$). Since there is no key used in the underlying block ciphers of Grøstl, both x_i and H_{i-1} can be considered as the plaintext, hence they are interchangeable.

Backward Analysis of Grindahl

While preserving the Merkle-Damgård padding rule, we obtain two messages $X \neq X'$ and their IV's , hashing to the same value H. With different concatenated 8-byte blocks, we give the messages X and X' with related IV's; hence we obtain pseudo-collisons.

Showing the construction steps of the fixed points for CPT structure, we also demonstrate how to find real collisions with the same IV's.

CHAPTER 5

CONCLUSION

In this work, we mainly aim at finding practical collisions for block cipher based hash functions, Whirlpool, Grøstl and Grindahl, by means of some attacks on the underlying block cipher as well as the backward cryptanalysis of the overall hash algorithm. All the hash functions that we choose utilize modified versions of the AES as their chain functions and an iterated principle. However, they are built using dissimilar constructions: the Merkle-Damgård construction for the Whirlpool hash function, the Wide-Pipe construction for the Grøstl hash function, and the Concatenate-Permute-Truncate (CPT) construction for the Grindahl hash function.

We start by giving a comparison of the Rijndael with the underlying block ciphers, *W* for Whirlpool, *P* & *Q* for Grøstl and *P* for Grindahl, of the considered hash algorithms. We chronologically classify the known attacks on hash algorithms and highlight the interrelation between these cryptanalytic methods.

In order to analyze the integral structures to be exploited by the integral cryptanalysis, corresponding to the chosen plaintexts as defined by Knudsen, which have all passive bytes except a single active byte that spans the whole field; we carry out a set of experiments and investigate the patterns of round outputs of the underlying block ciphers *W*, *P* & *Q* and *P*. In order to understand how integrals sum up to zero, we conduct a set of experiments. First, we calculate the frequencies, with which elements occur at a particular position in each integral for a fixed active byte position. The same frequency analysis of integral structures is done for all possible sets of active bytes in each position.

In order to see the diffusion flow in the integral structures, and to understand if the integrals can be constructed for a smaller collection of plaintexts; we find all the elements in integrals with high frequencies, their positions, as well as the value and position of the corresponding active bytes. We tabulate these frequencies for each of the hash functions

To understand the nonrandom behavior of the cipher and to investigate which of the operations lead to the systematic integral characteristics, we show the effect of core operations of underlying AES based functions on the 1st, 2nd and 3rd round output states of the overall cipher.

In hash function constructions, an adversary has the full knowledge of initial vectors and round constants used as cipher keys, at least for the first chain of a given hash function, since the initial vectors, IV, of the algorithms are fixed and not secret. Besides, for the algorithms constructed using an iterated structure, the alteration of the messages due to fixed points can become a major threat to the security of the algorithms. In Whirlpool, Grøstl and Grindahl, the underlying block ciphers are invertible, and this is believed to make them vulnerable to the attacks.

We examine some schemes of the Merkle-Damgård construction, which are modified versions of the Whirlpool hash function, to analyze the overall algorithm for collision resistance. We consider three different Merkle-Damgård construction modes, corresponding to different choice of feed-forward paths, that we call Scheme 1 (with no feed- forward path), Scheme 2 (with the feed-forward path $FF = H_{i-1}$) and Scheme 3 (with $FF = x_i$, and by using x_i as the key and H_{i-1} as the plaintext of the W cipher). For Scheme 1, we obtain pseudo-collisions by omitting one of the Merkle-Damgård strengthening rules, which requires the IV of the algorithm to be fixed. For Scheme 2 we construct multi-collisions and, as an example, we give 6 messages having the same hash output. The Merkle-Damgård strengthening is not violated and the IV remains the same as in the original design of the Whirlpool hash algorithm. For Scheme 3, we obtain a set of pseudo multi-collisions for different messages with corresponding IV's.

As for the Wide-Pipe construction, we attain fixed points on the modified version of Grøstl; obtain collisions with the respective IV's and show the impossibility of building the backward attack on the original version of the algorithm. When Grøstl hash function is used in Scheme 1, making use of this XOR operation defined within the chain function of the Grøstl, we obtain two different ciphertexts v'_i and u'_i such that $y_i = v'_i \oplus u'_i$. By calculating the inverse cipher outputs, we obtain two different messages X and X', yielding to the same hash value H, with two IV's; so we obtain pseudo-collisions again.

For Scheme 2, we show that obtaining a collision by using backward analysis is equivalent to the breaking the underlying block cipher P. This result also shows that Scheme 2, when used in Wide-Pipe construction, is more secure than Merkle-Damgård construction.

With regard to the Concatenate Permute Truncate (CPT) construction, the backward analysis on the original version of Grindahl and the role of Merkle-Damgård strengthening in this construction are shown. We obtain two messages $X \neq X'$ and their IV's , hashing to the same

value *H*. With different concatenated 8-byte blocks, we give the messages X and X′ with related IV's, indicating pseudo-collisions. Showing the steps of constructing the fixed points for CPT construction, we also demonstrate how to find real collisions with the same IV's.

As future work, we believe that the statistics of the chosen hash functions that we study can be exploited in rebound attack, as well as in the distinguishing attack and Super-Sbox attacks, retrieved from the rebound cryptanalysis; or for finding multi-collisions in herding attacks. More precisely, the most expensive part of the rebound attack, inbound phase, where the truncated differences should be linked, can be done more efficiently by making use of the non-randomness, studied in the integrals. Among the 49 non-trivial schemes of the iterated hash functions obtained by different choices of the cipher inputs and the feed-forward path, we analyze only 3 schemes for the Merkle-Damgård construction and 2 for the Wide-Pipe. In the literature, there are few works that consider the confirmation or rejection of the security of a scheme, except for the black-box proof of the security of 12 schemes by Black, *et al.* in 2002. The proof is given with the assumption that the underlying chain functions are ideal, collision resistant functions, and no influence of chain functions' weaknesses are included into the cryptanalysis of schemes. Thus, this field of study has also importance as a future work.

REFERENCES

[1] E. Andreeva, C. Bouillaguet, O. Dunkelman, J. Kelsey. Herding, Second Preimage and Trojan Message Attacks Beyond Merkle-Damgård, presented at Selected Areas in Cryptography 2009.

[2] E. Andreeva, C. Bouillaguet, P.-A. Fouque, J. J. Hoch, J. Kelsey, A. Shamir, and S. Zimmer. Second Preimage Attacks on Dithered Hash Functions. In *Advances in cryptology - Eurocrypt 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2008.

[3] E. Aras. Analysis of Security Criteria for Block Ciphers, MS Thesis, Middle East Technical University, September 1999

[4] Bakhtiari, Savafi-Naini, Pieprzyk, Cryptographic Hash Functions: A Survey

[5] P.S.L.M.Barreto. An observation on Grøstl. An e-mail to the hash-forum. Details in http://www.larc.usp.br/~pbarreto/Grizzly.pdf

[6] P.S.L.M.Barreto, V.Rijmen. The Whirlpool Hashing Function. NESSIE Project, September 2000. http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html (2008/12/11).

[7] R.Benadjila, O.Billet, H.Gilbert, G.Macario-Rat, T.Peyrin, M.Robshaw, Y.Seurin, *SHA-3 Proposal: ECHO. Submission to NIST* (2008), available online at http://crypto.rd.francetelecom.com/echo/

[8] T. A. Berson. Differential cryptanalysis mod 2^{32} with applications to MD5. In R. A.Rueppel, ed., EUROCRYPT, vol. 658 of LNCS, pp. 71-80 (1993).

[9] E. Biham. New techniques for cryptanalysis of hash functions and improved attacks on Snefru. In Fast Software Encryption { FSE 2008, volume 5086 of LNCS, pages 444-461. Springer, 2008.

[10] E. Biham and R. Chen. Near-Collisions of SHA-0. In *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer-Verlag, 2004.

[11] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of SHA-0 and reduced SHA-1. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005.

[12] E. Biham, A. Shamir. Differential cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer. In Advances in Cryptology “CRYPTO'91”, volume 576 of LNCS, pages 156-171.Springer, 1991.

[13] J. Black and M. Cochran. *A study of the MD5 attacks: Insights and improvements*. In Fast Software Encryption, pages 262-277. SpringerVerlag, 2006.

[14] J. Black, P. Rogaway, and T. Shrimpton. *Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV*. Advances in Cryptology - CRYPTO '02, Lecture Notes in Computer Science, vol. 2442, pp. 320-335, Springer, 2002.

[15] B. den Boer and A. Bosselaers. *An Attack on the Last Two Rounds of MD4*. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO ’91, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 194–203. Springer, 1992.

[16] B. den Boer and A. Bosselaers. Collisions for the Compression Function of MD5. In T. Helleseth, editor, *Advances in Cryptology – EUROCRYPT ’93, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 1994.

[17] A. Bogdanov, D. Khovratovich, C. Rechberger. *Biclique cryptanalysis of the full AES.* ASIACRYPT'11**:** Proceedings of the 17th international conference on The Theory and Application of Cryptology and Information Security. Springer-Verlag, December, 2011

[18] C. De Cannière, F. Mendel, and C. Rechberger. *Collisions for 70-Step SHA-1: On the Full Cost of Collision Search.* In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2007.

[19] C. De Cannière and C. Rechberger. Finding SHA-1 characteristics: General results and applications. In *Advances in Cryptology - ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[20] F. Chabaud and A. Joux. Differential Collisions in SHA-0. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO ’98, Proceedings,* volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.

[21] J. Daemen, V. Rijmen, *The Design of Rijndael*. In: Information Security and Cryptography. Springer, Heidelberg (2002), ISBN 3-540-42580-2

[22] J. Daemen, V. Rijmen, *Two-Round AES Differentials*. Cryptology ePrint Archive, Report 2006/039 (2006)

[23] J. Daemen, V. Rijmen, *Understanding Two-Round Differentials in AES*. In: DePrisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 78–94. Springer, Heidelberg (2006)

[24] J. Daemen, L. Knudsen, and V. Rijmen. *The block cipher Square*. [ed.] E. Biham. Fast Software Encryption, Fourth International Workshop, Haifa, IsraeLNCS 1267, January 1997, pp. 149–165.

[25] I. Damgård, *Collision Free Hash Functions and Public Key Signature Schemes*. In Advances in Cryptography, EUROCRYPT 87, Springer LNCS 304, pp 203-212.

[26] I. Damgård. *A Design Principle for Hash Functions*. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1990.

[27] M. Daum and S. Lucks. *Hash Collisions (The Poisoned Message Attack): The Story of Alice and her Boss.* Rump Session of EUROCRYPT 2006, http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/, 2006.

[28] G. I. Davida, J.A. Hansen, *A Four-Component Framework for Designing and Analyzing Cryptographic Hash Functions.* IACR Eprint archive, 2007

[29] D. Davies and W. L. Price, *Digital signatures, an update*, Proc. 5th International Conference on Computer Communication, October 1984, pp. 845-849

[30] R.D. Dean: *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University (January 1999)

[31] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976, p. 650

[32] H. Dobbertin. *Cryptanalysis of MD4.* In D. Gollmann, editor, *Fast Software Encryption 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 53–69. Springer, 1996.

[33] H. Dobbertin. *Cryptanalysis of MD4*. *Journal of Cryptology*, 11(4):253–271, 1998.

[34] A. Evans Jr., W. Kantrowitz, E. Weiss. *A User Authentication Scheme Not Requiring Secrecy in the Computer*. Communications of the ACM, 17(8):437–442, 1974.

[35] H. Feistel, *Cryptography and Computer Privacy*, Scientic American, vol. 228, no. 5, pp. 15-23.

[36] M. Gebhardt, G. Illies, and W. Schindler. *A Note on Practical Value of Single Hash Collisions for Special File Formats*. NIST - First Cryptographic Hash Workshop, October 31-November 1, 2005.

[37] H. Gilbert and T. Peyrin, *Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations*, In Proc. FSE, 2010, pp.365-383.

[38] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schlaffer, and S. S. Thomsen. *Grøstl - a SHA-3 candidate*. Available online at http://www.groestl.info, 2008

[39] R. Ismailova, M.D. Yücel. *Statistics of the Whirlpool Hash Function under Integral Cryptanalysis*. Proceedings from ISC Turkey-2010, pages 221-227. Ankara, Turkey, 2010

[40] A. Joux. *Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions.* In Advances in Cryptology - CRYPTO 2004, volume 3152 of Lecture Notes in Computer Science, pages 306–316. Springer, 2004.

[41] A. Joux, P. Carribault, W. Jalby, and C. Lemuet. *Collisions in SHA-0.* Presented at the rump session of CRYPTO 2004, August 2004.

[42] A. Joux and T. Peyrin. *Hash Functions and the (Amplified) Boomerang Attack*. In Advances in cryptology - CRYPTO 2007, volume 4622 of Lecture Notes in Computer Science, pages 244–263. Springer, 2007.

[43] B. S. Kaliski Jr. *The MD2 Message-Digest Algorithm*. Internet Request for Comments (RFC) 1319, April 1992.

[44] S. Kavut, M. D. Yücel. *On Some Cryptographic Properties of Rijndael*. MMM-ACNS '01 Proceedings of the International Workshop on Information Assurance in Computer Networks: Methods, Models, and Architectures for Network Security, Springer-Verlag London, 2001

[45] J. Kelsey and T. Kohno. *Herding Hash Functions and the Nostradamus Attack*. In EUROCRYPT, volume 4004 of Lecture Notes in Computer Science, pages 183–200. Springer, 2006.

[46] J. Kelsey and B. Schneier. *Second Preimages on n-Bit Hash Functions for Much Less than 2n Work*. In Advances in cryptology - EUROCRYPT 2005, volume 3494 of Lecture Notes in Computer Science, pages 474–490. Springer, 2005.

[47] P. Kjellbergr, Zahle Torben U. *Cascade hashing*. Proceedings of the Tenth International Conference on Very Large Data Bases, August, 1984

[48] V. Klima. *Tunnels in hash functions: MD5 collisions within a minute*. Cryptology ePrint Archive, Report 2006/105, 2006. http://eprint.iacr.org/2006/105.

[49] L. R. Knudsen. *Non-random properties of reduced-round Whirlpool*. NESSIE public report, NES/DOC/UIB/WP5/017/1. 2002.

[50] L. R. Knudsen and D. Wagner. *Integral cryptanalysis*. FSE : Springer Verlag, 2002. LNCS 2365.

[51] L. R. Knudsen, C. Rechberger, and S. S. Thomsen. *The Grindahl Hash Functions*. In A. Biryukov, editor, Fast Software Encryption 2007, Proceedings, volume 4593 of Lecture Notes in Computer Science, pages 39–57. Springer, 2007.

[52] D. Khovratovich, *Cryptanalysis of Hash Functions with Structures*, Selected Areas in Cryptography, 2009, pages 108-125

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

[54] X. Lai and J. Massey. *Hash function based on block ciphers*. Berlin: s.n., 1992, Lecture Notes in Computer Science, Vol. 658, pp. 55-70.

[55] M. Lamberger, F. Mendel, C. Rechberger, V. Rijmen, M. Schläffer. *Rebound Distinguishers: Results on the Full Whirlpool Compression Function*. In Proceedings of ASIACRYPT'2009. pp.126~143

[56] J. Linn. Privacy *Enhancement for Internet Electronic* Mail: Part III – Algorithms, Modes, and Identifiers. Internet Request for Comments (RFC) 1115, August 1989.

[57] S. Lucks. *Design Principles for Iterated Hash Functions*. Cryptology ePrint Archive, Report 2004/253 (2004). [Online] http://eprint.iacr.org.

[58] S. Lucks. *A Failure-Friendly Design Principle for Hash Functions*. s.l. : Lecture Notes in Computer Science, Springer, 2005, Vols. 3788, pp.474-494.

[59] S. M. Matyas, C. H. Meyer, and J. Oseas. *Generating strong oneway functions with crypographic algorithm*. IBM Technical Disclosure Bulletin, 27(10A):5658–5659, 1985.

[60] C. McDonald, P. Hawkez and J. Pieprzyk. SHA-1 collisions now 2^{52}. In Rump Session of EuroCrypt '09. Annoucement, 2009.

[61] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. The impact of carries on the complexity of collision attacks on SHA-1. In *Fast Software Encryption - FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 278–292. Springer, 2006.

[62] F. Mendel, C. Rechberger, and Vincent Rijmen. Update on SHA-1. Rump Session of CRYPTO 2007, 2007.

[63] F. Mendel, C. Rechberger, M. Schlaffer, S. S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) FSE. LNCS, vol. 5665, pp. 260-276. Springer (2009)

[64] A.J. Menezes, P. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton, FL, 1997.

[65] R. C. Merkle. Secure Communications Over Insecure Channels. *Communications of the ACM*, 21(4):294–299, 1978.

[66] R. C. Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD Thesis, 1979

[67] R. C. Merkle, A Digital Signature Based on Conventional Encryption Functions," Advances in Cryptology|CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 369-378.

[68] R. C. Merkle, *A Certifed Digital Signature Scheme*. Advances in Cryptology|CRYPTO '89

[69] R. C. Merkle, *A fast software one-way hash function*. Journal of Cryptology, 3(1):43-58, 1990. Proceedings, Springer-Verlag, 1990, pp. 218-238.

[70] R. C. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1990.

[71] C. H. Meyer and M. Schilling. Secure program load with manipulation detection code. In *SECURICOM 88, Proceedings*, pages 111–130, 1988.

[72] National Institute of Standards and Technology/U.S. Department of Commerce. Federal Information Processing Standards Publication (FIPS PUB) 180. Secure Hash Standard, May 1993.

[73] National Institute of Standards and Technology/U.S. Department of Commerce. Federal Information Processing Standards Publication (FIPS PUB) 180-1. Secure Hash Standard, April 1995.

[74] National Institute of Standards and Technology/U.S. Department of Commerce. Federal Information Processing Standards Publication (FIPS PUB) 180-2. Secure Hash Standard, August 2002.

[75] T. Peyrin. Cryptanalysis of Grindahl. In *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 551–567. Springer, 2007.

[76] B. Preneel, *The State of Cryptographic Hash Functions*. Published in: Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aurhus, Denmark, July 1998, pages 158-182, Springer-Verlag London, UK

[77] B. Preneel, *Design Principles for Dedicated Hash Functions*. Published in: Proceeding, Fast Software Encryption, Cambridge Security Workshop Pages 71 - 82 Springer-Verlag London, UK,1994

[78] B. Preneel, *Analysis and Design of Cryptographic Hash Functions* (PhD Thesis)

[79] B. Preneel, R. Govaerts, J. Vandewalle. *Hash Functions Based on Block Ciphers: A Synthetic Approach.* New-York : Springer-Verlag, 1993. ISBN:0-387-57766-1.

[80] G. B. Purdy. *A High Security Log-in Procedure*. Communications of the ACM, 17(8):442–445, 1974.

[81] M. O. Rabin. *Digitalized Signatures and Public Key Functions as Intractable as Factorization.* Technical Report 212, MIT, 1979. Available: http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-212.pdf (2008/09/17).

[82] V. Rijmen and E. Oswald. Update on SHA-1. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2005.

[83] R. L. Rivest. The MD4 Message Digest Algorithm. Internet Request for Comments (RFC) 1186, October 1990.

[84] R. L. Rivest. *The MD4 Message Digest Algorithm*. In A. Menezes and S. A. Vanstone, editors, *Advances in Cryptology – CRYPTO '90, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311. Springer, 1991.

[85] R. L. Rivest. *The MD5 Message-Digest Algorithm*. Internet Request for Comments (RFC) 1321, April 1992

[86] R. L. Rivest. *The MD4 Message-Digest Algorithm*. Internet Request for Comments (RFC) 1320, April 1992.

[87] M. Schläffer. *Updated Differential Analysis of Grøstl*. Available at http://www.groestl.info/analysis.html January, 2011

[88] B. Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C, second edition, John Wiley & Sons, Inc., New York, NY, 1996.

[89] M. Stevens, A. K. Lenstra, and B. deWeger. Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In *Advances in Cryptology - EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007.

[90] D.R. Stinson, Cryptography: Theory and Practice. CRC Press, Inc., Boca Raton, FL, 2000

[91] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.

[92] X. Wang, Y.L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[93] X. Wang and H. Yu. How to break MD5 and other hash functions. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[94] X. Wang, H. Yu, and Y.L. Yin. Efficient collision search attacks on SHA-0. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

[95] X. Wang, A. Yao, and F. Yao. New Collision Search for SHA-1. Presented at the rump session of CRYPTO 2005, August 2005. http://www.iacr.org/conferences/crypto2005/rumpSchedule.html.

[96] M. V. Wilkes. *Time-Sharing Computer Systems*. Macdonald and Jane's, 1968.

[97] R.S. Winternitz, *Producing a one-way hash function from DES*, Advances in Cryptology, Proceedings Crypto'83, D. Chaum, Ed., Plenum Press, New York, 1984, pp. 203-207.

[98] H. Yu, X. Wang, A. Yun, and S. Park. Cryptanalysis of the full HAVAL with 4 and 5 passes. In *Fast Software Encryption - FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 89–110. Springer, 2006.

APPENDIX A

STATISTICAL ANALYSIS OF BLOCK CIPHER BASED HASH FUNCTIONS

In this appendix, statistical properties of the three hash functions, the Whirlpool, Grøstl and Grindahl, are studied in terms of the weight and distance of message digests in section A.1. The analysis of correlations between messages and message digests are given in section A.2. Also, in this section there is analysis of correlation between inputs and round outputs of underlying block cipher of hash algorithms. Since P permutation of Grindahl hash function consists of one round only, the tests are done for W of Whirlpool and P and Q of Grøstl hash functions.

Section A.3 is about the analysis of the Whirlpool hash function. Since Grøstl and Grindahl use the s-box of AES, which satisfies the criteria of avalanche, strict avalanche and bit independence within very small values of relative absolute errors [44], in section A.3.1 the s-box of the Whirlpool hash function is analyzed and compared with those of AES finalists. Besides, since there are several schemes proposed under Merkle-Damgård constructions, in this section statistical analysis of the Whirlpool hash function under different hashing modes and use of two different s-boxes are presented.

A.1. Statistical Analysis of Weights and Distances

The *Hamming weight* of a vector *f*, denoted by $\mathrm{wt}(f)$, is the number of ones in the vector. If f and g are functions on V_n, then $\mathrm{d}(f,g) = \mathrm{wt}(f \oplus g)$ is called the *Hamming distance* between f and g. Since the message digest length is 512 bits for all three hash functions, the expected weights of the outputs are about 256 under the assumption of random distribution, which should be satisfied for a hash algorithm with good diffusion properties.

The choices of the test vectors are made according to the message block inputs for each of the hash algorithms. Since the length of the input message is different for the three hash functions considered at this work (512 bits for Whirlpool, 1024 bits for Grøstl and 64 bits for Grindahl), the

general data used in the statistical analysis was adapted for every case as shown in Table A-1. In this section, we only give the details of the weight distributions for the Whirlpool hash function and then, summarize our similar results for all three hash functions in Figure A-5.

Table A-1 Size and element weights of the input data sets used for the statistical analysis

Whirpool		Grøstl		Grindahl	
Set size	Weight	Set size	Weight	Set size	Weight
512	1 (or 511)	1024	1 (or 1023)	64	1 (or 63)
1536	2 (or 510)	2054	2 (or 1022)	704	2 (or 62)
2048	3 (or 509)	26573	3 (or 1021)	2240	3 (or 61)
93350	Random with Min=160 Av=230 Max= 390	64806	Random with Min=256 Av=461 Max= 516	100,000	Random with Min=16 Av=29 Max= 43

The Whirlpool hash algorithm works for 2 chains (i.e., the message is segmented into two blocks) for any 512-bit message; the first chain input is the message itself and the second chain input is the length of the message padded to 512 bits.

First, we find the weight distribution of Whirlpool hashes corresponding to the weight-1 inputs. We observe that the Whirlpool hash function produces almost randomly distributed digests, with mean weight of 256 bits, as expected from a random set of 512-bit vectors. However, these weights are somewhat concentrated around the mean value (minimum weight is 213 and the maximum is 288 bits). The histogram in Figure A-1 shows the weight distribution of message digests for weight-1 inputs.

When we use weight-511 inputs instead of weight-1 inputs, the mean, maximum and thep minimum values of the message hashes become 256, 300, and 228 respectively.

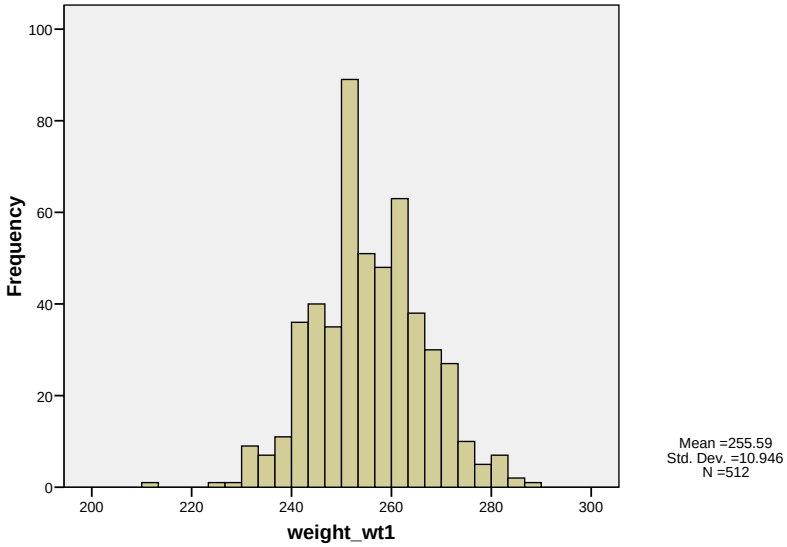

Figure A-1 Weight histogram of the Whirlpool hash function for 512 weight-1 inputs.

We have then found the weights of the message digests corresponding to 93350 random messages. The observed average weight of the hashed values is 256, minimal and maximal weights are 206 and 303, respectively. Results show that there is no meaningful weight difference between the hashed values of weight-1 or random messages. The histogram of output weights, when the inputs are random vectors, is given in Figure A-2.

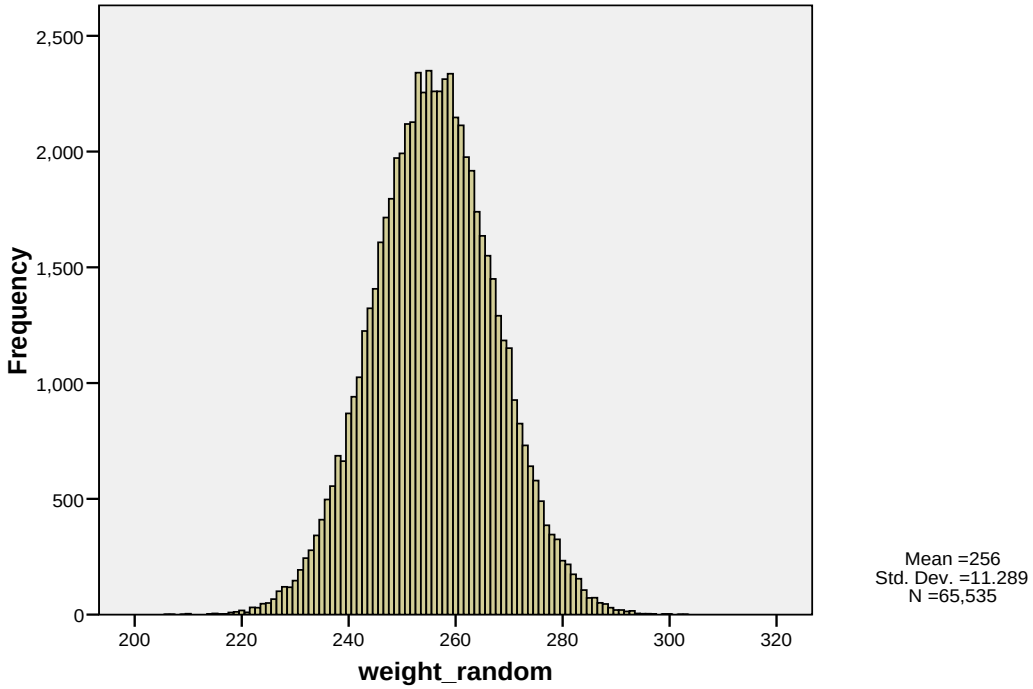

Figure A-2 Weight histogram of the Whirlpool hash function for 93350 random inputs.

The average, minimal and maximal weights of message digests for input vectors of different weights are shown in Figure A-3. As one can see, the mean weight is about 256, the minimum weight is not less than 200 (lower bound of 206 is obtained when 93350 random messages are hashed) and maximum is about 300 (303 in the case of random messages hashed values).

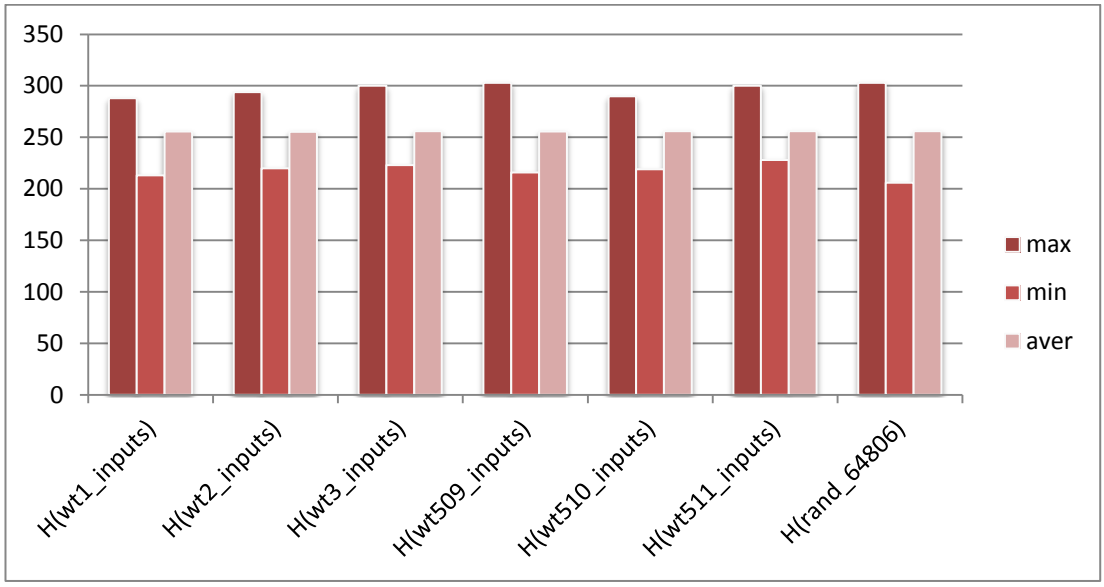

Figure A-3 Maximal, minimal and average weights of message digests for the Whirlpool hash function

Next, we analyze the Hamming distances between the obtained hashed values. Our input data consists of 512 weight-1 vectors $e_i, 1 \leq i \leq 512$, and all 0 vector v. The distance between vectors $e_i$and v is 1, and we would like to see the distance range between message digests corresponding to those messages. The histogram below shows that distance between message digests of vectors e_i and that of all-0 vector lie between 210 and 290 bits.

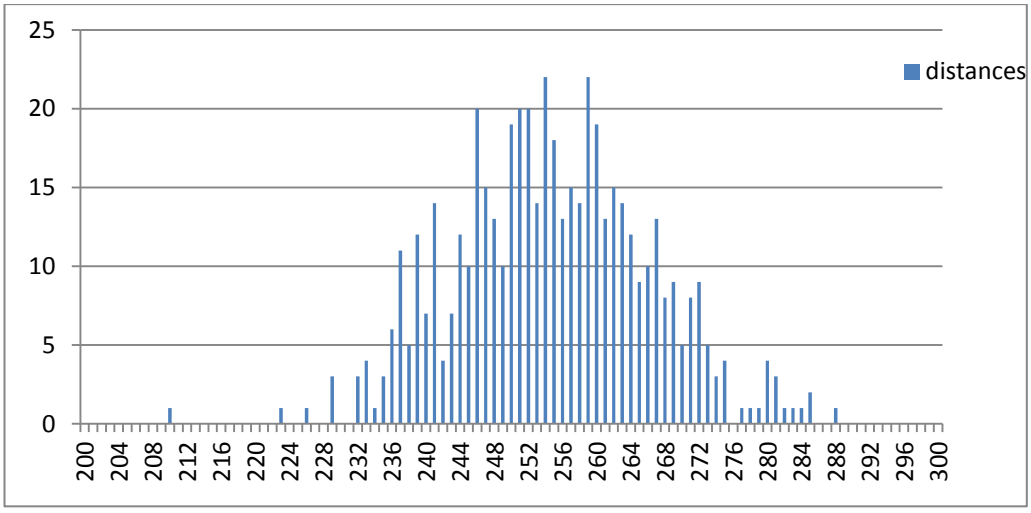

Figure A-4 Distances between the hashes of 512 weight1 vectors and the all-zero vector for the Whirlpool hash function.

Despite the difference in the size of input messages, results of the weight analysis obtained for the 512-bit message digests of the Grøstl and Grindahl hash functions are in the same range with Whirlpool as can be observed in Figure A-5.

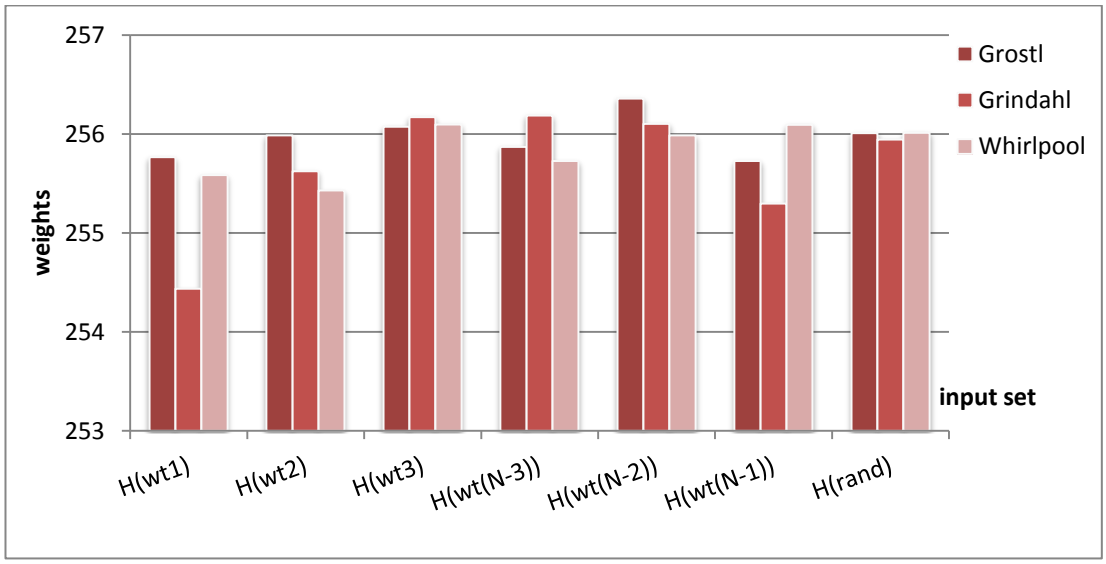

Figure A-5 Average message digest weights for the Whirlpool, Grøstl and Grindahl hash functions

A.2. Statistical Analysis of Correlation

In this section the correlation analysis is conducted. The correlation between message and their message digests of all three hash functions are analyzed in section A.2.1. Correlations between successive round outputs and between input and round outputs are given in sections A.2.2. and A.2.3. for W cipher of Whirlpool and P and Q permutations of Grøstl, respectively. For the Grindahl hash function we omit this analysis, since it consists of only one single round.

A.2.1 Correlation Between Message and Message Digests of the Hash Functions

Let f and g be two n-bit vectors in V_n. Relation between the correlation $c_{f,g}$ and the Hamming distance $d(f,g)$ is given by $c_{f,g} = n - 2d(f,g)$.

Normalized Correlation between f and g can be obtained by

$$C_{f,g} = \frac{c_{f,g}}{n} = \frac{1}{n}\sum_{x \in F_2^n}(-1)^{f(x)}(-1)^{g(x)}$$

We have analyzed the normalized correlation between weight-1 inputs and corresponding message digests. According to the result, after all 10 rounds Whirlpool shows good correlation properties.

Table A-2 Statistics of correlation for the Whirlpool hash function

	N	Minimum	Maximum	Mean	Std. Deviation
correlation_single_1_messages	512	-0.128	0.16	0.001	0.042
Valid N (listwise)	512				

Among observed 512 weigth-1 messages and their message digests, the maximal normalized correlation magnitude is at 0.16, and average is close to zero, as it is expected for a cryptographically strong hash functions.

For weight-511messages maximal normalized correlation is 0.175, average is again close to zero – 0.036.

To observe correlation behavior under equal conditions for the hash functions, the test is performed using the data set including 93350 random strings of length 512 bits. For Whirlpool and Grøstl hash algorithms, the hashed values are obtained after two chains due to padding rules, while for Grindahl it takes 8 chains of hashing the message blocks, plus one chain for padding.

Figure A-6 displays the average correlation magnitude between message and its message digest. The values are taken by absolute value. Although we see that maximal correlation is high enough, we can conclude that correlations are not high in general as average values are small.

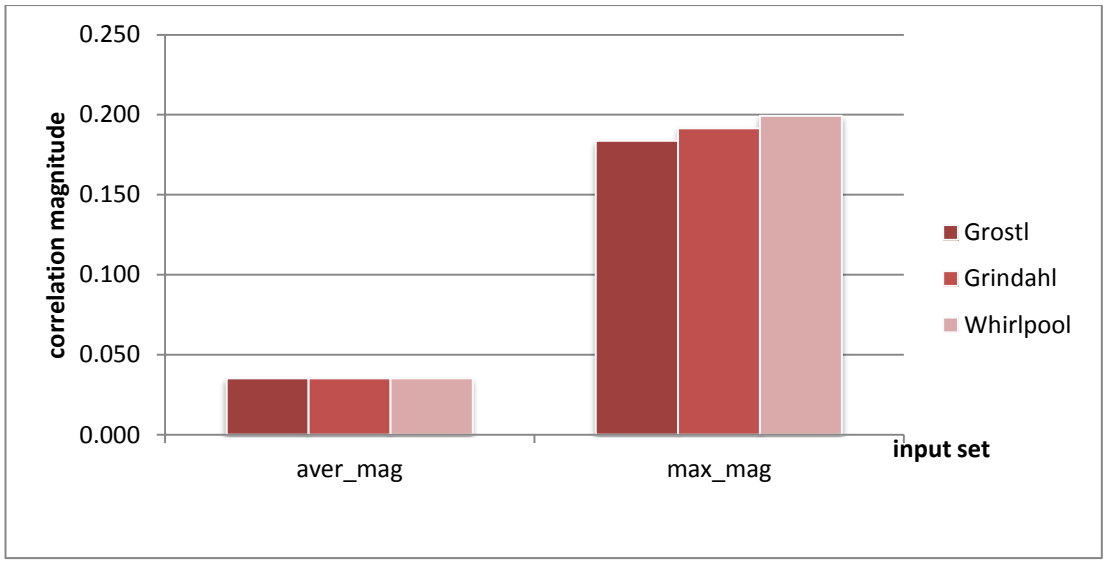

Figure A-6 Normalized correlation magnitudes (average and maximum values) between messages and message digests

A.2.2 Correlation Between Round Outputs of the W Cipher

Although the initial values are fixed in the design of all three hash functions, in Whirlpool hash function there is a key generation algorithm, which depends on chain outputs. That is, the key of W cipher is not always all zero vector. In this section we analyze the correlation within the W cipher when cipher key is all 0, all 1 and in a random case.

Correlation between Successive Round Outputs with Different Keys

In this test we also used weight 1 vectors of length 512. As an input of this test takes round outputs of Whirlpool algorithm and gives us correlation between them. Although in the previous test we have seen good results for correlation properties, the analysis of correlation between intermediate values, that is outputs of each round, do not pass the test for the first round. Correlation is very high between inputs and round-1 outputs, achieving 0.89 (maximum). Normalized correlations between round outputs are given in Table A-3.

Table A-3 Statistics of intermediate values normalized correlation

	N	Minimum	Maximum	Mean	Std. Deviation
round1_correlation	512	0.67188	0.89453	0.75822	0.03827
round2_correlation	512	-0.13281	0.12891	-0.00203	0.04398
round3_correlation	512	-0.12500	0.11719	-0.00291	0.04361
round4_correlation	512	-0.16406	0.12500	0.00170	0.04573
round5_correlation	512	-0.12891	0.13672	0.00188	0.04443
round6_correlation	512	-0.12109	0.12500	-0.00051	0.04525
round7_correlation	512	-0.12891	0.17969	-0.00098	0.04235
round8_correlation	512	-0.13672	0.12109	-0.00092	0.04467
round9_correlation	512	-0.15234	0.10938	0.00074	0.04371
round10_correlation	512	-0.14063	0.13281	-0.00002	0.04461
Valid N (listwise)	512				

Such behavior of correlation between inputs and round-1 outputs can be explained by the AddCipherKey operation in W cipher. This operation is applied to the state matrix before rounds start. Then both message and key are proceeded trough the same operations (since the key expansion of the W cipher is a copy of encryption algorithm itself with the only difference at AddRounKey operation, where round constants are used in key management). Remembering that round constants are matrices with nonzero entries only on the first row, we see that a message and the round key differ on that single row. So, it turns out that at AddRoundKey operation all changes are annulled, but the first row. Therefore the result is the high correlation.

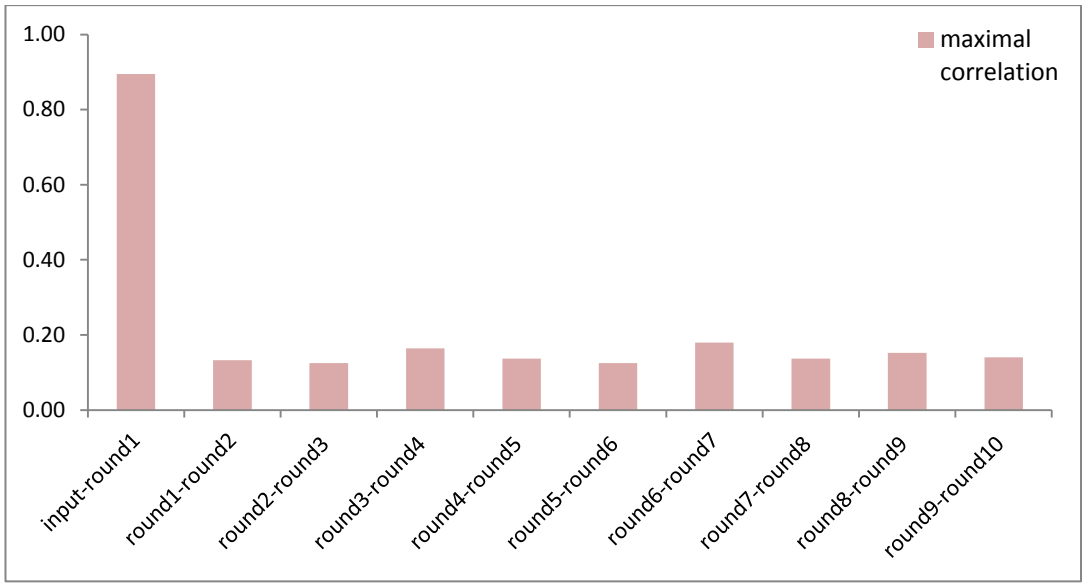

Figure A-7 Maximal normalized correlations between intermediate values

When the original key (i.e., all-0 initial vector) of Whirlpool Hash Function is used, we see high correlation between inputs and round-1 outputs. Normalized maximal correlation between round-1 and round-2 outputs is 0.12, and it remains the same between all following successive round outputs.

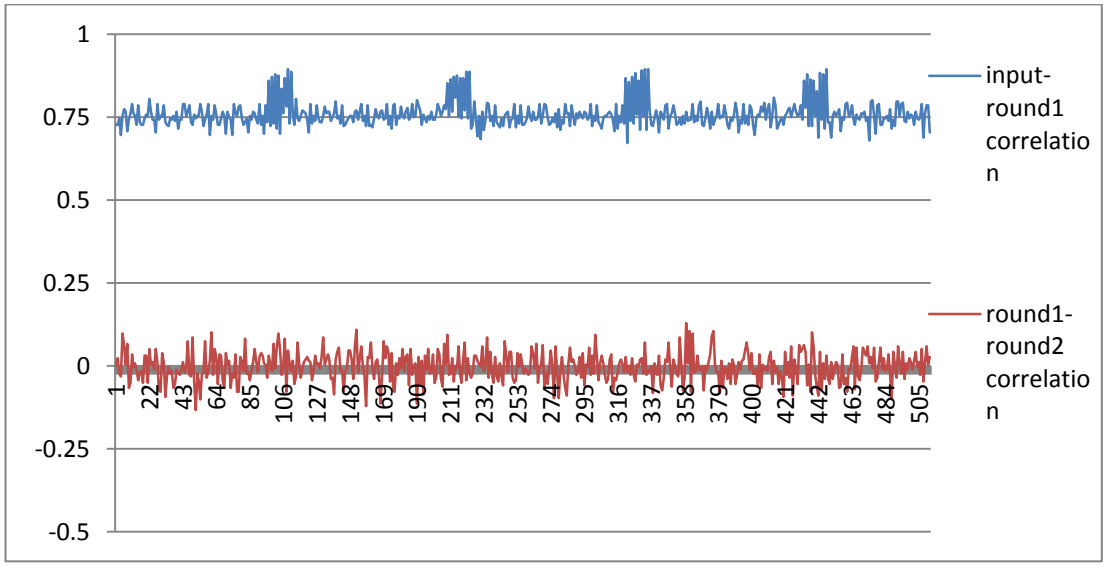

Figure A-8 Normalized correlation between successive round outputs over 512 inputs of weight 1 for all-0 key

We have tried to change the hash function's key to see if it affects the result. With all-1 key we observed the same result as in case of original key. Again, correlation between inputs and round-1 outputs is high and it diminishes between round 1 and round 2 and following rounds.

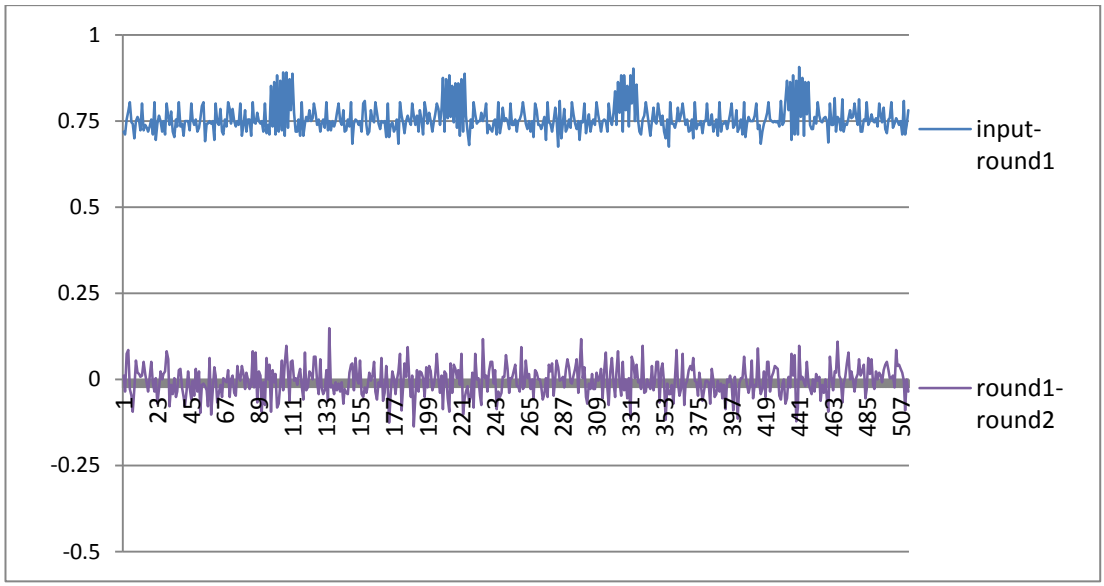

Figure A-9 Normalized correlation between successive round outputs over 512 inputs of weight 1 for all-1 key

With a totally random key, we still observe the same result (Figure A-10). So, we can conclude that changing the Whirlpool's key does not affect the strength of the hash function.

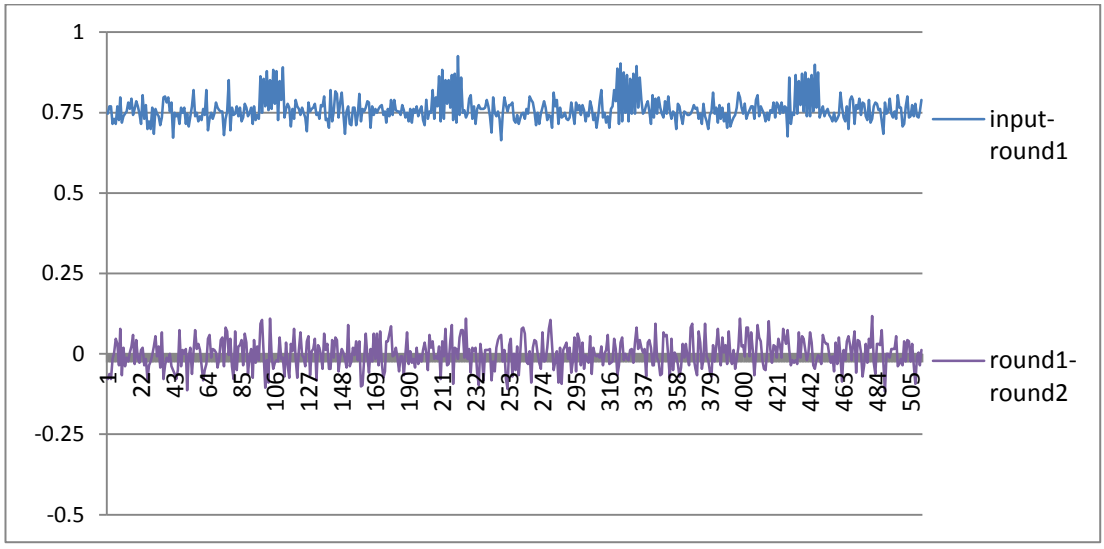

Figure A-10 Normalized correlation between successive round outputs over 512 inputs of weight 1 for a random key

Correlation between Input and Round Outputs with Different Keys

The previous test has been repeated for the analysis of correlation between input and round outputs. In this test we also used weight 1 vectors of length 512 as input of the Whirlpool hash function. Three different key were used, namely, the original key of the hash function (all-0 key), the all-1 key and random key. Result of this test also showed that changing Whirlpool's key has no effect on the diffusion properties of the hash function, with maximal normalized correlation at 0.18, 0.19 and 0.14 for all-0, all-1 and a random key, respectively. In Figure A-11 we give

normalized correlations between inputs and round-2 outputs only, since the range remains the same for the correlation between inputs and round outputs up to round-10.

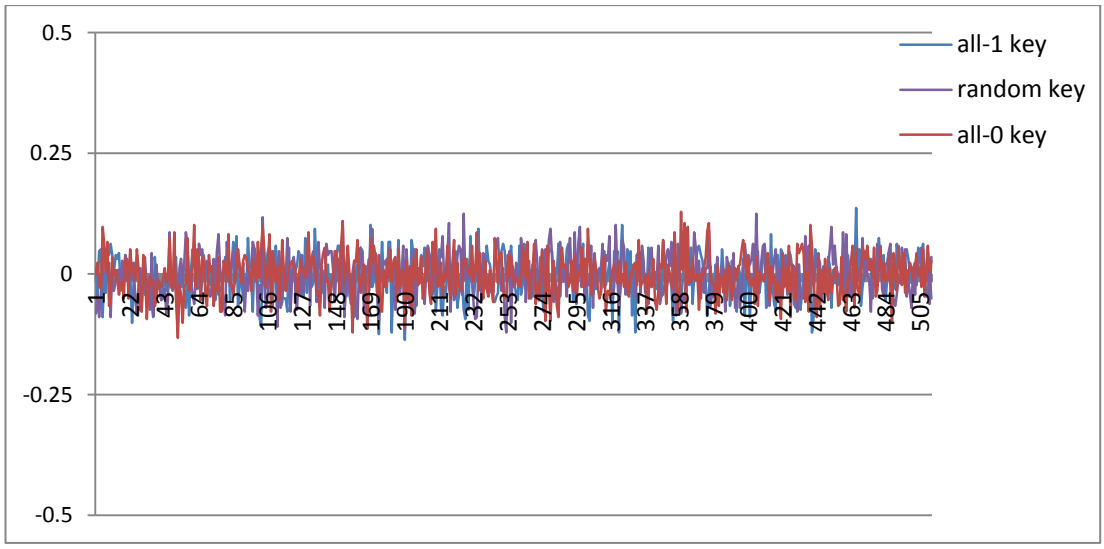

Figure A-11 Normalized correlation between inputs and round-2 outputs over 512 inputs of weight 1 different keys

A.2.3 Correlation Between Round Outputs of Grøstl Permutations P and Q

The chain function of the Grøstl hash algorithm exploits two identical ciphers P and Q with the only difference in round constants, which play the role of cipher key in AddRoundKey (AddRoundConstant) operation. In this section we examine the correlation properties within 14 rounds of P and Q permutations.

Correlation between Successive Round Outputs of P and Q Permutations

In this test we used weight 10000 vectors of length 512 and processed them though both permutations P and Q. Unlike W cipher of the Whirlpool hash function, P and Q of Grøstl show good correlation properties starting from the first round. That can be explained by the fact, that there are no key addition operations before rounds in these permutations. Maximal normalized correlation is almost twice lower for both P and Q compared to W (0.17 in W compared to 0.08 in P and Q).

In the rest, correlation behavior of permutation P and Q are much similar to that of W cipher of the Whirlpool hash function. In Figure A-12 and Figure A-13 the test results are given for the first 512 vectors (out of 10000) from the data set.

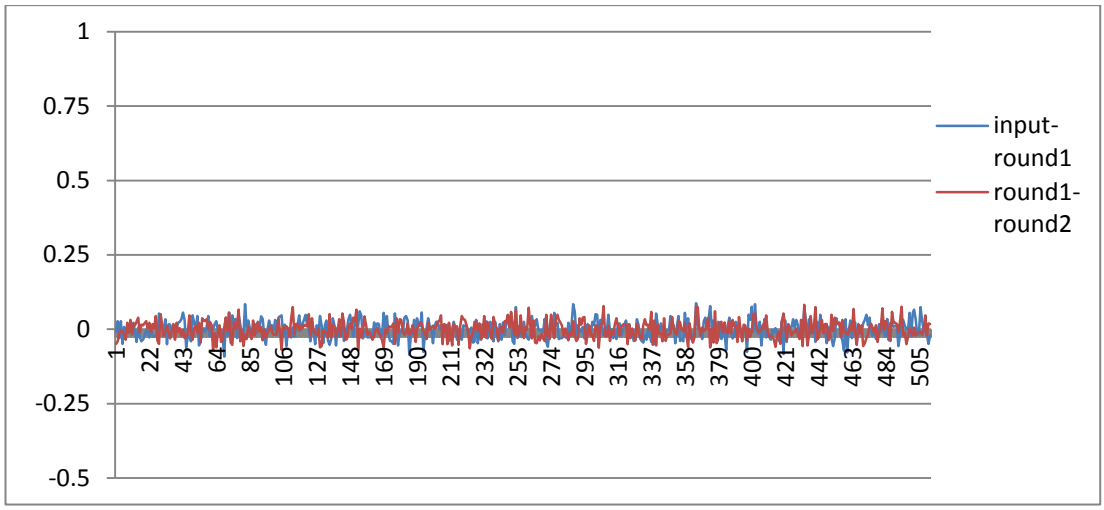

Figure A-12 Normalized correlation between successive round outputs of permutation P over 512 random inputs

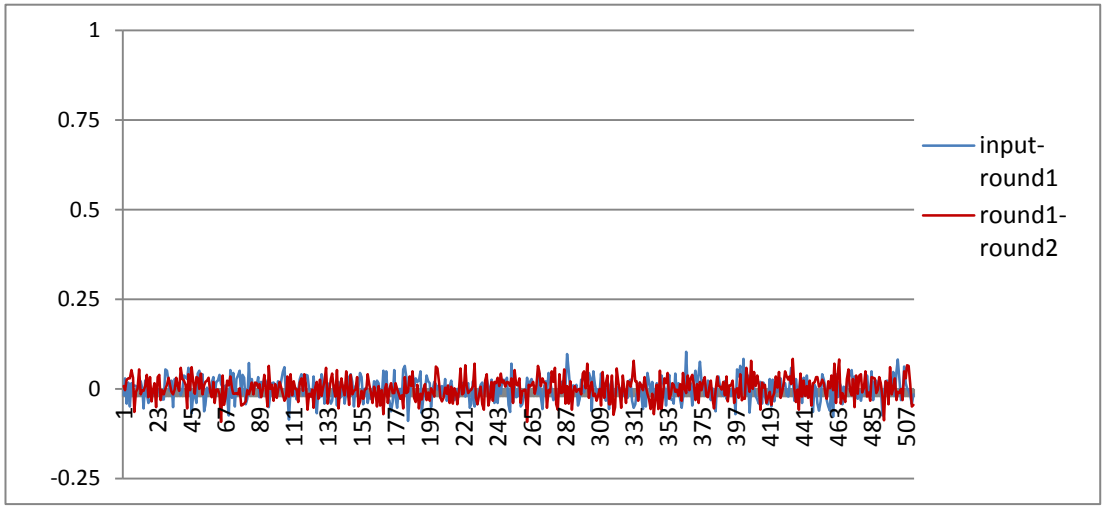

Figure A-13 Normalized correlation between successive round outputs of permutation Q over 512 random inputs

The aim of the next test is analysis of correlations between round outputs of permutations P and Q, since, to produce a next chain input, outputs of these permutations are XORed, therefore, the correlation between them is supposed to be low. As it was expected, correlation is high between round-1 outputs, since the only operation, responsible for difference, is the AddRoundConstant, which changes only one byte so far, and the MixBytes operation changes one column of the state (i.e., 8 bytes out of 128).

As an input to the round-2 we have the states, which differ from each other 8 bytes, one at each row (and exactly at the half of all columns). AddRoundConstant operation makes it 9 bytes, but still there are 8 columns (out of 16) with no difference, which results in the states of permutation with normalized correlation about 0.5 after the second round (Figure A-14).

However, after the 3rd round there is a good diffusion, and a low correlation between states of permutations P and Q.

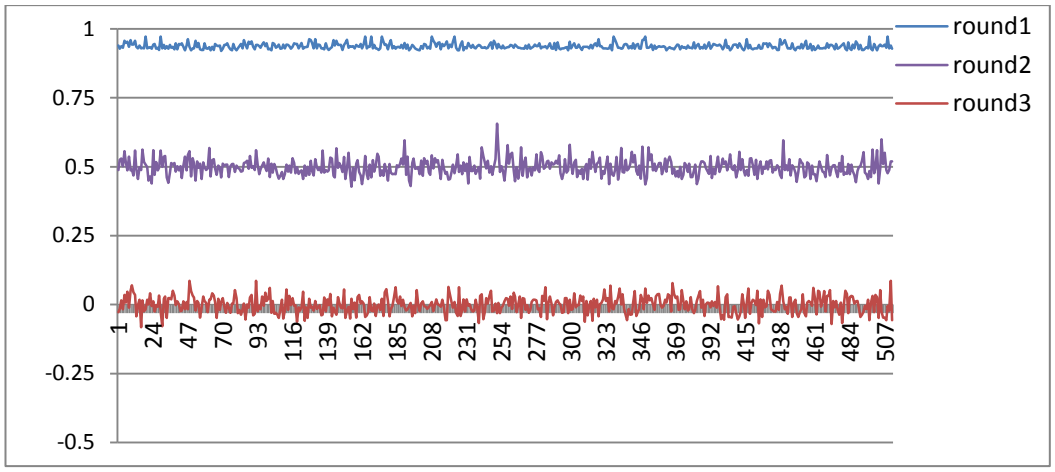

Figure A-14 Normalized correlation between round outputs of permutations P and Q over 512 random inputs

Correlation between Input and Round Outputs of P and Q Permutations

Since there is no key used in Grøstl hash function, this test includes the analysis of the original design over 10000 random vectors.

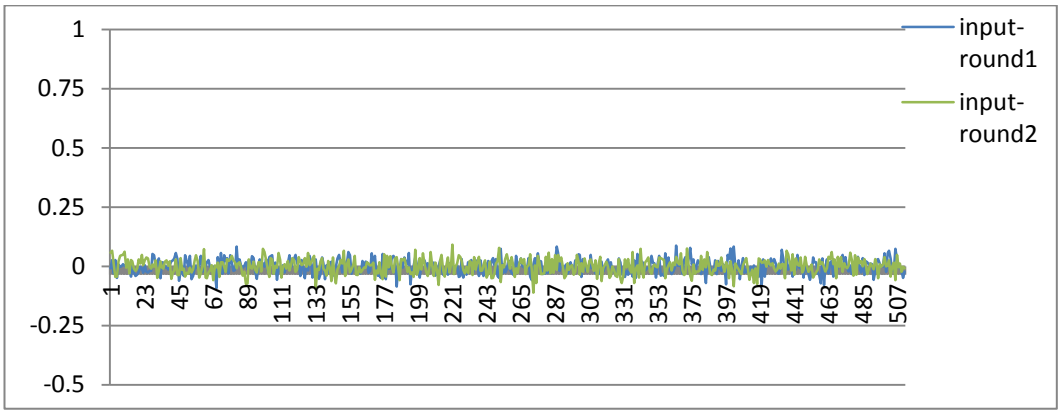

Figure A-15 Normalized correlation between input and round outputs of permutation P over 512 random inputs

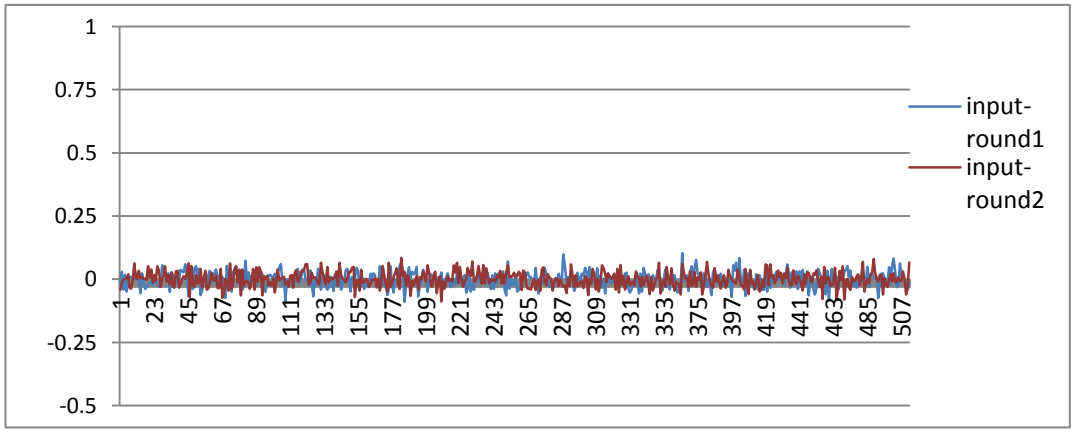

Figure A-16 Normalized correlation between input and round outputs of permutation Q over 512 random inputs

Maximal normalized correlation magnitude between inputs and round outputs are at 0.109 and 0.113 for P and Q, respectively, which is twice as low as for W of the Whirlpool algorithm. Averages are at 0.0006 for P and 0.0002 for Q. In Figure A-15 and Figure A-16 we give normalized correlations between inputs and round-2 outputs for these permutations. Input-rounds correlations remain the same for the other 14 round.

A.3. Whirlpool with Different Chaining Schemes with Original S-Box and AES S-Box

A.3.1 Comparison with the S-Boxes of Serpent and AES

The polynomial used in the design of the Whirlpool cipher is $x^4 + x + 1$ over GF(2^4). Three miniboxes are used, namely E, E^{-1} and R, where E^{-1} is the inverse of E box.

We have analyzed LAT and XOR tables of the Whirlpool cipher in terms of

- Nonlinearities of the s-boxes
- Differential uniformities of the s-boxes
- XOR table elements corresponding to input and output difference vectors of weight 1

The absolute value of maximal elements in the LATs of all three s-boxes equals to 4, so, the nonlinearity of all of them is $N_S = 2^{4-1} - 4 = 4$.

Analysis of the XOR tables shows that mini-S-boxes of the Whirlpool has differential uniformity equal to 4, meaning each differential characteristic has a probability of at most 1/4, and a one bit input difference will not result in one bit output difference. Thus, the probability that the

$S(x) + S(x + i) = j$ can be at most 4/16=1/4.

Besides, in all three s-boxes, the XOR(i,j) is not all zero for i, j – with the weight 1, i.e., on the intersection of i and j, with $\mathrm{wt}(i) = 1$ and $\mathrm{wt}(i) = 1$ there are elements different from zero. This implies that Whirlpool's mini s-boxes do not satisfy SAC, that is, there is bias when flipping one input bit, which allows applying differential cryptanalysis. If Whirlpool's mini s-boxes are compared with the s-boxes of the Serpent cipher, we see that Serpent's s-boxes show better properties.

When the whole s-box of Whirlpool cipher is analyzed, the LAT and XOR tables show the following properties. The absolute value of maximal element in the LAT appears to be 28, so, the nonlinearity is $N_S = 2^{8-1} - 28 = 100$, which is less than that of the Rijndael.

Table A-4 Comparison of the Whirlpool block cipher W and Serpent's s-boxes

	Whirlpool	**Serpent**
Maximal element in LAT	4	4
Nonlinearity of the S-box	4	4
Differential Uniformity of the S-box	4	4
Weight-1 elements of the XOR table	No	Yes

Analysis of the XOR table shows that the s-box of the Whirlpool has differential uniformity equal to 8 and, given one bit difference between two inputs, the difference between the corresponding outputs can also be one bit. In Table A-5 there is comparison of the s-boxes of the Whirlpool cipher W and the AES.

Table A-5 Comparison between the s-boxes of the Whirlpool block cipher W and the AES

	Whirlpool Cipher W	**AES**
Origin of the S-box	recursive structure	multiplicative inverse in $GF(2^8)$ plus affine transformation
Maximal element in LAT table	28	16
Nonlinearity of the S-box	100	112
Differential Uniformity of the S-box	8	4

A.3.2 Statistical Analysis of Weights

To see statistical behavior of message digests' weights when Whirlpool cipher is used in different chaining modes and with AES s-box, we used several data sets. As in case of the original scheme (Miyaguchi-Preneel Scheme), expected weights of outputs are about 256. As a test vectors we have taken inputs of size 512 (the hash algorithm works for 2 chains for any 512-bit input). The list of data used for statistical analysis is the same as we used in the analysis of original scheme (Table A-1), with the only difference at the number of random vectors (10000 random vectors of length 512 with average, minimal and maximal weights 231, 169 and 256, respectively). Table A-6 shows the general statistical behavior of message digests weights for a given data set.

According to result of statistical analysis, there is no significant difference between the test value and the observed mean in weights of message digests. For all schemes we can observe that the average weight is about 256. In Matyas-Meyer-Oseas sheme we see the maximal standard deviation, where minimum and maximum of weight deviate from the mean to 50 and 45 respectively, which is 5 units greater than this of the original scheme.

Table A-6 Statistics of weights for different schemes

		Input random	Devies Meyer Scheme	Matyas Meyer Oseas Scheme	Rabin Scheme	Original Scheme
N	Valid	10000	10000	10000	10000	10000
	Missing	0	0	0	0	0
Mean		231.44	255.93	256.23	255.75	256.00
Median		232.00	256.00	256.00	256.00	256.00
Std. Deviation		9.495	11.310	11.265	11.281	11.230
Minimum		169	214	206	214	206
Maximum		265	297	301	295	294

Figure A-17 also shows average, minimal and maximal weights of message digests when different schemes are used; the mean weight is about 256, the minimum weight is not less than 200 (lower bound of 206 is obtained when random messages are hashed using Matyas-Meyer-Oseas scheme and the original one) and maximum is about 300 (301 in the case of Matyas-Meyer-Oseas scheme).

The analysis in section A.3.1. shows that parameters of W cipher's s-boxes are inferior to those of AES finalists. For this reason, our next test is W cipher with AES's s-box and analysis of statistical behavior of hash function's outputs for different chaining modes.

Table A-7 Statistics of weights for different schemes with AES's s-box

		Input random	Davies Meyer Scheme AES	Matyas Meyer Oseas Scheme AES	Rabin Scheme AES	Original Scheme AES
N	Valid	10000	10000	10000	10000	10000
	Missing	0	0	0	0	0
Mean		231.44	256.21	256.03	255.95	255.91
Median		232.00	256.00	256.00	256.00	256.00
Std. Deviation		9.495	11.349	11.338	11.264	11.394
Minimum		169	211	216	209	208
Maximum		265	302	295	298	300

In this testing we again use as input 10000 random vectors of length 512. According to result of statistical analysis, average weight is 256, which is equal to the expected value. We can see that original scheme with AES's s-box gives greater deviations than it gives with the original s-box.

The histograms in Figure A-18 show the frequencies of message digests' weights for different schemes, when the original s-box is substituted with those of AES.

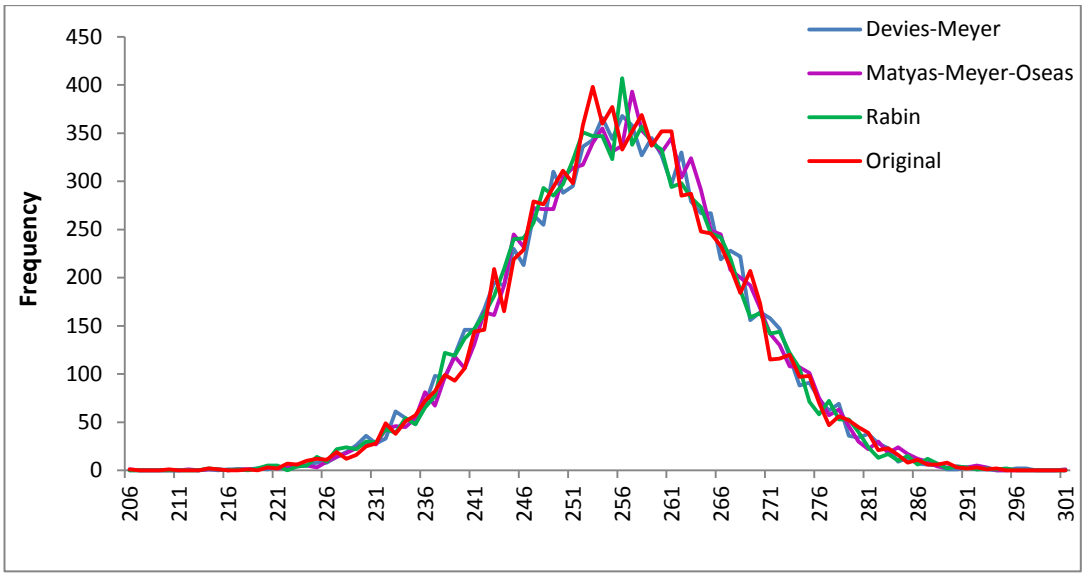

Figure A-17 Frequency of weights of MD's for the different schemes with the original s-box

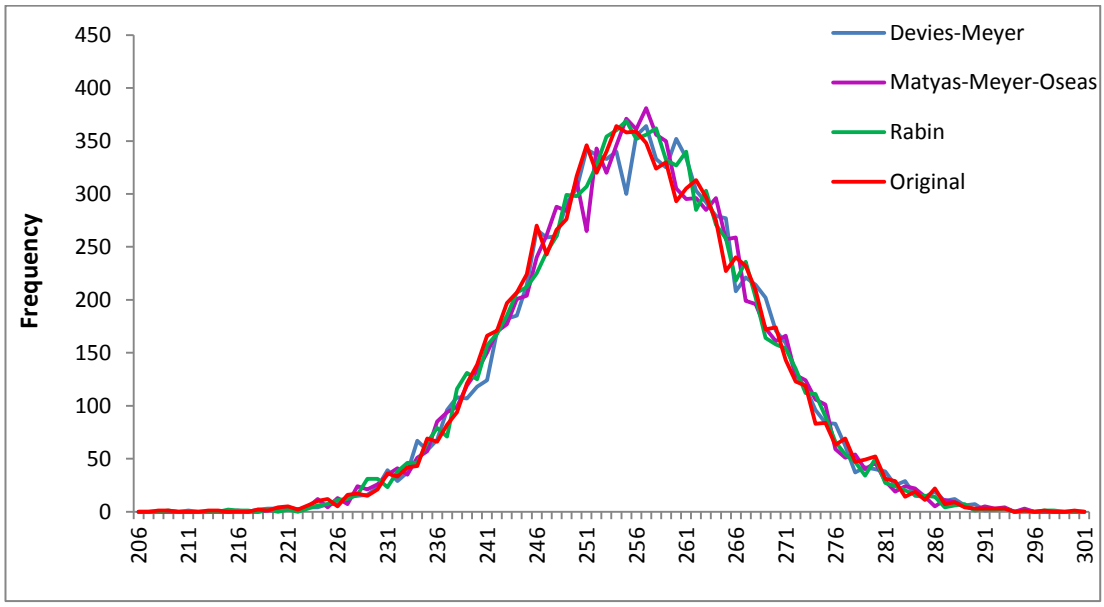

Figure A-18 Frequency of weights of MD's for the different schemes with s-box of AES

From the analysis above we can see that that maximal weight distribution is obtained when Davies-Meyer chaining mode is used with the original Whirlpool's s-box. However, average weights are almost the same for all chaining modes.

A.3.3 Statistical Analysis of Correlations between Message and Message Digests of the Whirlpool Hash Function

We have analyzed correlation between input data and corresponding message digest using 10000 random vectors as our sample data.

Table A-8 Statistics of normalized correlations between message and message digest for different schemes

		Davies Meyer Scheme	Matyas Meyer Oseas Scheme	Rabin Scheme	Original Scheme
N	Valid	10000	10000	10000	10000
	Missing	0	0	0	0
Mean		0.00051	0.00111	0.00002	-0.00008
Median		0	0	0	0
Std. Deviation		0.04389	0.04457	0.04397	0.04420
Minimum		-0.15625	-0.17188	-0.16406	-0.16797
Maximum		0.18359	0.16016	0.17188	0.16406

According to the result, after all 10 rounds the Whirlpool hash function shows good correlation properties with all schemes used. Maximal normalized correlation magnitude is observed when Whirlpool cipher is used in Davies-Meyer chaining mode and equals 0.18, while in the case of original scheme it is 0.16.

Next, we give the statistical analysis of correlation for different schemes with the original s-box substituted by AES's s-box.

Table A-9 Statistics for normalized correlation between message and message digest for different schemes with AES's s-box

		Davies Meyer Scheme AES	Matyas Meyer Oseas Scheme AES	Rabin Scheme AES	Original scheme AES
N	Valid	10000	10000	10000	10000
	Missing	0	0	0	0
Mean		-0.00053	-0.00008	0.00016	0.00033
Median		0	0	0	0
Std. Deviation		0.04463	0.04459	0.04396	0.04386
Minimum		-0.16406	-0.16406	-0.17969	-0.18359
Maximum		0.20313	0.16406	0.16406	0.17578

Again, after all 10 rounds Whirlpool shows good correlation properties with all schemes used. Maximal normalized correlation magnitude is observed when W cipher is used in Davies-Meyer chaining mode and equals 0.2, while in the case of original scheme it is 0.183. Note that in

Davies-Meyer scheme we observed maximal correlation magnitude when the original s-box was used (maximal was 0.18).

In terms of correlation the best result is obtained when Matyas-Meyer-Oseas scheme is used with AES's s-box with maximal normalized correlation magnitude at 0.164, while the original scheme with original s-box shows maximal normalized correlation magnitude equal to 0.168. Nevertheless, the best average correlation magnitude is obtained the original scheme is used with AES's s-box.

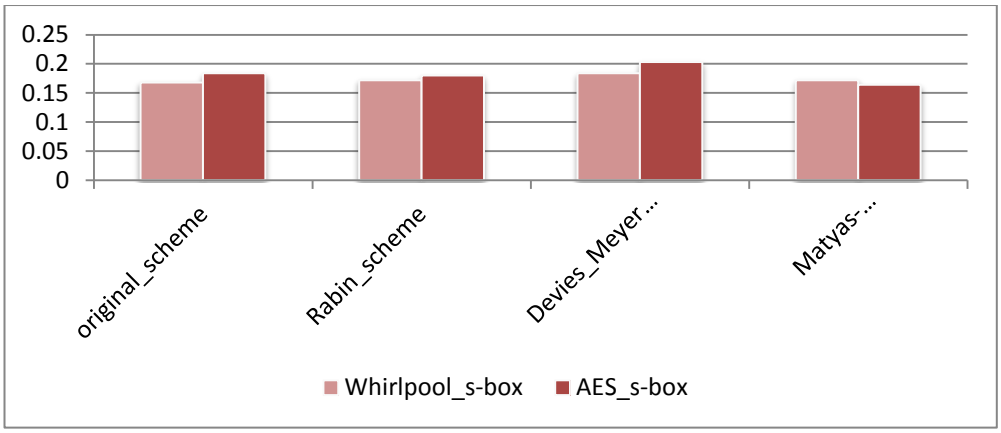

Figure A-19 Maximal correlation magnitudes between message and message digests

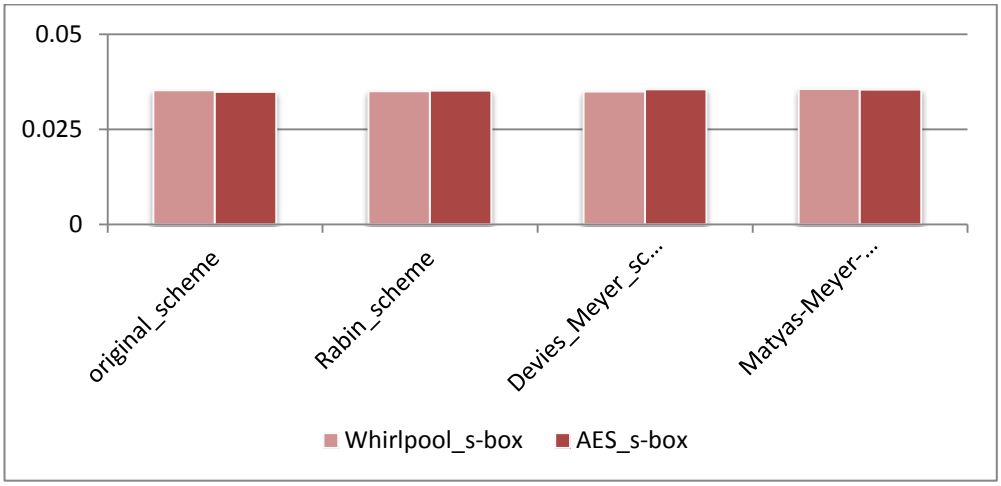

Figure A-20 Average correlation magnitudes between message and message digests

A.4. Conclusion

In this section statistical behavior of hash functions are analyzed. Although three different hashing techniques are considered, the weight and correlation in all these cases show high characteristics. However, in the analysis of weights it can be of interest the fact that outputs of one chain have weight distribution close to average value. As block ciphers are to map 1:1, and a fixed initial vectors are used, this range is due to hash functions' construction.

Besides, correlation between input and round-1 output appears to be high in W cipher of Whirlpool hash function. Although W is a modified version of Rijndael, the same behavior is monitored neither in P and Q permutations of the Grøstl hash function, nor in P cipher of Grindahl algorithm, which consists of just one round. Also, we study weight and correlation distributions of different schemes in Merkle-Damgård construction.

W cipher takes two inputs of 512 bits and outputs the ciphertext of the same length. As a last check, we examine how the outputs of the W cipher and Whirlpool hashes diffuse, we compute the weights and distances among all cipher outputs and among all Whirlpool hash values. We also compute the correlations between cipher outputs and messages and the correlations between hash values and messages. The results are given in Table A-10.

Table A-10 Statistics of the W cipher

Input weight	**Output weight**		**Distances between outputs**		**Maximum correlations with the inputs**	
	W	*Hash*	W	*Hash*	W	*Hash*
1 (512 vectors)	227-289	213-288	208-306	205-303	64	84
2 (1530 vectors)	221-292	220-294	201-313	201-296	76	72

As can be seen from Table A-, the maximum correlation between Whirlpool hash outputs and weight-1 inputs is slightly more than the maximum correlation between weight-1 inputs and W cipher outputs.

APPENDIX B

16-BIT VERSION OF WHIRLPOOL HASH FUNCTION

The aim of implementing a 16-bit version of Whirlpool hash function is to find colliding messages easily, since the small size of the hash function allows a brute force attack. We also describe a backward attack on the 16-bit version. Our intention is to use this information to analyze the original 512-bit version of the Whirlpool hash function.

B.1. Implementation

The following version of Whirlpool hash function takes an input of arbitrary length and outputs a hash of 16 bit length. The design of W has a 16-bit string as input and output.

In our implementation of 16 bit Whirlpool hash function, we use the same chaining mode as in the original one. The padding is again 1 bit, followed by necessary number of 0's. The binary representation of the length of original message is in last 8 bits of padding.

The initial value of H_0 is set to all 0, blocks of cipher key and plaintext for W cipher are of 16 bit length. H_0 is the cipher key for the first encryption by W cipher. Being XORed with the previous cipher key and plaintext, ciphertext becomes the cipher key for the next block. The message digest is an 16-bit output of the last block.

We try to keep the design of block cipher W used in the hash function close to the one of the original cipher. The input and output states are 2×2 matrices:

$$\begin{pmatrix} a_0 & a_1 \\ a_2 & a_3 \end{pmatrix} \rightarrow \begin{pmatrix} b_0 & b_1 \\ b_2 & b_3 \end{pmatrix}, \text{ where } a_i, b_i \in \mathrm{GF}(2^4), i = 0, \ldots, 3.$$

Changes have been made in SubByte and MixColumn operations, since the size of input is set to 16 bits. The MixColumn operation exploits an MDS matrix

$$C = \begin{pmatrix} F & 5 \\ 6 & 8 \end{pmatrix}$$

with irreducible polynomial $x^2 + x + 1$ (0x19).

As an s-box we use the first 4×4 s-box of Serpent block cipher (Figure B-1).

	00_x	01_x	02_x	03_x
00_x	3_x	8_x	F_x	1_x
10_x	A_x	6_x	5_x	B_x
20_x	E_x	D_x	4_x	2_x
30_x	7_x	0_x	9_x	C_x

Figure B-1 S-box used in the design

B.2. Test of Collisions

General Analysis

The number of possible hash values in 16 bit version of Whirlpool hash function is $2^{16} = 65536$. For our analysis we use a data set consisting of all possible 16 bit tuples. With the padding rule used in Whirlpool hash functions, these messages give us a hash values after 2 chains of W cipher. The average weight of hash values is 8.

The analysis of correlation shows a message and its digest having the same value, that is, the message

$$\begin{pmatrix} 1 & 2 \\ 6 & 2 \end{pmatrix}$$

maps to itself. Besides, the message

$$\begin{pmatrix} 1 & E \\ E & 0 \end{pmatrix}$$

maps to its complement vector. This behavior is not monitored at the original design of Whirlpool hash function. Although mapping in W cipher is 1:1, there is an XOR operation after each chain, which leads, in some cases, to a collision after one chain of Whirlpool hash function.

Analysis of Collisions

The aim of the test is to find the number of collision. The first data set consists of all possible 16 bit tuples, meaning there are $65536 (= 2^{16})$ input messages. The percent of colliding messages appeared to be equal to %0.003061. Further analysis shows that the weight of messages does not affect the percent of collisions, and it stays almost at the same range for messages of length 4 up to 8.

As our next step we check the structure of messages. The state of the message where diagonal bytes are fixed is analyzed. The choice of bytes at positions a_0 and a_3 is based on the previous analysis, since the most number of collisions is monitored with the values of these bytes being $a_0 = 0$ and $a_3 = 5$. Two other bytes spin the field GF(2^4), i.e. $a_i \in GF(2^4)$ for $i = 1,2$.

0	a_1
a_2	5

Figure B-2. Diagonal structure of an input

The number of colliding pairs with bytes, fixed to these values. is 4, which give us %0.012255 of collisions (compared to %0.003061 in general case). However, the further analysis of the messages with the same structure shows the percent of collisions close to general pattern (see Table B-1).

Table B-1 The number of collisions

input	**data set**	**number of collisions**	**% of collisions**
full	65536	65736	0.003061
Of weight 4	2517	102	0.003221
Of weight 5	6885	726	0.003064
Of weight 6	14893	3461	0.003121
Of weight 7	26333	10783	0.003110
Of weight 8	39303	28825	0.003732
Diagonal_0_5	256	4	0.012255
Diagonal_0_*	4096	22	0.004213

B.3. Analysis of Whirlpool Hash Function in Backward Direction

We denote the input of W cipher by x_i, where i represents the chain of Whirlpool hash function, i.e. x_1 is the input to the first W cipher, and by y_i the output of W cipher. We start by calculating a message digest for the all zero message. The previous analysis shows that a collision can be found even for one chain of Whirlpool hash function. Using this knowledge, we construct a one chain-collision. In Figure B-3 Output of W cipher for all zero input, the output of 10-round W cipher is given.

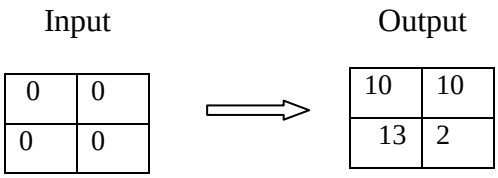

Figure B-3 Output of W cipher for all zero input

Now, we try to construct another message which gives the same hash value after one chain and XOR operation of Whirlpool hash function. Since the key of the cipher is fixed to all zero vector and the key scheduling algorithm is independent on the message, the only possibility to construct the same output of the hash function's first chain is to find message, which gives the same result after XOR operation. Going backward from the output, the message, which meets this requirement, is found (see Figure B-4 Backward search of the second preimage)

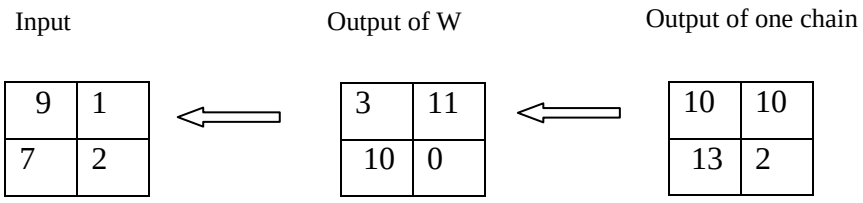

Figure B-4 Backward search of the second preimage

Since the input to the second chain is the same, this gives us a full collision after two chains.

VITA

PERSONAL INFORMATION

Surname, Name: Ismailova, Rita

Nationality: Kyrgyz (KR)

Date and Place of Birth: 14 June 1979 , Kyrgyz Republic, Naryn

email: **e146322@metu.edu.tr**

EDUCATION

Degree	Institution	Year of Graduation
MS	Applied Mathematics, Kyrgyz State National University, The center of the Magistracy, Postgraduate study and National Educational Programs	2003
BS	B.Sc.: Applied Mathematics, Kyrgyz State National University, faculty of Mathematics, Informatics and Cybernetics,	2001

FOREIGN LANGUAGES

Advanced English, Turkish, Russian, basic Korean

PUBLICATIONS

R. Ismailova, M.D. Yücel. *Statistics of the Whirlpool Hash Function under Integral Cryptanalysis*. Proceedings from ISC Turkey-2010, pages 221-227. Ankara, Turkey, 2010