Analysis of iterated block ciphers

Wei, Lei

2012

Wei, L. (2012). Analysis of iterated block ciphers. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/49503>

<https://doi.org/10.32657/10356/49503>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

ANALYSIS OF ITERATED BLOCK CIPHERS

WEI LEI

**SCHOOL OF PHYSICAL
AND MATHEMATICAL SCIENCES**

2012

ANALYSIS OF ITERATED BLOCK CIPHERS

WEI LEI

School of Physical and Mathematical Sciences

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy in Mathematical Sciences

2012

Acknowledgements

First, I am truly thankful to Prof. Wang Huaxiong, for all the encouragements and guidance in the four years, for creating opportunities for me to interact with researchers on cryptanalysis, for initiating research collaborations, for always being nice and responsive, I can not be more thankful to him as my graduate supervisor.

I appreciate Prof. Ling San for being my co-supervisor and sparing valuable time in research meetings and sharing ideas and comments. I am grateful to both my supervisors for their generous support for conference trips and providing comments to paper preprints and this thesis.

I want to specially thank Prof. Wu Hongjun for willingly to be an external supervisor of mine, for many helpful research discussions, for sharing his expertise in both cryptology and computer security. I also want to thank Prof. Josef Pieprzyk for the introduction and initial pointers when I started cryptanalysis and for all the interesting discussions we had during his visits to NTU.

I am infinitely grateful to Thomas Peyrin, for the inspiring and fruitful collaborations, for the way he practices the art of cryptanalysis, the passion, the rigorous attitude and the perseverance of always pushing the results to a higher standard and being a nice friend. These will always inspire me in my future research and pursuits.

I want to thank all my collaborators including Alexandre Duc, Guo Jian, Wu Hongjun, Khoo Khoongming and Nguyen Phuong Ha for the discussions and for the experiences in working on cryptanalysis together. I also want to thank Praveen Gauravaram, Wang Lei, Axel Poschmann, Christian Rechberger, Matt Robshaw, Li Ruilin, Przemysław Sokołowski and Wu Wenling for interesting discussions of various topics on cryptanalysis.

I would like to also thank my undergraduate research advisor Jussipekka Leiwo and final year supervisor Leong Peng-Chor for catering my initial interests in computer security and cryptography, for their encouragements and frank opinions when I decided to do research.

I am thankful to the anonymous examiners of this thesis for their valuable time spent on this thesis and the comments they provide. I also thank Dr. Ivica Nikolic for proof-reading parts of this thesis.

Lastly, I want to express my utmost gratitude to my family. I am heavily indebted to them, for their unconditional support, understanding and trust, for their professionalism and industrious character that will continue as one of my biggest sources of inspiration.

List of Publications

1. Lei Wei, Thomas Peyrin, Przemyslaw Sokolowski, San Ling, Josef Pieprzyk, and Huaxiong Wang. *On the (In)Security of IDEA in Various Hashing Modes. FSE 2012.*
2. Alexandre Duc, Jian Guo, Thomas Peyrin, and Lei Wei. *Unaligned Rebound Attack - Application to Keccak. FSE 2012.* Full version available on Cryptology ePrint Archive, Report 2011/420.
3. Lei Wei, Christian Rechberger, Jian Guo, Hongjun Wu, Huaxiong Wang, and San Ling. *Improved Meet-in-the-Middle Cryptanalysis of KTANTAN (Poster).* In proceedings of *ACISP, LNCS*, Vol. 6812, pp. 433-438. Springer, 2011. Full version available on Cryptology ePrint Archive, Report 2011/201.
4. Praveen Gauravaram, Lars R. Knudsen, Nasour Bagheri, and Lei Wei. *Improved Security Analysis of Fugue-256 (Poster).* In proceedings of *ACISP, LNCS*, Vol. 6812, pp. 428-432. Springer, 2011.
5. Phuong Ha Nguyen, Lei Wei, Huaxiong Wang, and San Ling. *On Multi-dimensional Linear Cryptanalysis.* In proceedings of *ACISP, LNCS*, Vol. 6168, pp. 37-52. Springer, 2010.
6. Aileen Zhang, Chu-Wee Lim, Khoongming Khoo, Lei Wei, and Josef Pieprzyk. *Extensions of the Cube Attack Based on Low Degree Annihilators.* In proceedings of *CANS, LNCS*, Vol. 5888, pp. 87-102. Springer, 2009.

Abstract

A block cipher is the foundation stone of symmetric-key cryptography. Due to its simplicity and high performance, it is often the workhorse for providing confidentiality - one of the primary goals of cryptography. Hence the security of a block cipher is of fundamental importance in the entire infrastructure of cryptography, and therefore block ciphers shall be analyzed and evaluated. This practice is called block cipher cryptanalysis. In this thesis, we analyze a few block ciphers in the classic meet-in-the-middle model and in the recently proposed multidimensional linear cryptanalysis model.

Besides for encryption, block ciphers are also one of the most versatile building blocks used for constructing many other cryptographic primitives. One such example is the compression function of cryptographic hash functions, and there is a close relation between the security analysis of block ciphers and hash functions. In addition, many dedicated cryptographic hash functions are designed with ideas used in block ciphers. Therefore, it is natural that many block cipher cryptanalysis techniques can be transferred to hash function analysis. In this thesis, we analyze hash functions with differential cryptanalysis and techniques inspired by differential cryptanalysis. On the other hand, recent advances in hash function cryptanalysis contribute to the analysis of block ciphers. We give one such example too.

In total we have four main topics on (or closely related to) the security analysis of block ciphers.

- We study the multidimensional extension to Matsui's Algorithm 1 and find improvements that lower the attack's costs. The new attacks are applied to 9-round and 4-round SERPENT, with interesting observations on these improvements and the framework.
- We study meet-in-the-middle attacks and their application to the hardware-oriented block cipher KTANTAN family and reduced DES. Several recent hash function analysis techniques are used for improving the first attacks.
- We conduct differential cryptanalysis for the internal permutation of the SHA-3 finalist KECCAK and propose a method named "unaligned rebound attack", for working with bitwise differential characteristics that have weak or no alignment.
- We investigate the block cipher IDEA under weak-key scenarios with respect to mainly differential properties, and find collision attacks for IDEA in single-length and double-block-length hashing modes. Similar investigation on a T-function property leads to a series of (pseudo-) preimage attacks in these modes.

Contents

Acknowledgements	i
List of Publications	iii
Abstract	v
List of Figures	xiii
List of Tables	xv
Glossary	xvii
1 Introduction	1
1.1 Thesis Outline	2
1.2 Contributions	3
2 Symmetric-Key Cryptology	5
2.1 Block Ciphers and Attacks	6
2.1.1 Definitions and Properties	6
2.1.2 Constructions of Block Ciphers	7
2.1.3 Attacks: Scenarios, Results and Methods	10
2.1.4 Distinguishers and Key Recovery	12
2.2 Concrete Block Ciphers	14
2.2.1 Block Cipher: DES	15
2.2.2 Block Cipher: IDEA	17
2.2.3 Block Cipher: SERPENT	18

2.2.4	Block Cipher: the KATAN/KTANTAN Family	20
2.3	Cryptanalysis of Block Ciphers	22
2.3.1	Differential Cryptanalysis	23
2.3.2	Linear Cryptanalysis	24
2.4	Cryptographic Hash Functions	26
2.4.1	Security and Attacks	26
2.4.2	Using Block Ciphers as Compression Functions	27
2.4.3	The SHA-3 Finalist KECCAK	28
2.4.3.1	The Domain Extension Algorithm	29
2.4.3.2	The KECCAK-f Permutations	29
2.4.4	Hash Functions Based on IDEA	31
2.4.4.1	Simple-Length Compression Functions	31
2.4.4.2	Double-Block-Length Compression Functions	32
3	Multidimensional Linear Cryptanalysis of SERPENT	35
3.1	Historical Developments	36
3.2	Background on Multidimensional Algorithm 1	38
3.2.1	Notions and Definitions	38
3.2.2	Multidimensional Probability Distribution	40
3.2.3	Multidimensional Algorithm 1	41
3.3	A Circulant Matrix Property	43
3.4	Attacking 4-Round SERPENT and Analysis	45
3.5	Attacking 9-Round SERPENT	46
3.6	Conclusions	48
4	Differential Cryptanalysis of KECCAK	51
4.1	Background on Cryptanalysis of KECCAK	52
4.2	Finding Differential Characteristics	53
4.2.1	Modeling Special properties of θ and χ	53
4.2.2	First Tools: Simplify the Nonlinear Layer	54
4.2.3	Algorithm for Differential Path Search	55
4.3	The Rebound Attack on KECCAK	56

4.3.1	Building Differential Distinguishers	56
4.3.2	The Rebound Attack Setting	58
4.3.2.1	The Original Rebound Attack	59
4.3.2.2	The Case of KECCAK Internal Permutations	60
4.3.3	A Buckets-and-Balls Problem	62
4.4	Constructing the Differential Paths Sets	64
4.4.1	The Forward Paths Set	64
4.4.2	The Backward Paths Set	65
4.5	Inbound Matching of Weakly Aligned Paths	70
4.6	The Outbound Phase	73
4.7	Finalizing the Rebound Attack	75
4.7.1	Solving the Dependency Chain	75
4.7.2	The Rebound Distinguisher	77
4.8	Further Improvements	78
4.9	Results and Conclusion	82
5	Meet-in-the-Middle Attacks on DES and KTANTAN	83
5.1	MITM Attacks on Round-Reduced DES	84
5.1.1	MITM Attacks on Block Ciphers	84
5.1.2	7-Round MITM Attacks on DES	87
5.2	Cryptanalysis of the KTANTAN Family	90
5.2.1	Previous Meet-in-the-Middle Attacks on KTANTAN	90
5.2.2	New Key Schedule Analysis and Results	92
5.2.3	Low Complexity Implementation of the Attack	93
5.3	From Hash to Cipher: Generalized MITM Attacks	95
5.3.1	Developments in Hash Preimage Attacks	95
5.3.2	Improving MITM Attacks on KTANTAN Family	97
5.3.3	Results and Conclusions	100
6	Cryptanalysis of IDEA in Hashing Modes	103
6.1	Introduction and Background	103
6.2	IDEA Weak-Keys in Hashing Modes	105

6.2.1	Analysis of IDEA Internal Operations	105
6.2.2	An Almost Half-Involution Property	106
6.3	Simple Collision Attacks	108
6.3.1	Davies-Meyer and Hirose Modes	108
6.3.2	Peyrin <i>et al.</i> (II)	109
6.3.3	Abreast-DM, Tandem-DM and MJH-Double	110
6.4	Improved Collision Attacks	110
6.4.1	Exploiting the Almost Half-Involution	111
6.4.2	Practical Hash Collision for Davies-Meyer	113
6.4.3	Collision Attacks on Abreast-DM and Tandem-DM	114
6.4.4	Collision Attack on Peyrin <i>et al.</i> (II)	115
6.4.5	Collision Attack on MJH-Double	116
6.5	Preimage Attacks with Practical Complexities	117
6.6	Results and Conclusions	119
7	Conclusions	121
7.1	Future Work	123
	Bibliography	125
A	Unaligned Rebound Attack	139
A.1	KECCAK Sbox, translation and transposition constants	139
A.2	Differential distribution table of χ and χ^{-1}	140
A.3	Differential paths for KECCAK-f[100]	141
A.4	Differential paths for KECCAK-f[200]	145
A.5	Differential paths for KECCAK-f[400]	149
A.6	Differential paths for KECCAK-f[800]	153
A.7	Plot for the pattern-match problem	156
A.8	Distributions for the KECCAK-f[1600] rebound attack	156
A.8.1	P_{out}	156
A.8.2	$\Pr[\text{Hw}_{\text{total}} = w \text{full}]$	156
A.8.3	$\mathbb{E}[N_{\text{SBox}} \text{match}, \text{Hw}_{\text{SBox}} = i]$	157

A.8.4	Behavior of N_w and $p_{\text{match}}(w)$	157
A.8.5	Different $N_{w_{\text{max}}}$ and p_{match} for relevant w_{max}	158
A.9	Differential paths for KECCAK- $f[1600]$	160
A.10	Example of valid pairs for KECCAK- $f[100]$	163
B	Meet-in-the-Middle Attacks	165
B.1	Derived sub-keys of KTANTAN	165
B.2	Derived sub-keys of DES	166
B.3	Indirect-partial-matching of the KTANTAN-32 attack	167

List of Figures

2.1	One round of IDEA	18
2.2	Illustration of KATAN/KTANTAN round transformation	21
2.3	Davies-Meyer mode	32
2.4	Abreast-DM DBL mode	33
2.5	Tandem-DM DBL mode	33
2.6	Hirose DBL mode	33
2.7	Peyrin <i>et al.</i> (II) mode	34
2.8	MJH-Double mode	34
4.1	Rebound attack on KECCAK	60
4.2	Number of differences for the rebound attack on KECCAK.	61
4.3	The forward trails. Values are taken from the 3 round differential path from Table A.23. The distance between the two lines reflects the number of differences.	64
4.4	The backward trails. The distance between the two arrows reflects the number of differences.	66
4.5	The forward trails with an extended round. The distance between the two lines reflects the number of differences.	78
4.6	The forward trails when we relax the 3rd round. The distance between the two lines reflects the number of differences.	79
4.7	The backward trails when we relax the first round. The distance between the two arrows reflects the number of differences.	80
5.1	Basic MITM attacks	85

5.2	A general setup for MITM attacks	95
5.3	Illustration of MITM attack with splice-and-cut	99
A.1	Average number of non empty buckets when a pattern-match occurs for $x_B =$ x_F and $s = 320$	156
A.2	Hw_{total} (in gray) is distributed following a Gaussian distribution(in black). . .	158
A.3	Behavior of N_w and $p_{\text{match}}(w)$	158

List of Tables

2.1	The first DES Sbox S_1	16
2.2	The E permutation of DES	16
2.3	The P permutation of DES	16
2.4	Key bits used for sub-keys $Z_j^{(i)}$ in the i -th round of IDEA	18
2.5	Parameters defined for the KATAN/KTANTAN family of ciphers	21
2.6	Round constant: the irregular update sequence	21
2.7	Parameters for KECCAK hash functions	28
3.1	Attack complexities on 4-round SERPENT	46
3.2	Comparisons of data complexities with different attack frameworks	48
3.3	Complexity comparison between different algorithms	49
4.1	Best differential paths for versions of KECCAK internal permutations	55
4.2	Best differential distinguishers for KECCAK internal permutations	82
5.1	Comparison of attacks on round reduced DES	87
5.2	DES Permutation Choice 1	88
5.3	DES Permutation Choice 2	88
5.4	Bits dependency map after round 7 for DES reduced to round 5 to 11	89
5.5	Neutral bits and attacks for different versions of 7-round DES	90
5.6	The new set of MITM attacks of KTANTAN family with corrected analysis	92
5.7	MITM attack with splice-and-cut and indirect-partial-matching	100
6.1	Summary of results for block cipher based compression function modes when instantiated with IDEA	119

A.1	The 5-bit Sbox applied to each row of the state	139
A.2	Translation constants of ρ for KECCAK- f [1600]	139
A.3	Transposition constants of π for KECCAK- f [1600]	139
A.4	The differential distribution table of the χ when viewed as Sbox	140
A.5	2-Round KECCAK- f [100] differential path with probability 2^{-8}	141
A.6	3-Round KECCAK- f [100] differential path with probability 2^{-19}	142
A.7	4-Round KECCAK- f [100] differential path with probability 2^{-37}	143
A.8	5-Round KECCAK- f [100] differential path with probability 2^{-67}	144
A.9	2-Round KECCAK- f [200] differential path with probability 2^{-8}	145
A.10	3-Round KECCAK- f [200] differential path with probability 2^{-20}	146
A.11	4-Round KECCAK- f [200] differential path with probability 2^{-56}	147
A.12	5-Round KECCAK- f [200] differential path with probability 2^{-121}	148
A.13	2-Round KECCAK- f [400] differential path with probability 2^{-8}	149
A.14	3-Round KECCAK- f [400] differential path with probability 2^{-24}	150
A.15	4-Round KECCAK- f [400] differential path with probability 2^{-84}	151
A.16	5-Round KECCAK- f [400] differential path with probability 2^{-245}	152
A.17	2-Round KECCAK- f [800] differential path with probability 2^{-8}	153
A.18	3-Round KECCAK- f [800] differential path with probability 2^{-32}	154
A.19	5-Round KECCAK- f [800] differential path with probability 2^{-459}	155
A.20	Distribution of P_{out} for our attack on KECCAK- f [1600]	157
A.21	Values for $N_{w_{\text{max}}}$ and p_{match} for different relevant w_{max}	159
A.22	3-round differential path for KECCAK- f [1600] with probability 2^{-32}	160
A.23	4-round differential path for KECCAK- f [1600] with probability 2^{-142}	161
A.24	5-round differential path for KECCAK- f [1600] with probability 2^{-709}	162
A.25	Example of 4-round differential path for KECCAK- f [100] and valid pairs verifying this path	163
B.1	The list of derived sub-keys of the KTANTAN family	165
B.2	Derived sub-key bits for all the 16 rounds of DES	166

Glossary

AES	Advanced Encryption Standard [58]	KL distance	Kullback-Leibler distance
ARX	Addition-Rotation-XOR	KP	Known Plaintext
CC	Chosen Ciphertext	KTANTAN	A family of lightweight block ciphers [45]
CP	Chosen Plaintext	LAT	Linear Approximation Table
CP-Kernel	Column parity kernel 4.2.1	LFSR	Linear Feedback Shift Register
CV	Chaining Variable	LSB	Least Significant Bit
DBL	Double-Block-Length	MDS	Maximum Distance Separable
DDT	Differential Distribution Table	MITM	Meet-in-the-middle
DES	Data Encryption Standard	MSB	Most Significant Bit
DM	Davies-Meyer, a hash compression function construction	NIST	National Institute of Standards and Technology
DP	Differential Probability	PGP	Pretty Good Privacy
ECC	Elliptic Curve Cryptography	PRESENT	A Family of Lightweight Block Ciphers [40]
FIPS	Federal Information Processing Standard	PRP	Pseudorandom Permutation
GE	Gate Equivalent	RFID	Radio-frequency identification
IDEA	International Data Encryption Algorithm [119]	Sbox	Substitution Box
Keccak	SHA-3 Finalist [19]	Serpent	AES Finalist [5]
		SHA	Secure Hash Algorithm
		SPN	Substitution-Permutation Network
		TDEA	Triple Data Encryption Algorithm [146]
		TEA	Tiny Encryption Algorithm [184]
		XOR	eXclusive OR

Chapter 1

Introduction

Cryptography is perhaps one of the oldest disciplines that are akin to mathematics, since there was the need of secure information exchange in ancient times, while the communication channels were still primitive and largely open to all adversaries. The use of cryptography started with methods of sending confidential message by means of secret writing in the pre-historic age ¹. In modern times, the demand for cryptography grew mostly with the need of highly secure and reliable communication for mission critical occasions, *e.g.*, warfare and diplomacy. The subject has advanced greatly since the emergence of computers. In the recent years, various communication networks have become pervasive. Civilian demand for cryptography over the Internet and the telecommunication networks alone introduces complex application scenarios for businesses and individuals. Modern cryptography has evolved to a state that systematic methods of providing confidentiality, data integrity, authenticity and more are readily available from academia, and can be applied in the industry.

The most common cryptographic techniques are methods of encryption and decryption, each always done with a key. Symmetric-key cryptography refers to the case when the encryption and decryption keys are the same or they can be derived from each other easily. On the other hand, public-key cryptography is established on the computational hardness assumption that deducing the private key from the public key is extremely difficult. Block ciphers and stream ciphers are the most important symmetric-key primitives in use, well known for their high performance and often well-studied security.

In general, the security of a block cipher is established after extensive analysis of its

¹We refer to [101] as a historical account on classic cryptology from ancient times to after WWII.

resistance against various cryptanalytic attacks, usually by the means of heuristic study of its mathematical properties, in the hope to find short-cut attacks that demonstrate weakness of the design. This study is usually called block cipher cryptanalysis. A block cipher is perceived and accepted as “secure” by the cryptology community, if it shows resistance to all known cryptanalytic attacks. The trust in its security increases when it shows good resistance over a stretch of time (usually in years), even against advances in cryptanalysis that were not previously conceived.

1.1 Thesis Outline

In this thesis, we present improvements in cryptanalytic attack algorithms, new ideas in methodology, and results on cryptanalysis of a few block ciphers and cryptographic hash functions akin to, or constructed from block ciphers. Our main results are presented in Chapter 3, 4, 5 and 6. Each chapter provides literature review, followed by the cryptanalysis results and concluding discussions. The outline is as follows:

- In Chapter 2 we introduce block ciphers, hash functions and their techniques of cryptanalysis. The most relevant block ciphers and hash functions are introduced compactly.
- In Chapter 3 we propose an improvement to one of the multidimensional linear cryptanalysis algorithms and derived improved attacks on the block cipher SERPENT reduced to 4 and 9 rounds.
- In Chapter 4 we present a differential cryptanalysis to the internal permutation of KECCAK. First we give algorithms for search of differential characteristics. Then we derive differential distinguishers for up to 8 rounds of KECCAK with our method called “unaligned rebound attack”.
- In Chapter 5 we deal with the meet-in-the-middle attacks and give key recovery attacks on 7-round DES variants. Then we show a cryptanalysis of the KTANTAN family obtained by correcting one recent attack. Lastly, we identify hash function cryptanalysis inspired generalizations to MITM attacks and obtain improvements to these attacks on KTANTAN.

- In Chapter 6, we analyze the block cipher IDEA and derive practical collision attacks and preimage attacks for several simple- and double-block-length hash functions constructed from IDEA.
- In Chapter 7, we conclude the thesis by discussing the results and give some future directions of research.

1.2 Contributions

Among the list of publications presented on page iii, all reported results in [182, 183, 181], most results in [70] and some results in [150] form the basis of this thesis. In these joint works, the author of this thesis played a leading/active role in obtaining the results included in this thesis. Below is the detailed description of these contributions.

- Chapter 3 studies the multidimensional extension of Matsui’s Algorithm 1 proposed in [88]. The results include
 - ▷ proof of a circulant matrix property that leads to improved attacks,
 - ▷ two improved attacks on 4-round SERPENT,
 - ▷ an improved attack on 9-round SERPENT and
 - ▷ analysis of the improvements. Based on the improved algorithms in [150], the multidimensional Alg. 1 can improve the basic linear cryptanalysis significantly when the number of linear approximations that can be formed is exponential to the dimension of the attack.
- Chapter 4 studies the internal permutation of the SHA-3 finalist hash function KECCAK with respect to differential cryptanalysis inspired techniques. The results include
 - ▷ a new and generic algorithm for differential paths search,
 - ▷ the current best differential paths for up to 5 rounds ¹,
 - ▷ a new method for inbound matching under weak alignment,
 - ▷ simple distinguishers for up to 7 rounds on almost all variants, and

¹Note that more results on KECCAK are obtained in [148] and [68].

- ▷ an 8-round rebound distinguisher for KECCAK- $f[1600]$ with complexity $2^{491.5}$, while the generic complexity is $2^{1057.6}$. The results have been verified on a small version of KECCAK.
- Chapter 5 evaluates the security of KTANTAN with respect to the meet-in-the-middle (MITM) attacks. Several techniques developed from hash function preimage attacks are identified and employed to improve the basic MITM attacks on KTANTAN. To the best of our knowledge, it is the first time that a *splice-and-cut* MITM attack is applied to block ciphers. The results include
 - ▷ low data complexity MITM attacks for all 7-round DES variants ¹,
 - ▷ the first non-marginal break of the whole KTANTAN family ² and
 - ▷ improved MITM attacks with *splice-and-cut* and *indirect-partial-matching* ³.
- Chapter 6 studies collision and preimage security of hash functions constructed from the IDEA block cipher in Davies-Meyer mode and double-block-length modes including Hirose, Abreast-DM, Tandem-DM, Peyrin *et al.*(II) and MJH-Double. This example shows an interesting perspective on how weak-keys of a block cipher may have disastrous consequence if the cipher is used for building a hash function. It is also a practical example of the gap that exists between secret-key and known- or chosen-key security for block ciphers. Moreover, the 20-year-old standing open question concerning the security of the Abreast-DM and Tandem-DM double-length compression functions is settled, the two modes were originally invented to be instantiated with IDEA. The results include
 - ▷ instantaneous free-start collisions for Davies-Meyer, Hirose and Peyrin *et al.*(II),
 - ▷ semi-free-start collisions attacks with practical complexity for Abreast-DM, Tandem-DM and MJH-Double,
 - ▷ practical hash function collisions for IDEA in Davies-Meyer mode and
 - ▷ preimage attacks of practical complexity for all modes.

¹These previously unpublished results are due to a joint work with Dr. Christian Rechberger.

²As the best known-plaintext attacks before recently attacks [189] by Zhu and Gong.

³As the best chosen-ciphertext attacks before recently attacks in [189] by Zhu and Gong.

Chapter 2

Symmetric-Key Cryptology

A *cryptosystem* [2, 176] is a general term referring to a set of cryptographic primitives used to provide information security services. The term is most often used in conjunction with primitives providing confidentiality, *i.e.*, encryption. This thesis primarily focuses on block ciphers, which are a type of *secret-key cryptosystems*, also known as *symmetric-key cryptosystems*. Examples of public-key cryptosystems are RSA [162], the elliptic curve cryptosystem [115, 140], etc.

The term *cryptography* is used to refer to building secure cryptosystems with mathematical techniques, while the term *cryptanalysis* is used to refer to mathematical analysis of the security of cryptosystems. *Cryptology* consists of *cryptography* and *cryptanalysis*.

Symmetric-key cryptology studies the design and analysis of symmetric-key cryptosystems, which include block ciphers, stream ciphers, message authentication codes and cryptographic hash functions. Note that only a cryptographic hash function does not involve secret keys. Historically the compression functions of cryptographic hash functions have been constructed from block ciphers with established security. For performance critical scenarios, *dedicated hash functions* are proposed for the sole purpose of hashing, usually constructed from operations that are directly supported by hardware.

This chapter provides an overview of the necessary background for results included in this thesis, and the specifications of block ciphers and hash functions that will be analysed in later chapters. In particular, Section 2.1 gives background on block cipher design and analysis, followed by several concrete block ciphers in Section 2.2. Differential cryptanalysis

and linear cryptanalysis as two of the most classic methods for block ciphers analysis are introduced in Section 2.3, they also serve as the basis to the majority of the attacks presented in this thesis. Lastly, in Section 2.4, we give a general introduction to cryptographic hash functions on their security and analysis, followed by hash function constructions that we analyse.

2.1 Block Ciphers and Attacks

2.1.1 Definitions and Properties

A *block cipher* is a family of keyed permutations that satisfy a set of security requirements. The mathematical specification is simple: a block cipher is a transformation $E : \{0, 1\}^{n_B} \times \{0, 1\}^{n_K} \mapsto \{0, 1\}^{n_B}$, *i.e.*, an n_B -bit ciphertext C is computed from n_B -bit plaintext P and n_K -bit key K as $C = E(P, K)$, where n_B is known as the block size and n_K the key size. $E(\cdot, K = k)$ is usually written as E_k which we call an *instantiation* of the block cipher, with $E_k(\cdot)$ being the encryption and $D_k := E_k^{-1}$ the decryption. Note that the transformation E_k is always a bijection, so is D_k . However, the security requirements can not be easily specified as a comprehensive and complete list, due to the fundamental limitation that our understanding in the security of block ciphers is based and advanced from heuristic analysis.

Nevertheless, a secure block cipher necessarily has a sufficiently large key size n_K and block size n_B . An *exhaustive search* or *brute-force* attack needs no more than 2^{n_K} calls to the cipher encryption routine to determine the correct key with a few plaintext-ciphertext pairs¹. As of now, a key space of size 2^{64} barely provides adequate security for any serious encryption. In addition, to make it hard to perform a *codebook attack* (sometimes called a *dictionary attack*), the block size has to be big enough. In fact an adversary has significant chance of success thanks to the birthday-paradox, if he is able to collect more than $2^{n_B/2}$ plaintext-ciphertext pairs. Rekeying is hence a good practice after no more than $2^{n_B/2}$ encryptions performed under the same key, especially when n_B is relatively small.

An *ideal block cipher* or a *random permutation* is a permutation E_k uniformly chosen from the set of all $2^{n_B}!$ permutations that operate on the message space $\{0, 1\}^{n_B}$, such that each k is uniform in the key space of 2^{n_K} keys. From the notion of descriptive complexity

¹A lower bound is the *unicity distance* due to Shannon [171].

theory, it is impossible to build such a block cipher in practice. The average descriptive complexity of a random E_k is around $\log_2(2^{n_B}!)$ bits, and this is way beyond the computing resources conveniently available for a compact implementation [37]. The size of any modern block cipher program is much smaller, we can see the implementable block ciphers are far from being truly random from this perspective. In practice, cryptographers usually design a block cipher with the intention that when the secret key is kept secret, the block cipher appears to be a random permutation with respect to cryptanalytic attacks.

2.1.2 Constructions of Block Ciphers

Modern block ciphers adopt the design of a *product cipher* (sometimes called a *cascade cipher* or an *iterative cipher*). Informally, the idea is to try to build a strong cipher by iterating a simple and cryptographically weak round functions for sufficiently high number of rounds. For an r -round block cipher with i -th *round function* F_i , let x_0 be the plaintext and x_i the state after the round i ($1 \leq i \leq r$). In the i -th round, the state x_{i-1} is mixed with the *sub-key* (also known as a *round key*) k_i , which is derived from the *secret key* (also known as a *user key* or a *master key*) using a *key schedule* algorithm. After r iterations of $x_i := F_i(x_{i-1}, k_i)$, the block cipher derives the ciphertext as x_r . Note that the function $G_i(\cdot) := F_i(\cdot, k_i)$ is necessarily bijective, for any fixed k_i in F_i .

The round function is necessarily *nonlinear*, otherwise the round key bits are efficiently derivable from a few plaintext-ciphertext pairs as a system of linear equations. Usually the round function involves a combination of *transpositions* and *substitutions*. Two conceptual ideas termed as *diffusion* and *confusion* are heavily used, due to the pioneering work of [171]. Good diffusion means spreading the influences of each input bit to preferably many output bits in a random way, whereas good confusion is about making the relationship between sub-key bits and the state bits as complex as possible after the round transformation, such that the relation between the ciphertext bits and the secret key bits is as complex as possible. If the round function achieves good diffusion and confusion, after sufficiently many rounds, each instantiation of the block cipher is expected to behave like a random permutation, with respect to the properties exploitable by cryptanalytic attacks. As a side note, diffusion and confusion are only interpretive descriptions of the inner workings of ciphers and they are by no means concrete methodologies or guarantee of achieving security. Criteria like the

strict avalanche criterion (SAC), the *bit independence criterion* (BIC) and the *guaranteed avalanche* (GA) are examples of more specific quantitative interpretation of diffusion and confusion, we refer to [174, pp 88-89] for an introduction of these criteria.

Primitive Components. Round functions are usually constructed from a dedicated set of linear and nonlinear components; building desirable cryptographic strength necessarily requires a careful combination of such components. Common primitives include (but not limited to) linear transformations, substitution functions and arithmetic operations. Linear transformations contribute to global diffusion inside the state space or a subspace that they operate on. Typical linear transformations range from simple transpositions like bit-wise permutations, to more complex mathematical functions with well understood diffusion properties, for instance an MDS matrix. Meanwhile, nonlinear substitution functions are crucial to make the whole cipher not a trivial linear system. A very popular choice of such functions is an Sbox, which is extensively studied as a cryptographic Boolean function. Sboxes are dedicated to providing confusion after the state bits are mixed with key bits, and at the same time they contribute to local diffusion in the substitution blocks they operate on. Arithmetic operations include modular multiplications, modular additions, shifts, rotations (sometimes called *cyclic shifts*) and XOR's. A combination of carefully selected arithmetic operations may also provide good nonlinear functions and local diffusion. In addition, they have an excellent performance profile since these operations are often directly supported by hardware architectures. In practice, Sboxes are often implemented as look-up tables, so frequently operate on smaller blocks than arithmetic operations. The theory of cryptographic Boolean functions has built dedicated tools for investigating the cryptographic properties. We refer to [46] as a comprehensive introduction to the theory of cryptographic Boolean functions.

High Level Constructions. The primitive components are glued together by well studied high level constructions - two notable examples are the Feistel network and the substitution-permutation (SP) network. The former was named after the cryptographer Horst Feistel. Similar to Shannon's original proposal of an SPN in [171], both constructions implement the idea of building a product cipher with alternating confusion functions and diffusion

functions and they have received extensive studies both in theory and practice.

Feistel Ciphers. A Feistel cipher [2] is an iterated cipher mapping a $2t$ -bit plaintext (L_0, R_0) , for t -bit blocks L_0 and R_0 , to a $2t$ -bit ciphertext (R_r, L_r) , through an r -round process where $r \geq 1$. For $1 \leq i \leq r$, round i maps $(L_{i-1}, R_{i-1}) \xrightarrow{K_i} (L_i, R_i)$ as follows: $L_i = R_{i-1}$, $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$, where each sub-key K_i is derived from the cipher key K . The ciphertext is outputted as (R_r, L_r) , the decryption can be done in exactly the same steps as encryption with the sub-keys in reverse order. The decryption is possible without f being a bijection since the same function f is used in both encryption and decryption. Feistel network has various generalizations. For example, in an so-called *unbalanced Feistel network* [168], the original algorithm is modified such that L_i and R_i registers are of unequal lengths. For a comprehensive classification of Feistel-like structures one can refer to the recent work on the security of generalized Feistel networks [92]. Concrete examples of Feistel ciphers include DES and its precedent Lucifer [173], the Soviet and Russian standard GOST-28147-89 [78], TEA [184], XTEA [149], RC5 [160], RC6 [161], Camellia [7], etc. The NSA declassified cipher Skipjack [152] is an unbalanced Feistel network, the recent Chinese WAPI cipher SMS4 [67] is a generalized Feistel network and MISTY1 [132] is a nested Feistel network.

SPN Ciphers. The substitution-permutation (SP) network is a highly organized way of constructing a product cipher. In each round there is usually an alternation of a substitution layer consisting of several Sboxes, and a permutation layer consisting of a global permutation or several smaller permutations. Its basic structure is best illustrated by the ciphers that adopt an SPN structure. For example, an early version of Lucifer adopts the SPN structure [74]. A few highly regarded SPN ciphers are AES [57, 58], the AES finalist SERPENT [5], the lightweight block cipher PRESENT [40] and others.

Other Ciphers. Besides the two popular choices, many ciphers are designed in high level dedicated constructions. However it is crucial that the generic construction has withstood years of cryptanalysis and is preferably well understood to gain acceptance. The Feistel cipher MISTY1 is one such example with a novel design in terms of security and performance.

Other examples are the block cipher IDEA [119] which remains unbroken¹ against at least 20 years of accumulated advances in cryptanalysis, and the KATAN/KTANTAN family of lightweight block ciphers [45] which base their round functions on LFSR.

2.1.3 Attacks: Scenarios, Results and Methods

Scenarios. In real world applications of block ciphers, an attacker may have very different access to the target cipher. However in theoretical analysis, we accept the *Kerckhoff's principle* which best encapsulates the consensus that “security by obscurity” is a bad practice, and the algorithm itself shall be public. Besides this, an attack specific to a scenario is often only limited to what kind of access the attacker already has with respect to the cipher instance, the plaintexts and the ciphertexts. In a *ciphertext-only* attack, the attacker is limited to access of a set of ciphertexts that he has no control over; while in a *known-plaintext* attack, he knows additionally the corresponding plaintexts of these ciphertexts. When the attacker is able to choose at his wish a set of ciphertexts or plaintexts before launching the attack, we call it a *chosen-ciphertext* or *chosen-plaintext* attack, respectively. The most powerful is an adaptively chosen text setting, where the attacker (wisely) chooses additional plaintext or ciphertext one by one, based on the knowledge already obtained from the use of earlier texts in the attack. A more recent line of research on *related-key* attacks [21, 22] empowers the attacker with more freedom in conjunction with the above mentioned scenarios, namely he has more than one target cipher instances whose keys are under a simple relation. This is a plausible attack model especially when the target block cipher is used with its keys exposed to certain manipulation directly by the attacker or indirectly by its application.

Results (Goals). Block cipher attacks are admissible with a variety of results or goals, among which recovering the secret key is the most desirable. An attack may pose a practical threat for a specific use of the cipher, showing a weakness that the designer overlooked, disprove a conjectured security level, or render an assumed property (*e.g.*, being a pseudorandom permutation (PRP)) as untrue. One attempt of a hierarchical classification was

¹The status may have been changed recently due to the Eurocrypt 2012 paper: Narrow Bicliques: Cryptanalysis of Full IDEA, by Khovratovich, Leurent, and Rechberger.

given by Knudsen in his PhD thesis [107] and in [109], summarized as what an attacker can gain **via performing an attack**.

- Total break - full recovery of the secret key.
- Global deduction - discovery of an equivalent encryption (decryption) algorithm.
- Instance (local) deduction - obtaining the plaintext (or ciphertext) of an intercepted ciphertext (plaintext).
- Information deduction - gaining some (Shannon) information about the secret key, the plaintexts or the ciphertexts.
- Distinguishing algorithm - differentiate the target cipher with respect to a random permutation.

We note again here that it is impossible to work directly with a truly random permutation since there is no such implementation available. Any distinguishing attack is effectively conducting a comparison with certain idealized properties possessed by a random permutation. Some distinguishers can be effectively converted to key recovery attacks while some distinguishers are solely for the purpose of challenging a chosen or known cipher instance with respect to an idealized randomness property. One such example is the so-called known-key model. We will discuss distinguishing attacks in Section 2.1.4, and known-key distinguishers in Section 2.4.2.

Complexity. The complexity of a cryptanalytic attack is primarily the amount of time required for computations, usually calculated in a basic unit such as an encryption, a memory access (MA) or a particular operation that would cost the majority of the computer cycles needed for such an attack. In addition, two other important factors are the *data complexity* as the number of plaintexts and ciphertexts required, and *memory complexity* as the amount of memory storage needed. In the related-key model, one also has to specify the number of related-keys needed. The complexity of the attack is no less than the maximum of these complexities.

Methods. It is often easier to discuss security with respect to specific attacks, and attacks are usually compared with the conjectured security level, which is the security parameter (often the key size). Usually a security level of 2^{n_K} is expected by the design, hence any attack that breaks this claim is a valid cryptanalytic attack.

A *generic attack* is carried out regardless of the internal structures, or one that concerns very little of the design itself. The (serial) exhaustive search is the most typical example. In addition, variants of exhaustive search are also classified as generic attacks such as parallelization [3, 185] and time-memory-data trade-off [87]. It is arguable that such attacks do not lead to asymptotic reduction in the complexity if compared to a serial exhaustive search, however in reality the razor’s edge is often the time, namely, one aims to disclose the secret within the lifespan of the piece of information protected by instantiations of the cipher.

On the other hand, a *shortcut attack* exploits mathematical patterns with respect to specific design details and it outperforms the brute-force attack. Even if the workload is far from being practical, such an attack is still considered valid as an *academic break* since it raises concerns on the conjectured security level that the cipher designers fail to deliver. Such a weakness of the cipher is usually called a *certificational weakness*.

2.1.4 Distinguishers and Key Recovery

In the theory of cryptography, randomness is an issue of central importance. Each instance of an n -bit block cipher is preferably a permutation that appears to be randomly selected from the space of all $2^n!$ possible permutations. However, this notion of randomness can not be simply studied when we analyze the security of block ciphers. Although the previously mentioned Kolmogorov-complexity (*i.e.*, descriptive complexity) offers an “ontological” treatment of randomness, it is not really practical in the theory and practice of cryptography. The Kolmogorov-complexity is a function that can not be computed, in addition, objects that can not be efficiently distinguished may have fundamentally different Kolmogorov-complexity in nature [80, pp 102-103]. Theorists hence adopt more practical alternatives from a complexity-theoretic perspective [39, 81, 82]. Luby and Rackoff were the first to study idealized round-reduced DES with respect to the notion of *pseudorandomness* in their 1988 work [128].

Informally, a block cipher *distinguisher* is an algorithm $\mathcal{D}$, by sending queries and analyzing the responses. It interacts with two black box permutations: a random permutation $\mathcal{R}$, and a block cipher $\mathcal{B}$ which is designed to emulate $\mathcal{R}$. If $\mathcal{D}$ is able to tell $\mathcal{B}$ from $\mathcal{R}$ with a non-negligible deviation termed *advantage*, then with the distinguisher $\mathcal{D}$ we have a *distinguishing attack* on the non-random behavior of $\mathcal{B}$. We refer to [100] for precise mathematical definitions and a rigorous study of various statistical distinguishers on block ciphers.

The Luby-Rackoff result studies the randomness of Feistel construction from a perspective that distinguishers are used as a tool to define pseudorandomness. For a permutation implemented as a block cipher instance $\mathcal{P}_0$, let $\mathcal{P}_1$ be a random permutation. If both $\mathcal{P}_0$ and $\mathcal{P}_1$ are able to answer queries as an encryption and decryption oracle, then $\mathcal{P}_0$ is

- *pseudorandom* if no distinguishing algorithm $\mathcal{D}$ can tell $\mathcal{P}_0$ from $\mathcal{P}_1$ in polynomial¹ number of encryption queries (an adaptive chosen-plaintext setting);
- *strong-pseudorandom* if no distinguishing algorithm $\mathcal{D}$ can tell $\mathcal{P}_0$ from $\mathcal{P}_1$ in polynomial number of encryption and decryption queries (an adaptive chosen-text setting).

From Distinguishers to Key Recovery. Despite its applications in the theory of block ciphers, a distinguishing attack is often viewed as one of the weakest forms of attacks, as we discussed in Section 2.1.3. However, in the realm of cryptanalysis, many powerful attacks can be viewed as a natural extension from a *distinguishing property*. One such example is the so-called *last-round attack*, where distinguishing attacks are converted to key recovery attacks. Suppose F_k is the round function (with sub-key k) of an r -round iterative block cipher instance $\mathcal{B}$ keyed by K , from which a list of sub-keys $(k_1, \dots, k_r)$ (without key-whitening) are derived from the key-schedule. We suppose that (without loss of generality) under a known-plaintext setting, a cryptanalyst knows a distinguishing property $\mathcal{P}$ of the block cipher $\mathcal{B}$ reduced to the first $(r - 1)$ rounds, such that he can distinguish the reduced cipher from a random permutation in N queries. Then for each guess k_r^* of the last sub-key, he can obtain N pairs of $(x_0, F_{k_r^*}^{-1}(x_r))$ from N plaintext-ciphertext pairs (x_0, x_r) . Using the property $\mathcal{P}$, with a mild assumption that a wrong guess k_r^* of k_r will result in random N

¹Polynomial in the block size n_B .

pairs with respect to $\mathcal{P}$, he can distinguish the correct guess of k_r^* from the wrong guesses. Thus he can recover the last-round sub-key k_r . If the key-schedule is invertible, he can directly deduce part of K from the recovered k_r . He may then “peel” the last round off the original cipher and continue searching the unknown part of K in a similar manner, or he can simply conduct an exhaustive search to the rest of the bits, in order to achieve full recovery of K . In the case of a non-invertible key-schedule, he may deduce the rest of the sub-keys similarly, recovering the list of sub-keys $(k_1, \dots, k_r)$ achieves a global deduction of $\mathcal{B}$ as we discussed in Section 2.1.3. In actual cryptanalysis scenarios the procedure of a last-round attack is often more flexible, *e.g.*, the distinguishing attack with $\mathcal{P}$ may not need the full knowledge of k_r^* , it may not be necessary that only the last round is handled, it could be the first few rounds, the last few rounds or a combination of them - whichever way that maximizes the overall efficiency of the attack.

This section only serves as a high level introduction to block cipher distinguishers and we show specific examples and results of distinguishers in the later parts of this thesis. In Section 2.3 we review several classic block cipher cryptanalysis techniques built on distinguishing properties. Recent advances in *open-key distinguishers* will be discussed in Section 2.4.2, including the *known-* and *chosen-key* distinguishers. We discuss *limited-birthday distinguishers* of a compression function in Chapter 4 in details and we also show how to construct them from simple differential paths or complex paths constructed by a rebound attack.

2.2 Concrete Block Ciphers

We introduce in this section the specifications of block ciphers that are directly related to the research contributions of this thesis. The list includes the Data Encryption Standard DES (Section 2.2.1), the International Data Encryption Algorithm IDEA (Section 2.2.2), the AES competition finalist SERPENT (Section 2.2.3) and the hardware-oriented block cipher KATAN/KTANTAN family (Section 2.2.4). The analysis of IDEA in this thesis is however targeting security of cryptographic hash functions based on IDEA, where the block cipher is used to construct compression functions in various modes.

2.2.1 Block Cipher: DES

DES was the most widely used block cipher in more than 20 years time before the adoption of current AES (Advanced Encryption Standard) algorithm in 2004. Although deemed insecure for many applications, DES remains influential and its strengthened variant TDEA¹ [146] is still effective and widely deployed in many systems. DES was initially developed by IBM as a commercial encryption product and its origins from the block cipher Lucifer², which is conceived from a late 1960s research project led by Horst Feistel. In 1974, IBM submitted DES for the call for proposals of a national cipher standard for civilian applications from the National Bureau of Standards (NBS)³, and it was adopted in 1977 as the Data Encryption Standard in FIPS PUB 46 [143].

DES is a 16 rounds Feistel cipher with 64-bit block size and 56-bit secret key, each round uses a 48-bit sub-key as a selection of 48 secret key bits by the key-schedule. In conjunction with the standard Feistel structure in Section 2.1.2, it has an *initial permutation* IP and a *final permutation* FP (with $FP := IP^{-1}$). The specification of the DES round function involves a 32-bit permutation P , a substitution function S implemented by a collection of eight 6-bit to 4-bit substitutions (Sboxes) and an expansion function E that linearly expands a 32-bit value to 48-bit. Let (L_0, R_0) be the two 32-bit halves of $IP(p)$ for a plaintext p , (L_i, R_i) be the state after i -th round, k_i be the sub-key of the i -th round for $1 \leq i \leq 16$, then each DES round can be described as:

$$L_i = R_{i-1}; \tag{2.1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i), \text{ where } f(R_{i-1}, k_i) = P(S(E(R_{i-1}) \oplus k_i)). \tag{2.2}$$

After 16 rounds, the ciphertext is output as $c := FP(R_{16}, L_{16})$. In fact, the initial and final permutations have no impact on the security of the cipher and they are crafted solely for hardware implementation efficiency. In function f , after the 32-bit value $R_{i-1} \oplus k_i$ gets expanded by E to 48 bits, the substitution layer divides them into 6-bit chunks and applies eight Sboxes $S_1, S_2, \dots, S_8$ in parallel. Since the Sboxes are 6-to-4, the S layer eventually outputs a 32-bit value, which is then fed into P . If an Sbox receives a 6-bit input of the

¹More commonly referred as Triple DES.

²Note that the name Lucifer was shared by a few early ciphers, such as in [74] and [173]

³Now known as the National Institute of Standards and Technology (NIST).

binary form $v_5v_4v_3v_2v_1v_0$, the output is then the entry with v_5v_0 as the row index and $v_4v_3v_2v_1$ as column index as the S_1 we describe in Table 2.1. An example is $S_1(100101) = 8$, with row index 11 and column index 0010. We describe E in Table 2.2 and P in Table 2.3, with bit 0 the LSB.

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Table 2.1: The first DES Sbox S_1

E						
31	0	1	2	3	4	
3	4	5	6	7	8	
7	8	9	10	11	12	
11	12	13	14	15	16	
15	16	17	18	19	20	
19	20	21	22	23	24	
23	24	25	26	27	28	
27	28	29	30	31	0	

Table 2.2: The E permutation of DES

P				
15	6	19	20	
28	11	27	16	
0	14	22	25	
4	17	30	9	
1	7	23	13	
31	26	2	8	
18	12	29	5	
21	10	3	24	

Table 2.3: The P permutation of DES

DES has seen an extensive amount of cryptanalysis since the late 70s and remains unbroken until early 1990s, a few short-cut attacks to the full cipher were discovered, including differential cryptanalysis (DC) [31], linear cryptanalysis (LC) [130] and improved Davies' attack [25]. Among these attacks, DC and LC are two powerful techniques general applicable to many ciphers, we will introduce them in Section 2.3.1 and Section 2.3.2 respectively. As these attacks are statistical in nature, it takes a huge amount of plaintext-ciphertext pairs to mount an attack. Another line of attacks effective on round-reduced DES variants and multiple cascaded DES are the meet-in-the-middle attacks, which are often of low data-complexity. We present in Chapter 5 such results. The DES key-schedule specification is presented in Section 5.1.2 since it is closely related to the security analysis in Section 5.1.2.

2.2.2 Block Cipher: IDEA

The International Data Encryption Algorithm (IDEA) is a 64-bit block cipher handling 128-bit keys and designed by Lai and Massey [119, 187] in 1990. Although it is patented ¹ in a number of countries and only freely available for non-commercial use, IDEA still received many applications such as PGP (Pretty Good Privacy) and OpenSSL. While its use is reducing over the recent years, it remains deployed in practice and has not been broken yet after extensive cryptanalysis efforts. It has a very simple design, performing 8.5 rounds composed of only 16-bit wide XOR, additions and multiplications. More precisely, one round is composed of three layers: first the key addition layer (denoted KA), a multiplication-addition layer (denoted MA) and a middle words switching layer (denoted S). For the eighth round, the switching is omitted.

Let X^i represent the 64-bit internal state of IDEA before application of the i -th round and we can view it as four 16-bit subwords $X^i = (X_1^i, X_2^i, X_3^i, X_4^i)$, with $1 \leq i \leq 9$. Also, $Y^i = (Y_1^i, Y_2^i, Y_3^i, Y_4^i)$ will stand for the intermediate internal state value of IDEA during the i -th round, right between the KA and the MA layers ². We denote by $\oplus$ the bitwise XOR operation, by $\boxplus$ the addition modulo 2^{16} and by $\odot$ the multiplication modulo $2^{16} + 1$, where the value 0 is considered as 2^{16} and vice-versa. Finally, $Z^i = (Z_1^i, Z_2^i, Z_3^i, Z_4^i, Z_5^i, Z_6^i)$ represents the six 16-bit sub-keys used during the i -th round (only the first four sub-keys for the last half round).

The KA layer simply incorporates four sub-keys:

$$Y_1^i = X_1^i \odot Z_1^i, \quad Y_2^i = X_2^i \boxplus Z_2^i, \quad Y_3^i = X_3^i \boxplus Z_3^i, \quad Y_4^i = X_4^i \odot Z_4^i.$$

The MA layer first computes $B = Z_6^i \odot ((Y_2^i \oplus Y_4^i) \boxplus (Z_5^i \odot (Y_1^i \oplus Y_3^i)))$ and $A = B \boxplus (Z_5^i \odot (Y_1^i \oplus Y_3^i))$. Then, after application of the S layer we have:

$$X_1^{i+1} = Y_1^i \oplus B, \quad X_2^{i+1} = Y_3^i \oplus B, \quad X_3^{i+1} = Y_2^i \oplus A, \quad X_4^{i+1} = Y_4^i \oplus A.$$

All the sub-keys are simply determined by choosing consecutive bits in the 128-bit master key according to the Table 2.4. Finally, ciphering the plaintext P with IDEA to obtain the

¹The patents are now expiring around 2011 to 2012.

²Note that the MA layer is different from what is usually called the “MA structure”.

ciphertext C is defined as: $C = \text{KA} \circ \text{S} \circ \{\text{S} \circ \text{MA} \circ \text{KA}\}^8(P)$. Figure 2.1 provides a schematic view of one round of IDEA.

Figure 2.1: One round of IDEA

i -th round	$Z_1^{(i)}$	$Z_2^{(i)}$	$Z_3^{(i)}$	$Z_4^{(i)}$	$Z_5^{(i)}$	$Z_6^{(i)}$
1	0-15	16-31	32-47	48-63	64-79	80-95
2	96-111	112-127	25-40	41-56	57-72	73-88
3	89-104	105-120	121-8	9-24	50-65	66-81
4	82-97	98-113	114-1	2-17	18-33	34-49
5	75-90	91-106	107-122	123-10	11-26	27-42
6	43-58	59-74	100-115	116-3	4-19	20-35
7	36-51	52-67	68-83	84-99	125-12	13-28
8	29-44	45-60	61-76	77-92	93-108	109-124
<i>OT</i>	22-37	38-53	54-69	70-85		

Table 2.4: Key bits used for sub-keys $Z_j^{(i)}$ in the i -th round of IDEA

Currently, the best cryptanalysis work published on IDEA [29] can reach 7.5 rounds with 2^{63} data (half the codebook) and 2^{114} computations. Concerning weak-keys, the current biggest weak-key class contains 2^{64} elements and has been published in [36].

2.2.3 Block Cipher: SERPENT

SERPENT is a block cipher designed by Ross Anderson, Eli Biham and Lars Knudsen [5]. It

was submitted to the Advanced Encryption Standard competition ¹ in 1997 and selected as a finalist after several years of public evaluation. The design philosophy is highly conservative and the designers expect the cipher to have a lifespan of more than 50 years [4]. As of today, only around 1/3 of its rounds can be attacked with certification weaknesses as for key-recovery attacks.

SERPENT is an SPN of 32 rounds and it has a 128-bit block size and supports secret key of sizes from 0 to 256 bits (while 128-, 192- and 256-bit keys are suggested as an AES submission). The cipher is of a simple design consisting an initial permutation IP , 32 rounds of SP-network and a final permutation FP ($FP := IP^{-1}$). Similar to the case of DES, IP and FP have no impact on security. To encrypt a 128-bit plaintext P , the key schedule first derives the list of sub-keys $\hat{K}_0, \dots, \hat{K}_{32}$ and the encryption is as follows. Let R_i be the i -th round function for $0 \leq i \leq 31$, $\hat{B}_i$ and $\hat{B}_{i+1}$ be the 128-bit states before and after R_i , then the cipher can be compactly described as follows: $\hat{B}_0 = IP(P)$, $\hat{B}_{i+1} = R_i(\hat{B}_i)$ and finally the ciphertext $C = FP(\hat{B}_{32})$. Specifically for a 128-bit state X , $R_i(X) = L(\hat{S}_i(X \oplus \hat{K}_i))$ for $i = 0, 1, \dots, 30$ and $R_{31}(X) = \hat{S}_{31}(X \oplus \hat{K}_{31}) \oplus \hat{K}_{32}$, where $\hat{S}_i$ is the parallel application 32 Sbox S_i and L is the linear transformation.

The linear transformation can be compactly implemented by a sequence of logical operations in the *bit-slice* mode, *i.e.*, with $B_i = IP^{-1}(\hat{B}_i)$ and $K_i = IP^{-1}(\hat{K}_i)$, and S_i the bit-slice version of the substitution. The hardware friendly bit-sliced round transformation (with L) can be described as:

¹An initial version known as SERPENT-0 was published in [24].

$$\begin{aligned}
X_0, X_1, X_2, X_3 &= S_i(B_i \oplus K_i) \\
X_0 &= X_0 \lll 13 \\
X_2 &= X_2 \lll 3 \\
X_1 &= X_1 \oplus X_0 \oplus X_2 \\
X_3 &= X_3 \oplus X_2 \oplus (X_0 \ll 3) \\
X_1 &= X_1 \lll 1 \\
X_3 &= X_3 \lll 7 \\
X_0 &= X_0 \oplus X_1 \oplus X_3 \\
X_2 &= X_2 \oplus X_3 \oplus (X_1 \ll 7) \\
X_0 &= X_0 \lll 5 \\
X_2 &= X_2 \lll 22 \\
B_{i+1} &= X_0, X_1, X_2, X_3.
\end{aligned}$$

In the last round, the linear transformation L is replaced by an additional key mixing: $B_{32} := S_7(B_{31} \oplus K_{31}) \oplus K_{32}$. We refer to other details such as the decryption, Sbox, and key-schedule to the specification [5].

2.2.4 Block Cipher: the KATAN/KTANTAN Family

KATAN and KTANTAN are two flavors of a family of hardware oriented block ciphers designed by Christophe De Cannière, Orr Dunkelman and Miroslav Knežević in 2009 [45]. Similar to the lightweight block cipher PRESENT [40], the design KATAN/KTANTAN is aiming at encryption needs on low-end devices such as RFID tags hence the cipher is targeting low gate count and reduced power consumptions. The ciphers come with 80-bit security level and are backed up by a large security margin on differential cryptanalysis and linear cryptanalysis, a few other attacks are considered additionally.

Both KATAN and KTANTAN have three variants with block sizes of 32-, 48- and 64-bit and each uses an 80-bit secret key. Moreover, KATAN and KTANTAN actually share the same data path specification, including the round transformation and round constants. The data path is best illustrated with Fig. 2.2 cited from the design paper. The round transformation is repeated for 254 rounds with a sequence of values of IR_i for $i = 1 \dots 254$.

The cipher state consists two parts L_1 and L_2 with different lengths. During the encryption, the plaintext is loaded into the two registers for L_1 and L_2 in little-endian. In each *round evaluation* of KTANTAN-32, L_1 (resp. L_2) is shifted to the left by 1 position with the

Figure 2.2: Illustration of KATAN/KTANTAN round transformation

Blocksize	$ L_1 $	$ L_2 $	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
32	13	19	12	7	8	5	3	18	7	12	10	8	3
48	19	29	18	12	15	7	6	28	19	21	13	15	6
64	25	39	24	15	20	11	9	38	25	33	21	14	9

Table 2.5: Parameters defined for the KATAN/KTANTAN family of ciphers

new LSB evaluated from $f_{b,r}(\cdot)$ (resp. $f_{a,r}(\cdot)$). $f_{a,r}$ and $f_{b,r}$ are the only nonlinear functions in the round transformation, defined as

$$f_{a,r}(L_1) = L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR_r) \oplus k_{a,r} \quad (2.3)$$

$$f_{b,r}(L_2) = L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_{b,r} \quad (2.4)$$

where $(k_{a,r}, k_{b,r})$ is a 2-bit sub-key of round r and IR_r is the r -th round constant. For KTANTAN-48 (resp. KTANTAN-64), each of the 254 rounds consists 2 (resp. 3) times of the round evaluation. Version specific constants can be found in Table 2.5 and IR in Table 2.6.

The round transformations are made different first by the introduction of IR , a 254-bit binary string termed the *irregular update sequence*. In addition, a sub-key (k_a, k_b) is

1111111000	1101010101	1110110011	0010100100	0100011000	1111000010
0001010000	0111110011	1111010100	0101010011	0000110011	1011111011
1010010101	1010011100	1101100010	1110110111	1001011011	0101110010
0100110100	0111000100	1111010000	1110101100	0001011001	0000001101
1100000001	0010				

Table 2.6: Round constant: the irregular update sequence

generated from the 80-bit secret key for each round. In fact the only difference between KATAN and KTANTAN is the key schedule. In this thesis, we focus on KTANTAN and hence discuss only its the key schedule. Two bits of the 80-bit $K = k_{79}k_{78} \dots k_1k_0$ are selected each round, by the state of an 8-bit LFSR defined by the feedback polynomial $x^8 + x^7 + x^5 + x^3 + 1$. With an initial value 0xFF, it is clocked 254 times. In each round the state of the LFSR is used to select two bits from K .

First divide K into five 16-bit words, let $K = w_4 || w_3 || w_2 || w_1 || w_0$, where the LSB of w_0 is the LSB of K and the MSB of w_4 is the MSB of K . Let $T_7T_6 \dots T_1T_0$ be the state of LFSR at a particular round r , then let $a_i = \text{MUX16to1}(w_i, T_7T_6T_5T_4)$ where MUX16to1 is a multiplexer that outputs the value of a bit in w_i with position specified by $T_7T_6T_5T_4$. After that, the sub-key bits $k_{a,r}$ and $k_{b,r}$ are selected as

$$k_{a,r} = \bar{T}_3 \cdot \bar{T}_2 \cdot (a_0) \oplus (T_3 \vee T_2) \cdot \text{MUX4to1}(a_4a_3a_2a_1, T_1T_0)$$

$$k_{b,r} = \bar{T}_3 \cdot T_2 \cdot (a_4) \oplus (T_3 \vee \bar{T}_2) \cdot \text{MUX4to1}(a_3a_2a_1a_0, \bar{T}_1\bar{T}_0)$$

where MUX4to1 is a multiplexer that selects 1 bit from 4, similar as MUX16to1. In fact, when the LFSR is clocked each round, the MSB is taken to form the irregular update sequence. The reference IR may be useful to verify the implementation of the key schedule. It is noted that the reference implementation has been updated several times, we list the entire key schedule in Table B.1 of Appendix B.1 to avoid ambiguity. A reference implementation can be found in [10].

2.3 Cryptanalysis of Block Ciphers

This section follows up the discussions on generic attacks in Section 2.1.3 by a few important cryptanalysis methods that require careful analysis of the internal workings of block ciphers. Perhaps it is not necessary to emphasize the importance of cryptanalysis beyond the simple fact that the state-of-art in designing secure ciphers is being consistently pushed forward by cryptanalysis. Indeed, cryptanalysis has been the dominating drive in advances of symmetric-key cryptology and the trust in the security of a cipher is completely established on cryptanalysis - the more quality analysis a cipher withstands, the more confidence

we have in its security.

As the world is today, ciphers are demanded and deployed almost everywhere, the need of secure ciphers can be so pivotal that cryptanalysis is often an important tool for security evaluations. Meanwhile, cryptanalyzing ciphers is an engaging problem solving process in its own right. As put by Alex Biryukov, cryptanalysis “has not, and will probably never become a routine”. Cryptanalytic results are presented as scientific discoveries while the process of doing cryptanalysis is essentially artistic. These are the best facets of problem solving.

2.3.1 Differential Cryptanalysis

Differential cryptanalysis was first announced by Eli Biham and Adi Shamir [30] around 1990 and it enables the first shortcut attack to the full DES [31]. In 20 years since it was rediscovered ¹, the technique has been researched and extended in many different ways, some of which are highlighted at the end of this section.

The basic idea is to analyze how random pairs of values with fixed *difference* propagate through components of the cipher in a non-random way, such that an important part of the cipher can be studied under a statistical model. For a vectorial Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ for $n, m > 0$, a *differential* of f (with respect to XOR) is written as $\Omega : \Delta_I \rightsquigarrow \Delta_O$, where Δ_I is the *input difference* and Δ_O is the *output difference* and the probability of Ω is defined as

$$Pr(\Omega) = |\{x \in \{0, 1\}^n : f(x) \oplus f(x \oplus \Delta_I) = \Delta_O\}|/2^n,$$

i.e., the probability that a random pair of values with input difference Δ_I result in an output difference Δ_O through f . The notion of differential is a convenient yet very powerful tool to study statistical behavior of the cipher with respect to random values with certain fixed input differences. When f is linear, differential of probability 1 can be easily found; when f is nonlinear, we usually search for differentials of good probabilities.

A cryptanalyst usually starts from searching a *differential characteristic*, which is a list of connected steps of difference propagation through components of the cipher, such that only

¹According to [53], the method was known during the design of DES but kept as a secret until 1994.

a small part of the nonlinear components are *activated* (i.e., probabilistic). A differential characteristic is a statistical modelization of the cipher and assumptions have to be made in order to obtain an estimation of its exact probability. A common practice requires active components with probabilistic differentials to behave independently, one such example is the general adoption of the *independent round key assumption*, which effectively treats the cipher rounds as independent layers. In fact the assumption holds to different extents on different block ciphers, accurate estimation of probabilities requires such assumptions to be well justified. In some ciphers a differential can be represented by more than one high probability differential characteristics, which have the same input/output differences but different intermediate round differences. At times more complicated models have to be used to evaluate the probabilities of differentials and characteristics. Once a cryptanalyst finds a good differential that covers the desired rounds of the cipher, he can then use it to carry out a last-round attack (see Section 2.1.4). We refer to [30] for greater details of differential cryptanalysis.

Differential cryptanalysis has been extensively studied with many extensions and generalizations. *Truncated differentials* was first used by Knudsen [108] in 1994 and *impossible differential cryptanalysis* first used by Knudsen [106] in 1998 which has been used in breaking reduced IDEA [27], 31 out of 32 rounds of Skipjack [26] and many others. In addition, there are *differential-linear cryptanalysis* [122], *higher-order differentials* [117], *higher-order differential cryptanalysis* [108], *boomerang attacks* [178] by David Wagner in 1999 and many more. It is also widely applied in cryptanalysis of hash functions. In Chapter 4, we employ basic tools of differential cryptanalysis to study the internal permutation of KECCAK in a *rebound attack* [137] setting, and in Chapter 6 we study special differential properties of IDEA in order to obtain collision and preimage attacks when IDEA is used to build compression functions in various hashing modes.

2.3.2 Linear Cryptanalysis

Linear cryptanalysis was formally introduced in 1993 by Matsui [130] as a subsequent work of similar techniques appeared in [134]. Second to differential cryptanalysis [31], the linear attack in [130] breaks full DES with an estimation of 2^{47} known plaintexts and was later improved to the first cryptanalytic break of the full DES by computer experiments [131],

requiring 2^{43} known plaintexts and a running time of 10 days to recover the 56-bit key with twelve computers (HP9735/PA-RISC 99MHz).

The most important notion in linear cryptanalysis is a *linear approximation*. For a vectorial Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ for $n, m > 0$, a linear approximation Ω of f is written as $\lambda_I \cdot X \oplus \lambda_O \cdot f(X) = 0$, where $\lambda_I \in \{0, 1\}^n$, $\lambda_O \in \{0, 1\}^m$ and Δ_I (resp. Δ_O) is called the *input mask* (resp. *output mask*). A mask λ for X ($\lambda, X \in \{0, 1\}^n$) is a *selection pattern* for computing the XOR-sum of selected bits in X , i.e., $\lambda \cdot X = \bigoplus_i \lambda_i \cdot X_i$. And the probability that Ω holds is computed as

$$Pr(\Omega) = |\{x \in \{0, 1\}^n : \lambda_I \cdot x \oplus \lambda_O \cdot f(x) = 0\}|/2^n,$$

i.e., the probability that the XOR-sum of the selected bits equals zero. When $Pr(\Omega) \neq 1/2$, the correlation between $\lambda_I \cdot X$ and $\lambda_O \cdot f(X)$ is usually quantified as a *bias* $\epsilon := Pr(\Omega) - 1/2$.

When analyzing block ciphers, a linear approximation usually involves bits of the plaintext, the ciphertext and the extended key. Similar to differential cryptanalysis, a cryptanalyst starts analyzing the cipher by searching a *linear characteristic*, which is derived from a connected series of linear approximations of components of the cipher. We refer the estimation of probabilities for linear characteristics to the famous *Piling-up Lemma* [130, Lemma 3].

Matsui suggests in [130] two algorithms for attacking block ciphers. We briefly introduce Algorithm 1. Suppose a block cipher $E_K(\cdot)$ has a linear approximation $\Omega : uX \oplus vY \oplus wK = 0$ and $Pr(\Omega) = 1/2 + \epsilon$, where u , v , and w are the selection patterns for plaintext X , ciphertext Y and the extended key K respectively and $\epsilon > 0$. For each pair (x, y) from a pool of random N known plaintexts, one can increase a counter T if $ux \oplus vy$ evaluates to zero. When N is sufficiently large, the value of T will have an observable deviation from $N/2$. Consequently, one can guess the key parity cK as 0 if $T > N/2$, and $cK = 1$ if $T < N/2$. Note that if $\epsilon < 0$ the guesses would be reversed. The amount of random known plaintexts needed is proportional to c/ϵ^2 , where c is a constant which determines the success probability of the attack.

Algorithm 2 uses linear approximations as distinguishers in a last-round style attack, we refer the details to [130]. Linear cryptanalysis was generalized and studied in various forms

such as with nonlinear approximations [84, 114, 172, 98], partitioning cryptanalysis [85], and with more than one approximations [131, 97, 34, 52, 88, 89] which we follow up in Chapter 3.

2.4 Cryptographic Hash Functions

2.4.1 Security and Attacks

Hash functions are considered as a very important building block for many security and cryptography applications. Informally, a hash function H is a function that takes an arbitrarily long message as input and outputs a fixed-length hash value of size n bits. In cryptography, we want these functions to fulfill three security requirements, namely collision resistance and (second) preimage resistance. It should be impossible for an adversary to find a collision (two different messages that lead to the same hash value) in less than $2^{n/2}$ hash computations, or a (second) preimage (a message hashing to a given challenge) in less than 2^n hash computations. Most of nowadays hash functions divide the whole input message into blocks after padding it, and then process the blocks in an iterative way. A very known and utilized example is the Merkle-Damgård algorithm [59, 138], which uses an n -bit compression function h in order to process the m message blocks M_i : $CV_{i+1} = h(CV_i, M_i)$, where CV_i is the n -bit internal state (or chaining variable) that is initialized by a fixed public value $CV_0 = IV$ and the final hash value is H_m . This algorithm is very interesting because it allows to reduce the collision security of the hash function to the collision security of the compression function. However, in order to guarantee the soundness of the construction, a designer must ensure that an attacker can not break the collision/preimage resistance of the compression function. One can

- *free-start collision*: in less than $2^{n/2}$ computations, find two different pairs $(CV, M) \neq (CV', M')$ such that they lead to the same compression function output value: $h(CV, M) = h(CV', M')$,
- *semi-free-start collision*: in less than $2^{n/2}$ computations, find one chaining variable CV and two different message blocks $M \neq M'$ such that they lead to the same compression function output value: $h(CV, M) = h(CV, M')$,

- *pseudo-preimage*: in less than 2^n computations, find one chaining variable CV and one message block M such that they lead to a given output challenge X : $h(CV, M) = X$.

Note that a semi-free-start collision for the compression function where the chaining variable CV is not chosen by the attacker directly leads to a collision for the whole hash function. In any case, a semi-free-start collision is very dangerous since it means that for some choices of IV , the attacker knows how to generate a collision. Even free-start collision are considered serious as they invalidate the collision resistance assumption on the compression function and we have seen many free-start collision attacks eventually turning into full hash collision attacks in the recent history (for example free-start collision attacks for MD5 were quickly identified [64], then upgraded to semi-free-start collision attacks [69] and eventually to full collision attacks [180]). As for preimage attacks on the compression function (also known as pseudo-preimages), they are very relevant since there exist a meet-in-the-middle algorithm that in most cases can turn them into a preimage attack for the full hash function.

2.4.2 Using Block Ciphers as Compression Functions

The separation between a block cipher and a compression function has always been blurry. Constructions are known to turn the former into the latter [38, 157] or the latter into the former [128]. For example, the Davies-Meyer mode [2] converts a block cipher E into a compression function and is incorporated in a large majority of the currently known hash functions. While very satisfying solutions exist to transform a secure n -bit block cipher into an n -bit compression function (Davies-Meyer, Miyaguchi-Preneel, Matyas-Meyer-Oseas modes [2] or see [38, 157] for a systematic study of this problem), there is still a lot of research being actively conducted on double-block-length compression functions (where the block cipher size is n bits and the compression function output size is $2n$), from simple-key block ciphers such as AES-128 or double-key such as AES-256 [58].

A major difference between the cryptanalysis of block ciphers and compression functions is that the attacker can fully control the inner behavior of the compression function. In other words, the attacker can use more efficiently the freedom degrees available on the input (*i.e.*, the number of independent binary variables he has to determine). A new security model for

block ciphers, the so-called *known-key model* [113], was recently proposed in order to fill the gap between these two situations. In this model, the secret key is known to the adversary and its goal is to distinguish the behavior of a random instance of the block cipher from the one of a random permutation by constructing a set of (plaintext, ciphertext) pairs satisfying an *evasive* property. Such a property is easy to check but impossible to achieve with the same complexity and a non-negligible probability using oracle accesses to a random permutation and its inverse. In general, these known-key attacks are not regarded as problematic when the block cipher is used in a classical “secret key” setting. Moreover, it is rare that such threats are extended to attacks on the compression function.

2.4.3 The SHA-3 Finalist KECCAK

KECCAK [19, 20] is a family of variable length output hash functions based on the sponge construction [14]. In KECCAK family, the underlying function is a permutation chosen from a set of seven KECCAK- f permutations, denoted as KECCAK- $f[b]$ where $b \in \{1600, 800, 400, 200, 100, 50, 25\}$ is the permutation width as well as the internal state size of the hash function. The KECCAK family is parametrized by an r -bit bitrate and c -bit capacity with $b = r + c$.

A variable length output hash function KECCAK- $[r, c]$ is obtained by applying the sponge construction to the permutation KECCAK- $f[b]$ and a padded input message. The KECCAK hash functions can be derived by assigning the desired r and c parameters that satisfy the constraint $b = r + c$.

b	n_r
1600	24
800	22
400	20
200	18
100	16
50	14
25	12

Table 2.7: Parameters for KECCAK hash functions

A fixed length n -bit hash value for a KECCAK hash function is obtained by truncating the output of the hash function to the first n bits and this function is denoted by KECCAK $_n$. For these variants, if at least 2^n (second) preimage resistance security is desired then the

parameter c is chosen such that $c \geq 2n$. The proposed version of KECCAK for the SHA-3 standardization uses KECCAK- $f[1600]$ as internal permutation. The default variant of KECCAK family is denoted by KECCAK- $[\]$ and it has parameters $r = 1024$, $c = 576$ for any output length n .

2.4.3.1 The Domain Extension Algorithm

Sponge is an iterated construction to build a function $F : \{0, 1\}^* \rightarrow \{0, 1\}^*$ by using a fixed length transformation or permutation f whose width is fixed to b bits. This construction operates on a state of b . The initial value of the state is zero and the input message is padded such that it is a multiple of r -bit message blocks (and the last message block is different than zero).

Message processing using sponge construction has two stages: absorbing phase and squeezing phase. In the absorbing phase, r -bit message blocks are XOR-ed with the first r bits of the b -bit state, interleaved with the applications of f until all blocks are processed. In the squeezing phase, depending on the required number of output bits n , the first r bits of the state are returned as output blocks, interleaved with the applications of f until n bits are returned.

2.4.3.2 The KECCAK-f Permutations

The internal state of the KECCAK family can be viewed as a bit array of 5×5 lanes, each of length $w = 2^\ell$ where $\ell \in \{0, 1, 2, 3, 4, 5, 6\}$ and $b = 25w$. The state can also be described as a three dimensional array of bits defined by $a[5][5][w]$. A bit position (x, y, z) in the state is given by $a[x][y][z]$ where x and y coordinates are taken over modulo 5 and the z coordinate is taken over modulo w . A *lane* of the internal state at *column* x and *row* y is represented by $a[x][y][\cdot]$, while a *slice* of the internal state at *width* z is represented by $a[\cdot][\cdot][z]$.

KECCAK- $f[b]$ is an iterated permutation consisting of a sequence of n_r rounds indexed from 0 to $n_r - 1$ and the number of rounds are given by $n_r = 12 + 2\ell$. A round $\mathbf{R}$ consists of a transformation of five step mappings and is defined by:

$$\mathbf{R} = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

These step mappings are discussed below where the operation $+$ indicates bitwise addition.

θ mapping. This linear mapping intends to provide diffusion for the state and is defined for every x, y and z by:

$$\theta : a[x][y][z] \leftarrow a[x][y][z] + \bigoplus_{y'=0}^4 a[x-1][y'][z] + \bigoplus_{y'=0}^4 a[x+1][y'][z-1]$$

That is, the bitwise sum of the two columns $a[x-1][\cdot][z]$ and $a[x+1][\cdot][z-1]$ is added to each bit $a[x][y][z]$ of the state.

ρ mapping. This linear mapping intends to provide diffusion between the slices of the state through intra-lane bit translations. For every x, y and z :

$$\rho : a[x][y][z] \leftarrow a[x][y][z + T(x, y)]$$

where $T(x, y)$ is a translation constant. That is, all bit positions in each lane are translated by a constant amount that depends on the column x and row y considered. The translation constants for KECCAK- $f[1600]$ are given in Table A.2 in the Appendix.

π mapping. This linear mapping intends to provide diffusion in the state through transposition of the lanes. More precisely, it is defined for every x, y and z as:

$$\pi : a[x'][y'][z] \leftarrow a[x][y][z], \text{ with } \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Since this results in transposition of bits into a same slice, this mapping is an intra-slice transposition. The transposition for KECCAK- $f[1600]$ is given in Table A.3 in the Appendix.

χ mapping. This is the only nonlinear mapping of KECCAK- f and is defined for every x, y and z by:

$$\chi : a[x][y][z] \leftarrow a[x][y][z] + ((\neg a[x+1][y][z]) \wedge a[x+2][y][z])$$

This mapping is similar to an Sbox applied independently to each 5-bit row of the state

(see Table A.1 in the Appendix) and can be computed in parallel to all rows. We represent by $s = 5w$ the number of Sboxes/rows in KECCAK internal state. Here $\neg$ denotes bit-wise complement, and $\wedge$ the bit-wise AND.

ι mapping. For every round $\mathbf{R}$ of the KECCAK- f permutation, this mapping adds constants derived from an LFSR (see [19] for details) to the lane $a[0][0][\cdot]$. These constants are different in every round i :

$$\iota : a[0][0][\cdot] \leftarrow a[0][0][\cdot] + \text{RC}[i]$$

This mapping aims at destroying the symmetry introduced by the identical nature of the remaining mappings in every round of the KECCAK- f permutation.

We refer to the KECCAK specifications document [19] for all the translation and round constants.

2.4.4 Hash Functions Based on IDEA

We will study the security of the various block cipher-based constructions that can use IDEA as the internal primitive. Therefore, we only consider the ones that use a double-key block cipher. More precisely, we denote $C = E_K(P)$ the process of ciphering the 64-bit plaintext P with IDEA using the 128-bit key K .

2.4.4.1 Simple-Length Compression Functions

A simple-length compression function construction with IDEA will provide a 64-bit output CV_{i+1} .

Davies-Meyer (Figure 2.3) is the most usual simple-length mode [2] and it handles 128-bit message blocks: $CV_{i+1} = E_M(CV_i) \oplus CV_i$. Most standardized hash functions are actually implementing this mode, with an ad-hoc internal block cipher. While some weaknesses such as fixed-points are known, its security in terms of preimage and collision resistance have been studied and proved in the ideal cipher model [38]. Namely, we should expect at least 2^{32} and 2^{64} computations respectively to generate a (semi)-free-start collision or preimage for the compression function. Note that Miyaguchi-Preneel and Matyas-Meyer-Oseas simple-block

length modes [2] are not considered in this thesis since they require the internal primitive to have the same block and key size, which is not the case for the IDEA block cipher.

Figure 2.3: Davies-Meyer mode

2.4.4.2 Double-Block-Length Compression Functions

A more interesting design strategy with IDEA would be to define double-block-length constructions, in order to get 128-bit output, represented by two 64-bit words $CV1_i$ and $CV2_i$. This problem has already been studied a lot and remains a very active research domain, even when the internal primitive is a double-key block cipher.

Abreast-DM and tandem-DM both have been especially designed for IDEA [120, 187].

Tandem-DM (Figure 2.5) handles a 64-bit message block M . We define $W = E_{CV1_i || M}(CV2_i)$ and then we have

$$\begin{aligned} CV1_{i+1} &= E_{M || W}(CV1_i) \oplus CV1_i, \\ CV2_{i+1} &= W \oplus CV2_i. \end{aligned}$$

Abreast-DM (Figure 2.4) also handles a 64-bit message block M :

$$\begin{aligned} CV1_{i+1} &= E_{M || CV2_i}(\overline{CV1_i}) \oplus CV1_i, \\ CV2_{i+1} &= E_{CV1_i || M}(CV2_i) \oplus CV2_i, \end{aligned}$$

where $\overline{X}$ stands for the bitwise complement of X .

Hirose proposed a construction (Figure 2.6) that contains two independent block cipher instances [90], later improved to only a single instance [91] by using a constant c to simulate

Figure 2.4: Abreast-DM DBL mode

Figure 2.5: Tandem-DM DBL mode

Figure 2.6: Hirose DBL mode

the two independent ciphers:

$$\begin{aligned} CV1_{i+1} &= E_{CV2_i || M}(CV1_i) \oplus CV1_i, \\ CV2_{i+1} &= E_{CV2_i || M}(CV1_i \oplus c) \oplus CV1_i \oplus c. \end{aligned}$$

Peyrin *et al.* described in [155] a compression function (denoted Peyrin *et al.*(II), see Figure 2.7) that utilizes 5 calls to independent $3n$ -to- n -bit compression functions, advising to be instantiated with double-key internal block ciphers such as AES-256 or IDEA. It handles two 64-bit message blocks $M1$ and $M2$:

$$\begin{aligned} CV1_{i+1} &= f_1(CV1_i, CV2_i, M1) \oplus f_2(CV1_i, CV2_i, M2) \oplus f_3(CV1_i, M1, M2), \\ CV2_{i+1} &= f_3(CV1_i, M1, M2) \oplus f_4(CV1_i, CV2_i, M1) \oplus f_5(CV2_i, M1, M2), \end{aligned}$$

where the functions f_i can be build for example by using the IDEA block cipher into a Davies-Meyer mode and we can simulate their independency by XORing distinct constants to the plaintext inputs, as it is done in [91]: $f_i(U, V, W) = E_{U || V}(W \oplus i) \oplus W$ (note that XORing the constants on the key input would be avoided in practice because it would lead to very frequent rekeying and therefore reduce the overall performance of the hash function).

Since no real candidate was proposed by the authors, all possible position permutations of the three f_i inputs will be considered. Note that when cryptanalyzing this scheme, we will attack the functions f_i independently. Thus, we will not use any weakness coming from potential dependencies between the functions f_i (apart of course that all 5 functions are based on IDEA).

Figure 2.7: Peyrin *et al.* (II) mode

MJH-Double is a rate 1 double-block-length compression function (Figure 2.8) recently published by Lee and Stam [125]. It uses a double-key block cipher and handles two 64-bit message blocks $M1$ and $M2$:

$$\begin{aligned} CV1_{i+1} &= E_{M2||CV2_i}(CV1_i \oplus M1) \oplus CV1_i \oplus M1, \\ CV2_{i+1} &= g \cdot (E_{M2||CV2_i}(f(CV1_i \oplus M1)) \oplus f(CV1_i \oplus M1)) \oplus CV1_i, \end{aligned}$$

where f is an involution with no fixed point and $g \neq 0, 1$ is a constant.

For all these double-block-length proposals, the conjectured security is 2^{64} and 2^{128} computations respectively to generate a (semi)-free-start collision or preimage for the compression function or hash function.

Figure 2.8: MJH-Double mode

Chapter 3

Multidimensional Linear Cryptanalysis of SERPENT

In this chapter, we study advances in linear cryptanalysis with a number of approximations, in particular we focus on the framework of multidimensional linear cryptanalysis proposed by Hermelin *et al.* [88]. Several improvements and observations on the multidimensional extension of Matsui’s Algorithm 1 are obtained [150] and we present in this chapter the following results: first, we prove a circulant matrix property, which enables the computation of the Kullback-Leibler distance being improved from $\mathcal{O}(2^{2m})$ to $\mathcal{O}(m2^m)$ in the Analysis phase of the algorithm. The algorithm for this improvement is named Method-B in [150]; we then analyze the block cipher SERPENT reduced to 4 rounds and compare the results with the attack in [88], illustrating the efficiency improvements over the Hermelin *et al.* version; at last, we show that the multidimensional Algorithm 1 has a significant advantage over the classic linear attacks by a much improved result on 9-round SERPENT, based on the linear approximations documented in [12]. Our analysis based on the improved attacks implies that multidimensional linear cryptanalysis with Algorithm 1 is particularly effective when there is a good trade-off between the magnitude of correlations and the amount of exploitable linear approximations that can be formed, *e.g.*, in the case of SPN block cipher with 4-bit Sboxes.

The roadmap for this chapter is as follows. In Section 3.1 we review historical developments on linear cryptanalysis with multiple approximations. In Section 3.2, we give the

necessary background for understanding the multidimensional Algorithm 1 and the motivation of proving the circulant matrix property, followed by a brief description of the algorithm Method-B. In Section 3.3 we give the details of the circulant matrix proof. Then we describe the attacks to reduced SERPENT in Section 3.4 and 3.5. At last the improvements to the multidimensional Algorithm 1 are discussed and compared with previous versions in the concluding section. The original contributions of this thesis are from Section 3.3 to Section 3.5.

3.1 Historical Developments

Since linear cryptanalysis emerged in [134, 130], the attack algorithms have been formally introduced by Matsui as Algorithm 1 (Alg. 1) and Algorithm 2 (Alg. 2) and the attack successfully breaks the full DES both in theory [130] and computer experiments [131]. Since then researchers started questioning possible improvements to the initial version of the attack. In particular, we show here several developments on improving the basic forms of Alg. 1 and 2, by making use of multiple linear approximations simultaneously. The earliest of such attempt was in [131], where 2 approximations are used to reduce the data complexity. In 1994, Kaliski and Robshaw [97] combine m linear approximations $g_1, \dots, g_m$, with biases of $\epsilon_1, \dots, \epsilon_m$ respectively. The linear approximations are required to have the same selection pattern c for the extended key. N can achieve an m -fold reduction while keeping the same success rate, with N proportional to $1/\sum_{i=1}^m \epsilon_i^2$. Due to the restriction on the key mask, it recovers at most 1-bit parity – $c \cdot K$. Intuitively speaking, asking 1 parity bit from m approximations simultaneously would require much fewer samples than from a single approximation, under the same success rate.

In 2004 Biryukov *et al.* [34] show a statistical framework for using multiple approximations in both Alg 1 and Alg 2. In their generalization to Alg. 1, m statistically independent linear approximation $g_1, \dots, g_m$ are used, with biases of $\epsilon_1, \dots, \epsilon_m$ respectively. The attack is able to recover at most m parity bits if successful, with N proportional to $1/\sum_{i=1}^m \epsilon_i^2$. In practice, m' linearly dependent masks are used with $m' > m$. It is expected that the performance in this case is strictly better than that of m independent approximations, although no explicit estimation of N was given. They confirmed the reduction in data complexity by

using $m' = 86$ approximations for 8-round DES, the 86 approximations only give 10 linearly independent key masks. In 2007, this reduction in N by using multiple approximations is further confirmed by experiments of Collard *et al.* [52], with $m' = 64$ for 4-round SERPENT from 10 linearly independent text masks. They also observed that the gain increases 8 times faster with 64 approximations than with 10 approximations. Both [34] and [52] did not analyze why such an advantage is present. Some intuitive guesses were given by Collard *et al.* [52], referring to linear hulls, error correcting codes, etc. They also noticed that in practice, m and m' cannot be too large due to computation limits.

Significant reduction in data complexity can be achieved by the methods in [34]. However, in practice, it is generally not easy to verify whether the m approximations are statistically independent. Instead, linear independence is used as a criterion for the m approximations, in both [34] and [52]. In the meantime, it is also helpful to think whether there is a better attack if the m approximations are linearly independent in text masks but statistically correlated, as shown by the experimental results in [34] and [52]. Notably, these experiments work with linearly dependent text masks.

Hermelin *et al.* [88] introduced a multidimensional framework, in which Matsui's Alg. 1 is generalized to m dimensions to exploit correlations between the m approximations, hence to achieve a higher capacity. Experiments on 4-round SERPENT have shown that this method reduces N compared with a similar attack in [52]. In this framework, the requirement for statistical independence of approximations is relieved. Instead, the approximations only need to have linearly independent text masks. This resembles the experiment scenarios in [34] and [52]. In fact, in [88] it is shown clearly that the statistical independence assumption of [34] does not hold.

In [88], the formula for N is derived as the amount of data needed to tackle the $|\mathcal{Z}|$ -ary hypothesis testing problem, where $\mathcal{Z}$ is the set of key classes. It does not reflect how it depends on the approximations $g_1, \dots, g_m$ and it is not efficiently computable. The reduction in N compared with [34] was observed empirically, rather than theoretically. The work [150] provides and proves a much simpler theoretical formula for the number of samples required, to complement this theoretical framework in [88]. This formula gives insights on how much can be achieved with multidimensional linear cryptanalysis. We can now easily estimate N given $g_1, \dots, g_m$, hence the attack complexity. The simplicity of the formula

eases cryptanalysis greatly since to compute N with the original formula in [88] (for N_{key}) requires a lot of computation when m is large.

A major obstacle for the method in [88] to be useful is that the number of approximations m is much limited due to complexity bottlenecks in the distillation phase ($2^m N$) and the analysis phase (2^{2m}) of the online stage. This limitation hinders the application of multidimensional Alg. 1 in practice when more approximations have to be used. We show in Section 3.3 the result that leads to Method-A proposed in [150]. For distributions in the distillation phase, Method-A speeds up the computation to $2m2^m + mN$, using ideas from [51]. Method-B in [150] reduces the complexity of the analysis phase to $3m2^m$. We will compare the improved algorithms for the multidimensional generalization of Matsui's Alg. 1 with previous algorithms in Section 3.3 and illustrate the performance of the improved algorithm with two attacks in Section 3.4 and 3.5.

3.2 Background on Multidimensional Algorithm 1

In this section, we first give the notations and some basic notions of vectorial Boolean functions, following [88]. Then we state the definitions and theorems necessary for understanding the multidimensional Alg. 1. Finally we state the attack algorithm and how it can be improved with the property in Section 3.3.

3.2.1 Notions and Definitions

Denote the space of m -dimensional binary vectors by V_m or $V_m := GF(2)^m$. The inner product of 2 vectors $a = (a_1, \dots, a_m), b = (b_1, \dots, b_m), a, b \in V_m$ is $ab = \bigoplus_{i=1}^m a_i b_i$. The function $f : V_m \rightarrow V_1$ is called a Boolean function and $f : V_n \rightarrow V_m, f = (f_1, \dots, f_m)$ is called a vectorial Boolean function, where each f_i is a Boolean function for all $i = 1, \dots, m$. Let X be a random variable (r.v.) in V_m . Let $p_\eta = Pr(X = \eta)$, with $\eta \in V_m$. Then $p = (p_0, p_1, \dots, p_{2^m-1})$ is the probability distribution (p.d) of r.v. X . If we associate with a vectorial Boolean function $f : V_n \rightarrow V_m$ an r.v. $Y := f(X)$, where X is uniformly distributed in V_m , then the p.d. of Y is $p(f) := (p_0(f), \dots, p_{2^m-1}(f))$ where $p_\eta(f) = Pr(f(X) = \eta)$, for all $\eta \in V_m$. Two Boolean functions f and g are called statistically independent if their associated r.v.'s $f(X)$ and $g(Y)$ are statistically independent, with X, Y uniform in V_n .

The correlation between a binary r.v. X and 0 is $\rho = Pr(X = 0) - Pr(X = 1) = 2\epsilon$, where ϵ is the bias of the r.v. X . Let $g : V_m \rightarrow V_1$ be a Boolean function. Its correlation with 0 is defined as

$$\rho = 2^{-m}(\#\{\eta \in V_m | g(\eta) = 0\} - \#\{\eta \in V_m | g(\eta) = 1\}) = 2Pr(g(X) = 0) - 1,$$

where X is uniformly distributed in V_m .

Definition 3.2.1. Let $p = (p_0, \dots, p_M)$ and $q = (q_0, \dots, q_M)$ be two p.d.'s. Then their (mutual) capacity is

$$C(p||q) = \sum_{\eta=0}^M \frac{(p_\eta - q_\eta)^2}{q_\eta}. \quad (3.1)$$

Definition 3.2.2. The relative entropy or the Kullback-Leibler (KL) distance between two distributions $p = (p_0, \dots, p_M)$ and $q = (q_0, \dots, q_M)$ is defined as

$$D(q||p) = \sum_{\eta=0}^M q_\eta \log \frac{q_\eta}{p_\eta}.$$

In [51], Collard *et al.* presented the following theorems concerning circulant matrices.

Theorem 3.2.1. A circulant $\mathbf{S}$ of level k and type $(m, n, o, \dots, r)$ is diagonalizable by the unitary matrix $\mathbf{F} = \mathbf{F}_m \otimes \mathbf{F}_n \otimes \mathbf{F}_o \otimes \dots \otimes \mathbf{F}_r$

$$\mathbf{S} = \mathbf{F}^* \text{diag}(\lambda) \mathbf{F},$$

where λ is the vector of eigenvalues of $\mathbf{S}$, the symbol $\otimes$ is the Kronecker product and $\mathbf{F}_n$ is the Fourier matrix of size $n \times n$ defined by:

$$\mathbf{F}_n(i, j) = \frac{1}{\sqrt{n}} \omega^{ij}, \quad (0 \leq i, j \leq n-1)$$

with

$$\omega = e^{\frac{2\pi\sqrt{-1}}{n}}.$$

Theorem 3.2.2. The eigenvalues vector λ of a circulant matrix $\mathbf{S}$ of level k and type

$(m, n, o, \dots, r)$ can be computed with the following matrix-vector product:

$$\lambda = \mathbf{FS}(:, 1) \sqrt{mno \dots r}$$

where $\mathbf{S}(:, 1)$ means we take the first column of $\mathbf{S}$.

We recall important results on the Fast Fourier Transform [54], Fast Walsh-Hadamard Transform [188] and Parseval's theorem. Given an M -dimensional vector $\mathbf{E} = (E_1, \dots, E_M)$ and a matrix $\mathbf{F}^{M \times M}$, we have M -dimensional vector

$$\mathbf{D} = \mathbf{F}\mathbf{E}^T,$$

where $\mathbf{E}^T$ is the transpose of $\mathbf{E}$ and $\mathbf{F}$ is a Hadamard matrix if $\mathbf{F}(i, j) = (-1)^{ij}$, for all $i, j = 0, \dots, M-1$. If matrix $\mathbf{F}$ is either Fourier or Hadamard, vector $\mathbf{D}$ can be computed with complexity $\mathcal{O}(M \log M)$ instead of $\mathcal{O}(M^2)$ by Fast Fourier Transform or Fast Walsh-Hadamard Transform, respectively. We recall basic facts due to Parseval:

Let $f : V_m \rightarrow \mathcal{R}$ where $\mathcal{R}$ is the real field, and $a \in V_m$. We define $f_1(a) = \sum_{b \in V_m} (-1)^{ab} f(b)$, $A = \sum_{a \in V_m} f^2(a)$ and $A_1 = \sum_{a \in V_m} f_1^2(a)$. Then

$$2^m A = A_1$$

or

$$2^m \left(\sum_{a \in V_m} f^2(a) \right) = \sum_{a \in V_m} f_1^2(a). \quad (3.2)$$

3.2.2 Multidimensional Probability Distribution

Let $f : V_l \rightarrow V_n$ be a vectorial Boolean function, binary vectors $w_i \in V_n$ and $u_i \in V_l$ for $i = 1, \dots, m$ be selection patterns such that pairs of input and output masks (u_i, w_i) are linearly independent. Define the functions g_i as

$$g_i(\eta) = w_i f(\eta) \oplus u_i \eta, \quad \forall \eta \in V_l, i = 1, \dots, m$$

and g_i has correlation ρ_i , $i = 1, \dots, m$. Then $\rho_1, \dots, \rho_m$ are called the base correlations, and $g_1, \dots, g_m$ are the base approximations of f . Let $g = (g_1, \dots, g_m)$ be an m -dimensional

vectorial Boolean function, and matrices $W = (w_1, \dots, w_m)$ and $U = (u_1, \dots, u_m)$ contain the output and input masks for each of the g_i , then we find the p.d. $p = (p_0, \dots, p_{2^m-1})$ of $g(\eta) = Wf(\eta) \oplus U\eta$.

Lemma 3.2.1. [88] *Let $g = (g_1, \dots, g_m) : V_l \rightarrow V_m$ be a vectorial Boolean function and $p = (p_0, \dots, p_{2^m-1})$ its p.d. Then*

$$2^l p_\eta = 2^{-m} \sum_{a \in V_m} \sum_{b \in V_l} (-1)^{a(g(b) \oplus \eta)}.$$

Define

$$\rho(a) = 2^{-l} \sum_{b \in V_l} (-1)^{ag(b)} = \Pr(ag(X) = 0) - \Pr(ag(X) = 1),$$

where X is an r.v. uniformly distributed in V_l .

Corollary 3.2.1. [88] *Let $g : V_n \rightarrow V_m$ be a Boolean function with p.d. p and correlations $\rho(a)$ of the combined approximations ag , for all $a \in V_m$. Then for $\eta \in V_m$,*

$$p_\eta = 2^{-m} \sum_{a \in V_m} (-1)^{a\eta} \rho(a). \quad (3.3)$$

3.2.3 Multidimensional Algorithm 1

The core idea of the multidimensional Alg. 1 in [88] is briefly introduced in this section. Let there be m linear approximations $g_i := u_i X \oplus v_i Y \oplus c_i K$, ($i = 1, \dots, m$). The corresponding correlation for g_i is

$$\rho_i := 2\Pr(u_i X \oplus v_i Y \oplus c_i K = 0) - 1$$

where the masks c_i for the extended key K are linearly independent. In addition, the pairs of input and output masks (u_i, v_i) are linearly independent.

Define $g := (g_1, \dots, g_m)$ with p.d. p , and $h := (h_1, \dots, h_m)$, where $h_i = u_i X \oplus v_i Y$. We call h an *experimental function*, as we use two of its probability distributions, namely, the theoretical p.d. q and the empirical p.d. $\hat{q}$. In the attack, q is approximated by $\hat{q}$, which is computed from N samples. Let $w_i = c_i K$, $i = 1, \dots, m$ be the parity bits of the extended key K . As the c_i 's are linearly independent, $w = (w_1, \dots, w_m)$ defines a key class of K .

Thus we have $g = h \oplus w$.

Hence, the p.d. q of experimental function h is a permutation of p . Since $\{c_i\}$ are linearly independent, with the 2^m possible parity vectors w , the key space K can be classified into 2^m classes. Let w^* be the correct key class, since p^{w^*} is the permutation of p corresponding to w^* , we have $q = p^{w^*}$. By [49], given w , the relationship between p^w and p is

$$p_\eta^w = \sum_{a \in V_m} (-1)^{a(\eta \oplus w)} \rho(a) = p_{\eta \oplus w}, \quad \forall \eta \in V_m. \quad (3.4)$$

The KL distance is then used to determine the correct key class w^* , by constructing a hypothesis testing problem of finding the closest distribution p^w with $\hat{q}$ among the 2^m possibilities of w . In [88] a theorem on the lower bound of N is given by solving a $|\mathcal{Z}|$ -ary hypothesis testing problem. However the method is neither efficient nor providing an accurate estimation.

Given $g = (g_1, \dots, g_m)$ and $\rho(a)$, for all $a \in V_m$, the capacity of the attack of [88] is $\min_{w \neq 0} C(p^0 || p^w)$. Let $p = p^0$ and $q = p^w$, Lemma 3.2.2 holds and further leads to Theorem 3.2.3.

Lemma 3.2.2. [150]

$$C(p || q) \geq 2 \sum_{\forall a \in V_m \setminus \{0\}} \rho^2(a).$$

Theorem 3.2.3. [150] *The estimation for N in the attack of [88] with m linear approximations g_i , ($i = 1, \dots, m$), where the $\{c_i\}$ are linearly independent and (u_i, v_i) are linearly independent, is given by*

$$N \approx \frac{m}{2 \sum_{\forall a \in V_m \setminus \{0\}} \epsilon^2(a)}. \quad (3.5)$$

Now we show the multidimensional Alg. 1 [88], which has an offline phase and an online phase. The inputs to the offline phase include linear approximations g_i with correlation ρ_i for $i = 1, \dots, m$ and p as the p.d. of g . During the offline, the algorithm computes N from p [88, Theorem 2]. Then it proceeds as follows in the online phase.

Online inputs: N pairs of (X, Y) , with N computed in offline stage.

Online phase:

1. *Distillation:* Compute empirical p.d. $\hat{q}$ of h using 2^m counters.
2. *Analysis:* Construct matrix $\mathbf{T}^{2^m \times 2^m}$, with cell $\mathbf{T}(w, \eta) = \log(\frac{\hat{q}_\eta}{p_\eta^w})$, for all $w, \eta \in V_m$.
Compute $D(\hat{q}||p^w)$ for all $w \in V_m$ in a list,

$$D = \mathbf{T}\hat{q}^T = (D(\hat{q}||p^0), \dots, D(\hat{q}||p^w), \dots, D(\hat{q}||p^{2^m-1})). \quad (3.6)$$

3. *Sorting:* Sort the list of w by values of $D(\hat{q}||p^w)$ in ascending order.
4. *Searching:* Starting from the top of the sorted list, search the correct key by key classes until the correct key is found.

Note: in the analysis phase the matrix $\mathbf{T}$ does not need to be stored. Storing a single row is sufficient. Other rows can be obtained as equation 3.4 by permuting this row when it's required for computation.

3.3 A Circulant Matrix Property

It is observed that when computing the KL distance between a p.d. $\hat{q}$ and p^w , one can expand $D(\hat{q}||p^w) = \sum_{\eta \in V_m} \hat{q}_\eta \log(\hat{q}_\eta/p_\eta^w) = \sum_{\eta \in V_m} \hat{q}_\eta \log \hat{q}_\eta - \sum_{\eta \in V_m} \hat{q}_\eta \log p_\eta^w$, the term $\sum_{\eta \in V_m} \hat{q}_\eta \log \hat{q}_\eta$ is a constant and hence does not affect the ranking of key classes. Then define $\bar{D}(\hat{q}||p^w) = \sum_{\eta \in V_m} \hat{q}_\eta \log p_\eta^w$ and use $\bar{D}(\cdot||\cdot)$ to rank the key class candidates. The list of w in the sorting phase of the online stage is now sorted by the values of $\bar{D}(\hat{q}||p^w)$ in descending order. To compute $\bar{D}$ as $\bar{D} = \bar{\mathbf{T}}\hat{q}^T$, the matrix $\bar{\mathbf{T}}$ can be constructed as

$$\bar{\mathbf{T}}(w, \eta) = \log(p_\eta^w), \quad \forall w, \eta \in V_m. \quad (3.7)$$

We show here that matrix $\bar{\mathbf{T}}$ is a circulant matrix.

Theorem 3.3.1. *The matrix $\bar{T}$ is level- m circulant with type $\underbrace{(2, 2, \dots, 2)}_{m\text{-times}}$.*

Proof. The structure of the modified matrix $\bar{T}^{2^m \times 2^m}$ is

$$\begin{pmatrix} \log(p_0^0) & \log(p_1^0) & \cdots & \log(p_{2^m-1}^0) \\ \log(p_0^1) & \log(p_1^1) & \cdots & \log(p_{2^m-1}^1) \\ \vdots & \vdots & \ddots & \vdots \\ \log(p_0^{2^m-1}) & \log(p_1^{2^m-1}) & \cdots & \log(p_{2^m-1}^{2^m-1}) \end{pmatrix}.$$

From the relation between p^w ($\forall w \neq 0$) and $p^0 = p$:

$$p_i^w = \sum_{a \in V_m} (-1)^{a(w \oplus i)} \rho(a) = \sum_{a \in V_m} (-1)^{aj} \rho(a) = p_j,$$

where $j = w \oplus i$.

Hence, $\log(p_i^w) = \log(p_j)$, $j = w \oplus i$. It means that $\bar{T}(w, i) = \bar{T}(0, j) = \bar{T}(0, w \oplus i)$.

Divide $\bar{T}$ into 4 blocks, each with size $(2^{m-1} \times 2^{m-1})$:

$$\begin{pmatrix} \bar{T}_{11} & \bar{T}_{12} \\ \bar{T}_{21} & \bar{T}_{22} \end{pmatrix}.$$

Then for $0 \leq i, j \leq 2^{m-1} - 1$:

- $\bar{T}_{11}(i, j) = \bar{T}(i, j) = \bar{T}(0, i \oplus j),$
- $\bar{T}_{21}(i, j) = \bar{T}(i + 2^{m-1}, j) = \bar{T}(0, (i + 2^{m-1}) \oplus j) = \bar{T}(0, i \oplus j \oplus 2^{m-1}),$
- $\bar{T}_{12}(i, j) = \bar{T}(i, j + 2^{m-1}) = \bar{T}(0, i \oplus (j + 2^{m-1})) = \bar{T}(0, (i + 2^{m-1}) \oplus j),$
- $\bar{T}_{22}(i, j) = \bar{T}(i + 2^{m-1}, j + 2^{m-1}) = \bar{T}(0, (i + 2^{m-1}) \oplus (j + 2^{m-1})) = \bar{T}(0, i \oplus j).$

Consequently, $\bar{T}_{11} = \bar{T}_{22}$ and $\bar{T}_{12} = \bar{T}_{21}$, hence $\bar{T}$ is 2-block circulant. We can inductively repeat the same argument to $\bar{T}_{11}$ with $m = m - 1$. Since $\bar{T}_{12} = \bar{T}_{11} \oplus 2^{m-1}$, the structure of $\bar{T}_{12}$ is similar to the circulant structure of $\bar{T}_{11}$. Hence, the matrix $\bar{T}$ is level- m circulant with type $\underbrace{(2, 2, \dots, 2)}_{m\text{-times}}$. $\square$

This property serves as the basis of the algorithm Method-B in [150], so that the fast Fourier transform can be applied for fast computation of the KL distance between two distributions. For $\bar{\mathbf{T}}$, only the first column ($\bar{\mathbf{T}}(w, 0) = \log(p_0^w) = \log(p_w)$, for all $w \in V_m$) needs

to be stored. The memory requirement is 2^m . In addition, in [150] the algorithm Method-A is proposed for more efficient computation of empirical distribution $\hat{q}$ of the experimental function h from the N samples. The improvements in [150] on the multidimensional Alg. 1 can be summarized as:

1. Method-A: calculate $\hat{q}$ from N samples given, with complexity $2m2^m + mN$.
2. Method-B: compute $\bar{D}$, with complexity $3m2^m$.

3.4 Attacking 4-Round SERPENT and Analysis

An attack to 4-round SERPENT is presented in [52], where 64 approximations were used to obtain 10 parity bits of 4-round SERPENT, from S_4 to S_7 . The approximations are modified from the first 4 rounds of the 6-round linear characteristic in [50], with the details described in [12]. As shown in [88], these approximations used are not linearly independent in text masks and key masks. There are 8 of them with a correlation of 2^{-11} and 56 with a correlation of 2^{-12} . This gives an overall capacity of $4 \sum_{i=0}^{63} \epsilon_i^2 = 2^{-17.54}$ hence estimation of $4m/4 \sum_{i=1}^{m'} \epsilon_i^2 = 2^{22.86}$ for N . By selecting 10 base approximations $L_0, \dots, L_9$ from the 64, with L_i being $u_i X \oplus w_0 Y \oplus c_i K = 0$, Hermelin *et al.* studied in [88] all the approximations generated as $\text{span}\{L_0, \dots, L_9\}$. Out of the 1023 combined approximations (including the 10 basis), 8, 64 and 128 are with non-negligible correlation of 2^{-11} , 2^{-12} and 2^{-13} respectively. Lemma 3.2.2 gives capacity $C(p||q)$ of at least $2 \sum_{a \neq 0} \rho^2(a) = 2 \cdot (8 \cdot 2^{-22} + 64 \cdot 2^{-24} + 128 \cdot 2^{-26}) = 2^{-16}$ and hence the estimation for N is $4m/C(p||q) = 2^{21.3}$, in perfect correspondence to the experimental results in [88]. With Method-A and Method-B of [150], we can set $m = 16$, yielding an attack with better complexity than the multidimensional Alg. 1 of [88] with $m = 10$.

The approximations $L_0, \dots, L_9$ are derived from the same linear characteristic with input masks $u_0, \dots, u_9$ and the same output mask w_0 . We obtain 6 additional ciphertext masks $w_1, \dots, w_6$ and use the following 16 approximations as base approximations: $(u_0, w_0), \dots, (u_9, w_0), (u_1, w_1), \dots, (u_1, w_6)$. Hence the key classes are now different, the key parities recovered will be a combination of first sub-key parity bits and last sub-key parity bits¹. An exhaustive check of all combined correlations shows that 32, 384, 1664, 3072 and

¹We do not discuss full key recovery here, in fact the attack derives parts of the two sub-keys. Neverthe-

2048 of the combined approximations have correlations of 2^{-11} , 2^{-12} , 2^{-13} , 2^{-14} and 2^{-15} respectively. This gives a capacity of $2^{-12.8}$, hence we estimate $N \sim 4m/C(p||q) = 2^{18.8}$. We tabulate the comparisons in Table 3.1. From the table we can conclude that it is clearly advantageous if we are able to use a larger m , *i.e.*, when using $m = 16$ instead of $m = 10$, in the case of 4-round SERPENT. A larger m in this case makes it possible to have a much larger number of non-negligible approximations, hence larger capacity, reduced data complexity and the overall time complexity. Setting $m = 16$ improves the cryptanalysis due to the fact that the number of non-negligible approximations in SERPENT is exponential in m . It may appear that the attack is slower with a larger m . However, as the analysis above, it is in fact the reduction in data complexity N that dominates the time complexity improvement. However, we find out by experiment that in DES, the number of high probability approximations is much fewer than that of SERPENT. Thus, larger values of m do not necessarily give better attacks. Essentially, this is a trade-off. In particular, with the improved attacks, a larger range of m can be accommodated and this allows the cryptanalyst to find an appropriate m for an optimal attack.

m	$C(p q)$	N	Distillation Phase		Analysis Phase		Memory
			Hermelin <i>et al.</i> [88]	This thesis	Hermelin <i>et al.</i> [88]	This thesis	
10	2^{-16}	$2^{21.3}$	$2^{31.3}$	$2^{24.6}$	2^{20}	$2^{14.9}$	2^{10}
16	$2^{-12.8}$	$2^{18.8}$	$2^{34.8}$	$2^{23.2}$	2^{32}	$2^{21.6}$	2^{16}

Table 3.1: Attack complexities on 4-round SERPENT

3.5 Attacking 9-Round SERPENT

To develop an attack to 9-round SERPENT with the multidimensional Alg. 1, we use the 9-round linear characteristic of [50], the details of the characteristic is described in [12]. This linear characteristic starts from S_3 and ends after the next S_3 , with an overall correlation of 2^{-49} . The first round has 11 active Sboxes, which results in a correlation of 2^{-11} and the remaining 8 rounds with correlation 2^{-38} . We can enumerate all possible input masks that match the characteristic in the remaining part, and there are 10^{11} such input masks in

less, the linear approximations can be chosen such that the key classes are fully defined by parities of bits from only the first sub-key, so that recovering part of the sub-key directly reduces the remaining key space for brute-force. If the key classes are determined by the last sub-key, it is easier to attack the secret key if the key-schedule is invertible.

total, with the first round correlations ranging from 2^{-11} to 2^{-22} .

Constructing the space of input masks. To exploit the correlations of the 10^{11} non-negligible approximations, we are to find 44 linearly independent input masks such that their linear combinations will cover the space of 2^{44} input masks, which correspond to all the combinations of all the possible 4-bit inputs to each of the 11 active Sboxes in the first round. This can be done easily since the base approximations are not required to have non-negligible correlations. We simply pick all the different input masks of weight 1 such that each input mask has one (and only one) of these 11 Sboxes active. Then the space of combined approximations will cover all the exploitable approximations.

The attack. We build 44 independent base approximations that use the input masks we described above, with the rest of the characteristic being the same. We hence have a 44-dimensional attack, with capacity

$$C(p||q) = 2 \cdot \left[\sum_{i=0}^{11} \binom{11}{i} \cdot 2^i \cdot 8^{11-i} \cdot ((2^{-1})^i \cdot (2^{-2})^{11-i})^2 \right] \cdot (2^{-38})^2 = 2^{-75}$$

which is 2^{22} times larger than the capacity 2^{-98} for the single approximation scenario. Correspondingly the estimation for N is $4m/C(p||q) = 2^{82.5}$. The time complexity for the distillation phase is 2^{88} and the analysis phase is 2^{51} . Around 2^{44} memory cells are needed. In [50], this 9-round linear characteristic is used to break a 10-round and an 11-round SERPENT with Matsui's Alg. 2 extended with multiple approximations. It requires at least 2^{99} known-plaintext which is much higher than our estimation. Moreover, the time complexities presented in [50] did not take into account the distillation phase, which should require no lower than mN which is greater than 2^{99} . Hence, it is much higher than our overall time complexity of 2^{88} .

Comparing with extensions of Alg. 2, it is less desirable that with Alg. 1, r -round linear characteristics are not used in the form of a last-round attack so as to cover more than r rounds. However, it is noted in [52] that optimal application of Algorithm 2 with multiple approximations requires accurate estimation of the biases, which can be less reliable than multidimensional Alg. 1 if accurate estimation of the biases is not available due to

inherent difficulties of analysis in certain cases, *e.g.*, when there exist several or more linear characteristics of similar correlations under the same input-output masks, *i.e.*, the linear hull effect. In fact, we believe that linear cryptanalysis should be used with great caution for both Alg. 1 and Alg. 2 on linear hulls. The linear characteristics used in our attacks on SERPENT are assumed to be with dominating correlations and the linear hull effect is assumed to be close to none. Since for SERPENT, the linear hull effect on low weight characteristics is in general much less phenomenal than for some other block ciphers, *e.g.*, the block cipher PRESENT [153].

3.6 Conclusions

We discuss in this section the implications of the improved multidimensional Alg. 1. From the result of N derived in [150], we confirm that the framework provided by Hermelin *et al.* in [88] is able to exploit all the combined approximations systematically, as compared to Biryukov *et al.* in [34], which offers experimental exploration and a heuristic argument on the issue.

The improved results are presented in Table 3.3 to compare with previous results on the complexities of the distillation phase and the analysis phase, following [88, Table 1]. In practice mN is always larger than $2m2^m$, and we estimate $\mathcal{O}(2m2^m + mN) = \mathcal{O}(mN)$.

Following the definitions of $N_{s.i.}$ and N_{plain} in [88], $N < N_{s.i.} < N_{plain}$, $m < m'$ (see Table 3.2 as well), we can observe significant improvement in the time complexities for the distillation phase and the analysis phase, due to Method-A and Method-B. In addition, the data complexity is also less than the attack of Biryukov *et al.* [34], thanks to the additional insights due to the closed form of Equation 3.5, while the same result was observed in [88] with only empirical arguments.

N	Framework	Comments
$1/\epsilon^2$	Matsui [130]	
$1/\sum_{i=1}^m \epsilon_i^2$	Kaliski and Robshaw [97]	
$1/\sum_{i=1}^m \epsilon_i^2$	Biryukov <i>et al.</i> [34]	
$m/2 \sum_{a \neq 0} \epsilon^2(a)$	Hermelin <i>et al.</i> [88]	See 3.5

Table 3.2: Comparisons of data complexities with different attack frameworks

As shown in the case of 4- and 9-round SERPENT, the increase in m brings significant

	Distillation Phase				Analysis Phase			
	Plain	Biryukov	Hermelin	Method-A	Plain	Biryukov	Hermelin	Method-B
Data	$\mathcal{O}(N_{plain})$	$\mathcal{O}(N_{s.i})$	$\mathcal{O}(N)$	$\mathcal{O}(N)$	-	-	-	-
Time	$\mathcal{O}(mN_{plain})$	$\mathcal{O}(m' N_{s.i})$	$\mathcal{O}(2^m N)$	$\mathcal{O}(mN)$	$\mathcal{O}(m2^m)$	$\mathcal{O}(m' 2^m)$	$\mathcal{O}(2^{2m})$	$\mathcal{O}(m2^m)$
Mem	$\mathcal{O}(m)$	$\mathcal{O}(m')$	$\mathcal{O}(2^m)$	$\mathcal{O}(2^m)$	$\mathcal{O}(2^m)$	$\mathcal{O}(2^m)$	$\mathcal{O}(2^m)$	$\mathcal{O}(2^m)$

Table 3.3: Complexity comparison between different algorithms

reduction on the data complexity and the time complexity. We expect the improved multi-dimensional Alg. 1 to outperform all the previous frameworks which make use of multiple linear approximations. We observe that the framework of Hermelin *et al.* [88], with the efficiency improvements of Method-A and Method-B, overcomes the limitations of the Biryukov *et al.* framework [34] that the statistical independence assumption can neither be verified nor guaranteed in practice. In fact, this independence requirement is not necessary, the number of combined approximations with non-negligible correlations can be exponential to m . Meanwhile, the series of experiments in [50] and [52] are excellent examples on this issue.

Chapter 4

Unaligned Rebound Attack: Differential Cryptanalysis of KECCAK

In this chapter, we analyze the differential cryptanalysis resistance of the internal permutation of KECCAK, one of the NIST SHA-3 competition finalists. More precisely, we first introduce a new and generic method that looks for good differential paths for all the KECCAK internal permutations, and we obtain the currently best known differential paths for up to 5 rounds. We use these differential paths in a rebound attack [137] setting and adapt this powerful freedom degrees utilization in order to derive distinguishers for up to 8 rounds of the internal permutations of the submitted version of KECCAK. The complexity of the 8-round distinguisher is $2^{491.47}$. Our results have been implemented and verified experimentally on a small version of KECCAK. This is currently the best known differential attack against the internal permutations of KECCAK. This application of rebound attack is far from being trivial and requires a careful analysis of many technical details in order to model the behavior of the attack. This technique is in particular much more complicated to apply to KECCAK than to AES or to other 4-bit Sbox hash functions [158, 102]. One reason for that is that KECCAK has *weak alignment* [18], *i.e.*, the propagation of truncated differences is hard to predict. This is why we call our attack “unaligned rebound attack”. The model introduced has been verified experimentally on a small version of KECCAK and we even-

tually obtained differential distinguishers for up to 8 rounds of the submitted version of KECCAK to the **SHA-3** competition. In order to demonstrate why differential analysis is in general more relevant than zero-sum ones in regards to the full hash function, we applied our techniques to the recent KECCAK challenges [17] and managed to obtain the currently best known collision attack for up to two rounds ¹.

4.1 Background on Cryptanalysis of KECCAK

Cryptographic hash functions are used in many applications such as digital signatures, authentication schemes or message integrity and they are among the most important tools in cryptography. Recently, most of the standardized hash functions [159, 144] have suffered from serious collision attacks [180, 179]. As a response the NIST launched in 2007 the **SHA-3** competition [145] that will lead to the future hash function standard. 5 candidates made it to the final round, and KECCAK [19] is among them. Compared to its opponents, this hash function employs the construction of a sponge function [14]. The submitted versions of KECCAK to the **SHA-3** competition use as main component an internal permutation P of 1600 bits. In the original submission [15] the internal permutation used 18 rounds and the tweaked versions [16] went up to 24 rounds.

Like any construction that builds a hash function from a subcomponent, the cryptographic quality of this internal permutation is very important for a sponge construction. Therefore, this permutation P should not present any structural flaw, or should not be distinguishable from a randomly chosen permutation. Previous cryptanalysis have not endangered the KECCAK security so far. Zero-sum distinguishers [9] can reach an important number of rounds, but generally with a very high complexity. For example, the latest results [43] provide zero-sum partitions distinguishers for the full 24-round 1600-bit internal permutation with a complexity of 2^{1575} . When looking at smaller number of rounds the complexity would decrease, but it is unclear how one can describe the partition of a 1600-bit internal state without using the KECCAK round inside the definition of the partition. Moreover, such zero-sum properties seem very hard to exploit when the attacker aims at the whole hash function. On the other side, more classical preimage attack on 3 rounds

¹We note that newer results on KECCAK are obtained in [148] and [68].

using SAT-solvers have been demonstrated [141]. Finally, Bernstein published [13] a $2^{511.5}$ computations (second)-preimage attack on 8 rounds that allows a workload reduction of only half a bit over the generic complexity with an important memory cost of 2^{508} .

4.2 Finding Differential Characteristics

Before describing how we use the freedom degrees in a rebound attack setting, we first study how to find “good” differential paths¹ for all KECCAK variants. In this section, we describe our differential finding algorithms. We start by recalling several special properties of the mappings θ and χ in the round function, followed by our algorithm which provides most of the best known differential paths for the KECCAK internal permutations. In particular, we provide the currently best known 3-, 4- and 5-round differential paths for KECCAK- f [1600], the internal permutation from the submitted version of KECCAK.

4.2.1 Modeling Special properties of θ and χ

The θ mapping updates each state bit $a[x][y][z]$ by adding the bitwise sum of the two columns $a[x-1][\cdot][z]$ and $a[x+1][\cdot][z-1]$. When every column sums to 0, θ acts as identity. This is noted by the KECCAK designers [19, Section 2.4.3], and the set of the states with all columns sums to 0 is called *column parity kernel*, or *CP-kernel* for short. Since θ is linear, this property applies not only to the state values, but also to differentials. While θ expands a single bit difference into at most 11 bits (2 columns and the bit itself), it has no influence, *i.e.*, acts as identity, on differences in the CP-kernel. This property will be intensively used in finding low Hamming weight bitwise differentials. Another interesting property is that θ^{-1} diffuses much faster than θ , *i.e.*, a single bit difference can be propagated to about half state bits through θ^{-1} [19, Section 2.3.2]. However, the output of θ^{-1} is extremely regular when the Hamming weight of the input is low.

The χ layer updates each bit $a[x, y, z]$ by adding $((\neg a[x+1, y, z]) \wedge a[x+2, y, z])$. It is a row-wise operation and thus can also be viewed as a 5-bit Sbox. Similar to the analysis of other Sboxes, we build the differential distribution table (DDT), as shown in Table A.4 in

¹Note that the terms “differential path”, “differential characteristic” and “differential trail” are essentially the same. Also as a side note, in the KECCAK permutations, the clustering of differential characteristics is not at all phenomenal.

Appendix A.2. We remark that when a single difference is present, χ acts as identity with the best probability 2^{-2} , while input differences with more active bits tends to lead to more possible output differences, but with lower probability. It is also interesting to note that given an input difference to χ , all possible output differences occur with same probability, however this is not the case for χ^{-1} (the DDT for χ^{-1} can be derived from Table A.4).

4.2.2 First Tools: Simplify the Nonlinear Layer

Our goal is to derive “good” bitwise differential paths by maintaining the bit difference Hamming weight as low as possible. The ι permutation adds predefined constants to the first lane, and hence has no essential influences when such differentials are considered. For the rest of this chapter, we will ignore this layer. We note that θ , ρ and π are all linear mappings, while χ acts as a nonlinear Sbox. Furthermore, ρ and π do not change the number of active bits in a differential path, but only bit positions. Hence, θ and χ are critical when looking for a “good” differential path. Since χ is followed by θ in the next round (ignoring ι), we consider these two mappings together by treating a slice of the state as a unit, and try to find the potential best mapping of the slice through χ with the following two rules.

1. Given an input difference of the slice, *i.e.*, 5 row differences, find all possible output differences by looking into the DDT table. Then among all combinations of solutions of the 5 rows, choose the output combinations with minimum number of columns with odd parity.
2. In case of a draw, we select the state with the minimum number of active bits.

Rule 1 aims at reducing the amount of active bits after applying θ by choosing each slice of the output of the χ closest to the CP-kernel (*i.e.*, with even parity for most columns), and rule 2 further reduces the amount of active bits within the columns. Although this strategy may not lead to the minimum number of active bits after θ in the entire state (the full KECCAK- $f[1600]$ state is too large to precompute the best mappings for the whole state), it finds the best slice-wise mappings with help of a table of size 2^{25} (tricks like removing the ordering of the rows reduce the table size to about 2^{18}). We call this table χ -*slice-table*.

4.2.3 Algorithm for Differential Path Search

Denote $\lambda = \pi \circ \rho \circ \theta$ (all linear mappings), and the state at round i before (resp. after) applying the linear layer λ as a_i (resp. b_i). We start our search from a_1 , *i.e.*, the input state to the second round, and compute backwards for one round, and few rounds forwards, as shown below.

$$a_0 \xleftarrow{\lambda^{-1}} b_0 \xleftarrow{\chi^{-1}} \mathbf{a}_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi} a_2 \xrightarrow{\lambda} b_2 \xrightarrow{\chi} a_3 \xrightarrow{\lambda} b_3 \dots$$

The forward part is longer than the backward part because the diffusion of θ^{-1} is much better than for θ , thus, it will be easier for us to control the bit differences Hamming weight for several forward rounds (instead of backward rounds).

We choose a_1 from the CP-Kernel. However, it is impossible to enumerate all $\left(\binom{5}{0} + \binom{5}{2} + \binom{5}{4}\right)^{320} = 2^{1280}$ combinations. We further restrict to a subset of the CP-Kernel with at most 8 active bits and each column having exactly 0 or 2 active bits. Note also that any bitwise differential path is invariant through position rotation along the z axis, so we have to run through a set of size about 2^{36} . A brute-force search on this set using our two rules stated previously finds 3-round differential paths with probability 2^{-32} , 4-round differential paths with probability 2^{-142} and 5-round paths with probability 2^{-709} for KECCAK- f [1600], with examples shown in Table A.22, A.23 and A.24 in Appendix A.9, respectively. We provide also in Table 4.1 all the best differential path probabilities found for all KECCAK internal permutation sizes.

b	best differential path [†] probability (successive differential complexity of the rounds)				
	1-round	2-round	3-round	4-round	5-round
100	2^{-2} (2)	2^{-8} (4-4)	2^{-19} (4-8-7)	2^{-30} (4-8-10-8)	2^{-54} (4-8-10-8-24)
200	2^{-2} (2)	2^{-8} (4-4)	2^{-20} (4-8-8)	2^{-46} (11-9-8-8)	2^{-108} (8-16-20-16-48)
400	2^{-2} (2)	2^{-8} (4-4)	2^{-24} (8-8-8)	2^{-84} [‡] (16-14-12-42)	2^{-216} (16-32-40-32-96)
800	2^{-2} (2)	2^{-8} (4-4)	2^{-32} (4-4-24)	2^{-109} (12-12-12-73)	2^{-432} (32-64-80-64-198)
1600	2^{-2} (2)	2^{-8} (4-4)	2^{-32} (4-4-24)	2^{-142} (12-12-12-106)	2^{-709} (16-16-16-114-547)

[†] The detailed differential paths for KECCAK- f [1600] are shown in Appendix A.9.

[‡] Paths in bold are new results we found with the method presented in this thesis.

Table 4.1: Best differential paths for versions of KECCAK internal permutations

Comment. In the reference document [19], among other methods, the KECCAK designers looked for special differential paths to motivate their choice of the step mappings. In their model, the input states to the χ of the first two rounds are forced to fall entirely inside the CP-kernel. The best 3-round path they could find for KECCAK- f [1600] has probability

2^{-35} . Note also that no 4-round and 5-round path is obtained. Their results were up to now the best published differential paths for KECCAK. As shown in Table 4.1, our model outperforms in most cases the KECCAK designers model by removing this strong restriction on χ and using a more appropriate slice-wise mapping. As a result, we found the best known 3-, 4- and 5-round differential paths for the bigger versions of KECCAK.

4.3 The Rebound Attack on KECCAK

4.3.1 Building Differential Distinguishers

We introduce here a simple and generic method of obtaining differential distinguishers by extending the best differential paths as summarized in Table 4.1. In the simplest case, we are able to find valid pairs of internal state values for some differential paths on a few rounds with a rather low complexity. Said in other words, we are able to compute internal state value pairs with a predetermined input/output difference. A direct application from this is to derive distinguishers. For a randomly chosen permutation of b bits, finding a pair of inputs with a predetermined difference that maps to a predetermined output difference costs 2^b computations. Indeed, since the input and the output differences are fixed, the attacker can not apply any birthday-paradox technique.

In addition, the values and difference at the end of the path can be controlled to cover more round(s) extended forward or backward if the remaining freedom degrees are enough. Hence it may allow one to find valid pairs of internal state values whose difference falls in predetermined sets, such that the complexity is still lower than the complexity of a generic algorithm, knowing the descriptions of the sets. An example would be that we start from the 4-round differential path for KECCAK- f [1600], with differential probability 2^{-142} . One can obtain a 5-round path by adding one round to the left but the number of conditions for this new round will be huge. Therefore, we simply identify all the active Sboxes for this fifth round and we fix the value for them so that the expected differential transitions are verified. Overall, the attacker finds a valid pair of internal state values for the 5-round differential path only with complexity 2^{142} (we are ensured to have at least 2^{320} freedom degrees to handle the 2^{-142} probability). Those distinguishers are called “limited-birthday distinguishers” and can be generalized in the following way (we refer to [79] for more details):

for a randomly chosen b -bit permutation, the problem of mapping an input difference from a subset of size I to an output difference from a subset of size J requires $\max\{\sqrt{2^b/J}, 2^b/(I \cdot J)\}$ calls to permutation (while assuming without loss of generality since we are dealing with a permutation that $I \leq J$).

Extending the Differential Path Since for many of our distinguishers, the gap between our attack and the generic case complexity is very big, we can try to reach a few more rounds without increasing the complexity. Indeed, by analyzing how the differences will propagate forward from the output and backward from the input of our differential path, we will be able to determine the size of the possible input differences set and the possible output differences set.

For the forward case (i.e. when adding a round to the right), we start from the fully determined difference on the output of the differential path. We first apply the linear layers θ , ρ and π on this output difference and we obtain the difference mask at the input of χ . Now, for each active Sbox, knowing exactly its input difference, we can check with the DDT from χ (Table A.4 in the Appendix A.2) that only a subset of the 2^5 possible output differences can be reached. Therefore, the size Γ^{out} of the set of reachable output differences after applying this extra round is bounded and this bound can be computed exactly using the DDT from χ .

For the backward case (i.e. when adding a round to the left), we start from the fully determined difference on the input of the differential path. Then, reading at the DDT from χ^{-1} , one can check that one active Sbox can produce at most a certain small subset of the 2^5 possible input differences. Therefore, the size Γ^{in} of the set of reachable input differences after inverting this χ layer is bounded and this bound can be computed exactly using the DDT from χ^{-1} . Note that continuing to invert the extra round by computing θ^{-1} , ρ^{-1} and π^{-1} will not affect the size of this set.

To conclude, using a r -round path from Table 4.1 with differential probability p , we extend it by one more round in order to find valid internal state pairs for this new $(r+1)$ -round differential path with p^{-1} computations. Then, using the limited-birthday distinguishers, one can derive a $(r+3)$ -round distinguisher for the KECCAK internal permutation with

complexity p^{-1} , if

$$p^{-1} < \max\{\sqrt{2^b/J}, 2^b/(I \cdot J)\}$$

where $I = \Gamma^{\text{out}}$ and $J = \Gamma^{\text{in}}$ if $\Gamma^{\text{out}} \leq \Gamma^{\text{in}}$; $I = \Gamma^{\text{in}}$ and $J = \Gamma^{\text{out}}$ otherwise.

Continuing our example with the 4-round differential path for KECCAK- $f[1600]$ from Table 4.1, we have $\Gamma^{\text{in}} = 2^{1064}$ and $\Gamma^{\text{out}} = 2^{576}$ for a 7-round distinguisher of the internal permutation. Thus the generic complexity to find such constraint pairs is 2^{268} computations which is much higher than 2^{142} . All the distinguishers we obtain with this method are summarized in Table 4.2.

Note that the reader might be concerned by the fact that the sizes Γ^{in} and Γ^{out} of the reachable differences sets can be very big and might be not easy to describe in a compact way in our distinguisher. However, we emphasize that all the reachable differences on the output (resp. input) are actually built from the independent combinations of all the possible output differences (resp. input differences) of all active Sboxes in the last round (resp. first round). Therefore, the description of this set is easily done by identifying the reachable output differences (resp. input differences) for all the Sboxes independently.

4.3.2 The Rebound Attack Setting

The rebound attack is a freedom degrees utilization technique that was first proposed by Mendel *et al.* in [137] as an analysis of round-reduced Grøstl and Whirlpool. It was then improved in [136, 121, 79, 167] to analyze AES and AES-like permutations and also ARX ciphers [103].

With the help of rebound techniques, we show in this section how to extend the number of attacked rounds significantly, but for a higher complexity. We will see that the application of the rebound attack for KECCAK seems quite difficult. Indeed, the situation for KECCAK is not as pleasant as the AES-like permutations case where the utilization of truncated differential paths (i.e. path for which one only checks if one cell is active or inactive, without caring about the actual difference value) makes the application of rebound attacks very easy to handle.

4.3.2.1 The Original Rebound Attack

Let P denote a permutation, which can be divided into 3 sub-permutations, *i.e.*, $P = E_F \circ E_I \circ E_B$. The rebound attack works in two phases.

- **Inbound phase or controlled rounds:** this phase usually starts with several chosen input/output differences of E_I that are propagated through linear layers forward and backward. Then, one can carry out meet-in-the-middle (MITM) match for differences through a single Sbox layer in E_I and generate all possible value pairs validating the matches.
- **Output phase or uncontrolled rounds:** with all solutions provided in the inbound phase, check if any pair validates as well the differential paths for both the backward part p_B and the forward part p_F .

In most cases, the inbound phase can be done fast due to the MITM nature and generates solution pairs with very low average complexity. Hence, attackers usually choose the position of E_I in the differential path so that it covers a low probability portion of the trail in order to increase the success probability of the outbound phase. For example when dealing with the 128-bit AES internal permutation, this MITM is performed on sixteen parallel 8-bit Sboxes. If the match is done with k Sboxes being active and since a random 8-bit input/output difference can be matched with probability $1/2$ through a single AES Sbox, one needs to try at least 2^k input/output difference pairs of E_I in order to hope having one matching candidate for the inbound phase. However, once a match is found one can generate about 2^k solution values from it, thus leading to an average cost of about one operation per solution for the controlled rounds. The final goal of the attacker is then to generate enough inbound phase solution values such that one of them also verifies the forward and backward outbound trails, *i.e.* at least $p_B^{-1} \cdot p_F^{-1}$ pairs need to be tested if p_B and p_F are the respective backward and forward differential probability (DP) of the outbound trail.

The SuperSbox technique [121, 79] extends the E_I from one Sbox layer to two Sbox layers for an AES-like permutation, by considering two consecutive AES-like rounds as one with column-wise SuperSboxes. This technique is possible due to the fact that one can swap few linear operations with the Sbox in AES, so that the two layers of Sboxes in two rounds

Figure 4.1: Rebound attack on KECCAK

become close enough to form one SuperSbox layer. However, in the case of KECCAK, it seems very hard to form any partition into independent SuperSboxes. For the same reason, using truncated differential paths seems very difficult for KECCAK, as it has recently been shown in [18].

During the application of rebound attacks, one has to start with several input/output differences of E_I to complete the inbound phase. For AES-like permutations one can start with truncated differences and thus it is much more handy because this view simplifies a lot the analysis. Indeed, a truncated differential for AES-like permutations can be seen as a collection of several bit differentials, all with the same success probability and the same properties in regards to rebound attacks. Thus, whatever the difference masks considered for the input/output of the controlled rounds, the probabilities p_B and p_F will remain the same, so will the probability to get a match in E_I or the number of solutions that can be generated from a match. This will not be the case for KECCAK as we can not use truncated differential paths and the analysis will be much more involved.

4.3.2.2 The Case of KECCAK Internal Permutations

Assume that we know a set of n_B differential trails (called *backward trails*) on nr_B KECCAK rounds and whose DP is higher or equal to p_B . For the moment, we want all these backward paths to share the same input difference mask Δ_B^{in} and we denote by $\Delta_B^{\text{out}}[i]$ the output difference mask of the i -th trail of the set. Similarly, we consider that we also know a set of n_F differential trails (called *forward trails*) on nr_F KECCAK rounds and whose DP is higher or equal to p_F . We want all those forward paths to share the same output difference mask Δ_F^{out} and we denote by $\Delta_F^{\text{in}}[i]$ the input difference mask of the i -th trail of the set.

Our goal here is to build a differential path on $nr_B + nr_F + 1$ KECCAK rounds (thus

Figure 4.2: Number of differences for the rebound attack on KECCAK.

one Sbox layer of inbound), by connecting a forward and a backward trail with the rebound technique, and eventually to find the corresponding solution values for the controlled round. We represent by p_{match} the probability that a match is possible from a given element of the backward set and a given element of the forward set, and we denote by N_{match} the number of solution values that can be generated once a match has been obtained.

For this connection to be possible, we need the inbound phase to be a valid differential path, that is we need to find a valid differential path from a $\Delta_B^{\text{out}*}$ to a Δ_F^{in} . By using random $\Delta_B^{\text{out}*}$ and Δ_F^{in} this will happen in general with very small probability, because we need the very same set of Sboxes to be active/inactive in both forward and backward difference masks to have a chance to get a match. Even if the set of active Sboxes matches, we still require the differential transitions through all the active Sboxes to be possible.

We can generalize a bit this approach by allowing a fixed set of differences Δ_B^{in} (resp. Δ_F^{out}) instead of just one. We call Γ_B^{in} (resp. Γ_B^{out}) the *size* of the set of possible Δ_B^{in} (resp. Δ_B^{out}) values for the backward paths. Similarly, we call Γ_F^{in} (resp. Γ_F^{out}) the size of the set of possible Δ_F^{in} (resp. Δ_F^{out}) values for the forward paths. In fact, the number of possible differences in the backward or forward parts will form a butterfly shape (see Figure 4.2). We call Γ_B^{mid} (resp. Γ_F^{mid}) the minimum number of differences in the backward (resp. forward) part.

The total complexity C to find one valid internal state pair for the $(nr_B + nr_F + 1)$ -round path is

$$C = n_F + n_B + \frac{1}{p_{\text{match}}} \cdot \left\lceil \frac{1}{p_F \cdot p_B \cdot N_{\text{match}}} \right\rceil + \frac{1}{p_B \cdot p_F}, \quad (4.1)$$

with

$$\Gamma_B^{\text{out}} \cdot \Gamma_F^{\text{in}} = \frac{1}{p_{\text{match}}} \cdot \left\lceil \frac{1}{p_F \cdot p_B \cdot N_{\text{match}}} \right\rceil. \quad (4.2)$$

The first two terms are the costs to generate the backward and forward paths. The term $\left\lceil \frac{1}{p_F \cdot p_B \cdot N_{\text{match}}} \right\rceil$ denotes the number of time we will need to perform the inbound and each

inbound costs $1/p_{\text{match}}$. The last term is the cost for actually performing the outbound phase. The condition (4.2) is needed since we need enough differences to perform the inbound phase.

Roadmap. To better understand the pattern of the activate Sboxes in the rebound attack, we will need some useful counting lemmas in Section 4.3.3. We explain how to prepare the forward and backward differential paths in Section 4.4 and describe the inbound and outbound phases in Section 4.5 and 4.6 respectively. We explain how to relate Sections 4.4, 4.5 and 4.6 in Section 4.7.1, we show also how we can reduce the complexity of the attack and we give a numerical application of our model. Finally in Section 4.7.2 we show the 8-round rebound distinguisher, and its further improvements in Section 4.8.

4.3.3 A Buckets-and-Balls Problem

We model the active/inactive Sboxes match as a *limited capacity ordered buckets and balls problem*: the $s = 5w$ ordered buckets ($s = 320$ for KECCAK- $f[1600]$) limited to capacity 5 will represent the s 5-bit Sboxes and the x_B (resp. x_F) balls will stand for the Hamming weight of the difference in $\Delta_B^{\text{out}*}$ (resp. in Δ_F^{in}). Given a set B of s buckets in which we randomly throw x_B balls and a set F of s buckets in which we randomly throw x_F balls, we call the result a *pattern-match* when the set of empty buckets in B and F after the experiment are the same.¹ Before computing the probability of having a pattern-match, we need the following Lemma 4.3.1.

Lemma 4.3.1. [70] *The number of possible combinations $b_{\text{bucket}}(n, s)$ to place n balls into s buckets of capacity 5 such that no bucket is empty is*

$$b_{\text{bucket}}(n, s) := \begin{cases} \sum_{i=\lceil n/5 \rceil}^s (-1)^{s-i} \binom{s}{i} \binom{5i}{n} & \text{if } s \leq n \leq 5s \\ 0 & \text{else.} \end{cases} \quad (4.3)$$

We can solve the counting problem by recursion ², i.e., the number of combinations

¹Note that the *position of the balls in the buckets* is significant. This is why we refer to an ordered buckets and balls problem.

²The closed form in Lemma 4.3.1 is attributed to Alexandre Duc, we refer to the proof with generating

verifies the following recurrence relation:

$$b_{\text{bucket}}(n, s) = \sum_{i=1}^5 \binom{5}{i} b_{\text{bucket}}(n-i, s-1),$$

with $b_{\text{bucket}}(x, s) = 0$ when $x \leq 0$ and $b_{\text{bucket}}(x, 1) = \binom{5}{x}$ when $0 < x \leq 5$ and 0 else.

Using (4.3), we can derive the probability p_{bucket} that every bucket contains at least one ball when n balls are thrown into s buckets with capacity 5:

$$p_{\text{bucket}}(n, s) := \frac{b_{\text{bucket}}(n, s)}{\binom{5s}{n}}. \quad (4.4)$$

The expected number of active buckets when n balls are thrown into s buckets is given by

$$\frac{\sum_{i=0}^s b_{\text{bucket}}(n, s-i) \cdot \binom{s}{i} \cdot (s-i)}{\binom{5s}{n}}. \quad (4.5)$$

We can now relate this lemma to the more general pattern-match problem.

Lemma 4.3.2. *Given a set of s buckets B of capacity 5 in which we throw x_B balls and a set of s buckets F in which we throw x_F balls, the probability that B and F have the same pattern of empty buckets is given by*

$$p_{\text{pattern}}(s, x_B, x_F) = \frac{1}{\binom{5s}{x_B} \binom{5s}{x_F}} \sum_{i=0}^s b_{\text{bucket}}(x_B, s-i) b_{\text{bucket}}(x_F, s-i) \binom{s}{i},$$

where $b_{\text{bucket}}(x, s)$ is defined as in (4.3). The average number n_{pattern} of non-empty buckets if both experiments results follow the same pattern is given by

$$n_{\text{pattern}}(s, x_B, x_F) = \frac{\sum_{i=0}^s b_{\text{bucket}}(x_B, s-i) b_{\text{bucket}}(x_F, s-i) \binom{s}{i} (s-i)}{\sum_{i=0}^s b_{\text{bucket}}(x_B, s-i) b_{\text{bucket}}(x_F, s-i) \binom{s}{i}}.$$

This model tells us that when the number of balls (*i.e.*, active bits) is not too small on both sides, most of the matches happen when (almost) all the Sboxes are active. Figure A.1 in Appendix A.7 depicts this behavior.

functions in [70].

A More General Problem. We can also look into a more general problem, i.e., we characterize more precisely how the bits are distributed into the Sboxes.

Lemma 4.3.3. *The probability p_{dist} of distributing randomly n active bits into s 5-bit Sboxes such that exactly A_i Sboxes contain i bits, for $i \in [1, 5]$ is*

$$p_{\text{dist}}(A_1, A_2, A_3, A_4, A_5) := \frac{s! \binom{5}{1}^{A_1} \binom{5}{2}^{A_2} \binom{5}{3}^{A_3} \binom{5}{4}^{A_4} \binom{5}{5}^{A_5}}{(s - A_1 - A_2 - A_3 - A_4 - A_5)! A_1! A_2! A_3! A_4! A_5! \binom{5s}{n}}, \quad (4.6)$$

with $n = A_1 + 2A_2 + 3A_3 + 4A_4 + 5A_5$.

Important Remark. Since most matches happen when all the Sboxes are active, in order to simplify the analysis, we will consider from now on that we will only use forward and backward paths such that *all Sboxes are active in the χ layer of the inbound phase*.

4.4 Constructing the Differential Paths Sets

In this section, we explain how to construct the forward and backward ¹ paths because this will have an impact on the derivation of p_{match} and N_{match} (this will be handled in the next two sections).

4.4.1 The Forward Paths Set

Figure 4.3: The forward trails. Values are taken from the 3 round differential path from Table A.23. The distance between the two lines reflects the number of differences.

For the forward paths set, we start by choosing a differential trail computed from the previous section and we derive a set from it by exhausting all the possible Sbox differential

¹Note that the initial idea of constructing the set of backward paths is the credit of [70].

transitions for the inverse of the χ layer in its first round (all the paths will be the same except the differences on their input and on the input of the χ layer in the first round). For example, we can use the 3 first rounds of the 4-round differential path from Table A.23 in Appendix A.9 which have a total success probability 2^{-36} and present 6 active Sboxes during the χ layer of the first round. Note that we didn't choose the best 3-round differential path (with success probability 2^{-32}) since it will not provide enough paths (due to its input difference Hamming weight being too small). We randomize the χ^{-1} layer differential transitions for the 6 active Sboxes of the first round, and we obtain about 2^{19} distinct trails in total. We analyzed that all the trails of this set have a success probability of at least 2^{-48} (this is easily obtained with the χ^{-1} DDT). Moreover, note that they will all have the same output difference mask (at the third round), but distinct input masks (at the first round). Since we previously forced the requirement that all Sboxes must be active for the inbound match, we check experimentally that $2^{17.3}$ of the 2^{19} members of the set fulfill this condition.¹ We call τ_F the ratio of paths that verify this condition over the total number of paths, i.e., $\tau_F = 2^{-1.7}$. Overall, we built a set of $2^{17.3}$ forward differential paths on $nr_F = 3$ KECCAK- $f[1600]$ rounds, all with DP higher or equal to $p_F = 2^{-48}$. We can actually generate 64 times more paths by observing that they are equivalent by translation along the KECCAK lane (the z axis). However, these paths will have distinct output difference masks (the same difference mask rotated along the z axis), and we have $\Gamma_F^{\text{out}} = \Gamma_F^{\text{mid}} = 2^6$. The *total amount of input differences* Γ_F^{in} is, hence, $\Gamma_F^{\text{in}} := \Gamma_F^{\text{mid}} \cdot 2^{17.3} = 2^{23.3}$ and we have to generate in total $n_F = \tau_F \cdot \Gamma_F^{\text{in}} = 2^{25}$ forward differential paths.

4.4.2 The Backward Paths Set

Applying the same technique to the backward case does not generate a sufficient amount of output differences Γ_B^{out} , crucial for a rebound-like attack. Thus, concerning the backward paths set, we build another type of 3-round trails. We need first to ensure that we have *enough differential paths* to be able to find a match in the inbound phase, i.e., we want $\Gamma_B^{\text{out}} \cdot \Gamma_F^{\text{in}} = 1/p_{\text{match}} \cdot \lceil \frac{1}{p_F \cdot p_B \cdot N_{\text{match}}} \rceil$ following (4.2). Moreover, we will require these path to verify two conditions:

¹The small amount of filtered forward paths (a factor $2^{1.7}$) is due to the regularity of the output of θ inverse. Thus, most of the paths have all Sboxes active when the Hamming weight of the input is low.

Figure 4.4: The backward trails. The distance between the two arrows reflects the number of differences.

1. First, we need to filter paths that have not all Sboxes active in the χ layer of the inbound phase. Using (4.4), this happens with a probability about $\tau_B^{\text{full}} := p_{\text{bucket}}(800, 320) = 2^{-15.9}$ if we assume that about half of the bits are active. This assumption will be verified in our case (and was verified in practice) since our control on the diffusion of the active bits will be reduced greatly.
2. Moreover, *all* the paths we collect should have a *DP* of at least p_B such that the number of solutions N_{match} generated in the inbound phase is sufficient. Indeed, we must have $N_{\text{match}} \geq 1/(p_F \cdot p_B)$ in order to have a good success probability to find one solution for the entire path. We call τ_B^{DP} the probability that a path verifies this property. Hence, we need $p_B \geq p_B^{\text{min}} = 1/(p_F \cdot N_{\text{match}})$. We will show in Section 4.7.1 that $N_{\text{match}} = 2^{509}$ and we previously showed that $p_F = 2^{-48}$. Hence, $p_B^{\text{min}} = 2^{48-509} = 2^{-461}$.

These two filters induce a ratio $\tau_B := \tau_B^{\text{full}} \cdot \tau_B^{\text{DP}}$ of “good” paths. We have $n_B \cdot \tau_B = \Gamma_B^{\text{out}}$, where n_B is the number of paths we need to generate. Thus, we need to generate $n_B^{\text{min}} := 1/(p_{\text{match}} \cdot \lceil \frac{1}{p_F \cdot p_B \cdot N_{\text{match}}} \rceil \cdot \Gamma_F^{\text{in}} \cdot \tau_B)$ trails to perform the rebound. We will show in Section 4.7.1 that $p_{\text{match}} = 2^{-498.11}$, that $\lceil \frac{1}{p_F \cdot p_B \cdot N_{\text{match}}} \rceil = 1$ and that $\tau_B = 2^{-17.37}$. We also know that $\Gamma_F^{\text{in}} = 2^{23.3}$. Hence, $n_B^{\text{min}} = 2^{498.11+17.37-23.3} = 2^{492.18}$.

We show now how we generated these paths. Fig. 4.4 can help the reading. We start at the beginning of the second round by forcing X columns of the internal state to be active and each active column will contain only 2 active bits (thus a total of $2X$ active bits). Therefore, we can generate $\binom{5}{2}^X \cdot \binom{s}{X}$ distinct starting differences and each of them will lead to a distinct input difference of the backward path. This implies that $\Gamma_B^{\text{in}} = \binom{5}{2}^X \cdot \binom{s}{X}$. Note

also that all active columns are in the CP-Kernel and thus applying the θ function on this internal state will leave all bit-differences at the same place. Then, applying the ρ and π layers will move the $2X$ active bits to random locations before the Sbox layer of the second round. If X is not too large, we can assume that for a good fraction of the paths, all active bits are mapped to distinct Sboxes and thus we obtain $2X$ active Sboxes, each with one active bit on its input. We call ϵ this fraction of paths which is given by

$$\epsilon := p_{\text{dist}}(2X, 0, 0, 0, 0) , \quad (4.7)$$

where p_{dist} is given by Lemma 4.3.3.¹ We will need to take ϵ into account when we count the total number of paths we can generate. This position in the paths, i.e., after the linear layer of the second round, is the part with the lowest amount of distinct differences. Hence, we call the number of differences at this point $\Gamma_B^{\text{mid}}(X) = \Gamma_B^{\text{in}}(X) \times \epsilon$.

Looking at the DDT from χ (Table A.4 in Appendix A.2), one can check that with one active input bit in an Sbox, there always exists:

- two distinct transitions with probability 2^{-2} for the KECCAK Sbox such that we observe 2 active bits on its output (we call it a $1 \mapsto 2$ transition)
- one single transition with probability 2^{-2} and one single active bit on its output (a $1 \mapsto 1$ transition). This transition is in fact the identity.

We need to define how many $1 \mapsto 1$ and $1 \mapsto 2$ transitions we have to use, since there is a trade-off between the number of paths obtained and the DP of these paths. Whatever choices we make, we always have that the success probability of this χ transition (in the second round) is 2^{-4X} . Let k be the number of $1 \mapsto 2$ transitions among the $2X$ possible ones. We will observe $2X + k$ active bits after χ . Before the χ transition, we have $\Gamma_B^{\text{mid}}(X)$ different paths from the initial choice. For each of these paths, we can now select $\binom{2X}{k}$ distinct sets of $1 \mapsto 2$ transitions each of which can generate 2^k different paths. These $2X + k$ bits are expanded through θ to *at most* $11 \cdot (2X + k) = 22X + 11k$ bits. However, this expansion factor (every active bit produces 11 one) is smaller when the number of bits increases. Let n be the number of obtained active bits at the input of the Sboxes in the third round. At

¹Simulations verified this behavior in practice for the parameters we use in our attack.

the beginning of the third round, we have $2X + k$ active bits. For $\text{KECCAK-}f[1600]$, given $2X + k$ active bits at the input of θ , we get

$$n \approx u - \frac{u \cdot (u - 1)}{1600} \quad (4.8)$$

bits at the output, with $u := 11(2X + k)$ for X small enough. Indeed, the $2X + k$ bits are first multiplied by 11 due to the property of θ . We suppose now that these u active bits are thrown randomly and we check for collisions. Given u bits, we can form $u \cdot (u - 1)/2$ different pairs of bits. The probability that a pair collides is 2^{-1600} , hence, we have about $u \cdot (u - 1)/(2 \cdot 1600)$ collisions of two bits. In a 2-collision, two active bits are wasted (they become inactive). Hence, we can remove $u \cdot (u - 1)/1600$ from the overall number of active bits. For small X , we can neglect collisions between three, four and five active bits, since the bits before θ are most likely separated and will not collide. Hence we verify (4.8). This model has been verified in simulations for the values we are using.

We need now to evaluate the number of active Sboxes in the χ layer of the third round. However, in order to precisely evaluate the DP of this layer (that we want to be higher than $p_B^{\min}$) and the expansion factor we get on the amount of distinct differential paths, we also need to look at how the bits are distributed into the input of the Sboxes. The probability p_{dist} of distributing randomly n active bits into s 5-bit Sboxes such that exactly A_i Sboxes contain i bits, for $i \in [1, 5]$ is given by Lemma 4.3.3.

Lemma 4.4.1. *Suppose that we have n active bits before χ in the third round. Then, if $n \leq s$, the expected number of useful (i.e., which have $\text{DP} \geq p_B^{\min}$) paths $G_B(n)$ we can generate verifies*

$$G_B(n) \geq \sum_{A_5=0}^{\lfloor n/5 \rfloor} \sum_{A_4=0}^{\lfloor (n-5A_5)/4 \rfloor} \sum_{A_3=0}^{\lfloor (n-5A_5-4A_4)/3 \rfloor} \sum_{A_2=0}^{\lfloor (n-5A_5-4A_4-3A_3)/2 \rfloor} F(X, A_1, A_2, A_3, A_4, A_5) \cdot 2^{\sum_{i=1}^5 a_i A_i}, \quad (4.9)$$

where $A_1 := n - 5A_5 - 4A_4 - 3A_3 - 2A_2$, $\bar{a} = (a_1, a_2, a_3, a_4, a_5) = (2, 3, 3.58, 4, 4)$ and

$$F(X, A_1, A_2, A_3, A_4, A_5) := \begin{cases} p_{\text{dist}}(A_1, A_2, A_3, A_4, A_5) & \text{if } 2^{-8X-2A_1-3A_2-4(A_3+A_4+A_5)} \geq p_B^{\min} \\ 0 & \text{else.} \end{cases} \quad (4.10)$$

Note that we use $F(\dots)$ to filter the paths that have a too low DP.

Proof. Given the number of active input bits in every Sbox, it is easy to compute the number of paths we can generate by looking into the DDT.¹ We find that for an input Hamming weight of 1 (resp. 2), there are always 2^2 (resp. 2^3) possible output differences. For an Hamming weight of 3, half of the input differences can produce 2^3 differences and half 2^4 differences. Hence, the expected value is $2^{3.58}$. For input Hamming weights of 4 and 5, we can always produce 2^4 differences. Thus, the total expected number of paths we can generate when we have A_i Sboxes with an input Hamming weight of i is $2^{2A_1+3A_2+3.58A_3+4(A_4+A_5)}$.

Moreover, we count only the paths that verify $p_B \geq p_B^{\min}$ by discarding all the paths that have a DP smaller than $p_B^{\min}$ using the filter $F(\dots)$. The DP of the complete path is given by

$$2^{-4X-4X-2A_1-3A_2-4(A_3+A_4+A_5)}. \quad (4.11)$$

Indeed, in the first round, since $2X$ Sboxes are active with one active bit in each active Sbox, we can choose a transition that has a probability 2^{-2} per active Sbox (see the DDT of χ^{-1}). Hence, the DP for the first round is 2^{-4X} . For the second round, since we still have one active bit per Sbox, we have a DP of 2^{-4X} as well. For the third round, an analysis of the DDT shows that, when we have 1 (resp. 2) active bit in the input, the DP of the Sbox is always 2^{-2} (resp. 2^{-3}). For a Hamming weight of 3, there are two different DPs depending on the input. We considered the worst case, which is 2^{-4} . For a Hamming weight of 4 and 5, the DP is always 2^{-4} . Hence, the DP of the complete path verifies (4.11).

Now, from Lemma 4.3.3, we deduce that the paths occur with probability

$$p_{\text{dist}}(A_1, A_2, A_3, A_4, A_5).$$

Hence, the expected number of paths we will get is the sum of all the probabilities of the path that are not discarded by the filter. $\square$

In practice, we compute $G_B(n)$ by summing over all possible values of $A_1, \dots, A_5$, such that $n = A_1 + 2A_2 + 3A_3 + 4A_4 + 5A_5$.

¹We considered the average case here since we already have a lot of paths to start with at the input of the third round.

We have now reached the inbound round and we discard all the paths that do not have all Sboxes active. Hence, we keep only a fraction of $\tau_B^{\text{full}} = 2^{-15.9}$ paths.

It is now easy to see that

$$\tau_B^{\text{DP}} := \sum_{A_5=0}^{\lfloor (n)/5 \rfloor} \sum_{A_4=0}^{\lfloor (n-5A_5)/4 \rfloor} \sum_{A_3=0}^{\lfloor (n-5A_5-4A_4)/3 \rfloor} \sum_{A_2=0}^{\lfloor (n-5A_5-4A_4-3A_3)/2 \rfloor} F(X, A_1, A_2, A_3, A_4, A_5) \quad (4.12)$$

with $F(\dots)$ defined in (4.10) since this is exactly the fraction of path we keep.

To summarize, we have now reached the inbound round and we are able to generate

$$\Gamma_B^{\text{out}} = \epsilon \cdot \binom{5}{2}^X \cdot \binom{s}{X} \cdot \binom{2X}{k} \cdot 2^k \cdot G_B(n) \cdot \tau_B^{\text{full}} \quad (4.13)$$

differences that have a good DP and all Sboxes active and the total number of paths we have to generate is $n_B = \Gamma_B^{\text{out}} / \tau_B = \Gamma_B^{\text{out}} / (\tau_B^{\text{full}} \cdot \tau_B^{\text{DP}})$.

By playing with the filter bound, we noticed the following behavior. The stronger the filter is (i.e., the higher we set the bound on the DP), the higher the expected value of the Hamming weight at the input of the Sboxes of the inbound phase will be. This behavior will allow us to reduce the complexity of our attack in Section 4.7.1, where we discuss the numerical application. Hence, instead of filtering at p_B^{min} , we will filter at a higher value to get better results.

Summary. At this point, we started with n_F (resp. n_B) forward (resp. backward) paths from which we kept only Γ_F^{in} (resp. Γ_B^{out}) candidates that have a DP greater than p_F (resp. p_B) and all Sboxes actives during the inbound.

4.5 Inbound Matching of Weakly Aligned Paths

Now that we have our forward and backward sets of differential paths, we need to estimate the average probability p_{match} that two trails can match during the inbound phase of the rebound attack. We recall that we already enforced all Sboxes to be active during this match, so p_{match} only takes into account the probability that the differential transitions through all the s Sboxes of the internal state are possible.

A trivial method to estimate p_{match} would be to simply consider an average case on the

KECCAK Sbox. More precisely, the average probability that a differential transition is possible through the KECCAK Sbox, given two random non-zero 5-bit input/output differences is equal to $2^{-1.605}$. Thus, one is tempted to derive $p_{\text{match}} = 2^{-1.605 \cdot s}$. However, we observed experimentally that the event of a match greatly depends on the **Hamming weight of the input of the Sboxes** and this can also be indicated from the DDT of the χ layer (for example with an input Hamming weight of one the match probability is $2^{-2.95}$, while for an input Hamming weight of four the match probability is $2^{-0.95}$). Note that *this effect is only strong regarding the input of the Sbox* (i.e. the backward paths), and there is comparatively much lower bias on the differential matching probability concerning the output Hamming weight.

Therefore, in order to model more accurately the input Hamming weight effect on the matching event, we first divide the backward paths *depending on their Hamming weight* and treat each class separately. More precisely, we look at each possible input Hamming weight division among the s Sboxes. To represent this division, we only need to look at the number of Sboxes having a specific input Hamming weight (their relative position do not matter). We denote by c_i the number of Sboxes having an input Hamming weight i and we need the following equations to hold

$$\sum_{i=1}^5 c_i = s \quad (4.14)$$

since we forced that all Sboxes are active during a match. Moreover, for a Hamming weight value w , we have

$$\sum_{i=1}^5 i \cdot c_i = w. \quad (4.15)$$

The set of divisions c_i verifying (4.14) and (4.15) is denoted by C_w . The number of possible 5s-bit vectors satisfying $\bar{c} := (c_1, \dots, c_5)$ (i.e., c_1 Sboxes with 1 active bit, c_2 with 2 etc.) is denoted $b_c(\bar{c}) := b_c(c_1, \dots, c_5)$ and

$$\begin{aligned} b_c(\bar{c}) &= \binom{s}{c_1} \cdot \binom{s-c_1}{c_2} \cdot \dots \cdot \binom{s-c_1-c_2-c_3-c_4}{c_5} \cdot \binom{5}{1}^{c_1} \cdot \dots \cdot \binom{5}{5}^{c_5} \\ &= \frac{s!}{c_1! c_2! \dots c_5!} \cdot 5^{c_1+c_4} \cdot 10^{c_2+c_3}. \end{aligned} \quad (4.16)$$

We can now compute the probability of having a match p_{match} depending on the input

Hamming weight divisions:

Theorem 4.5.1. *The probability p_{match} of having a match is*

$$p_{\text{match}} = \sum_{w=s}^{5s} \Pr[\text{Hw}_{\text{total}} = w | \text{full}] \cdot \sum_{\bar{c} \in C_w} \frac{b_c(\bar{c})}{b_{\text{bucket}}(w, s)} \prod_{i=1}^5 \left(\sum_{y \in \{0,1\}^5} \sum_{\substack{v \in \{0,1\}^5: \\ \text{Hw}(v)=i}} \frac{P_{\text{out}}(y) \cdot \mathbb{1}_{\text{DDT}[v][y]}}{\binom{5}{i}} \right)^{c_i}, \quad (4.17)$$

where $P_{\text{out}}(y)$ is the measured probability distribution of having y at the output of an Sbox when we enforce all Sboxes to be active, $\Pr[\text{Hw}_{\text{total}} = w | \text{full}]$ is the measured probability distribution of the Hamming weight of the input of the Sboxes when all Sboxes are active, $b_c(\dots)$ is given by (4.16), $b_{\text{bucket}}(w, s)$ by Lemma 4.3.1 and $\mathbb{1}_{\text{DDT}[v][y]}$ is set to one if the entry $[v][y]$ is non-zero in the DDT of the χ layer and zero otherwise.¹

Proof. Let full be the event denoting that all Sboxes are active at the inbound phase. We have

$$p_{\text{match}} := \Pr[\text{match} | \text{full}] = \sum_w \Pr[\text{match} | \text{Hw}_{\text{total}} = w, \text{full}] \cdot \Pr[\text{Hw}_{\text{total}} = w | \text{full}].$$

We define $p_{\text{match}}(w) := \Pr[\text{match} | \text{Hw}_{\text{total}} = w, \text{full}]$. We have

$$p_{\text{match}}(w) = \sum_{(c_1, \dots, c_5) \in C_w} \Pr[\text{match} | (c_1, \dots, c_5), \text{Hw}_{\text{total}} = w, \text{full}] \cdot \Pr[(c_1, \dots, c_5) | \text{Hw}_{\text{total}} = w, \text{full}]. \quad (4.18)$$

We easily find that

$$\Pr[(c_1, \dots, c_5) | \text{Hw}_{\text{total}} = w, \text{full}] = \frac{b_c(c_1, \dots, c_5)}{b_{\text{bucket}}(w, s)}, \quad (4.19)$$

since $b_c(c_1, \dots, c_5)$ is the number of possible combinations of vectors verifying $c_1, \dots, c_5$ and $b_{\text{bucket}}(w, s)$ the number of possible combinations of vectors for which all Sbox are active. It remains to compute $\Pr[\text{match} | (c_1, \dots, c_5), \text{Hw}_{\text{total}} = w, \text{full}] = \Pr[\text{match} | (c_1, \dots, c_5), \text{full}]$, since $(c_1, \dots, c_5)$ have all a total Hamming weight of w . We can now consider every Sbox

¹Note that $\Pr[\text{Hw}_{\text{total}} = w | \text{full}]$ greatly depends on the backward paths we choose and that these paths depends on p_{match} . We explain how to solve this cyclic dependency in Section 4.7.1.

independently. Hence,

$$\Pr[\text{match} | (c_1, \dots, c_5), \text{full}] = \prod_{i=1}^5 (\Pr[\text{match} | \text{Hw}_{\text{SBox}} = i, \text{full}])^{c_i} \quad (4.20)$$

and

$$\Pr[\text{match} | \text{Hw}_{\text{SBox}} = i, \text{full}] = \sum_{y \in \{0,1\}^5} \sum_{\substack{v \in \{0,1\}^5: \\ \text{Hw}(v)=i}} \frac{P_{\text{out}}(y) \cdot \mathbb{1}_{\text{DDT}[v][y]}}{\binom{5}{i}}.$$

□

We continue now with our example of the KECCAK- $f[1600]$ internal permutation. The measured distributions along with some intermediate values are given in Appendix A.8. Applying Theorem 4.5.1, we find that $p_{\text{match}} = 2^{-490.15}$. Said in other words, we require to test $1/p_{\text{match}}$ backward/forward paths combinations in order to have a good chance for a match. Note that in the next section, we will actually put an extra condition on the match in order to be able to generate enough values in the worst case during the outbound phase.

4.6 The Outbound Phase

Now that we managed to obtain a match with complexity $1/p_{\text{match}}$, we need to estimate how many solutions can be generated from this match. Again, one is tempted to consider an average case on the KECCAK Sbox: the average number of Sbox values verifying a non-zero random input/output difference such that the transition is possible is equal to $2^{1.65}$. The overall number of solutions would then be $2^{1.65 \cdot s}$. However, as for p_{match} , this number highly depends on the Hamming weight of the input of the Sboxes and this can be easily observed from the DDT of the χ layer (for example with an input Hamming weight of one the average number of solutions is 2^3 , while for an input Hamming weight of four the average number of solutions is 2^1).

In order to obtain the expected number of values N_{match} we can get from a match, we proceed like in the previous section and divide according to the input Hamming weight.

Theorem 4.6.1. *Let N be a random variable denoting the number of values we can generate. Let also full be the event denoting that all the Sboxes are active for the inbound phase. Given a Hamming weight of w at the input of the Sboxes, we can get $N_w := \mathbb{E}[N | \text{match}, \text{Hw}_{\text{total}} =$*

$w, \text{full}]$ values from a match, with

$$N_w = \frac{1}{p_{\text{match}}(w)} \sum_{(c_1, \dots, c_5) \in C_w} \prod_{i=1}^5 Z^{c_i} \cdot \frac{b_c(c_1, \dots, c_5)}{b_{\text{bucket}}(w, s)}, \quad (4.21)$$

with

$$Z := \frac{1}{\binom{5}{i}^2} \left(\sum_{\substack{v \in \{0,1\}^5: \\ \text{Hw}(v)=i}} \text{DDT}[v] \right) \sum_{y \in \{0,1\}^5} \sum_{\substack{v \in \{0,1\}^5: \\ \text{Hw}(v)=i}} P_{\text{out}}(y) \cdot \mathbb{1}_{\text{DDT}[v][y]},$$

where $\text{DDT}[v]$ is the value of the non-zero entries in line v of the DDT, $P_{\text{out}}(y)$ is the measured probability distribution of having y at the output of an Sbox when we enforce all Sboxes to be active, $p_{\text{match}}(w)$ is given by (4.18), $b_c(\dots)$ is given by (4.16), $b_{\text{bucket}}(w, s)$ is given by Lemma 4.3.1 and $\mathbb{1}_{\text{DDT}[v][y]}$ is set to one if the entry $[v][y]$ is non-zero in the DDT of the χ layer and zero otherwise.

Proof. We have

$$\begin{aligned} N_w &= \sum_{\bar{c} \in C_w} \mathbb{E}[N | \text{match}, \bar{c}, \text{Hw}_{\text{total}} = w, \text{full}] \cdot \Pr[\bar{c} | \text{match}, \text{Hw}_{\text{total}} = w, \text{full}] \\ &= \sum_{\bar{c} \in C_w} N_{\text{match}}(\bar{c}) \cdot \frac{\Pr[\text{match} | \bar{c}, \text{Hw}_{\text{total}} = w, \text{full}] \cdot \Pr[\bar{c} | \text{Hw}_{\text{total}} = w, \text{full}]}{\Pr[\text{match} | \text{Hw}_{\text{total}} = w, \text{full}]} \\ &= \sum_{\bar{c} \in C_w} N_{\text{match}}(\bar{c}) \cdot \frac{\Pr[\text{match} | \bar{c}, \text{full}] \cdot \Pr[\bar{c} | \text{Hw}_{\text{total}} = w, \text{full}]}{p_{\text{match}}(w)}, \end{aligned}$$

where $N_{\text{match}}(\bar{c}) = N_{\text{match}}(c_1, \dots, c_5) := \mathbb{E}[N | \text{match}, (c_1, \dots, c_5), \text{full}]$. Note that the remaining terms can be computed from (4.19) and (4.20). Like before, we can now consider each Sbox independently. Thus

$$N_{\text{match}}(c_1, \dots, c_5) = \prod_{i=1}^5 (\mathbb{E}[N_{\text{SBox}} | \text{match}, \text{Hw}_{\text{SBox}} = i, \text{full}])^{c_i},$$

where N_{SBox} is a random variable denoting the number of values we can obtain for a single Sbox. Note that no output distribution needs to be considered, since for a fixed input the non-zero values of the DDT are always the same. We call this non-zero value $\text{DDT}[v]$. Then,

$$\mathbb{E}[N_{\text{SBox}} | \text{match}, \text{Hw}_{\text{SBox}} = i, \text{full}] = \frac{1}{\binom{5}{i}} \sum_{\substack{v \in \{0,1\}^5: \\ \text{Hw}(v)=i}} \text{DDT}[v].$$

□

One would be tempted to take the expected value of all the N_w and compute N_{match} as

$$\sum_w \mathbb{E}[N | \text{match}, \text{Hw}_{\text{total}} = w, \text{full}] \cdot \Pr[\text{Hw}_{\text{total}} = w | \text{match}, \text{full}] .$$

This expectancy would be fine if we were expecting a high number of matches. This is however not necessarily our case. Hence, we need to ensure that the number of values we can generate from the inbound is sufficient. To do this, first note that N_w decreases exponentially while w increases. Similarly, $p_{\text{match}}(w)$ increases exponentially while w increases. This is depicted in Figure A.3 in Appendix A.8. Thus, we are more likely to obtain a match at a high Hamming weight which will lead to an insufficient N_{match} .

To solve this issue, we proceed as follows. First, we compute N_w for every w . We check then the maximum Hamming weight w_{max} we can afford such that $N_{w_{\text{max}}} \geq 1/(p_B \cdot p_F)$. This way, we are ensured to obtain enough solutions from the match. However, we need to update our definition of a match: a match occurs only when the Hamming weight of the input is lower than w_{max} . Hence, instead of summing over all possible values of w , we sum only up to w_{max} and (4.17) becomes

$$p_{\text{match}} = \sum_{w=s}^{w_{\text{max}}} \Pr[\text{Hw}_{\text{total}} = w | \text{full}] \cdot \sum_{\bar{c} \in C_w} \frac{b_c(\bar{c})}{b_{\text{bucket}}(w, s)} \prod_{i=1}^5 \left(\sum_{y \in \{0,1\}^5} \sum_{\substack{v \in \{0,1\}^5: \\ \text{Hw}(v)=i}} \frac{P_{\text{out}}(y) \cdot \mathbb{1}_{\text{DDT}[v][y]}}{\binom{5}{i}} \right)^{c_i} . \quad (4.22)$$

The number of values we can then obtain from the inbound is $N_{\text{match}} \geq N_{w_{\text{max}}}$.

We can now apply this model to the KECCAK- $f[1600]$ internal permutation. Some useful intermediate results and relevant $N_{w_{\text{max}}}$ (with their associated p_{match}) are shown in Appendix A.8.

4.7 Finalizing the Rebound Attack

4.7.1 Solving the Dependency Chain

In Section 4.4, we showed how to choose the backward paths given the probability of having a match in the inbound phase (p_{match}) and the number of solution we can generate from

this match (N_{match}). In Sections 4.5 and 4.6, we showed how to compute p_{match} and N_{match} . However, in these computations, we needed the probability distribution of the Hamming weight of the input of the Sbox, $\Pr[\text{Hw}_{\text{total}} = w | \text{full}]$. This probability depends greatly on the paths we select in Section 4.4.

To solve this circular dependency, we performed several iterations of the following algorithm until we found some parameters that verify all equations. First, we estimated roughly $\Pr[\text{Hw}_{\text{total}} = w | \text{full}]$ by taking some random backward paths with limited complexity. Using the worst case cost of these paths, we were able to select w_{max} from Table A.21 such that the number of values generated from the inbound is sufficient. Then, we computed p_{match} and N_{match} . With this first guess, we searched for an X and a k such that we can find a match with a good probability and such that we can generate enough values from the inbound. Then, we computed $\Pr[\text{Hw}_{\text{total}} = w | \text{full}]$ using these new paths generated by X , k and p_B and started our algorithm again with this new distribution. After some iterations, we found a set of filtered backwards paths that provided a sufficient p_{match} and N_{match} .

As discussed in Section 4.4, we noticed the following interesting behavior. By increasing p_B , the expectation of $\Pr[\text{Hw}_{\text{total}} = w | \text{full}]$ is higher. This leads then to a smaller N_{match} and a greater p_{match} . Furthermore, less values need to be generated from the inbound phase since the worst case cost of the backward paths is lower. By taking advantage of this behavior, we were able to reduce significantly the complexity of our attack.

When $(X, k) = (8, 9)$, we have that the number of backward input differences is $\Gamma_B^{\text{in}}(X) = 2^{77.7}$ and $\epsilon = 0.736$. Thus, we have $\Gamma_B^{\text{mid}} = \Gamma_B^{\text{in}} \cdot \epsilon = 2^{77.26}$. If we filter all paths that have a DP smaller than 2^{-461} , i.e., we set $p_B = 2^{-461}$, we get for $(X, k) = (8, 9)$ at least $\epsilon \cdot \Gamma_B^{\text{in}}(X) \cdot \binom{2X}{k} \cdot 2^k \cdot G_B(n) \cdot \tau_B^{\text{full}} = 2^{475.07}$ distinct differences using (4.13) for the inbound (for these parameters, the difference Hamming weight at the input of the χ layer in the third round is $n = 227.9$). With these parameters, since we remove the paths with a DP lower than p_B , we keep $\tau_B^{\text{DP}} = 36\%$ of the paths, following (4.12). Hence, we filter the backward paths with a ratio $\tau_B = \tau_B^{\text{full}} \cdot \tau_B^{\text{DP}} = 2^{-15.9} \cdot 0.36 = 2^{-17.37}$. We have also $p_B = 2^{-461}$ and $p_F = 2^{-48}$. Therefore, we need $N_{\text{match}} \geq 2^{509}$. If we consult Table A.21, we find that we have to set $w_{\text{max}} = 956$. This leads to $p_{\text{match}} = 2^{-498.11}$. This implies that the minimum total number of backward paths we need to generate is $n_B^{\text{min}} = 2^{492.18}$. All these paths apply on $nr_B = 3$ KECCAK- $f[1600]$ rounds, all with DP higher or equal to $p_B^{\text{min}} = 2^{48-509} = 2^{-461}$.

To summarize, we have that the number of backward output differences is $\Gamma_B^{\text{out}} = n_B^{\text{min}} \cdot \tau_B = 2^{492.18-17.37} = 2^{474.81}$ and that the number of forward input differences is $\Gamma_F^{\text{in}} = 2^{23.3}$. Hence, there is a total of $2^{498.11}$ couples of $(\Delta_B^{\text{out}}, \Delta_F^{\text{in}})$ for the inbound phase, which is enough since it is equal to $1/p_{\text{match}}$. Once a match is found, the worst case complexity of the connected path is $1/(p_B \cdot p_F) \leq 2^{461+48} = 2^{509}$ which is equal to N_{match} . Hence, we can generate enough values from the inbound phase to find with a good probability values verifying the differential path.

The *overall complexity for the rebound attack* given by (4.1) is $C = 2^{509}$.

This model was verified on the KECCAK- $f[100]$ internal permutation. By applying this attack on it, we found a 4-round result together with solution pairs, which are shown in Appendix A.10. This gives a 6-round distinguisher with complexity $2^{28.76}$ which is higher than the simple distinguishers for 6 rounds. However, our goal for KECCAK- $f[100]$ was to verify our model in practice, so that we can be confident for applying it to the KECCAK- $f[1600]$ version. Moreover, finding a solution as in Appendix A.10 is hard since all s Sboxes are active in the middle of the path.

4.7.2 The Rebound Distinguisher

We will use exactly the same type of limited-birthday distinguishers as in Section 4.3.1. Our rebound attack finds pairs of internal state values such that the input and output difference masks are fully predetermined and we already showed that this should require 2^b operations in the generic case. Therefore, we obtain a $(nr_B + nr_F + 1)$ -round distinguisher for the b -bit KECCAK internal permutation considered if the total cost of the rebound attack to find one solution is lower than 2^b .

In the case of KECCAK- $f[1600]$, we have $nr_B = nr_F = 3$ but note that the backward paths utilized do not have the same input difference Δ_B^{in} and the forward paths do not have the same output difference Δ_F^{out} . Indeed, the backward paths have been generated with $\Gamma_B^{\text{in}}(X)$ distinct starting difference patterns in the second round and thus we have $\Gamma_B^{\text{in}}(X)$ possible input difference masks Δ_B^{in} . Regarding the forward paths, we utilized Γ_F^{out} distinct starting points and thus we have Γ_F^{out} possible output difference masks Δ_F^{out} . Then, our rebound attack finds 7-round pairs of internal state values such that the input and output

difference masks lie in a subspace of size $\Gamma_B^{\text{in}}(8) = 2^{77.7}$ and $\Gamma_F^{\text{out}} = 2^6$ respectively, with total complexity 2^{509} (whereas the generic case complexity is $2^{1516.3}$, as explained in Section 4.3.1).

Figure 4.5: The forward trails with an extended round. The distance between the two lines reflects the number of differences.

However, we can attack one more round by adding an extra round to the right of the 7-round path exactly as we did for the distinguishers in Section 4.3.1. Namely, we have Γ_F^{out} possible output difference masks, all with 6 active bits. Figure 4.5 depicts the new forward paths. We computed that each one of these output differences for the 7-round path can be mapped to only 2^{106} output differences for the extended 8-round path. One can calculate this by analyzing b_3 in Table A.23. We have 50 active Sboxes with one active bit that can produce 2^2 transitions and 2 active Sboxes with two active bits that can produce 2^3 transitions (the number of transitions can be checked in Table A.4). Therefore, our rebound attack finds 8-round pairs of internal state values such that the input and output difference masks lie in a subspace of size $\Gamma_B^{\text{in}}(8) = 2^{77.7}$ and $\Gamma_F^{\text{out}} = 2^6 \cdot 2^{106} = 2^{112}$ respectively, with total complexity 2^{509} (whereas the generic case complexity is $2^{1410.3}$).

4.8 Further Improvements

In the distinguisher presented in the previous section, the bottleneck of the attack came from the outbound phase. However, the gap between its complexity and the generic complexity was enormous. In this section, we reduce the complexity of the attack by narrowing this gap. We reduce the complexity of the distinguisher by relaxing one round in both the forward and the backward direction.

Relaxing the Forward Paths The idea is to take the same forward paths as in Section 4.4 but we allow any possible transition in the last round to avoid the cost of this round (see Figure 4.6). Hence we get this round for free except that we will reduce the generic complexity of the generic distinguisher since we will have more possible output differences. In short, we have to pay only for two rounds with a worst case DP of 2^{-36} .

Figure 4.6: The forward trails when we relax the 3rd round. The distance between the two lines reflects the number of differences.

We analyze now the impact of this modification on Γ_F^{out} , the set of reachable output differences. At the entrance of the third round, every Sbox has one single active bit. Hence, according to the DDT, there are only 4 different possibilities at the output of the Sboxes. Since we have 6 active Sboxes in the third round, the number of possible differences at the output of the third round is multiplied by $4^6 = 2^{12}$. Thus, the number of differences at the output of the third round is $\Gamma_F^{\text{mid}} \cdot 2^{12} = 2^6 \cdot 2^{12} = 2^{18}$.

We need now to look at the fourth round to obtain Γ_F^{out} and compute the generic complexity of the distinguisher. In the third round, every active Sbox can produce at most 3 active bits at its output, since each active Sbox has only one single active bit at its input. Hence, the maximum Hamming weight at the output is $3 \cdot 6 = 18$. Each of these active bits can be expanded to at most 11 bits through θ and hence, we have at most $11 \cdot 18 = 198$ active bits at the input of the Sboxes of the fourth round. In the worst case, each of these bits will be in a different Sbox and will produce four possible differences. Hence, we have $\Gamma_F^{\text{out}} \leq \Gamma_F^{\text{mid}} \cdot 2^{12} \cdot 4^{198} = 2^{18} \cdot 2^{396} = 2^{414}$.

To conclude, we reduced the complexity of the attack by a factor 2^{12} and increased the number of possible outputs for the distinguisher by a factor 2^{302} .

Relaxing the Backward Paths We can do the same kind of operation for the backward paths by relaxing the first round of each path (see Figure 4.7). Instead of picking only $1 \mapsto 1$ transitions, i.e., transition that maps one bit to a single bit in the χ^{-1} layer, we allow any possible transition and, hence, do not have to pay for it. This reduces the complexity of the backward paths by a factor 2^{4X} . Each Sbox with one single active bit at its output can have 9 possible input differences and the maximum possible of input differences that can occur for a given input difference is 12 (see χ^{-1} DDT). Since we have $2X$ active Sboxes, the number of possible input differences is increased by a factor of at most 9^{2X} . Therefore, $\Gamma_B^{\text{in}} \leq \Gamma_B^{\text{mid}} \cdot 9^{2X}/\epsilon$ and we reduced the complexity by a factor 2^{4X} .

Figure 4.7: The backward trails when we relax the first round. The distance between the two arrows reflects the number of differences.

Finding New Parameters Now that we have reduced the cost of the forward paths by 2^{12} and the cost of the backward paths by 2^{4X} , we do not need to generate that many values in the outbound phase. Furthermore, the bottleneck of the attack is now in the inbound phase, i.e., it comes from p_{match} .

Our goal is now to increase p_{match} so that the complexity of the attack is reduced. There are two ways of increasing p_{match} .

- Recall that a way of increasing p_{match} is to allow again a match to happen at a higher Hamming weight at the input of the Sboxes of the inbound round. These matches are much more probable and will increase p_{match} . However, as discussed in Section 4.6, N_{match} drops in this case. This is not an issue anymore, since we need much less values to verify the differential paths.

- Another solution is to modify the Hamming weight at the input of the Sboxes of the inbound round. This can be done by modifying X or k .¹ We will see that we can reduce k since less paths will be filtered because of their DP.

We found that for $(X, k) = (8, 8)$, we can get at least $\epsilon \cdot \binom{5}{2}^X \cdot \binom{s}{X} \cdot \binom{2X}{k} \cdot 2^k \cdot G_B(n) \cdot \tau_B^{\text{full}} = 2^{493.88} \cdot 2^{-15.9} = 2^{477.98}$ distinct differences with worst case DP $p_B = 2^{-450}$ and this, without almost any filtering on the DP, i.e., $\tau_B^{\text{DP}} \approx 1 - 10^{-10}$. Note that for these parameters, $\epsilon = 0.736$ as before and $n = 220.61$.

The worst case total cost of the outbound phase is $1/(p_B \cdot p_F) = 2^{450+36} = 2^{486}$. Remark that we need a N_{match} much lower than before (2^{486} instead of 2^{509}). By looking into Table A.21 in Appendix A.8 we remark that any w_{max} in the table will produce enough values in the outbound phase and we select $w_{\text{max}} = 1000$. The Hamming weight distribution at the input of the Sboxes of the inbound phase ($\Pr[\text{Hw}_{\text{total}} = w | \text{full}]$) when $k = 8$ behaves like $\mathcal{N}(847.47, 651.51)$. With this new distribution, we can compute the new p_{match} for $w_{\text{max}} = 1000$ and we obtain $p_{\text{match}} = 2^{-491.47}$ and $N_{\text{match}} = 2^{486.81}$. Since Γ_F^{in} is still equal to $2^{23.3}$, the total number of backward paths we have to generate is $n_B^{\text{min}} = 1/(p_{\text{match}} \cdot \Gamma_F^{\text{in}} \cdot \tau_B) = 1/(p_{\text{match}} \cdot \Gamma_F^{\text{in}} \cdot \tau_B^{\text{full}}) = 2^{484.07}$. We have then that the total number of backward output differences is $\Gamma_B^{\text{out}} = n_B^{\text{min}} \cdot \tau_B = 2^{468.17}$. Hence, there is a total of $2^{491.47}$ couples of $(\Delta_B^{\text{out}}, \Delta_F^{\text{in}})$ for the inbound phase, which is enough since it is equal to $1/p_{\text{match}}$. Once a match is found, the worst case complexity of the connected path is $1/(p_B \cdot p_F) = 2^{486}$ which is lower than N_{match} . Thus, we can generate enough values from the inbound phase to find with a good probability values verifying the differential path.

The new overall complexity for the rebound attack given by (4.1) is $C = 2^{491.47}$. Note that the bottleneck now comes from p_{match} .

We have $\Gamma_B^{\text{in}} \leq \Gamma_B^{\text{mid}}(8) \cdot 9^{2 \cdot 8} / \epsilon = 2^{77.7+50.7} = 2^{128.4}$ and $\Gamma_F^{\text{out}} \leq 2^{414}$. The new generic complexity of the distinguisher is, hence, greater than $2^{1057.6}$.

b	best differential distinguishers complexity [†]							
	1-round	2-round	3-round	4-round	5-round	6-round	7-round	8-round
100	1	1	1	2^2	2^8	2^{19}	-	-
200	1	1	1	2^2	2^8	2^{20}	2^{46}	-
400	1	1	1	2^2	2^8	2^{24}	2^{84}	-
800	1	1	1	2^2	2^8	2^{32}	2^{109}	-
1600	1	1	1	2^2	2^8	2^{32}	2^{142}	$2^{491.47} [2^{1057.6}]$

[†] Note that due to its technical complexity when applied on KECCAK, the rebound attack has only been applied to KECCAK- $f[100]$ and KECCAK- $f[1600]$. In brackets, the generic complexity.

Table 4.2: Best differential distinguishers for KECCAK internal permutations

4.9 Results and Conclusion

In this chapter, we analyzed the internal permutations used in the KECCAK family of hash functions in regards to differential cryptanalysis. We first proposed a generic method that looks for the best differential paths using CP-kernel considerations and better χ mapping. This new method provides some of the best known differential paths for the KECCAK internal permutations and we derived distinguishers with rather low complexity exploiting these trails. In particular we were able to obtain a practical distinguisher for 6 rounds of the KECCAK- $f[1600]$ permutation. Then, aiming for attacks reaching more rounds, we adapted the rebound attack to the KECCAK case. This adaptation is far from trivial and contains many technical details. Our model was verified by applying the attack on the reduced version KECCAK- $f[100]$. The main final result is a 8-round distinguisher for the KECCAK- $f[1600]$ internal permutation with a complexity of $2^{491.47}$. All our distinguisher results are summarized in Table 4.2. Note that our attack does not endanger the security of the full KECCAK. We believe that this work will also help to apply the rebound attack on a much larger set of primitives.

This work might be extended in many ways, in particular by further refining the differential path search or by improving the inbound phase of the rebound attack such that the overall cost is reduced. Moreover, another research direction would be to analyse how the differential paths derived in this chapter can lead to collision attacks against reduced versions of the KECCAK hash functions. Also, the bottleneck of our attack is now p_{match} . Using the techniques presented in [147] could help reducing the complexity of it.

¹In fact experiments have shown that lowering k leads to a better Hamming weight distribution for $X = 8$.

Chapter 5

Meet-in-the-Middle Attacks on DES and KTANTAN

This chapter is dedicated to meet-in-the-middle attacks. In Section 5.1, we review meet-in-the-middle attacks on block ciphers, then we analyze the DES key schedule and present low data complexity attacks on 7-round variants of DES. These attacks are currently the best such attacks in number of rounds and require only one known plaintext.

Section 5.2 illustrates our first contribution on cryptanalysis of the lightweight block cipher KTANTAN family. In particular, we show corrections to previous cryptanalysis and point out that the key schedule is in fact more vulnerable to MITM attacks than previously reported. The correction leads to the first non-marginal break of the entire KTANTAN family, reducing a factor of 2^6 on KTANTAN-32, KTANTAN-48 and $2^{5.5}$ on KTANTAN-64 from the exhaustive search complexity of 2^{80} . The reduction of the attack complexity is also confirmed by a low complexity computer experiment.

In Section 5.3 we introduce some of the recent advances in meet-in-the-middle preimage attacks of hash functions. Despite the presence of a secret key in the block cipher case, we identify techniques that can also be mounted on block ciphers, thus allowing us to improve the cryptanalysis results on KTANTAN family further. Especially, the technique *indirect-partial-matching* can be used to increase the number of matched bits significantly, as exemplified by our attacks. To the best of our knowledge, this is the first time that a *splice-and-cut* meet-in-the-middle attack is applied to block ciphers. When the splitting

point is close to the start or the end of the cipher, the attack remains to be at very low data complexity. The MITM key recovery attacks on KTANTAN-32, KTANTAN-48 and KTANTAN-64 work with time complexities of $2^{72.9}$, $2^{73.8}$ and $2^{74.4}$ respectively, in number of full round encryptions. Each of the three attacks works with 4 chosen ciphertexts only. These results are the best cryptanalysis on KTANTAN family before the recent improvement presented in [189].

5.1 MITM Attacks on Round-Reduced DES

Meet-in-the-middle (MITM) is a generic cryptanalytic approach originally developed from cryptanalysis of block ciphers. Early development starts with cryptanalysis of DES, which dates back to 1977. Similar ideas have been applied to round-reduced variants of AES in last 12 years. We will review some of the results and techniques developed on DES, AES and a few other block ciphers in Section 5.1.1, especially the specific line of research on round-reduced DES is introduced at the end of Section 5.1.1. Finally we analyze the DES key schedule in Section 5.1.2, which leads to our results on 7-round variants of DES.

5.1.1 MITM Attacks on Block Ciphers

In an early cryptanalysis of DES by Diffie and Hellman [66] in 1977, a meet-in-the-middle attack was proposed to examine the security of a cascade of two instances of the DES encryption routine $E_K(\cdot)$, where two independent 56-bit keys K_1 and K_2 are used. In this 2-key scheme, a plaintext P has correspondingly a ciphertext $C := E_{K_2}(E_{K_1}(P))$ and the decryption is simply the inverse. Although the effective key-length is 112-bit, the actual security is shown to be far below 112-bit, based on the following observation: given a pair (P, C) , an intermediate value M can be computed as $M := E_{K_1}(P)$ and $M := E_{K_2}^{-1}(C)$ in parallel with independent guesses of K_1 and K_2 .

An adversary first stores all the 2^{56} values of K_1 in a table $\mathcal{T}$, with each entry indexed by $M := E_{K_1}(P)$. He then tests one value of K_2 at a time to compute $M' := E_{K_2}^{-1}(C)$ and compare M' with the table entry $\mathcal{T}[M']$, on an event of a match he can simply try a second known plaintext pair (P', C') to test if $(\mathcal{T}[M'], K_2)$ is the real key. Since the *unicity distance* is 2 ($\lceil (56 \times 2)/64 \rceil = 2$), the correct 112-bit key pair (K_1, K_2) can be deduced with 2 known

Figure 5.1: Basic MITM attacks

plaintexts and the time complexity is only 2^{57} DES encryptions, with a memory complexity of 2^{56} .

The attack exploits the weakness that two halves of the cipher use different sets of key bits. Figure 5.1 shows a more typical setup where E_K can be represented as $E := H_{K_2} \circ G_{K_1}$. In fact TDEA can also be attack by MITM attacks in a similar idea [139, 129].

MITM Attacks on AES and Other Block Ciphers. MITM attacks against AES are a bit different from the techniques just described. A cryptanalyst aims to find some special properties of $M := G_K(P)$, such as the *balanced property*, and then try to decrypt (*i.e.*, compute $M' := H_K^{-1}(C)$) the second half to match M' with the special properties so that wrong key guesses can be filtered out. This type of attacks on AES start with a 6-round result in the Rijndael submission [56], then they have been improved to 7- and 8-round results in a series of work [60, 63, 72, 75]. More dedicated attacks with meet-in-the-middle technique can be found in the practical attack to KeeLoq [96] and cryptanalysis of reduced round IDEA [61].

More Recent MITM Attacks. In [41] the framework is made clearer with different notions as compared with the basic idea in [48]. Some key bits of K are not used in the first S_1 rounds, *i.e.*, $G_K = G_{K_1}$ with $K_1 \subsetneq K$ being the key bits essentially used in the first S_1 rounds. Similarly, $H_K = H_{K_2}$ with $K_2 \subsetneq K$ for the last S_2 rounds. In addition, the matching is done in a separate partial matching phase of the middle $R - S_1 - S_2$ rounds transformation $I(\cdot)$ from two fully determined states $M := G_{K_1}(P)$ and $M' := H_{K_2}^{-1}(C)$, *i.e.*, $E_K(\cdot) = H_{K_2} \circ I \circ G_{K_1}(\cdot)$. The formalization allows better understanding of the attack and may breed powerful extensions. Similar MITM attacks have been applied to block ciphers including round reduced GOST [169] and XTEA [170] recently.

MITM Attacks on Round-Reduced DES. In a modern block cipher, the sets of key bits used by two halves of the cipher are rarely disjoint like the case of doubly cascaded DES. Meet-in-the-middle remains effective even in less desirable scenarios. In 1985, Chaum and Evertse published several meet-in-the-middle attacks on round reduced DES [48]. The idea is to find “bits in the middle” that are independent of certain key bits when computed from both ends. For an R -round case with encryption function $E_K(\cdot)$, let the $G_K(\cdot)$ be the first S rounds and $H_K(\cdot)$ be the rest $R - S$ rounds, we have $E_K(\cdot) = H_K \circ G_K(\cdot)$. Given text pair (P, C) , we can compute the intermediate values $M := G_K(P)$ and $M' := H_K^{-1}(C)$ for a key guess K . The attack is effective if there are two different sets $\tilde{K}_1, \tilde{K}_2 \subsetneq K$, such that with a nonzero mask b , $b \& G_{\tilde{K}_1}(P) = b \& H_{\tilde{K}_2}^{-1}(C)$ for all (P, C) pairs, where $\&$ is the bitwise AND of two equal length binary strings. The $wt(b)$ -bit match can be used to filter wrong keys at a ratio of $1 - 2^{-wt(b)}$ during the MITM attack, $wt(\cdot)$ computes the weight of the binary vector b . The attacks are effective on DES reduced up to 6 rounds. For 7 rounds, only the case of 2-8 can be attacked by saving a factor of 2 from the exhaustive search.

Although DES is broken in full, it remains meaningful to study how many rounds of DES can be broken with one (or few) known plaintexts or chosen ciphertexts. This restriction is much more relevant to ciphers deployed in the real world, for a number of reasons. In practice a symmetric cipher is often used with a secret key agreed by a public-key key distribution protocol, and the lifespan of the key is within a session which generates limited amount of plaintext-ciphertext pairs. Moreover, as we discussed in Section 2.1.1, an n_B -bit block cipher is often required to re-key before $2^{n_B/2}$ encryptions to avoid codebook attacks. In either case, attacks that are statistical in nature would be unlikely if they require a considerable amount of plaintext-ciphertext pairs which are not available in the first place. In reality, an attack is much more likely to happen if an attacker only needs a handful of known plaintexts or chosen ciphertexts to launch the attack ¹.

In [73], the above mentioned approach is devised in a novel way and improved results are obtained on DES reduced to 4, 5, and 6 rounds. To compute the intermediate values, instead of guessing the key bits required, one can guess the actual intermediate state values to save the need of guessing many key bits for one state bit.

¹Although algebraic attacks usually require very few known plaintexts too, we do not discuss them here since we are not aware of any algebraic attack that reaches 7 rounds of DES.

We improve the MITM attack results on round reduced DES by one more round by directly working in bit level using a simple dependency map, thanks to the precise alternative formulation of MITM attacks in [41]. We cite [73, Table 1] the comparison of MITM attacks in [48] and [73] and add our results in Table 5.1 below and discuss the details of our attacks in Section 5.1.2.

Rounds	Attack	Data Complexity	Time Complexity	Comment
4	Differential	16 CP	Negligible [†]	[73]
	Linear	52 KP	$> 2^{13.7}$ [†]	[73]
	MITM	1 KP	2^{35} [†]	[48]
	MITM	1 KP	$2^{32.0}$	[73, Section 4.1]
	MITM	15 KP	$2^{20.0}$	[73, Section 4.2]
	MITM	6 CC	$2^{19.3}$	[73, Section 4.3]
5	Differential	64 CP	$> 2^{11.7}$ [†]	[73]
	Linear	72 KP	$> 2^{13.8}$ [†]	[73]
	MITM	1 KP	$2^{45.5}$ [†]	[48]
	MITM	51 KP	$2^{35.5}$	[73, Section 5]
	MITM	28 KP	$2^{37.9}$	[73, Section 5]
	MITM	8 CC	2^{30}	[73, Section 5.1]
6	Differential	256 CP	$2^{13.7}$ [†]	[73]
	Linear	> 104 KP	$2^{13.9}$ [†]	[73]
	MITM	1 KP	$2^{52.9}$ [†]	[48]
	MITM	1 KP	$2^{51.8}$	[73, Section 6]
7	MITM	1 KP	2^{56}	This thesis [‡]
8	Differential	2^{14} CP	2^9 [†]	[31]
9	Differential	2^{24} CP	2^{32} [†]	[31]

[†] The attack retrieves only parts of the key.

[‡] Although Feistel ciphers usually have an even number of rounds, security analysis can still be done in a round-wise setting.

Table 5.1: Comparison of attacks on round reduced DES

5.1.2 7-Round MITM Attacks on DES

The DES Key-Schedule. The DES key schedule algorithm generates 16 sub-keys of 48 bits each, when the user supplies a 64-bit secret key (including 8 parity bits) which has an effective length of 56-bit. We refer to [2, Section 7.4.2] for the following compact description of the procedure. Let the 64-bit key be $K := k_1 \dots k_{64}$ and K_i be the 48-bit sub-key of round i , for $1 \leq i \leq 16$. Define v_i for $1 \leq i \leq 16$ as $v_i = 1$ for $i \in \{1, 2, 9, 16\}$ and $v_i = 2$ otherwise. Each v_i value is used as a rotation amount. Let $T := PC1(K)$; represent T as

28-bit halves (C_0, D_0) . (Use PC1 in Table 5.2 to select bits from K : $C_0 = k_{57}k_{49} \dots k_{36}$ and $D_0 = k_{63}k_{55} \dots k_4$.) For i from 1 to 16, compute K_i as follows: $C_i = C_{i-1} \lll v_i$, $D_i = D_{i-1} \lll v_i$, $K_i = PC2(C_i, D_i)$. (Use PC2 in Table 5.3 to select 48 bits from the concatenation $C_i || D_i$.)

<i>PC1</i>							
C	57	49	41	33	25	17	9
	1	58	50	42	34	26	18
	10	2	59	51	43	35	27
	19	11	3	60	52	44	36
D	63	55	47	39	31	23	15
	7	62	54	46	38	30	22
	14	6	61	53	45	37	29
	21	13	5	28	20	12	4

Table 5.2: DES Permutation Choice 1

<i>PC2</i>					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Table 5.3: DES Permutation Choice 2

The MITM attacks under the framework of [41] are possible on DES variants due to two reasons. First the key schedule of DES is completely linear, in fact each sub-key bit is always selected from the secret key (see Appendix B.2 for the derived sub-keys in original secret key bits). Second, each sub-key uses 48 out of the 56 secret key bits, so that it is possible to find “bits in the middle” that are always neutral from a small set of key bits.

The 7-Round Results. We use a simple procedure to determine which are the key bits required to compute a certain state bit in the middle, *i.e.*, we use a *bit dependency map*. Let A be the set $\{1, 2, \dots, 64\}$ representing indices of secret key bits (with 8 of them being the parity bits and not actually used), in little-endian bit ordering. First we derive the 16 sets of sub-key bits as indices of A , $S_1, S_2, \dots, S_{16} \subset A$. Then we analyze both the encryption and decryption by associating each intermediate bit a 64-bit zero vector as the *bit dependency map*. If an intermediate bit depends on a certain bit k of A for encryption/decryption, the bit position is set in its bit dependency map. In round i , we set corresponding bits in S_i to the map of state bits for the sub-key mixing step. For the rest of the permutations and substitutions, the rule is simple: each bit inherits the map of a previous bit if it is a re-wiring (for the permutation case), otherwise the map is the bitwise logic OR of the maps of all

the inputs that it depends on (for the Sbox case). Given a setting (r_{start}, r_{end}, c) , we can run the procedure from round r_{start} for c rounds forward, and from round r_{end} backward to meet at the cut c . If the dependency map is not fully set for the 56 effective key bits, then the bit is neutral to some key bits.

State	A_1	A_2	State	A_1	A_2
33 [†]	14, 49, 52, 59	2, 23, 58, 63	34	49, 52, 59	2, 58
35	13,23	20,61	36	14	23, 63
37	13,53	20	40	13,23	20,61
41	25	26,36	42 [†]	14, 49, 52, 59	2, 23, 58, 63
43	14	23, 63	45	49, 52, 59	2, 58
46	13,23	20,61	47	13,53	20
49	25	26,36	50	49, 52, 59	2, 58
51	14	23, 63	52 [†]	14, 49, 52, 59	2, 23, 58, 63
53	13,53	20	55	25	26,36
57	13,23	20,61	58 [†]	14, 49, 52, 59	2, 23, 58, 63
59	13,53	20	60	49, 52, 59	2, 58
61	14	23, 63	63	25	26,36

[†] State bits with the biggest set of neutral key bits are chosen for the MITM attack to achieve optimal complexity.

Table 5.4: Bits dependency map after round 7 for DES reduced to round 5 to 11

From the procedure, we discover that for up to 7 rounds, we can always find some middle state bits that do not require the full set of key bits when computed from either the plaintext or ciphertext, the scenario allows an MITM attack. We show one such example for DES reduced to from round 5 to round 11 (inclusive) in Table 5.4, including in total 24 vulnerable state bits, let A_1 denote the set of neutral key bits if computed forward and A_2 the set for backwards, the state bits are after round 7, which we call a *cut* of 3 rounds (from the start of the 7 rounds). We select the state bits at the cut with biggest possible sets of neutral bits, *i.e.*, the m state bits (33, 42, 52, 58) for MITM matching can be computed without key bits (14, 49, 52, 59) from the plaintext, or without (2, 23, 58, 63) from the ciphertext. The attack would work with only 1 known plaintext since one plaintext-ciphertext pair carries enough information to deduce the key with a success probability close to 1. The memory complexity is $2^{\min\{|A_1|, |A_2|\}}$ for the MITM matching and the time complexity is

$$2^{56-|A_1|-|A_2|}(c/r \times 2^{|A_2|} + (r-c)/r \times 2^{|A_1|} + 2^{|A_1|+|A_2|-m}),$$

where c is the cut and $r = 7$ is the total number of rounds. For the attack in Table 5.4, the

time complexity is $2^{56-4-4} \times (3/7 \times 2^4 + 4/7 \times 2^4 + 2^4) = 2^{53}$. In fact all the variants of 7 consecutive rounds of DES can be attacked similarly, we show these results in Table 5.5.

Rounds	Cut	A_1	A_2	Bits to match	Time
1-7	3	7, 12, 43, 52	17, 27, 54	39, 44, 54, 64	$2^{53.36}$
	4	7, 12, 43, 52	17, 27, 54	7, 12, 22, 32	$2^{53.36}$
2-8	3	3, 35, 42, 62	5, 33, 42, 63	33, 42, 52, 58	2^{53}
3-9	3	22, 57	21, 25, 34, 58	38, 48, 56, 62	$2^{53.72}$
4-10	3	3, 10, 30, 36	9, 12, 18, 39	33, 42, 52, 58	2^{53}
5-11	3	14, 49, 52, 59	2, 23, 58, 63	33, 42, 52, 58	2^{53}
6-12	3	33, 36, 43, 61	7, 42, 47, 51	33, 42, 52, 58	2^{53}
7-13	3	17, 27, 45, 49	26, 31, 35, 54	33, 42, 52, 58	2^{53}
8-14	3	29, 33, 38, 42	13, 17, 43	39, 44, 54, 64	$2^{53.36}$
9-15	3	3, 21, 25, 58	3, 22, 59, 62	33, 42, 52, 58	2^{53}
10-16	3	12, 18	15, 19, 51, 60	38, 48, 56, 62	$2^{53.72}$

Table 5.5: Neutral bits and attacks for different versions of 7-round DES

5.2 Cryptanalysis of the KTANTAN Family

Low-end computing devices such as RFID tags are usually characterized by extremely tight cost and power consumption requirements. The needs of cryptography on such devices have been increasing with the growing pervasiveness and mass deployment of these devices. There are several block cipher primitives proposed in recent years, targeting very small footprint and reduced power consumption, in particular DESL [123], PRESENT [40], HIGHT [94], mCrypton [127], the KATAN/KTANTAN family [45] and PRINTCIPHER [111] which is proposed as a cryptography solution for IC-printing.

In this section we analyze the security of the KTANTAN family following the work in [41]. We briefly introduce the attack of [41] in Section 5.2.1. Then we point out that this initial cryptanalysis is incorrect and give corrected results in Section 5.2.2. An implementation of our attack confirms the reduction of time complexity in our attack, we show the result in Section 5.2.3. For further improvements to the attacks, we refer to Section 5.3.

5.2.1 Previous Meet-in-the-Middle Attacks on KTANTAN

The attack manages to recover the secret key with very low data complexity, as required by the *unicity distance* in the known plaintext setting. The attack to KTANTAN-32 runs at a

time complexity of 2^{79} , although arguably marginal, is the first claimed key-recovery attack applicable to the full 254 rounds faster than brute force. For block size b of 48 and 64, the attack manages to break 251 and 248 rounds respectively.

The main idea of this attack is to cut the R -round KTANTAN- b cipher into three parts for some $\alpha, \beta < R$, *i.e.*, the first α rounds as the *forward* phase, the last β rounds as the *backward* phase and the middle $R - \alpha - \beta$ rounds as the *partial matching (MITM)* phase. Let x_i be the state after round i , for $0 \leq i \leq 254$. Let $\varphi_{i,j}$ be the transformation from round i to round j (inclusive) of the R -round KTANTAN- b , under an unknown fixed key $K := k_{79}k_{78} \dots k_1k_0$. It is found that the key bits in $A_1 := \{k_{15}, k_{79}\}$ are neutral to $H := \varphi_{254-\beta+1, 254}$ (*i.e.*, the last β rounds) for $1 \leq \beta \leq 118$, and $A_2 := \{k_5, k_{37}, k_{69}\}$ neutral to $G := \varphi_{1,\alpha}$ (*i.e.*, the first α rounds) for $1 \leq \alpha \leq 105$. The rest of the bits $A_0 := \{k_0, k_1, \dots, k_{78}, k_{79}\} \setminus (A_1 \cup A_2)$ are used in both the forward and the backward phases. The attack to KTANTAN-32 proceeds with a text pair (P, C) .

For each guess of key bits in A_0 :

1. For each guess of A_1 , compute $M := G(P) = \varphi_{1,105}(P)$. Then 3-bit of x_{128} can be computed from M and used as an index to store value of A_1 in a table.
2. For each guess of A_2 , compute $M' := H^{-1}(C) = \varphi_{137,254}^{-1}(C)$. Compute the same 3-bit of x_{128} from M' . If the value is the index to some stored A_1 , do the *key testing*.

Key testing. Test whether $M' = \varphi_{106,136}(M)$ for K , *i.e.*, current guessed values for bits in A_0, A_1 and A_2 . If K passes, try additional pairs one by one. Continue the search if K is rejected. If K manages to survive all pairs, we conclude that it is the correct key with probability close to 1.

Data Complexity. $\lceil 80/b \rceil$ plaintext-ciphertext pairs are necessary and sufficient to identify the correct key for block size b . (*i.e.*, each pair has a filtering power of 2^{-b}). Each time the key passes the test of one pair, an additional pair is used, until the key is rejected or all text pairs exhausted - we conclude that K is the correct guess.

b	R	α	β	A_1	A_2	m	Time	Data	
32	254	105	118	15, 79	5, 37, 69	3	$2^{79.0}$ [†]	3 KP	[41]
48	251	107	112	11, 15, 75, 79	5, 69	1	$2^{79.7}$ [†]	2 KP	[41]
64	248	107	112	9, 73	5, 69	2	$2^{79.58}$ [†]	2 KP	[41]
32	254	111	122	3, 20, 41, 47, 63, 74	32, 39, 44, 61, 66, 75	12	$2^{73.88}$	3 KP	This thesis
32	254	110	122	3, 20, 41, 47, 63, 74	27, 32, 39, 44, 59, 61, 66, 75	4	$2^{73.88}$	3 KP	This thesis
32	254	109	122	3, 20, 41, 47, 63, 74	13, 27, 32, 39, 44, 59, 61, 66, 75	3	$2^{74.33}$	3 KP	This thesis
48	254	123	122	3, 20, 41, 47, 63, 74	32, 44, 61, 66, 75	37	$2^{74.53}$	2 KP	This thesis
48	254	111	121	3, 20, 41, 47, 63, 74	32, 39, 44, 61, 66, 75	4	$2^{73.97}$	2 KP	This thesis
64	254	123	122	3, 20, 41, 47, 63, 74	32, 44, 61, 66, 75	44	$2^{74.53}$	2 KP	This thesis

[†] These complexities are calculated in a slightly different manner, *e.g.*, with the method we described in Section 5.2.1, the attack complexity for KTANTAN-32 is $2^{77.6}$ encryptions.

Table 5.6: The new set of MITM attacks of KTANTAN family with corrected analysis

Time Complexity. The claimed complexity is computed as $2^{|A_0|}(2^{|A_1|} + 2^{|A_2|}) + 2^{80-3} = 2^{79}$ basic operations. Considering that a more accurate calculation will better reflect the attack at the marginal value, the number of full block encryptions that the attack works at is

$$2^{|A_0|}(2^{|A_1|} \cdot \alpha/R + 2^{|A_2|} \cdot \beta/R + 2^{|A_1|+|A_2|-m}(R - \alpha - \beta)/R) \doteq 2^{77.6}.$$

5.2.2 New Key Schedule Analysis and Results

In [41] it is shown that the way KTANTAN selects the sub-key bits is vulnerable to MITM attacks, the weakness can be exploited for the full KTANTAN-32, 251 rounds of KTANTAN-48 and 248 rounds of KTANTAN-64. Our analysis of the KTANTAN key schedule and round transformation from scratch shows that the key schedule is much weaker than reported in [41]. In particular, we find different sets of neutral key bits and different partial matching bits as well. These findings lead to the first non-marginal cryptanalysis result of the cipher family for the full rounds of all the three block sizes. The new set of neutral key bits are presented in Table 5.6, in comparison with the previous attacks. As an example, for the case of KTANTAN-32, the set of key bits neutral to $\varphi_{1,105}$ is $A_2 = \{13, 27, 32, 39, 44, 59, 61, 66, 75\}$, and $A_1 = \{3, 20, 41, 47, 63, 74\}$ neutral to $\varphi_{137,254}$, with at most 3 bits to be matched in the middle. In [41] it is given as $A_1 = \{15, 79\}$ and $A_2 = \{5, 37, 69\}$ ¹.

¹The results were presented on SAC 2011 and the article was compiled in the pre-proceeding version of the conference. The authors later updated the proceeding version and the ePrint report 2010/532 [42] for this correction too.

For an attack with m -bit matching value and b -bit block size we follow a more detailed calculation of the time complexity in number of full block encryptions (for $m > |A_1| + |A_2|$ the last term can be dropped):

$$2^{|A_0|}(2^{|A_1|} \cdot \alpha/R + 2^{|A_2|} \cdot \beta/R) + 2^{80-m} \cdot (R - \alpha - \beta)/R.$$

5.2.3 Low Complexity Implementation of the Attack

The attack works at time complexity above 2^{70} which is still out of reach for nowadays computing power. Nevertheless, due to the very low data complexity at merely the unic-ity distance, it is feasible to launch the attack with just a handful of text pairs (*e.g.*, by eavesdropping a small amount of ciphertexts that correspond to known protocol headers). Moreover, in the cases that the secret keys are not derived with full 80-bit entropy, such an attack may become a real threat when time complexity becomes 2^6 to 2^7 times less due to the attack we presented.

We implement such an attack scenario as a cryptanalysis game. Without loss of generality, we assume the attacker has already gained knowledge of part of the key bits in A_0 . The attack works on the full 254-round KTANTAN-32, with $\alpha = 111, \beta = 122, A_1 = \{3, 20, 41, 47, 63, 74\}, A_2 = \{32, 39, 44, 61, 66, 75\}$. By computing forward up to the end of round 126 and backward before round 127, there are 12 bits match. The details of the matching is given in the same notations used in [41]. *e.g.*, $k_1 = 1$ means the key bit is active (unknown) and $k_2 = 0$ means the key bit is inactive (known). An 1 in the state means the bit is affected by an active key bit, and 0 means it is not affected (*i.e.*, this state bit can be fully determined from previous state).

forward part: Neutral bits: 32 39 44 61 66 75

```
forward R=112: k1=1, k2=1, 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
forward R=113: k1=0, k2=0, 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
forward R=114: k1=0, k2=0, 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
forward R=115: k1=0, k2=0, 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
forward R=116: k1=0, k2=0, 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
forward R=117: k1=0, k2=0, 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
forward R=118: k1=0, k2=0, 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1
forward R=119: k1=0, k2=0, 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0
forward R=120: k1=0, k2=0, 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1
forward R=121: k1=0, k2=0, 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1
forward R=122: k1=0, k2=0, 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 1
forward R=123: k1=0, k2=0, 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0
forward R=124: k1=1, k2=0, 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1
forward R=125: k1=0, k2=0, 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 1
```

```

forward R=126: k1=0, k2=0,  0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 1 0 1 1 1
backward part: Neutral bits: 3 20 41 47 63 74

backward R=132: k1=0, k2=1,  0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
backward R=131: k1=1, k2=1,  1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
backward R=130: k1=0, k2=0,  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
backward R=129: k1=0, k2=0,  0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
backward R=128: k1=0, k2=0,  0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
backward R=127: k1=0, k2=0,  1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0

Bit map of 12 matched bits:  1 0 1 0 1 0 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 1 0 1 0 1 1 1 0 1 1 1

```

The game works as follows:

1. First, player A chooses an 80-bit secret key and generate 3 text pairs. Then he sends the lower 40 bits of A_0 together with the text pairs to Player B. The 80-bit secret key in bit stream order looks as: 100*1101 01001010 0101*111 01010110 *111101* 1*01*0** ***** ***** ***** ***** , where * denotes unknown bit and 0/1 the known bit. The text pairs are (0x1314B499, 0xE995A2BE), (0x39AF5959, 0xA7B2C77C), and (0xF5128319, 0x1A54138F).
2. Second, player B recovers the bits under * with the following results: A_1 : 111110, A_2 : 010100 and A_0 : 1001100111001011011100001000, where each bit from low to high order corresponds to the masked bits in A_1 , A_2 and A_0 .
3. Lastly, player A verifies the recovered bits and acknowledges the correctness.

Result. The attack in Step 2 successfully recovers the 40-bits in 5 hours 34 seconds on a Quad-core HP xw4600 workstation of 2.40GHz. The implementation makes use of the reference bit-slice code which fits well in this attack as 2^6 instances of partial transformations are required on both the forward and backward phase, *i.e.*, for $\varphi_{1,111}$ and $\varphi_{133,254}^{-1}$ respectively. On each direction, the 64 bit-sliced instances are loaded with 64 guesses of A_1 and A_2 respectively. With 40 bits known, the estimated complexity 2^{74} is reduced to around 2^{34} full encryptions. The estimation matches the experiment well since the bit-sliced reference implementation encrypts 2^{26} text blocks in roughly 45 seconds on this workstation. As a comparison, recovering all the unknown 40 bits by brute-force will take roughly half a month using the same computational resources.

Figure 5.2: A general setup for MITM attacks

5.3 From Hash to Cipher: Generalized MITM Attacks

5.3.1 Developments in Hash Preimage Attacks

In 2008, Aoki and Sasaki noticed that the MITM attacks could be applied to hash functions, to find (second) preimages faster than brute-force. The attacks have successfully broken the one-wayness of several designs in the MD4 hash family including MD5 [166], round-reduced SHA-0 and SHA-1 [8], round-reduced SHA-2 [6], and also some other Davies-Meyer hash constructions, *e.g.*, Tiger [83], reduced HAS-160 [93] and HAVAL [165]¹. Besides the basic techniques, several new techniques have been developed in these hash attacks, namely, *splice-and-cut*, *initial structure* (IS), *partial-matching* (PM), and *indirect-partial-matching*. We describe them here in detail as in Figure 5.2, and later apply to KTANTAN family to improve the results. A detailed review of MITM preimage attacks on hash functions can also be found in [83].

Splice-and-Cut To prevent (second) preimage attacks for narrow-pipe (*i.e.*, the state size equals to digest size) hash designs, feedforward is usually required. Davies-Meyer is among the most popular ways to construct a compression function from a block cipher, *i.e.*, $h' = E_m(h) \oplus h$, where $\oplus h$ is called feedforward and E is a block cipher keyed by message m . In contrast to the MITM attacks against block ciphers, *splice-and-cut* makes use of the feedforward, and split the block cipher $E(\cdot)$ into three sub-ciphers, *e.g.*, $E(h) = H_2 \circ G \circ H_1(h)$. Let $h_{inter} = H_1(h)$, we can refine the expression to $H_2^{-1}(H_1^{-1}(h_{inter}) \oplus h') = G(h_{inter})$. With the information of h and h' , one can compute the output $h \oplus h'$ of the block cipher, this is called *splice*, which connects the input and output of E . The attack starts with the intermediate point h_{inter} of the E , and *cut* the compression function into two parts: $G(\cdot)$ and $H_2^{-1}(H_1^{-1}(\cdot) \oplus h')$, then carry out the MITM attacks (note in a preimage

¹There are some more results, we only cite the best attacks on each hash function here.

attack against hash functions, h' is given as the target, hence can be treated as a constant and values of h_{inter} can be chosen freely). This enables the attacker to choose the splitting point (position of h_{inter}) wherever he wants, this relaxation makes it possible to search for neutral bits settings for better attacks.

In the case of block ciphers, such feedforward does not exist. However, one can still build a *virtual* feedforward by making use of a lookup table between plaintext and ciphertext. Instead of using $H_1^{-1}(h_{inter}) \oplus h'$ as the output of E , we use $\mathcal{L}(H_1^{-1}(h_{inter}))$, where $\mathcal{L}(\cdot)$ outputs a ciphertext given the plaintext $H_1^{-1}(h_{inter})$ under the correct key. Hence the MITM attack can be carried out from the two sides of the equation $H_2^{-1}(\mathcal{L}(H_1^{-1}(\cdot))) = G(\cdot)$. The original MITM attacks can be viewed as a special case in this generalized setting with H_1 being the identity, *i.e.*, H_1 covers 0 round with h_{inter} being the plaintext.

The lookup table $\mathcal{L}$ can be constructed by computing from a randomly chosen h_{inter} . The set of possible $H_1^{-1}(h_{inter})$ for varying keys are the chosen plaintexts to index the ciphertexts accordingly. The entire codebook is necessary when H_1 involves more secret key bits than the block size. If much less entries are used in the table, it can be further compressed to a dense table to reduce memory requirement. The data complexity of the attack is the number of entries in this table. More details are in Section 5.3.2.

Initial Structure In a hash attack, one can choose the value of h_{inter} freely. It is noted that, under certain conditions, some neighboring key bits can be swapped, and yet leave the computation result unchanged. With help from this, some steps near h_{inter} can be skipped as demonstrated in Figure 5.2, and smaller size of K_1 and K_2 can be found for the forward computation G_{K_1} and backward H_{K_2} , hence a reduced time complexity.

Partial-Matching Let $K_c := K_1 \cap K_2$, *i.e.*, the key bits used in both computing forward for M and backward for M' from the same h_{inter} . It is noted that it is not necessary to match M and M' fully at the early stage of the attacks. For $2^{|K_1|-|K_c|}$ candidates of M , and $2^{|K_2|-|K_c|}$ candidates of M' , it is expected to have $2^{|K_1|+|K_2|-2|K_c|-m}$ pairs left if only m bits are checked and matched. Then one can further check the candidates left if the rest of bits in M and M' are also matched. This additional step takes about $2^{|K_1|+|K_2|-2|K_c|-m}$ full encryptions. Actually it is much less due to the fact that the re-check needs only to

repeat the portion of PM as demonstrated in Figure 5.2. Let α denotes the percentage of steps covered by PM over that of the full cipher. The re-check will not dominate the overall complexity when

$$(|K_1| + |K_2| - 2|K_c| - m) + \log_2(\alpha) < \max(|K_1| - |K_c|, |K_2| - |K_c|).$$

Indirect-Partial-Matching. The indirect partial matching is a way to extend the partial matching for more steps. Usually partial matching starts only when key bits in $K_2 \setminus K_c$ appear after the end of G (otherwise, one can extend G for more steps). Similarly for the other side, *i.e.*, key bits in $K_1 \setminus K_c$ appear just before H_2 . If one can find several bits in the IS portion such that it can be expressed as $G(K_1) + \phi(K_2)$ from one direction, and $H(K_2) + \mu(K_1)$ from the other direction. Instead of computing the actual values and doing the match directly, one can compute $G(K_1) - \mu(K_1)$ and $H(K_2) - \phi(K_2)$ independently from two directions to check for match. The power of this technique will be demonstrated in our results in Section 5.3.2.

There are more techniques such as partial-fixing, precomputed-initial-structure, probabilistic-partial-matching developed in these hash attacks, however they are not much related to our results presented here.

5.3.2 Improving MITM Attacks on KTANTAN Family

The attacks described in Section 5.2.1 are due to a weakness in the key schedule of the KTANTAN family, especially, the efficiency of the attacks depends crucially on how many neutral key bits can be found on the forward phase and the backward phase. Hence a natural question is, is there an optimal partitioning if we are allowed to relax some requirements? We employ the *splice-and-cut* technique described in Section 5.3.1, start by randomly choosing a value for the state at a middle round and compute both forward and backward to check how many bits can be matched by the *indirect-partial-matching* technique. The core idea of this setup is almost identical with the basic form. For m -bit match, under the correct key guess, the match and re-check followed will all pass while for a random wrong guess, the match will hold with probability 2^{-m} , *i.e.*, a ratio of 2^{-m} of the overall guesses will be filtered in the partial matching phase. This effectively reduces the amount of key testing

below brute-force.

Indirect Partial Matching. We employ this technique due to an important observation on the mixing of key bits. In each round the transformation with $f_{r,a}$ and $f_{r,b}$ are repeated 1, 2 and 3 times for respective block size 32, 48 and 64. For each of the evaluations of (2.3) and (2.4), a single round key bit is mixed XOR-linearly into the LSB of L_1 or L_2 , hence affecting the lowest 1 to 3 bits considering the shift(s). In the round that follows, very few bits of the state get involved in the nonlinear terms of (2.3) and (2.4) when computing $f_{r,a}$ and $f_{r,b}$. This means after the round key bits are mixed into the state, they will remain linear for quite a number of rounds. With this observation, the *indirect-partial-matching* technique is considered to try tracing more state bits during the partial matching. These additional state bits are those affected by active key bits, but just linearly affected.

Splice-and-Cut. Let x_i be the state after round i for $1 \leq i \leq 254$ then x_{254} is the ciphertext and denote the plaintext x_0 . Let b_0 and b_1 be the round numbers of the start and the end of backward computation, and f_0 and f_1 be that of the forward computation. The rounds between (exclusive) b_0 and f_0 are used for the *initial structure*. However in the case of KTANTAN, the search result shows that this technique is not particularly relevant to a better attack, therefore we are setting f_0 to $b_0 + 1$ to allow no initial structure. The indirect partial matching is performed between two fully determined states $M := x_{f_1}$ (after round f_1) and $M' := x_{b_1-1}$ (before round b_1). The search is done with the fact in mind, that a huge $\mathcal{L}$ table may be necessary to splice the two ends of cipher together as described in Section 5.3.1. Nevertheless, the block sizes of all three versions are less than the key size, in the worst case, the key recovery attack will have to use the entire codebook. Note that the attack is still strictly better than owning the codebook since it is able to recover the secret key.

From the search result shown in Table 5.7, in the optimal partition of KTANTAN-32 and KTANTAN-64 as well as a near-optimal partition of KTANTAN-48, the backward phase and the forward phase are split before round 254. In total only 4 ciphertexts have to be computed from the chosen state x_{253} as $\varphi_{254,254}(x_{253})$ with all possible values of the round key $k_{71}k_7$. Hence the data complexity is at 4 chosen ciphertexts. For each iteration of

Figure 5.3: Illustration of MITM attack with splice-and-cut

the attack, we refer the active entry of $\mathcal{L}$ the *active pair*, and other entries in the table as *non-active* pairs. During the key testing, besides the reuse of the active pair, one or two additional non-active pairs can be used to reject false positives. In the optimal partition of KTANTAN-48 that gives the least time complexity, the entire codebook is necessary since the chosen state x_{107} is in the middle of the cipher. In Figure 5.3 below we illustrate the cases in which the splitting point is at the end of the cipher. With notations of previous sections, $G := \varphi_{1,f_1} \circ \mathcal{L} \circ \varphi_{f_0,254}$ and $H := \varphi_{b_1,b_0}$.

The Attack with Splice-and-Cut and Indirect-Partial-Matching. The first step is to construct the table $\mathcal{L}$ as to splice the two ends of cipher. It is done with the chosen ciphertext (C, P) pairs, or the entire codebook. For the attack illustrated by Figure 5.3, the splitting point is at the end, we select a random value for x_{b_0} and compute $C := G_1(x_{b_0})$ for all the possible values of key bits involved in $G_1 := \varphi_{f_0,254}$. For each C find the corresponding plaintext P and add (C, P) to the table $\mathcal{L}$ with C as the index and P as the value. If the splitting point is in a middle round and we need the entire codebook, $\mathcal{L}$ is constructed by adding all the (C, P) pairs when $f_1 < f_0$ or all the (P, C) pairs when $b_0 < b_1$. We do not discuss this scenario since it is almost identical to the algorithm for the illustrated attack. Following the notations we use A_1 for the key bits neutral to the backward transformation $H^{-1} := \varphi_{b_1,b_0}^{-1}$ and A_2 for the key bits neutral to forward $G := G_2 \circ \mathcal{L} \circ G_1 = \varphi_{1,f_1} \circ \mathcal{L} \circ \varphi_{f_0,254}$.

Attack Algorithm. For each guess of key bits in A_0 :

1. For each guess of A_1 , compute $M := G(x_{b_0})$. Then, m -bit *partial matching signature* can be computed from M and used as an index to store value of A_1 in a table.
2. For each guess of A_2 , compute $M' := H^{-1}(x_{b_0})$. Then compute the same m -bit partial matching signature from M' . If the value is the index to some stored A_1 , do the *key testing*.

b	R	b_1	b_0	f_0	f_1	A_1	A_2	m	Time	Data
32	254	148	253	254	109	13, 27, 32, 39, 44, 59, 61, 66, 75	3, 20, 41, 47, 63, 74	11	$2^{72.93}$	4 CC This thesis
48	254	150	253	254	111	32, 39, 44, 61, 66, 75	3, 20, 41, 47, 63, 74	15	$2^{73.77}$	4 CC This thesis
64	254	151	253	254	112	32, 44, 61, 66, 75	3, 20, 41, 47, 63, 74	54	$2^{74.38}$	4 CC This thesis

Table 5.7: MITM attack with splice-and-cut and indirect-partial-matching

Indirect Partial Matching. The m -bit partial matching signature are m bits values at the position of the matched bits, computed as described in Section 5.3.1 on indirect partial matching. Searching of such signature requires to find state bits either independent of the active key bits in the IPM phase or dependent but only linearly affected. The technique significantly improved the number of bits that can be matched. This allows us to have more rounds for partial matching hence possibly more neutral bits. The detailed matching of the current optimal attack to KTANTAN-32 can be found in Appendix B.3. To compute the matching signature, we take KTANTAN-32 for example, the matching point in the optimal attack shown above is after round 115. To simplify the notations, let $x := x_{115}$ and $x[i]$ be the i -th bit of x , for $0 \leq i \leq 31$. For the forward part of the indirect partial matching, it is not hard to find the following key bits appear only as linear terms in corresponding state bits, k_{27} in $x[0]$, k_{13} in $x[1]$, k_{39} in $x[3]$, k_{59} in $x[4]$ and k_{39} in $x[22]$. For the backward part, similarly we have linear terms of k_{74} in $x[26]$, k_{74} in $x[21]$, k_{74} in $x[3]$ and k_{20} in $x[2]$. Hence the matching signature is $(x[26] - k_{74}, x[22] - k_{39}, x[21] - k_{74}, x[7], x[6], x[5], x[4] - k_{59}, x[3] - k_{39} - k_{74}, x[2] - k_{20}, x[1] - k_{13}, x[0] - k_{27})$ which can be computed from both sides without knowing the value for the active key bits from that side.

Key testing. Test whether $M' = \varphi_{f_1+1, b_1-1}(M)$ for K , *i.e.*, current guessed values for bits in A_0, A_1 and A_2 . If K passes, try non-active pairs. Continue the search if K is rejected. If K manages to survive a number of pairs at the unicity distance, we conclude that it is the correct key with probability close to 1.

Table 5.7 shows the result of search that leads to better complexities.

5.3.3 Results and Conclusions

We presented new experimental observations that lead to improved cryptanalytic results for the KTANTAN family. In addition, we have implemented the attack to KTANTAN-32 in a low complexity setting. The result confirms the validity of such attacks and the reduction on

time complexity. Moreover, the techniques developed from cryptanalysis of hash functions in the MITM framework are shown to be effective on analysis of block ciphers as well, despite the presence of a secret key, as we obtained the best cryptanalysis on KTANTAN family. In particular, the *splice-and-cut* technique allows searching of optimal partitioning on neutral bits and matching. In conjunction with *indirect-partial-matching*, significantly more bits can be matched in our attacks, this helps to have more rounds covered by partial matching hence it becomes possible to have more neutral bits to improve the time complexity. Other techniques like *precomputed-initial-structure* do not contribute to our attack, but we expected them to be effective in analysis of other block ciphers in the MITM framework. It is yet open to examine the real potential of MITM attacks in general on block cipher cryptanalysis if more powerful enhancements can be discovered. More dedicated matching techniques might be possible for particular block ciphers.

Chapter 6

Cryptanalysis of IDEA in Hashing Modes

In this chapter, we study the security of the IDEA block cipher when it is used in various simple-length or double-length hashing modes. Even though this cipher is still considered as unbroken, we show that one should avoid its use as internal primitive for block cipher based hashing. In particular, we are able to generate instantaneously free-start collisions for most modes, and even semi-free-start collisions, pseudo-preimages or hash collisions in practical complexity. This work shows a practical example of the gap that exists between secret-key and known- or chosen-key security for block ciphers. Moreover, we also settle the 20-year-old standing open question concerning the security of the Abreast-DM and Tandem-DM double-length compression functions, originally invented to be instantiated with IDEA. Our attacks have been verified experimentally and work even for strengthened versions of IDEA with any number of rounds.

6.1 Introduction and Background

A potential candidate for hashing is the 64-bit block cipher IDEA [119, 187] that uses 128-bit keys. While a simple-length hashing mode would only provide a 64-bit hash output, insufficient for most of nowadays security applications, a double-block-length construction (DBL) would allow 128-bit hash outputs which can be sufficient in some scenarios. As IDEA handles double-length keys, more freedom in the constructions is possible. In fact, the well

known Abreast-DM and Tandem-DM modes were specifically created to perform hashing with IDEA [120, 187]. These modes were later studied in much details [76, 77, 124, 126], but the security they provide when instantiated with IDEA remains a 20-year-old standing open question. In classical “secret key” setting, IDEA has already been studied a lot [11, 28, 32, 29, 36, 47, 55, 62, 86] and is still considered as unbroken despite its age. The current best attack [29] requires 2^{63} data (half the codebook) and 2^{114} computations to recover the secret key for IDEA reduced to 7.5 rounds over a total of 8.5. One can also cite the work of [36], that exposes a weak-key class of size 2^{64} . Note also that a first step towards analysis of IDEA in hashing mode was done in [47], where the authors show how to find a free-start near collision (only a subset of the output collides) when IDEA is plugged into the Hirose DBL mode [47] (and also a free-start collision if the internal constant c is controlled by the attacker).

In what follows, we study the security of the IDEA block cipher [119, 187] when plugged into various block cipher based compression function constructions, such as the classical Davies-Meyer mode [2], also DBL constructions such as Hirose [90, 91], Abreast-DM and Tandem-DM [120, 187], Peyrin *et al.* (II) [155] or MJH-Double [125]. Even if this cipher is still considered as unbroken in the classical “secret key” setting, its security remains an open problem in hashing mode. Depending on the IDEA-based hash construction, we show that an attacker can find free-start collisions instantaneously, preimages or semi-free-start collisions practically. For some modes, we even describe a method to compute collisions for the whole hash function. These attacks are based on weak-keys utilization, but in contrary to the “secret key” setting where the goal of the attacker is to exhibit the biggest weak-key class possible, in hashing mode the goal is to find and exploit the weakest of all keys. We show that the key 0 in IDEA is extremely weak, actually rendering the whole encryption process a T-function [105], already known as dangerous for building a hash function [142]. Even strengthened versions of the cipher with any number of rounds can be attacked with about the same complexities. This work indicates that one has to be very careful when hashing with a block cipher that presents any weakness when the key is known or controlled by the attacker. In particular, one should strictly avoid the use of a block cipher for which weak-keys exist, even if only a single weak-key is known.

6.2 IDEA Weak-Keys in Hashing Modes

Weak-keys for IDEA has already been studied in details [36, 55, 86], but what we are looking for is slightly different. Indeed, for block cipher cryptanalysis, since the attacker can not control the key input he looks for the biggest possible class of weak-keys, so as to get the highest possible probability that a weak-key will indeed be chosen. In the case of compression function cryptanalysis, the key input is fully known or even controlled by the attacker. The goal is therefore not to find the biggest possible class of weak-keys, but to find the weakest possible key. As we will show for IDEA, even if only one weak-key is found, its weakness might directly lead to successful attacks on the whole compression or hash function.

6.2.1 Analysis of IDEA Internal Operations

When looking at the internal round function of IDEA, one might wonder what would be a weak-key. In IDEA, the most annoying functions for the cryptanalyst are clearly the multiplications in $\mathbb{Z}_{2^{16}+1}$. Indeed, these operations are strongly non-linear and provide good diffusion between the different bit positions. On the contrary, XOR operations are linear and do not provide any diffusion between the bit positions, while the additions in $\mathbb{Z}_{2^{16}}$ can be easily approximated linearly and the diffusion between the bit positions only happens through the carry. Moreover, XOR and additions are even weaker in IDEA since no rotations are present, comparing with Addition-Rotation-XOR (ARX) designs. Here the rotation is done through the multiplications in $\mathbb{Z}_{2^{16}+1}$ and our goal is therefore to avoid them.

When adding $(a + b) \bmod 2^{16}$, we can avoid any diffusion by forcing one operand to 0. When multiplying $(a \odot b) = (a \cdot b) \bmod 2^{16} + 1$, the good diffusion will happen especially when $(a \cdot b) \geq 2^{16} + 1$. An easy way to avoid this is to fix one of the two operands to 1. In that case, we have $(a \odot 1) = (a \cdot 1) \bmod 2^{16} + 1 = a \bmod 2^{16}$. As already remarked in [55],

a good choice is also 0, since

$$\begin{aligned}
(a \odot 0) \bmod 2^{16} &= ((a \cdot 2^{16}) \bmod (2^{16} + 1)) \bmod 2^{16} \\
&= (((a \cdot 2^{16} + a) + (2^{16} + 1) - a) \bmod (2^{16} + 1)) \bmod 2^{16} \\
&= (0 + 2^{16} + 1 - a) \bmod 2^{16} = 1 - a \bmod 2^{16} \\
&= 2 + (2^{16} - 1 - a) \bmod 2^{16} = (2 + \bar{a}) \bmod 2^{16}
\end{aligned}$$

and the multiplication is reduced to only a complement and an addition with a constant.

Weak-Key Classes Based on the remark that the operand 0 is very weak for both multiplications and additions, Daemen *et al.* [55] generated a class of weak-keys. A first obvious candidate is the null key (all bits set to zero), which will force all the sub-keys to zero as well. As a consequence, all sub-keys additions can be simply removed and all sub-keys multiplications can be replaced by a complement (or XOR with 0xffff) and an addition with value 2. At this point, all the operations in IDEA with null key are either XOR or additions. Therefore, by inserting differences only on the Most Significant Bit (MSB) of the four 16-bit plaintext input words, the attacker is ensured that only the MSB of the four output words will contain a difference. Even better, the mapping from an MSB input difference pattern to an MSB output difference pattern is completely deterministic (is it linear since on the MSB no carry is propagated). Such a property is largely sufficient to consider the null key as weak. This reasoning can be generalized by observing that the attacker does not necessarily need all sub-keys to be null, but only the ones that are multiplied to an internal word which contains a MSB difference. Since the MSB differential paths are quite sparse, many of the null constraints on the sub-keys are relaxed and one finally gets 2^{35} weak-keys.

6.2.2 An Almost Half-Involution Property

We show that the null key is particularly weak for hash function utilization. Even if other keys belong to a weak-key class, they do not present the same special properties as the null key.

Almost Half-Involution When using the null key, we remark that all sub-keys will be null as well. Then, all rounds layers will be the same and we write KA_0 and MA_0 the KA and MA layers with null sub-keys. A nice practical feature of **IDEA** is that the decryption is done using the very same algorithm as encryption, but with different sub-keys. The decryption sub-keys for the MA layer are the same as the encryption ones since the MA layer is an involution (i.e. $MA=MA^{-1}$). The decryption sub-keys for the KA layer are the respective multiplicative and additive inverses of the encryption sub-keys. However, note that a null sub-key is both its own multiplicative and additive inverse and the KA layer becomes an involution as well (i.e. $KA_0=KA_0^{-1}$). To summarize, using the null key, we are ensured that $KA_0=KA_0^{-1}$ and $MA_0=MA_0^{-1}$. Note that we trivially have $S=S^{-1}$.

Now, since the KA layer and S layer commute, **IDEA** with null key can be rewritten as

$$\begin{aligned}
C &= KA_0 \circ S \circ \{S \circ MA_0 \circ KA_0\}^8(P) \\
&= KA_0 \circ S \circ \{S \circ MA_0 \circ KA_0\}^3 \circ S \circ MA_0 \circ KA_0 \circ \{S \circ MA_0 \circ KA_0\}^4(P) \\
&= \underbrace{KA_0 \circ MA_0 \circ \{S \circ KA_0 \circ MA_0\}^3}_{\sigma^{-1}} \circ \underbrace{KA_0 \circ S}_{\theta} \circ \underbrace{\{MA_0 \circ KA_0 \circ S\}^3 \circ MA_0 \circ KA_0}_{\sigma}(P)
\end{aligned}$$

which eventually gives $C = \sigma^{-1} \circ \theta \circ \sigma(P)$. One can check that since KA_0 , MA_0 and S are involutions, the operation denoted by σ^{-1} is indeed the inverse of the one denoted by σ . Thus, using the notation

$$P \xrightarrow{\sigma^{-1}} U \xrightarrow{\theta} V \xrightarrow{\sigma} C$$

where U and V are internal state values, we have

$$P \xleftarrow{\sigma} U \xrightarrow{\theta} V \xrightarrow{\sigma} C.$$

We will use this almost half-involution property in Section 6.4 to find free-start collisions and even hash function collisions for some **IDEA**-based constructions.

T-function: When using the null key, we have already described that all operations remaining are either XOR or additions. These operations are triangular functions [105] (or T-functions) in the sense that any output bit at position i only depends on the input bits located at a position i or lower. A composition of T-functions is itself a T-function,

therefore the whole permutation defined by IDEA with the null key is a T-function. As shown in [142], this property might be very dangerous in a hash function design. We will explain in Section 6.5 how to exploit this weakness and compute preimages by guessing the input words bit layer by bit layer.

6.3 Simple Collision Attacks

As shown by Daemen *et al.* [55], when using the null key for the encryption process of IDEA, differences inserted uniquely on the MSB of the four 16-bit input plaintext words will lead to differences on the MSB of the four 16-bit output ciphertext words. Moreover, since this difference mapping is linear (the difference on the carry is not propagated further than the MSB), all possible differential characteristics have a differential probability 1. For example, we denote by $\delta_{MSB} = 0x8000$ the 16-bit word with difference only on the MSB and by $\Delta_{MSB} = (\delta_{MSB}, \delta_{MSB}, \delta_{MSB}, \delta_{MSB})$ the 64-bit difference composed of 4 words with difference δ_{MSB} . Then, Δ_{MSB} propagates to itself with probability 1 through one round of IDEA, or through its last half-round. Therefore, we have with probability 1

$$\Delta_{MSB} \xrightarrow{\text{IDEA}_{K=0}} \Delta_{MSB}.$$

Note that instead of using δ_{MSB} only, one can generalize the input difference space and obtain other very good differential paths for the encryption of IDEA with the null key. However, we omit this generalization here since the methods described in later sections already provide much better attacks.

6.3.1 Davies-Meyer and Hirose Modes

The two modes can be treated similarly since the null key property can be mounted easily.

Davies-Meyer Mode Finding a free-start collision on Davies-Meyer mode instantiated with IDEA is simple. Since the difference Δ_{MSB} is mapped to itself through the IDEA encryption process with the null key, the attacker only has to pick $M = 0$. Then, any value of CV with difference Δ_{MSB} applied to it will lead to a collision with probability 1. We give here example of such a free-start collision.

$$\begin{aligned}
CV_i &: 0x9efc \ 0x14ef \ 0x85d6 \ 0xc557 \\
CV'_i &: 0x1efc \ 0x94ef \ 0x05d6 \ 0x4557 \\
M = M' &: 0 \\
CV_{i+1} = H(CV_i, M) &: 0x7f11 \ 0x83f1 \ 0x7617 \ 0x8af3 \\
CV'_{i+1} = H(CV'_i, M') &: 0x7f11 \ 0x83f1 \ 0x7617 \ 0x8af3
\end{aligned}$$

Hirose The same method as for Davies-Meyer mode can be applied to the Hirose mode in order to find free-start collisions. The attacker fixes $CV2 = 0$ and $M = 0$ so as to force the null key to both encryptions. Then, any value of $CV1$ with a difference Δ_{MSB} applied to it will lead to a collision with probability 1, since Δ_{MSB} will appear on the plaintext input of both encryptions with the null key.

We give here example of such a free-start collision. For Hirose mode, we used as constant c the first 64 output bits of the SHA-2 computation of the string “IDEA”.

$$\begin{aligned}
\text{SHA-2(“IDEA”)} &: 9f8c \ 7b26 \ cde5 \ 9ca3 \ dacc \ 74ec \ 7afd \ a737 \ \dots \\
c &: 0x9f8c \ 0x7b26 \ 0xcde5 \ 0x9ca3 \\
CV1_i &: 0x93e8 \ 0x4d86 \ 0x45a5 \ 0xa829 \\
CV1'_i &: 0x13e8 \ 0xcd86 \ 0xc5a5 \ 0x2829 \\
CV2_i = CV2'_i &: 0 \\
M = M' &: 0 \\
CV1_{i+1} &: 0x2101 \ 0x23c9 \ 0xde42 \ 0xdc96 \\
CV1'_{i+1} &: 0x2101 \ 0x23c9 \ 0xde42 \ 0xdc96 \\
CV2_{i+1} &: 0x0009 \ 0x0401 \ 0x3d38 \ 0x3934 \\
CV2'_{i+1} &: 0x0009 \ 0x0401 \ 0x3d38 \ 0x3934
\end{aligned}$$

6.3.2 Peyrin *et al.*(II)

We have to separate in two groups the possible instances of this construction, obtained by permuting the position of the three inputs of each internal function f_i . If all compression function inputs $CV1$, $CV2$, $M1$ and $M2$ appear in at least one of the IDEA key inputs of any f_i internal function, then the attack will not apply. Indeed, since all inputs will be involved at least one time, the attacker will necessarily have to insert a difference in at least one IDEA key input and he will not be able to use the differential path with probability 1. Note that these instances would be avoided in practice because they would lead to more

frequent rekeying and therefore reduce the overall performance of the hash function. If this condition is not met, then we can apply the following free-start collision attack. Let $X \in \{CV1, CV2, M1, M2\}$ denote the input that is missing in all the IDEA key inputs of the compression function. The attacker simply fixes the difference Δ_{MSB} on X (one can give any value to X) and all other inputs are set to 0 in order to get the null key in every internal IDEA. The attacker ends up with several Davies-Meyer in parallel, with either no difference at all or with null key and Δ_{MSB} as plaintext input difference. Thus, he obtains a collision with probability 1. If $X \notin \{CV1, CV2\}$, then this attack finds semi-free-start collisions.

6.3.3 Abreast-DM, Tandem-DM and MJH-Double

This technique seems impossible to apply to the Abreast-DM mode since forcing a difference Δ_{MSB} on any of the two encryptions plaintext input will imply a difference inserted in the key input of the other encryption block. Therefore, one cannot use Δ_{MSB} difference on plaintext input with null key in both encryption blocks. Even if the attacker tries to attack only one encryption block with this method, the other block will not be controlled and he will have to deal with random differences on its output. These random differences cannot be dealt with some birthday technique because fixing all inputs of one encryption block will fix all inputs of the other one as well. This technique seems impossible to apply to the Tandem-DM mode for the exact same reasons as for Abreast-DM.

The MJH-Double mode prevents this simple attack since even if we fix $CV2 = 0$ and $M2 = 0$ in order to get the null key in both encryptions, it is hard to force the difference Δ_{MSB} on both their plaintext inputs. Indeed, the f operation will randomize the difference and in order for the attack to run, we would require $\Delta_{MSB} \xrightarrow{f} \Delta_{MSB}$ which is unlikely to happen.

6.4 Improved Collision Attacks

In this section, using the almost half-involution property with the null key, we will show how to get the same difference on the input and on the output of the IDEA ciphering process with good probability. Then, we will use this weakness to derive our collision attacks, for

any number of rounds.

6.4.1 Exploiting the Almost Half-Involution

We have already shown in Section 6.2 that when the key is null, IDEA encryption process can be rewritten as

$$P \xleftarrow{\sigma} U \xrightarrow{\theta} V \xrightarrow{\sigma} C$$

where

$$\sigma = \{\text{MA}_0 \circ \text{KA}_0 \circ \text{S}\}^3 \circ \text{MA}_0 \circ \text{KA}_0 \quad \text{and} \quad \theta = \text{KA}_0 \circ \text{S}.$$

We denote ΔU the XOR difference between two 64-bit internal state values U and U' , i.e. $\Delta U = U \oplus U'$, and δU_i represents the 16-bit difference on the i -th word of ΔU , that is $\Delta U = (\delta U_1, \delta U_2, \delta U_3, \delta U_4)$. Let us consider two random 64-bit internal state values U and U' such that $\delta U_2 = \delta U_3$ and we denote this 16-bit difference δ_M . For truly random values U and U' , this condition happens with probability 2^{-16} . One can check that applying θ on U and U' to obtain V and V' respectively will lead to $\delta V_2 = \delta V_3 = \delta_M$ since layer S only switches the two middle words and layer KA_0 has no effect on them (addition of null sub-keys).

Let δ_L and δ_R represent the difference on δU_1 and δU_4 respectively, i.e. $\Delta U = (\delta_L, \delta_M, \delta_M, \delta_R)$. Applying function θ to U and U' , we would like the same differences to appear on internal state V and V' : $\Delta V = (\delta_L, \delta_M, \delta_M, \delta_R)$. The previous condition with probability 2^{-16} already ensures the two middle differences being the same δ_M . Concerning differences δ_L and δ_R , they will both be unaffected by layer S, but they might be modified through layer KA_0 that applies a multiplication with a null sub-key. Therefore, we need to study the probability that a random difference δ is mapped to itself through a multiplication by the null sub-key. We show below in Theorem 6.4.1 that this probability is equal to $2^{-1.585}$ and finally we have $\Pr[(\delta_L, \delta_M, \delta_M, \delta_R) \xrightarrow{\theta} (\delta_L, \delta_M, \delta_M, \delta_R)] = 2^{-3.17}$.

Theorem 6.4.1. (*Difference preservation through $\odot 0$*) For randomly chosen values a and a' with $a \oplus a' = \delta$, the probability that the difference δ is preserved after multiplication by the null sub-key is equal to $2^{-1.585}$.

Proof. The condition we expect can be translated into the following equation

$$\delta = a \oplus a' = (a \odot 0) \oplus (a' \odot 0).$$

Since the $\odot$ operation is equivalent to a complement (or XOR with `0xffff`) and an addition with value 2, we can rewrite

$$\begin{aligned} \delta &= ((a \oplus 0xffff) + 2) \oplus ((a' \oplus 0xffff) + 2) \\ \delta &= ((a \oplus 0xffff) + 2) \oplus ((a \oplus \delta \oplus 0xffff) + 2) \\ \delta &= (b + 2) \oplus ((b \oplus \delta) + 2) \\ \delta \oplus (b + 2) &= (b \oplus \delta) + 2 \end{aligned}$$

where $b = a \oplus 0xffff$. One can check that the least significant bit condition of this equation is always fulfilled.

If the second least significant bit of b is 0 (probability $1/2$), then $(b + 2) = b \oplus 2$ and the equation is fulfilled if and only if the second least significant bit of $(b \oplus \delta)$ is also 0 (probability $1/2$). Overall, this situation happens with probability $1/4$.

If the second least significant bit of b is 1 (probability $1/2$), then we will have a carry propagating and we require the second least significant bit of $(b \oplus \delta)$ to be also 1 (probability $1/2$). If the third least significant bit of b is 0 (probability $1/2$), then $(b + 2) = b \oplus 6$ and the equation is fulfilled if and only if the third least significant bit of $(b \oplus \delta)$ is also 0 (probability $1/2$). Overall, this situation happens with probability $(1/4)^2$.

Continuing this reasoning over all the bits layers, we obtain that the success probability is equal to

$$\sum_{i=1}^{14} (1/4)^i = 2^{-1.585}.$$

□

At this point, we proved that for randomly chosen internal state values U and U' , we will observe with probability $2^{-19.17}$ the same difference on U and V , i.e. $\Delta U = \Delta V$.

One can see that computing backward from internal states U to P or forward from V to C , the function σ is applied. Our final goal is to have the same difference on P and C .

However, this seems unlikely to happen since U and V have different values, the forward and backward computations of σ should be completely unrelated, even with the same input difference. Yet, this reasoning does not take in account the fact that while U and V have distinct values, they are far from being independent: $V = \theta(U)$ with θ being a very light function. Moreover, we remarked that almost each time that we got the same difference on P and C , the same differences were observed as well in all rounds of the forward and backward σ computations (the round success probability increasing with the number of rounds already processed). Because all the rounds are not independent and because U and V are strongly related, it is very difficult to compute theoretically the probability of observing the same difference on P and C . Therefore, we measured it by choosing random values of U , δ_L , δ_M , δ_R , computing $V = \theta(U)$, and checking for collisions on the difference of P and C . The probability obtained was $2^{-16.26}$ for about 2^{28} tests (note that this probability somehow contains the $2^{-3.17}$ probability computed previously, but we can not separate them because the two events are not independent).

To conclude, the probability that two randomly chosen internal state values U and U' give the same difference on P and C is equal to $2^{-16-16.26} = 2^{-32.26}$ (instead of 2^{-64} expected for a random function). In other words, using the birthday paradox, one can find such a pair with about $2^{16.13}$ computations.

Interestingly, we have observed that most of the pairs fulfilling the differential path for the full IDEA will also be valid for a strengthened version of the cipher with any number of additional rounds. Since the sub-keys are always null, strengthening the cipher would mean that $\sigma = \{\text{MA}_0 \circ \text{KA}_0 \circ \text{S}\}^t \circ \text{MA}_0 \circ \text{KA}_0$ for any $t > 3$. We checked that the probability that two randomly chosen internal state values U and U' give the same difference on P and C tends to $2^{-32.54}$ when t tends to infinite. Thus, similarly to the method presented in the previous section, the attacks using this almost half-involution property will work for any number of rounds.

6.4.2 Practical Hash Collision for Davies-Meyer

A first obvious application of having the same difference in P and C is collision search on Davies-Mayer mode, where the feed-forward will cancel the two differences in the output.

The attack find collisions for the whole hash function and the procedure is very simple: we start from the IV and add random differences in the first message block M_0 . This will cause random differences in the the first chaining variable CV_1 . For the second message block M_1 , we will set all its bits 0 ($M_1 = 0$), forcing the internal IDEA computation to use the null key. Since we proved in the previous section that with the null key a random pair of inputs has a probability $2^{-32.26}$ to give the same input/output difference, one can use the birthday paradox to generate a collision on CV_2 with only $2^{16.13}$ distinct message blocks M_0 . We give here example of hash collisions for the Davies-Meyer mode. Note that finding semi-free-start collisions with this technique is impossible since we would have to insert differences in the message input, which forbids the use of the null key in the internal cipher.

We use as initial value the first 64 output bits of the SHA-2 computation of the string “IDEA”:

```

SHA-2(“IDEA”) : 9f8c 7b26 cde5 9ca3 dacc 74ec 7afd a737 ...
IV1, IV2, IV3, IV4 : 0x9f8c, 0x7b26, 0xcde5, 0x9ca3
M1 : 0xdacc, 0xdacc, 0xdacc, 0xdacc
      : 0xdacc, 0xdacc, 0xcadc, 0x0282
M'1 : 0xdacc, 0xdacc, 0xdacc, 0xdacc
      : 0xdacc, 0xdacc, 0xcade, 0x1a3f
CV1 = H(IV, M1) : 0xb782, 0x4583, 0x83b6, 0x0bef
CV'1 = H(IV, M'1) : 0x1ce2, 0x8553, 0xe656, 0x4387
CV2 = H(CV1, 0) : 0xdffd, 0x3ffd, 0x8e7d, 0x6e7d
CV'2 = H(CV'1, 0) : 0xdffd, 0x3ffd, 0x8e7d, 0x6e7d

```

6.4.3 Collision Attacks on Abreast-DM and Tandem-DM

Abreast-DM. One can derive a free-start collision attack for the Abreast-DM compression function using this technique. The attacker first fixes $CV_1 = 0$ and $M = 0$. Then, he builds a set of $2^{48.13}$ distinct values CV_2 and checks if a pair of this set leads to a collision. The probability that a pair leads to a collision on the first (top) branch is $2^{-32.26}$ (since the internal cipher on this part has the null key), and 2^{-64} on the other half. Overall, using the birthday paradox on the set of $2^{48.13}$ values CV_2 is sufficient to have a good chance to obtain a collision. Note that finding a semi-free-start collision for the compression function or a collision for the hash function seems impossible with this method, for the same reasons

as the Hirose mode.

Tandem-DM. The situation of Tandem-DM is identical to the Abreast-DM one: one can find free-start collisions for compression function using this technique. The attacker first fixes $CV1 = 0$ and $M = 0$. Then, he builds a set of $2^{48.13}$ distinct values $CV2$ and checks if a pair of this set leads to a collision. The probability that a pair leads to a collision on the first (top) branch is $2^{-32.26}$ (since the internal cipher on this part has the null key), and 2^{-64} on the other half. Overall, using the birthday paradox on the set of $2^{48.13}$ values $CV2$ is sufficient to have a good chance to obtain a collision. Again, finding a semi-free-start collision for the compression function or a collision for the hash function seems impossible with this method, for the same reasons as the Hirose mode.

A Note on Hirose. We already showed how to find free-start collisions for the Hirose mode. However, finding semi-free-start collisions with this technique is impossible since we would have to insert differences in the message input, which forbids the use of the null key in the internal cipher. Also, concerning hash collisions, it seems hard as well because forcing the null key during iteration i requires us to obtain a chaining variable $CV2_{i-1} = 0$ during the previous iteration. This half-preimage already costs the same complexity as a generic collision search on the entire compression function.

6.4.4 Collision Attack on Peyrin *et al.*(II)

We showed in previous section how to find (semi)-free-start collisions with probability 1 for a certain subset of Peyrin *et al.*(II) constructions, but here we provide attacks on a bigger subset. If all compression function inputs $CV1$, $CV2$, $M1$ and $M2$ appear in at least one of the IDEA key inputs of f_1 , f_2 , f_3 (left side) and in at least one of the IDEA key inputs of f_3 , f_4 , f_5 (right side), then the attack will not apply. Indeed, for both left side and right side of the compression function, the attacker will necessarily have to insert a difference in at least one key input (since all inputs will be involved) and he will not be able to use the null key completely. Note that these instances would be avoided in practice because they would lead to more frequent rekeying and therefore reduce the overall performance of the hash function. However, if this condition is not met, then we can apply the following free-start

collision attack. Let $X \in \{CV1, CV2, M1, M2\}$ denote the input that is missing in all the IDEA key inputs of f_1, f_2, f_3 (wlog the reasoning is the same with f_3, f_4, f_5). The attacker first fixes all inputs but X to 0 in order to get the null key in every internal IDEA on the left side. Then he chooses $2^{48.13}$ random values for X and checks among them if any pair collides on the whole compression function output. Since he has a probability $2^{-32.26}$ to get a collision on the left side and 2^{-64} on the right side, using a birthday search the attacker finds a solution with complexity $2^{48.13}$. Again, if $X \notin \{CV1, CV2\}$, then this attack finds semi-free-start collisions. However, finding a collision for the hash function seems impossible with this method, because at least one of the chaining variable inputs $CV1$ and $CV2$ will be present as key input for one of the IDEA internal encryption. Setting this word to 0 is equivalent to a half-preimage that already costs the same complexity as a generic collision search on the entire hash function.

6.4.5 Collision Attack on MJH-Double

One can derive a semi-free-start collision attack on the MJH-Double compression function instantiated with IDEA. The attacker first fixes $CV2 = 0$ and $M2 = 0$ and this will force the null key in both encryptions. Now he chooses a random value for $CV1$ (note that actually this value could be fixed by the challenger) and builds a set of $2^{32.26}$ values $M1$. In this configuration, it is easy to see that one will have random differences on the plaintext inputs to both encryptions. Since the null key is used for both, we have a probability $2^{-64.52}$ that a pair of $M1$ leads to a collision after the feed-forward of both encryptions (on the output of the bottom block and just before the application of g on the top block). Therefore, with a birthday technique, one can find such a pair with only $2^{32.26}$ computations. Note that while this pair will directly lead to a collision on the bottom $CV1$ output, the difference on $M1$ is injected two times before computing the top $CV2$ output. Two times of the same difference will cancel themselves and we eventually get a full semi-free-start collision. Note that it seems hard to extend this attack to a hash collision since the attacker would require to force the incoming chaining variable $CV2$ to be equal to 0 and this half-preimage already costs the same complexity as a generic collision search on the entire hash function.

6.5 Preimage Attacks with Practical Complexities

We showed in Section 6.2 that if used with the null key, the whole permutation defined by IDEA is a T-function. Since any output bit at position i only depends on the input bits located at a position i or lower, we reuse the idea of preimage attack for hash functions based on T-functions [142] where the preimage is computed bit layer by bit layer, starting from the LSB. However, here our situation is different than the functions studied in [142] since we do not have any truncation or reduction of the internal state at the end of the process.

We denote by p the probability that given a random challenge, our algorithm outputs a preimage for this challenge. We denote by s the average number of preimage solutions that the algorithm will output, given that at least one is found. The average number of solutions outputted by our algorithm is then $A = s \cdot p$. For an n -bit ideal compression function, a generic attack restricted to C computations can generate $A = C \cdot 2^{-n}$ solutions on average. Thus, we can consider that a preimage attack is found if we exhibit an algorithm that outperforms this generic complexity.

Davies-Meyer. Since the key is fixed to 0 and since the plaintext and ciphertext sizes are the same, we trivially have that $A = 1$. We measured¹ that $p = 2^{-17.50}$, thus we directly deduce that $s = A/p = 2^{17.5}$. A straightforward implementation is a recursive depth first search, attacking the T-function by bit layer from the LSB to the MSB of the 16-bit state words. Wrong candidates at lower layers are discarded thanks to an early-abort strategy. On average, the amount of IDEA encryptions required to find all the possible preimages (if at least one can be found) can be estimated as $C \simeq 16 \cdot 2^4 \cdot s = 2^{25.5}$, since we have 16 bit layers, each having 4 bits of input, and on average the number of candidates in one layer is s . This is a very conservative estimation since only $p = 2^{-17.50}$ of the challenges on average will eventually lead to a solution and the early-abort strategy will make the actual search of very low complexity. In the ideal case, with $C = 2^{25.5}$ computations allowed, an attacker should only be able to generate $A = 2^{25.5-64} = 2^{-38.5}$ solutions on average for an ideal 64-bit compression function. We give an example of a preimage here.

Since a random 64-bit challenge has preimage(s) with a probability p , we show the

¹from 2^{31} random challenges, we measured that $p = 2^{-17.50}$ and $s = 2^{17.74}$.

preimage of a challenge which we are sure at least one preimage exists (similar to a second-preimage search). In order to get the challenge, we use as input the first 64 output bits of the SHA-2 computation of the string “IDEA”, and provide one of the preimages found:

SHA-2(“IDEA”) :	9f8c 7b26 cde5 9ca3 dacc 74ec 7afd a737 ...
The challenge $H(0x9f8c7b26cde59ca3, 0)$:	0x20ad1fc924e61ba2
$CV_{i+1} = H(CV_i, M)$:	0x20ad 0x1fc9 0x24e6 0x1ba2
M :	0
CV_i :	0x1860 0x002e 0x2d82 0x0200

CV_i is one preimage out of $2^{23.585}$ for CV_{i+1} , the search takes $2^{25.486}$ IDEA encryptions, and the average cost per preimage is around $2^{1.9}$.

Hirose. We can reuse the attack on Davies-Meyer, but only one of the two branches will be controlled, with the other behaving randomly. We first find a preimage for the first branch (with probability $2^{-17.5}$) and then use the $2^{17.5}$ solutions on average to also match the second branch (with probability $2^{17.5-64} = 2^{-46.5}$). Therefore, our preimage search algorithm have parameters $p = 2^{-17.5-46.5} = 2^{-64}$ and $s = 1$, while the average number of preimage solutions found is $A = 2^{-64}$. The complexity of the search is equivalent to the Davies-Meyer case, $C = 2^{25.5}$. For an attacker using at most $2^{25.5}$ computations on an ideal 128-bit compression function, the average number of solutions he could find is only $2^{-102.5}$.

Abreast-DM. Similarly to Hirose, by setting for example $M = CV1 = 0$, one can attack one branch bit layer by bit layer while the other branch will behave randomly. The complexity analysis is identical to Hirose’s case.

Tandem-DM. Similarly to Hirose, by setting $M = CV1 = 0$, one can attack one branch bit layer by bit layer while the other branch will behave randomly. The complexity analysis is identical to Hirose’s case.

Peyrin *et al.*(II). If all compression function inputs $CV1$, $CV2$, $M1$ and $M2$ appear in at least one of the IDEA key inputs of f_1 , f_2 , f_3 (left side) and in at least one of the IDEA

key inputs of f_3, f_4, f_5 (right side), then the attack will not apply (because the attacker will not be able to use the null key completely). Otherwise, similarly to Hirose, by setting all IDEA keys to 0 on one side, one can attack it bit layer by bit layer while the other side will behave randomly. The complexity analysis is identical to Hirose's case.

MJH-Double. The attacker first fixes $M2 = CV2 = 0$ so as to get the null key for both IDEA encryptions. Then, similarly to the Davies-Meyer case, he find a preimage with probability $p = 2^{-17.5}$ for one of the two sides and this defines the value of $M1 \oplus CV1$. In order to get the preimage on the second side as well, the attacker only has to modify the value of $M1$ accordingly. If a solution is found on the first side, the attacker therefore gets $s = 2^{17.5}$ preimages. On average, he finds $A = 1$ solutions and the complexity is again $2^{25.5}$ computations. For an attacker using at most $2^{25.5}$ computations on an ideal 128-bit compression function, the average number of solutions he should find is only $2^{-102.5}$.

6.6 Results and Conclusions

We depict in Table 6.1 our results for the block cipher to compression function modes considered in this thesis when instantiated with IDEA. We implemented all attacks of reasonable complexities and provide the collision/preimage examples obtained.

Mode [†]	hash output size	compression function			hash function
		free-start collision attack	semi-free-start collision attack	preimage attack [‡] complexity (s, p)	collision attack
Davies-Meyer [2]	64	2^1		$2^{25.5} (2^{17.5}, 2^{-17.5})$	$2^{16.13}$
Hirose [90, 91]	128	2^1		$2^{25.5} (1, 2^{-64})$	
Abreast-DM [120, 187]	128	$2^{48.13}$		$2^{25.5} (1, 2^{-64})$	
Tandem-DM [120, 187]	128	$2^{48.13}$		$2^{25.5} (1, 2^{-64})$	
Peyrin <i>et al.</i> (II) [155]	128	$2^1 / 2^{48.13*}$	$2^1 / 2^{48.13} *$	$2^{25.5} (1, 2^{-64})*$	
MJH-Double [125]	128	$2^{32.26}$	$2^{32.26}$	$2^{25.5} (2^{17.5}, 2^{-17.5})$	

[†] We did not include MDC-2 as it does not provide ideal collision resistance.

[‡] The preimage attacks find s preimages on average with probability p , for on average $A = s \cdot p$ solutions.

* The results depend on the instance considered (see Sections 6.3, 6.4 and 6.5 for more details).

Table 6.1: Summary of results for block cipher based compression function modes when instantiated with IDEA

In this chapter, we showed collision and preimage attacks for several single and double-length block cipher based compression function constructions when instantiated with the block cipher IDEA. Namely, we analyzed all known double-key schemes such as Davies-Meyer,

Hirose, Abreast-DM, Tandem-DM, Peyrin *et al.* (II) and MJH-Double. While most of these constructions are conjectured or proved to be secure in the ideal cipher model, we showed that their security is very weak when instantiated with the block cipher **IDEA**, which remains considered as unbroken in the secret key model. In particular, we answer in the negative for the 20-year-old standing open question concerning the security of the Abreast-DM and Tandem-DM instantiated with **IDEA**. All our practical attacks have been implemented and they can work even for any number of **IDEA** rounds. Our results indicate that one has to be very careful when hashing with a block cipher that presents any weakness when the key is known or controlled by the attacker. Also, since we extensively use the presence of weak-keys for **IDEA**, as a future work it would be interesting to look at the security of hash functions based on block ciphers for which some key sets are known to be weaker than others.

Chapter 7

Conclusions

A consensus is that a block cipher should be designed and critically analyzed to demonstrate resistance to known cryptanalytic attacks, before it is trusted, accepted and deployed. This similarly applies to cryptographic hash functions.

In this thesis we present original contributions in four broad areas on or related with block cipher cryptanalysis, namely, linear cryptanalysis, differential cryptanalysis, meet-in-the-middle attack, and weak-keys of block cipher (in block cipher based hashing).

In Chapter 3 we present some results on an improved version of the multidimensional version of Matsui's Algorithm 1. Estimations of the attack complexities on reduced SERPENT illustrate that the multidimensional framework of Hermelin *et al.* is the most general framework so far, in particular, with the improvements, the overall efficiency outperforms all previous results. It also implies that, for block cipher designs, bounding the maximum correlation for the best single linear characteristic is not sufficient to form security claims. For example, in an SP-network block cipher with small Sboxes, a single linear trail with multiple active S-boxes in the first or last round can be modified to be a family of approximations such that the total number is exponential to the number of active S-boxes in the outer rounds. When these approximations have similar magnitude of correlations, the multidimensional Algorithm 1 can be used to exploit these combined correlations systematically. It may be worthy to consider security margins of block ciphers with respect to multidimensional linear cryptanalysis, meanwhile, it may also be meaningful to find out specific mechanisms to prevent such an attack from forming an exponential number of good

linear approximations.

Two of the four contribution chapters, *i.e.*, Chapter 4 and 6, are studying the security of hash functions rather than block ciphers. For cryptographic hash functions we mainly consider the collision and preimage security, while for block ciphers the major concern is its resistance to key-recovery attacks. The strategies of analyzing of hash functions is therefore quite different from the analysis of block ciphers due to the distinct security goals of the two symmetric-key primitives. On the other hand, for historical reasons, cryptographic hash functions are built either directly from block ciphers in the case of block cipher based hashing, or indirectly employing constructions used in block ciphers, such as the SPN like construction of the internal permutation of KECCAK. Indeed, the two hash functions considered in this thesis are especially closely related to block ciphers. Moreover, the techniques involved in analyzing these two hash functions are exceptionally close to block cipher cryptanalysis.

KECCAK employs a sponge construction and has its security heavily counted on the cryptographic quality of its internal permutation KECCAK-f[1600]. This permutation in fact closely resembles an SPN, with full confusion and diffusion layers. It is highlighted by the KECCAK design team that KECCAK has weak alignment [18]. Observing that rebound attack heavily relies on constructing truncated differentials, the authors argue that due to the weak alignment of KECCAK, it is difficult to construction truncated differentials because the propagation of differences is hard to predict. Indeed, the original rebound attack is a convenient tool when good truncated differentials can be easily formulated, *e.g.*, for the case of AES, Grindahl [112, 154] and many other designs. Despite the fact that it is not so easy to construct high probability differential paths for a good number of rounds of the KECCAK internal permutation, we show that the weak alignment property is not sufficient for KECCAK to be free from rebound attacks, although the weak alignment indeed makes the analysis quite difficult. The unaligned rebound attack framework shown in this thesis enables the analysis of such permutations with weak alignments, including both hash functions and block ciphers. An example that rebound attack is applied to block cipher is the 8-round Super-Sbox distinguisher of AES [79].

The analysis of IDEA hash is entirely based on weak-key analysis of the block cipher IDEA. To some it may be an instant intuition that IDEA, shown to have large class of weak-keys,

shall not be used for hashing. However, it is an intuition rather than a scientific fact that block ciphers with weak-keys should not be used for hashing. It is certainly not a convincing argument that a block cipher with a single weak-key should not be used for hashing. The results presented in Chapter 6 serve as a concrete example to the concern: even when there is only one weak-key, the results can be disastrous. Hashing with block ciphers with weak-keys should be handled with great caution. We think from the perspective of a cryptanalyst that it would be interesting to look at security of hash functions based on block ciphers for which some key sets are known to be weaker than others. Finding more certain answers on such problems rather than avoiding them, may serve our understanding of block cipher based hashing better.

7.1 Future Work

There are some interesting directions of research as a continuation of the topics discussed in this thesis.

- Study multidimensional linear cryptanalysis with an experimental approach, as in actual ciphers there are often more complex scenarios, such as the linear hull phenomena.
- Analyze the security of hash functions and block ciphers that have weak alignment, with respect to rebound attacks. The current unaligned rebound attack framework can be used or further improved.
- Study block cipher based hash functions where the underlying block cipher has a subset of keys weaker than the rest. Or more generally, what type of weak-keys should we concern when we reduce the security of the hash function to the security of block ciphers and the construction.

Bibliography

- [1] Masayuki Abe, editor. *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*. Springer, 2010. [132](#), [135](#)
- [2] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *CRC-Handbook of Applied Cryptography*. CRC Press, 1996. [5](#), [9](#), [27](#), [31](#), [32](#), [87](#), [104](#), [119](#)
- [3] Hamid R. Amirazizi and Martin E. Hellman. Time-Memory-Processor Trade-Offs. *IEEE Transactions on Information Theory*, 34(3):505–512, 1988. [12](#)
- [4] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A flexible block cipher with maximum assurance. In *The First Advanced Encryption Standard Candidate Conference*, 1998. [19](#)
- [5] Ross J. Anderson, Eli Biham, and Lars R. Knudsen. Serpent: A Proposal for the Advanced Encryption Standard. Available from: <http://www.cl.cam.ac.uk/~rja14/serpent.html>. [xvii](#), [9](#), [18](#), [20](#)
- [6] Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang. Preimages for Step-Reduced SHA-2. In Matsui [[133](#)], pages 578–597. [95](#)
- [7] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In Douglas R. Stinson and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2000. [9](#)
- [8] Kazumaro Aoki and Yu Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2009. [95](#)
- [9] Jean-Philippe Aumasson and Willi Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. Presented at the rump session of CHES 2009, 2009. [52](#)
- [10] Jean-Phillipe Aumasson, Miroslav Knezevic, and Orr Dunkelman. Bit-sliced Reference Implementation of KATAN/KTANTAN Family. <http://www.cs.technion.ac.il/~orrd/KATAN/katan.c>, 2010. Accessed on 6 August 2010. [22](#)
- [11] Eyüp Serdar Ayaz and Ali Aydin Selçuk. Improved DST Cryptanalysis of IDEA. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006. [104](#)

- [12] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Improved and Multiple Linear Cryptanalysis of Reduced Round Serpent - Description of the Linear Approximations. Unpublished, 2007. [35](#), [45](#), [46](#)
- [13] Daniel J. Bernstein. Second preimages for 6 (?? (8??)) rounds of Keccak?, November 2010. Available at http://ehash.iaik.tugraz.at/uploads/6/65/NIST-mailing-list_Bernstein-Daemen.txt. [53](#)
- [14] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge functions. ECRYPT Hash Workshop 2007, May 2007. [28](#), [52](#)
- [15] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Keccak specifications. Submission to NIST (Round 1), 2008. Available at <http://keccak.noekeon.org/Keccak-specifications.pdf>. [52](#)
- [16] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Keccak specifications. Submission to NIST (Round 2), 2009. Available at <http://keccak.noekeon.org/Keccak-specifications-2.pdf>. [52](#)
- [17] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Keccak Crunchy Crypto Collision and Pre-image Contest, 2011. See http://keccak.noekeon.org/crunchy_contest.html. [52](#)
- [18] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On alignment in Keccak. ECRYPT II Hash Workshop, 2011. [51](#), [60](#), [122](#)
- [19] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. The KECCAK Reference. Submission to NIST (Round 3), 2011. Available at <http://keccak.noekeon.org/Keccak-reference-3.0.pdf>. [xvii](#), [28](#), [31](#), [52](#), [53](#), [55](#)
- [20] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. The KECCAK SHA-3 Submission. Submission to NIST (Round 3), 2011. Available at <http://keccak.noekeon.org/Keccak-submission-3.pdf>. [28](#)
- [21] Eli Biham. New Types of Cryptanalytic Attacks Using related Keys (Extended Abstract). In *EUROCRYPT*, pages 398–409, 1993. [10](#)
- [22] Eli Biham. New Types of Cryptanalytic Attacks Using Related Keys. *J. Cryptology*, 7(4):229–246, 1994. [10](#)
- [23] Eli Biham, editor. *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*. Springer, 1997. [130](#), [133](#)
- [24] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A new block cipher proposal. In Vaudenay [\[177\]](#), pages 222–238. [19](#)
- [25] Eli Biham and Alex Biryukov. An Improvement of Davies' Attack on DES. In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 461–467. Springer, 1994. [16](#)
- [26] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *EUROCRYPT*, pages 12–23, 1999. [24](#)
- [27] Eli Biham, Alex Biryukov, and Adi Shamir. Miss in the Middle Attacks on IDEA and Khufu. In Knudsen [\[110\]](#), pages 124–138. [24](#)
- [28] Eli Biham, Orr Dunkelman, and Nathan Keller. New Cryptanalytic Results on IDEA. In Lai and Chen [\[118\]](#), pages 412–427. [104](#)

- [29] Eli Biham, Orr Dunkelman, Nathan Keller, and Adi Shamir. New Data-Efficient Attacks on Reduced-Round IDEA. Cryptology ePrint Archive, Report 2011/417, 2011. <http://eprint.iacr.org/>. 18, 104
- [30] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991. 23, 24
- [31] Eli Biham and Adi Shamir. Differential Cryptanalysis of the Full 16-Round DES. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992. 16, 23, 24, 87
- [32] Alex Biryukov, editor. *A New Attack on 6-Round IDEA*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007. 104
- [33] Alex Biryukov, editor. *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007. 132, 133
- [34] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On Multiple Linear Approximations. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2004. 26, 36, 37, 48, 49
- [35] Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors. *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*. Springer, 2011. 127, 131
- [36] Alex Biryukov, Jorge Nakahara Jr., Bart Preneel, and Joos Vandewalle. New Weak-Key Classes of IDEA. In Robert H. Deng, Sihan Qing, Feng Bao, and Jianying Zhou, editors, *ICICS*, volume 2513 of *Lecture Notes in Computer Science*, pages 315–326. Springer, 2002. 18, 104, 105
- [37] John Black. The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function. In Robshaw [163], pages 328–340. 7
- [38] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002. 27, 31
- [39] Manuel Blum and Silvio Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM J. Comput.*, 13(4):850–864, 1984. 12
- [40] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Viskelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007. xvii, 9, 20, 90
- [41] Andrey Bogdanov and Christian Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In Biryukov et al. [35], pages 1–24. 85, 87, 88, 90, 92, 93
- [42] Andrey Bogdanov and Christian Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. Cryptology ePrint Archive, Report 2010/532, 2010. <http://eprint.iacr.org/2010/532.pdf>. 92

- [43] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-Order Differential Properties of Keccak and *Luffa*. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2011. 52
- [44] Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990. 129, 134
- [45] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009. xvii, 10, 20, 90
- [46] Claude Carlet. *Boolean Functions for Cryptography and Error Correcting Codes*, pages 257–397. Cambridge University Press, New York, NY, USA, 1st edition, 2010. 8
- [47] Donghoon Chang. Near-Collision Attack and Collision-Attack on Double Block Length Compression Functions based on the Block Cipher IDEA. Cryptology ePrint Archive, Report 2006/478, 2006. <http://eprint.iacr.org/>. 104
- [48] David Chaum and Jan-Hendrik Evertse. Cryptanalysis of DES with a Reduced Number of Rounds: Sequences of Linear Factors in Block Ciphers. In Williams [186], pages 192–211. 85, 86, 87
- [49] Joo Yeon Cho, Miia Hermelin, and Kaisa Nyberg. A New Technique for Multidimensional Linear Cryptanalysis with Applications on Reduced Round Serpent. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC*, volume 5461 of *Lecture Notes in Computer Science*, pages 383–398. Springer, 2008. 42
- [50] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Improved and Multiple Linear Cryptanalysis of Reduced Round Serpent. In Dingyi Pei, Moti Yung, Dongdai Lin, and Chuankun Wu, editors, *Inscrypt*, volume 4990 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 2007. 45, 46, 47, 49
- [51] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Improving the Time Complexity of Matsui's Linear Cryptanalysis. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *ICISC*, volume 4817 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2007. 38, 39
- [52] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Experiments on the Multiple Linear Cryptanalysis of Reduced Round Serpent. In Nyberg [151], pages 382–397. 26, 37, 45, 47, 49
- [53] D. Coppersmith. The Data Encryption Standard (DES) and Its Strength Against Attacks. *IBM J. Res. Dev.*, 38:243–250, May 1994. 23
- [54] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001. 40
- [55] Joan Daemen, René Govaerts, and Joos Vandewalle. Weak Keys for IDEA. In Stinson [175], pages 224–231. 104, 105, 106, 108
- [56] Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael. NIST AES proposal, 1998. 85
- [57] Joan Daemen and Vincent Rijmen. The Block Cipher Rijndael. In Jean-Jacques Quisquater and Bruce Schneier, editors, *CARDIS*, volume 1820 of *Lecture Notes in Computer Science*, pages 277–284. Springer, 1998. 9

- [58] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002. xvii, 9, 27
- [59] Ivan Damgård. A Design Principle for Hash Functions. In Brassard [44], pages 416–427. 26
- [60] Hüseyin Demirci and Ali Aydın Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In Nyberg [151], pages 116–126. 85
- [61] Hüseyin Demirci, Ali Aydın Selçuk, and Erkan Türe. A New Meet-in-the-Middle Attack on the IDEA Block Cipher. In Matsui and Zuccherato [135], pages 117–129. 85
- [62] Hüseyin Demirci, Ali Aydın Selçuk, and Erkan Türe. A New Meet-in-the-Middle Attack on the IDEA Block Cipher. In Matsui and Zuccherato [135], pages 117–129. 104
- [63] Hüseyin Demirci, Ihsan Taskin, Mustafa Çoban, and Adnan Baysal. Improved Meet-in-the-Middle Attacks on AES. In Bimal K. Roy and Nicolas Sendrier, editors, *INDOCRYPT*, volume 5922 of *Lecture Notes in Computer Science*, pages 144–156. Springer, 2009. 85
- [64] Bert den Boer and Antoon Bosselaers. Collisions for the Compressin Function of MD5. In *EUROCRYPT*, pages 293–304, 1993. 27
- [65] Yvo Desmedt, editor. *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*. Springer, 1994. 131, 133
- [66] Whitfield Diffie and Martin E. Hellman. Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10(6):74–84, 1977. 84
- [67] Whitfield Diffie and George Ledin (translators). SMS4 Encryption Algorithm for Wireless Networks. Cryptology ePrint Archive, Report 2008/329, 2008. <http://eprint.iacr.org/>. 9
- [68] Itai Dinur, Orr Dunkelman, and Adi Shamir. New attacks on Keccak-224 and Keccak-256. Cryptology ePrint Archive, Report 2011/624, 2011. <http://eprint.iacr.org/>. 3, 52
- [69] Hans Dobbertin. Cryptanalysis of MD5 Compress. Presented at the rump session of EUROCRYPT 1996, 1996. 27
- [70] Alexandre Duc, Jian Guo, Thomas Peyrin, and Lei Wei. Unaligned Rebound Attack - Application to Keccak. *Fast Software Encryption*, 2012. Full version available on Cryptology ePrint Archive, Report 2011/420. <http://eprint.iacr.org/2011/420.pdf>. 3, 62, 63, 64
- [71] Orr Dunkelman, editor. *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*. Springer, 2009. 130, 131, 134
- [72] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved Single-Key Attacks on 8-round AES. Cryptology ePrint Archive, Report 2010/322, December 2010. To appear, available <http://eprint.iacr.org/2010/322.pdf>. 85

- [73] Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007. 86, 87
- [74] Horst Feistel. Cryptography and Computer Privacy. *Scientific American*, 228(5):15–23, May 1973. 9, 15
- [75] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved Cryptanalysis of Rijndael. In Bruce Schneier, editor, *FSE*, volume 1978 of *LNCS*, pages 213–230. Springer, 2000. 85
- [76] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. On the Security of Tandem-DM. In Dunkelman [71], pages 84–103. 104
- [77] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Security of Cyclic Double Block Length Hash Functions including Abreast-DM. Cryptology ePrint Archive, Report 2009/261, 2009. <http://eprint.iacr.org/>. 104
- [78] Internet Engineering Task Force. GOST 28147-89 Encryption, Decryption and Message Authentication Code (MAC) Algorithms. Request for Comments (RFC) 5830, March 2010. <http://tools.ietf.org/html/rfc5830>. 9
- [79] Henri Gilbert and Thomas Peyrin. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In Hong and Iwata [95], pages 365–383. 56, 58, 59, 122
- [80] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Foundations of Cryptography. Cambridge University Press, 2007. 12
- [81] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions (Extended Abstract). In *FOCS*, pages 464–479. IEEE Computer Society, 1984. 12
- [82] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. *J. ACM*, 33(4):792–807, 1986. 12
- [83] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *LNCS*, Singapore, December 2010. Springer. 95
- [84] Carlo Harpes, Gerhard G. Kramer, and James L. Massey. A Generalization of Linear Cryptanalysis and the Applicability of Matsui’s Piling-Up Lemma. In *EUROCRYPT*, pages 24–38, 1995. 26
- [85] Carlo Harpes and James L. Massey. Partitioning Cryptanalysis. In Biham [23], pages 13–27. 26
- [86] Philip Hawkes. Differential-Linear Weak Key Classes of IDEA. In *EUROCRYPT*, pages 112–126, 1998. 104, 105
- [87] Martin E. Hellman. A Cryptanalytic Time-Memory Trade-Off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980. 12
- [88] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional Linear Cryptanalysis of Reduced Round Serpent. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP*, volume 5107 of *Lecture Notes in Computer Science*, pages 203–215. Springer, 2008. 3, 26, 35, 37, 38, 41, 42, 45, 46, 48, 49

- [89] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional Extension of Matsui's Algorithm 2. In Dunkelman [71], pages 209–227. 26
- [90] Shoichi Hirose. Provably Secure Double-Block-Length Hash Functions in a Black-Box Model. In Choonsik Park and Seongtaek Chee, editors, *ICISC*, volume 3506 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 2004. 32, 104, 119
- [91] Shoichi Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. In Robshaw [163], pages 210–225. 32, 33, 104, 119
- [92] Viet Tung Hoang and Phillip Rogaway. On Generalized Feistel Networks. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 613–630. Springer, 2010. 9
- [93] Deukjo Hong, Bonwook Koo, and Yu Sasaki. Improved Preimage Attack for 68-Step HAS-160. In Donghoon Lee and Seokhie Hong, editors, *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 332–348. Springer, 2009. 95
- [94] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jaesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2006. 90
- [95] Seokhie Hong and Tetsu Iwata, editors. *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *Lecture Notes in Computer Science*. Springer, 2010. 130, 135
- [96] Sebastiaan Indestege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel. A Practical Attack on KeeLoq. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008. 85
- [97] Burton S. Kaliski Jr. and Matthew J. B. Robshaw. Linear Cryptanalysis Using Multiple Approximations. In Desmedt [65], pages 26–39. 26, 36, 48
- [98] Jorge Nakahara Jr, Bart Preneel, and Joos Vandewalle. Experimental Non-Linear Cryptanalysis. Technical report, Katholieke Universiteit Leuven, Dept. Electrical Engineering, March 2003. Version 1.15. 26
- [99] Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors. *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*. Springer, 2009. 134, 135
- [100] Pascal Junod. *Statistical Cryptanalysis of Block Ciphers*. PhD thesis, Ecole Polytechnique Federale de Lausanne, Switzerland, 2004. 13
- [101] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Sphere, London :, 1973. 1
- [102] Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck, and Martin Schläffer. Cryptanalysis of *uffa* v2 Components. In Biryukov et al. [35], pages 388–409. 51

- [103] Dmitry Khovratovich, Ivica Nikolic, and Christian Rechberger. Rotational Rebound Attacks on Reduced Skein. In Abe [1], pages 1–19. 58
- [104] Aggelos Kiayias, editor. *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*. Springer, 2011. 133, 136
- [105] Alexander Klimov and Adi Shamir. Cryptographic Applications of T-Functions. In Matsui and Zuccherato [135], pages 248–261. 104, 107
- [106] Lars Knudsen. DEAL - A 128-bit Block Cipher. In *NIST AES Proposal*, 1998. 24
- [107] Lars R. Knudsen. *Block Ciphers - Analysis, Design and Applications*. PhD thesis, Aarhus University, Denmark, 1994. 11
- [108] Lars R. Knudsen. Truncated and Higher Order Differentials. In Preneel [156], pages 196–211. 24
- [109] Lars R. Knudsen. Contemporary Block Ciphers. In Ivan Damgård, editor, *Lectures on Data Security*, volume 1561 of *Lecture Notes in Computer Science*, pages 105–126. Springer, 1998. 11
- [110] Lars R. Knudsen, editor. *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*. Springer, 1999. 126, 136
- [111] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010. 90
- [112] Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl Hash Functions. In Biryukov [33], pages 39–57. 122
- [113] Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In Kurosawa [116], pages 315–324. 28
- [114] Lars R. Knudsen and Matthew J. B. Robshaw. Non-Linear Approximations in Linear Cryptanalysis. In *EUROCRYPT*, pages 224–236, 1996. 26
- [115] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987. 5
- [116] Kaoru Kurosawa, editor. *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*. Springer, 2007. 132, 135
- [117] Xuejia Lai. Higher Order Derivatives and Differential Cryptanalysis. In *Proc. "Symposium on Communication, Coding and Cryptography", in honor of J. L. Massey on the occasion of his 60'th birthday*, Monte-Verita, Ascona, Switzerland, Feb 1994. Kluwer Academic Publishers. 24
- [118] Xuejia Lai and Kefei Chen, editors. *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*. Springer, 2006. 126, 135

- [119] Xuejia Lai and James L. Massey. A Proposal for a New Block Encryption Standard. In *EUROCRYPT*, pages 389–404, 1990. [xvii](#), [10](#), [17](#), [103](#), [104](#)
- [120] Xuejia Lai and James L. Massey. Hash Function Based on Block Ciphers. In *EUROCRYPT*, pages 55–70, 1992. [32](#), [104](#), [119](#)
- [121] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schlffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Matsui [[133](#)], pages 126–143. [58](#), [59](#)
- [122] Susan K. Langford and Martin E. Hellman. Differential-Linear Cryptanalysis. In Desmedt [[65](#)], pages 17–25. [24](#)
- [123] Gregor Leander, Christof Paar, Axel Poschmann, and Kai Schramm. New Lightweight DES Variants. In Biryukov [[33](#)], pages 196–210. [90](#)
- [124] Jooyoung Lee and Daesung Kwon. The Security of Abreast-DM in the Ideal Cipher Model. Cryptology ePrint Archive, Report 2009/225, 2009. <http://eprint.iacr.org/>. [104](#)
- [125] Jooyoung Lee and Martijn Stam. MJH: A Faster Alternative to MDC-2. In Kiayias [[104](#)], pages 213–236. [34](#), [104](#), [119](#)
- [126] Jooyoung Lee, Martijn Stam, and John P. Steinberger. The Collision Security of Tandem-DM in the Ideal Cipher Model. In Rogaway [[164](#)], pages 561–577. [104](#)
- [127] Chae Hoon Lim and Tymur Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In JooSeok Song, Taekyoung Kwon, and Moti Yung, editors, *WISA*, volume 3786 of *LNCS*, pages 243–258. Springer, 2005. [90](#)
- [128] Michael Luby and Charles Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM J. Comput.*, 17(2):373–386, 1988. [12](#), [27](#)
- [129] Stefan Lucks. Attacking Triple Encryption. In Vaudenay [[177](#)], pages 239–253. [85](#)
- [130] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993. [16](#), [24](#), [25](#), [36](#), [48](#)
- [131] Mitsuru Matsui. The First Experimental Cryptanalysis of the Data Encryption Standard. In Desmedt [[65](#)], pages 1–11. [24](#), [26](#), [36](#)
- [132] Mitsuru Matsui. New Block Encryption Algorithm MISTY. In Biham [[23](#)], pages 54–68. [9](#)
- [133] Mitsuru Matsui, editor. *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*. Springer, 2009. [125](#), [133](#)
- [134] Mitsuru Matsui and Atsuhiro Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In *EUROCRYPT*, pages 81–91, 1992. [24](#), [36](#)
- [135] Mitsuru Matsui and Robert J. Zuccherato, editors. *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers*, volume 3006 of *Lecture Notes in Computer Science*. Springer, 2004. [129](#), [132](#)

- [152] National Institute of Standards and Technology. Skipjack and KEA Algorithm Specifications, Version 2.0, May 1998. Available from NIST, <http://csrc.nist.gov/encryption/skipjack/skipjack.pdf>. 9
- [153] Kenji Ohkuma. Weak Keys of Reduced-Round PRESENT for Linear Cryptanalysis. In Jr. et al. [99], pages 249–265. 48
- [154] Thomas Peyrin. Cryptanalysis of Grindahl. In Kurosawa [116], pages 551–567. 122
- [155] Thomas Peyrin, Henri Gilbert, Frédéric Muller, and Matthew J. B. Robshaw. Combining Compression Functions and Block Cipher-Based Hash Functions. In Lai and Chen [118], pages 315–331. 33, 104, 119
- [156] Bart Preneel, editor. *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*. Springer, 1995. 132, 135, 136
- [157] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In Stinson [175], pages 368–378. 27
- [158] Vincent Rijmen, Deniz Toz, and Kerem Varici. Rebound Attack on Reduced-Round Versions of JH. In Hong and Iwata [95], pages 286–303. 51
- [159] Ronald L. Rivest. The MD5 message-digest algorithm. Request for Comments (RFC) 1320, Internet Activities Board, Internet Privacy Task Force, April 1992. 52
- [160] Ronald L. Rivest. The RC5 Encryption Algorithm. In Preneel [156], pages 86–96. 9
- [161] Ronald L. Rivest, Matt Robshaw, Ray Sidney, and Yiqun L. Yin. The RC6 Block Cipher. AES proposal (v1.1). Available on <http://www.rsa.com/rsalabs/node.asp?id=2512>. 9
- [162] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. 5
- [163] Matthew J. B. Robshaw, editor. *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*. Springer, 2006. 127, 131
- [164] Phillip Rogaway, editor. *Advances in Cryptology - CRYPTO 2011, 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011. 133, 134
- [165] Yu Sasaki and Kazumaro Aoki. Preimage Attacks on 3, 4, and 5-Pass HAVAL. In Josef Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271. Springer, 2008. 95
- [166] Yu Sasaki and Kazumaro Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2009. 95
- [167] Yu Sasaki, Yang Li, Lei Wang, Kazuo Sakiyama, and Kazuo Ohta. Non-full-active Super-Sbox Analysis: Applications to ECHO and Grøstl. In Abe [1], pages 38–55. 58
- [168] Bruce Schneier and John Kelsey. Unbalanced Feistel Networks and Block Cipher Design. In Dieter Gollmann, editor, *FSE*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144. Springer, 1996. 9

- [169] Gautham Sekar, Nicky Mouha, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round GOST. *Information Processing Letters*, 2011. Submitted. 85
- [170] Gautham Sekar, Nicky Mouha, Vesselin Velichkov, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round XTEA. In Kiayias [104]. To appear, available <http://www.cosic.esat.kuleuven.be/publications/article-1505.pdf>. 85
- [171] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28:656–715, 1949. 6, 7, 8
- [172] Takeshi Shimoyama and Toshinobu Kaneko. Quadratic Relation of S-box and Its Application to the Linear Attack of Full Round DES. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 1998. 26
- [173] Arthur Sorkin. Lucifer, a Cryptographic Algorithm. *Cryptologia*, 8(1):22–41, January 1984. 9, 15
- [174] William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson Education, 3rd edition, 2002. 8
- [175] Douglas R. Stinson, editor. *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*. Springer, 1994. 128, 135
- [176] Douglas R. Stinson. *Cryptography: Theory and Practice, Third Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, November 2005. 5
- [177] Serge Vaudenay, editor. *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*. Springer, 1998. 126, 133
- [178] David Wagner. The Boomerang Attack. In Knudsen [110], pages 156–170. 24
- [179] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005. 52
- [180] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005. 27, 52
- [181] Lei Wei, Thomas Peyrin, Przemysław Sokołowski, San Ling, Josef Pieprzyk, and Huaxiong Wang. On the (In)Security of IDEA in Various Hashing Modes. *Fast Software Encryption*, 2012. 3
- [182] Lei Wei, Christian Rechberger, Jian Guo, Hongjun Wu, Huaxiong Wang, and San Ling. Improved Meet-in-the-Middle Cryptanalysis of KTANTAN. *Cryptology ePrint Archive*, Report 2011/201, 2011. <http://eprint.iacr.org/>. 3
- [183] Lei Wei, Christian Rechberger, Jian Guo, Hongjun Wu, Huaxiong Wang, and San Ling. Improved Meet-in-the-Middle Cryptanalysis of KTANTAN (Poster). In Udaya Parampalli and Philip Hawkes, editors, *ACISP*, volume 6812 of *Lecture Notes in Computer Science*, pages 433–438. Springer, 2011. 3
- [184] David J. Wheeler and Roger M. Needham. TEA, a Tiny Encryption Algorithm. In Preneel [156], pages 363–366. xvii, 9

- [185] Michael J. Wiener. The Full Cost of Cryptanalytic Attacks. *J. Cryptology*, 17(2):105–124, 2004. [12](#)
- [186] Hugh C. Williams, editor. *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218 of *Lecture Notes in Computer Science*. Springer, 1986. [128](#), [134](#)
- [187] Xuejia Lai. On the Design and Security of Block Ciphers. Hartung-Gorre Verlag, Konstanz, 1992. [17](#), [32](#), [103](#), [104](#), [119](#)
- [188] R. K. Rao Yarlagadda and John E. Hershey. *Hadamard matrix analysis and synthesis: with applications to communications and signal/image processing*. Kluwer Academic Publishers, Norwell, MA, USA, 1997. [40](#)
- [189] Bo Zhu and Guang Gong. Guess-then-Meet-in-the-Middle Attacks on the KTANTAN Family of Block Ciphers. Cryptology ePrint Archive, Report 2011/619, 2011. <http://eprint.iacr.org/>. [4](#), [84](#)

Appendix A

Unaligned Rebound Attack

A.1 KECCAK Sbox, translation and transposition constants

The first bit of a row is viewed as the most significant bit. Given a 5-bit input x , the output of the Sbox is given by y . Both x and y are in hex number.

x	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
y	00	09	12	0B	05	0C	16	0F	0A	03	18	01	0D	04	1E	07
x	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
y	14	15	06	17	11	10	02	13	1A	1B	08	19	1D	1C	0E	1F

Table A.1: The 5-bit Sbox applied to each row of the state

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

Table A.2: Translation constants of ρ for KECCAK- f [1600]

The value in the table is the new (x, y) coordinate of the point.

	$x = 0$	$x = 1$	$x = 2$	$x = 3$	$x = 4$
$y = 0$	(0, 0)	(0, 2)	(0, 4)	(0, 1)	(0, 3)
$y = 1$	(1, 3)	(1, 0)	(1, 2)	(1, 4)	(1, 1)
$y = 2$	(2, 1)	(2, 3)	(2, 0)	(2, 2)	(2, 4)
$y = 3$	(3, 4)	(3, 1)	(3, 3)	(3, 0)	(3, 2)
$y = 4$	(4, 2)	(4, 4)	(4, 1)	(4, 3)	(4, 0)

Table A.3: Transposition constants of π for KECCAK- f [1600]

A.2 Differential distribution table of χ and χ^{-1}

The first bit of a row is viewed as the least significant bit. Given input difference Δ_{in} and output difference Δ_{out} the number in the table shows the size of the solution set $\{v \mid \chi(v) \oplus \chi(v \oplus \Delta_{in}) = \Delta_{out}\}$. Differences are in hex number. We note that the DDT of χ^{-1} is the transpose of Table A.4.

$\Delta_{in} \backslash \Delta_{out}$	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
00	32	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
01	-	8	-	-	-	-	-	-	-	8	-	-	-	-	-	-	-	8	-	-	-	-	-	-	-	8	-	-	-	-	-	-
02	-	-	8	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	8	-	-	-	-	-	-	-	-	-	-	-	-
03	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-	-
04	-	-	-	-	8	8	8	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05	-	-	-	-	4	-	4	-	-	-	-	-	4	-	4	-	-	-	-	-	-	4	-	4	-	-	-	-	-	4	-	4
06	-	-	-	-	4	4	4	4	-	-	-	-	-	-	-	-	-	-	-	-	4	4	4	4	-	-	-	-	-	-	-	-
07	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2
08	-	-	-	-	-	-	-	8	-	8	-	8	-	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09	-	4	-	4	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-	4	-	-	-	-	-	-	-	-	4	-	4
0A	-	-	-	-	-	-	-	4	-	-	4	4	-	4	-	4	-	-	-	-	-	-	-	4	-	-	4	4	-	-	4	-
0B	-	4	4	-	-	-	-	-	-	-	-	-	-	4	4	-	-	4	4	-	-	-	-	-	-	-	-	-	-	4	4	-
0C	-	-	-	-	-	-	-	4	4	4	4	4	4	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0D	-	-	-	-	4	-	4	-	4	-	4	-	-	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-	-	-	-
0E	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2
0F	-	-	-	-	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	-	-	-	-
10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	-	-	-	-	8	-	-	-	8	-	-	8	-	-	-	-
11	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-
12	-	-	4	4	-	-	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	-	-	4	4
13	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	-	-	-	-	4	4	4	4	-	-	-	-	4	4
15	-	4	-	-	-	-	-	4	-	4	-	-	-	-	4	4	-	-	-	-	-	-	4	-	4	-	-	-	-	-	4	-
16	-	-	4	4	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	4	4	-	-
17	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	2	2	2	2	2	2	-	-	-	2	2	2	2	-	-
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-	4	-	4	-	4	-	4	-	-
19	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2
1A	-	-	-	-	-	-	-	4	-	-	4	4	-	-	4	4	-	-	4	4	-	-	4	-	-	-	-	-	-	-	-	-
1B	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-
1C	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1D	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-
1E	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-
1F	-	2	2	-	2	-	-	2	2	-	-	2	-	2	2	-	2	-	2	-	2	-	2	2	-	2	2	-	2	-	-	2

Table A.4: The differential distribution table of the χ when viewed as Sbox

A.3 Differential paths for KECCAK-f[100]

a_0	3	7	2	A	C	
	2	7	2	B	C	
	2	7	2	B	C	
	2	7	2	B	C	
	2	7	2	B	C	
b_0	1	-	-	-	-	2^{-4}
	1	-	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
a_1	1	-	-	-	-	
	1	-	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
b_1	1	-	-	-	-	2^{-4}
	-	-	-	-	-	
	-	-	-	-	-	
	-	1	-	-	-	
	-	-	-	-	-	
a_2	1	-	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
	1	1	-	-	-	
	-	-	-	-	-	

Table A.5: 2-Round KECCAK-f[100] differential path with probability 2^{-8}

a_0	D	-	E	5	C	
	D	-	E	7	D	
	D	-	E	5	C	
	D	-	E	5	C	
	D	-	E	5	C	
b_0	-	-	-	-	-	2^{-4}
	-	1	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
	-	1	-	-	-	
a_1	-	-	-	-	-	
	-	1	-	-	1	
	-	-	-	-	-	
	-	-	-	-	-	
	-	1	-	-	1	
b_1	-	1	-	-	4	2^{-8}
	-	1	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
	-	-	-	-	4	
a_2	-	1	-	-	4	
	-	1	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
	-	-	-	-	4	
b_2	-	1	-	-	1	2^{-7}
	-	-	-	-	-	
	2	-	-	-	-	
	2	-	-	-	-	
	-	-	-	-	-	
a_3	-	1	-	-	-	
	-	-	-	-	-	
	2	-	-	-	-	
	2	-	-	-	-	
	-	-	-	-	-	

Table A.6: 3-Round KECCAK-f[100] differential path with probability 2^{-19}

a_0	D	-	E	5	C	
	D	-	E	7	D	
	D	-	E	5	C	
	D	-	E	5	C	
	D	-	E	5	C	
b_0	-	-	-	-	-	2^{-4}
	-	1	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
	-	1	-	-	-	
a_1	-	-	-	-	-	
	-	1	-	-	1	
	-	-	-	-	-	
	-	-	-	-	-	
	-	1	-	-	1	
b_1	-	1	-	-	4	2^{-8}
	-	1	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
	-	-	-	-	4	
a_2	-	1	-	-	4	
	-	1	-	-	-	
	-	-	-	-	-	
	-	-	-	-	-	
	-	-	-	-	4	
b_2	-	1	-	-	1	2^{-7}
	-	-	-	-	-	
	2	-	-	-	-	
	2	-	-	-	-	
	-	-	-	-	-	
a_3	-	1	-	-	-	
	-	-	-	-	-	
	2	-	-	-	-	
	2	-	-	-	-	
	-	-	-	-	-	
b_3	2	-	8	-	-	2^{-18}
	-	-	-	-	2	
	2	4	-	-	8	
	-	2	-	8	-	
	4	-	-	-	-	
a_4	2	-	8	-	-	
	-	-	-	-	2	
	6	4	8	8	8	
	-	2	-	8	2	
	4	-	-	-	-	

Table A.7: 4-Round KECCAK-f[100] differential path with probability 2^{-37}

a_0	D	C	5	D	-	
	D	6	9	D	-	
	D	4	1	D	-	
	F	4	1	F	-	
	D	4	1	D	-	
b_0	-	2	-	4	-	2^{-12}
	-	-	-	-	-	
	1	2	-	-	-	
	-	-	-	-	-	
	1	-	-	4	-	
a_1	-	2	-	4	-	
	-	-	-	-	-	
	1	2	-	-	-	
	-	-	-	-	-	
	1	-	-	4	-	
b_1	-	-	-	-	-	2^{-11}
	4	-	8	-	-	
	4	-	-	-	4	
	-	-	8	-	4	
	-	-	-	-	-	
a_2	-	-	-	-	-	
	4	-	8	-	-	
	4	-	-	-	4	
	-	-	8	-	4	
	-	-	-	-	-	
b_2	-	-	-	-	-	2^{-11}
	-	-	2	-	-	
	-	2	-	4	-	
	-	4	-	4	-	
	-	-	2	-	-	
a_3	-	-	-	-	-	
	-	-	2	-	-	
	-	2	-	4	-	
	-	-	-	4	-	
	-	2	2	-	-	
b_3	-	-	-	8	-	2^{-11}
	-	-	-	-	4	
	-	8	8	-	-	
	-	-	8	-	-	
	-	-	-	-	8	
a_4	-	-	-	8	-	
	-	-	-	-	4	
	-	-	8	-	8	
	-	-	8	-	-	
	-	-	-	8	8	
b_4	4	-	4	1	2	2^{-22}
	-	4	2	-	-	
	-	-	1	-	1	
	-	4	-	4	-	
	-	4	4	8	-	
a_5	-	-	6	5	2	
	-	4	2	-	-	
	-	-	-	1	1	
	-	-	-	4	-	
	-	4	4	8	-	

Table A.8: 5-Round KECCAK-f[100] differential path with probability 2^{-67}

A.4 Differential paths for KECCAK-f[200]

a_0	74	CB	DF	-A	E2	
	75	CB	DF	1A	E2	
	75	CB	DF	1A	E2	
	75	CB	DF	1A	E2	
	75	CB	DF	1A	E2	
b_0	-1	--	--	--	--	2^{-4}
	-1	--	--	--	--	
	--	--	--	--	--	
	--	--	--	--	--	
	--	--	--	--	--	
a_1	-1	--	--	--	--	
	-1	--	--	--	--	
	--	--	--	--	--	
	--	--	--	--	--	
	--	--	--	--	--	
b_1	-1	--	--	--	--	2^{-4}
	--	--	--	--	--	
	--	--	--	--	--	
	--	1-	--	--	--	
	--	--	--	--	--	
a_2	-1	--	--	--	--	
	--	--	--	--	--	
	--	--	--	--	--	
	--	1-	--	--	--	
	--	--	--	--	--	

Table A.9: 2-Round KECCAK-f[200] differential path with probability 2^{-8}

a_0	2F	AA	C2	36	6A	
	2F	AA	C2	34	7A	
	2F	AA	C2	36	6A	
	2F	AA	C2	36	6A	
	2F	AA	C2	36	6A	
b_0	--	--	--	--	--	2^{-4}
	--	-1	--	--	--	
	--	--	--	--	--	
	--	--	--	--	--	
	--	-1	--	--	--	
a_1	--	--	--	--	--	
	-1	-1	--	--	--	
	--	--	--	--	--	
	--	--	--	--	--	
	-1	-1	--	--	--	
b_1	--	1-	--	--	--	2^{-8}
	--	--	--	--	--	
	--	--	--	--	-4	
	--	1-	--	--	--	
	--	--	--	--	-4	
a_2	--	1-	--	--	--	
	--	--	--	--	--	
	--	--	--	--	-4	
	--	1-	--	--	--	
	--	--	--	--	-4	
b_2	--	--	--	--	-1	2^{-8}
	--	--	--	-2	--	
	2-	--	--	--	--	
	--	--	--	--	--	
	--	--	-2	--	--	
a_3	--	--	--	--	-1	
	--	--	-2	-2	--	
	2-	--	--	--	--	
	--	--	--	--	--	
	--	--	-2	--	--	

Table A.10: 3-Round KECCAK-f[200] differential path with probability 2^{-20}

a_0	D2	3B	7C	31	A4	
	C3	3A	7C	33	A4	
	D2	3B	7C	31	A4	
	D2	3B	7C	21	A6	
	D2	3B	7C	31	A4	
b_0	--	1-	--	-2	--	2^{-12}
	--	--	--	--	--	
	--	--	--	-2	--	
	--	11	--	--	--	
	--	-1	--	--	--	
a_1	--	1-	--	-2	--	
	--	--	--	--	--	
	--	--	--	-2	--	
	-1	11	--	--	--	
	-1	-1	--	--	--	
b_1	--	--	--	--	--	2^{-14}
	2-	--	--	22	--	
	2-	--	-4	--	-4	
	--	--	--	--	--	
	--	--	--	-2	-4	
a_2	--	--	--	--	--	
	2-	--	--	-2	--	
	2-	--	--	--	-4	
	--	--	--	--	--	
	--	--	--	-2	-4	
b_2	--	--	--	--	-1	2^{-11}
	--	--	-1	--	--	
	--	--	--	--	--	
	--	-2	--	--	-2	
	--	-1	-2	--	--	
a_3	--	--	-1	--	-1	
	--	-1	-1	--	--	
	--	--	--	--	--	
	--	-2	-2	--	--	
	--	-3	-2	--	-1	
b_3	--	1-	--	--	4-	2^{-19}
	--	--	--	4-	4-	
	--	4-	--	--	--	
	-8	--	--	-1	--	
	4-	--	--	--	-C	
a_4	--	1-	--	--	4-	
	--	4-	--	--	4-	
	4-	4-	--	--	--	
	-8	--	--	-1	-8	
	4-	--	--	--	-C	

Table A.11: 4-Round KECCAK-f[200] differential path with probability 2^{-56}

a_0	BE	A-	29	E2	B3	
	9E	A-	29	E6	B3	
	9E	A-	-D	E2	F3	
	BE	A-	29	C2	B7	
	BA	A4	29	E2	B3	
b_0	--	--	21	-4	--	2^{-20}
	--	--	-1	--	--	
	--	--	--	-4	1-	
	--	-2	--	--	--	
	--	-2	2-	--	1-	
a_1	--	--	21	-4	--	
	--	--	-1	--	--	
	--	--	--	-4	1-	
	--	-2	--	--	--	
	--	-2	2-	--	1-	
b_1	--	--	--	--	-4	2^{-16}
	4-	--	--	4-	-4	
	--	4-	-8	--	--	
	--	--	--	--	--	
	48	--	-8	--	-8	
a_2	--	--	--	--	-4	
	4-	4-	--	--	-4	
	-8	48	-8	--	--	
	--	--	--	--	--	
	48	-8	-8	--	--	
b_2	--	-4	4-	--	--	2^{-22}
	--	4-	4-	--	-1	
	--	--	--	--	21	
	2-	-4	21	--	--	
	--	--	--	--	2-	
a_3	--	-4	4-	--	--	
	--	--	4-	--	-1	
	--	--	-1	--	21	
	--	-4	21	--	--	
	--	--	2-	--	2-	
b_3	--	--	-8	--	-8	2^{-22}
	--	1-	--	8-	-4	
	-8	1-	--	--	--	
	--	--	--	9-	--	
	1-	--	9-	--	--	
a_4	-8	--	--	--	-8	
	--	1-	--	8-	-4	
	-8	1-	--	--	-8	
	--	--	9-	9-	--	
	--	--	9-	1-	--	
b_4	-C	-1	--	13	--	2^{-41}
	8-	4-	6-	--	12	
	--	--	1-	--	1-	
	4-	4-	4-	48	18	
	--	44	-4	-8	--	
a_5	-C	--	--	13	-8	
	8-	--	6-	-2	12	
	--	--	--	1-	1-	
	--	4-	4-	-8	18	
	-4	4-	-4	-8	--	

Table A.12: 5-Round KECCAK-f[200] differential path with probability 2^{-121}

A.5 Differential paths for KECCAK-f[400]

a_0	3C48	9358	5788	6268	1AF8	
	3C49	9358	5788	6278	1AF8	
	3C49	9358	5788	6278	1AF8	
	3C49	9358	5788	6278	1AF8	
	3C49	9358	5788	6278	1AF8	
b_0	---1	----	----	----	----	2^{-4}
	---1	----	----	----	----	
	----	----	----	----	----	
	----	----	----	----	----	
	----	----	----	----	----	
a_1	---1	----	----	----	----	
	---1	----	----	----	----	
	----	----	----	----	----	
	----	----	----	----	----	
	----	----	----	----	----	
b_1	---1	----	----	----	----	2^{-4}
	----	----	----	----	----	
	----	----	----	----	----	
	----	--1-	----	----	----	
	----	----	----	----	----	
a_2	---1	----	----	----	----	
	----	----	----	----	----	
	----	----	----	----	----	
	----	--1-	----	----	----	
	----	----	----	----	----	

Table A.13: 2-Round KECCAK-f[400] differential path with probability 2^{-8}

a_0	B4D6	1E7F	D574	C5-5	8C28	
	B4D6	1E7F	D574	C5-5	8C28	
	B4D6	1E7F	D554	C585	8C28	
	B4D6	1E7F	D574	E5-5	8828	
	B4D6	1E7F	D574	C5-5	8C28	
b_0	----	----	---1	---4	----	2^{-8}
	----	----	----	----	----	
	----	----	---1	---4	----	
	----	----	----	----	----	
	----	----	----	----	----	
a_1	----	----	---1	---4	----	
	----	----	----	----	----	
	----	----	---1	---4	----	
	----	----	----	----	----	
	----	----	----	----	----	
b_1	----	----	-8--	----	----	2^{-8}
	4---	----	----	----	----	
	----	----	-8--	----	----	
	----	----	----	----	----	
	4---	----	----	----	----	
a_2	----	----	-8--	----	----	
	4---	----	----	----	----	
	----	----	-8--	----	----	
	----	----	----	----	----	
	4---	----	----	----	----	
b_2	----	----	--4-	----	----	2^{-8}
	----	----	----	----	----	
	----	----	----	----	---1	
	----	---4	----	----	----	
	-2--	----	----	----	----	
a_3	----	----	--4-	----	----	
	----	----	----	----	----	
	----	----	----	----	---1	
	----	---4	----	----	----	
	-2--	----	----	----	----	

Table A.14: 3-Round KECCAK-f[400] differential path with probability 2^{-24}

a_0	12EF	4577	6BCC	1FCD	BA-3	
	12EE	C477	6BCC	1FCD	BA-2	
	12EE	C577	6BCC	9FCD	BA-1	
	12EE	D577	6BCC	1FDD	BA-3	
	12EE	C577	6BCC	1FCD	BA-3	
b_0	---1	--1-	----	-2--	----	2^{-16}
	----	--1-	----	-2--	----	
	---1	----	-1--	----	----	
	----	----	----	----	----	
	----	----	-1--	----	----	
a_1	---1	--1-	----	-2--	----	
	----	--1-	----	-2--	----	
	---1	----	-1--	----	----	
	----	----	----	----	----	
	----	----	-1--	----	----	
b_1	---1	---1	---8	----	----	2^{-14}
	--2-	----	---8	----	--2-	
	--2-	----	----	----	----	
	----	----	----	----	----	
	----	---1	----	----	----	
a_2	----	---1	---8	----	----	
	--2-	----	---8	----	----	
	--2-	----	----	----	----	
	----	----	----	----	----	
	----	---1	----	----	----	
b_2	----	----	----	----	----	2^{-12}
	----	----	-1--	----	----	
	---2	-2--	----	----	----	
	----	-2--	----	----	----	
	---2	----	----	----	---4	
a_3	----	----	----	----	----	
	----	----	-1--	----	----	
	---2	-2--	----	----	----	
	----	-2--	----	----	----	
	---2	----	----	----	---4	
b_3	---4	--2-	----	21--	---1	2^{-42}
	8-1-	----	--3-	----	----	
	-4--	4---	1--2	----	--18	
	----	--4-	----	----	-8-1	
	----	84--	----	-8--	-8--	
a_4	---4	--2-	----	21--	---1	
	8---	--2-	--3-	----	----	
	-4--	4---	1-12	----	--18	
	----	--4-	----	----	-8-1	
	84--	84--	----	----	-8--	

Table A.15: 4-Round KECCAK-f[400] differential path with probability 2^{-84}

The first 4 rounds of this path form a best 4-round path with probability 2^{-84} .

a_0	12EF	4577	6BCC	1FCD	BA-3	
	12EE	C477	6BCC	1FCD	BA-2	
	12EE	C577	6BCC	9FCD	BA-1	
	12EE	D577	6BCC	1FDD	BA-3	
	12EE	C577	6BCC	1FCD	BA-3	
b_0	---1	--1-	----	-2--	----	2^{-16}
	----	--1-	----	-2--	----	
	---1	----	-1--	----	----	
	----	----	----	----	----	
	----	----	-1--	----	----	
a_1	---1	--1-	----	-2--	----	
	----	--1-	----	-2--	----	
	---1	----	-1--	----	----	
	----	----	----	----	----	
	----	----	-1--	----	----	
b_1	---1	---1	---8	----	----	2^{-14}
	--2-	----	---8	----	--2-	
	--2-	----	----	----	----	
	----	----	----	----	----	
	----	---1	----	----	----	
a_2	----	---1	---8	----	----	
	--2-	----	---8	----	----	
	--2-	----	----	----	----	
	----	----	----	----	----	
	----	---1	----	----	----	
b_2	----	----	----	----	----	2^{-12}
	----	----	-1--	----	----	
	---2	-2--	----	----	----	
	----	-2--	----	----	----	
	---2	----	----	----	---4	
a_3	----	----	----	----	----	
	----	----	-1--	----	----	
	---2	-2--	----	----	----	
	----	-2--	----	----	----	
	---2	----	----	----	---4	
b_3	---4	--2-	----	21--	---1	2^{-42}
	8-1-	----	--3-	----	----	
	-4--	4---	1--2	----	--18	
	----	--4-	----	----	-8-1	
	----	84--	----	-8--	-8--	
a_4	---4	--2-	----	21--	---1	
	8---	--2-	--3-	----	----	
	-4--	4---	1-12	----	--18	
	----	--4-	----	----	-8-1	
	84--	84--	----	----	-8--	
b_4	889D	-2-6	94B2	-242	-A42	2^{-161}
	2311	1-82	64CC	-4--	1-C8	
	4-C-	9C21	242-	-929	3264	
	49-8	899-	-181	432-	121-	
	219-	-9-8	881-	3311	91-2	
a_5	-A-9	82-6	9CB-	-A--	--46	
	-351	1-86	6--4	-4-8	--48	
	68C-	B8-8	2425	4929	22-4	
	4-18	8A9-	4-81	412-	92--	
	218-	A-18	981-	-2-1	B--A	

Table A.16: 5-Round KECCAK-f[400] differential path with probability 2^{-245}

A.6 Differential paths for KECCAK-f[800]

a_0	EBC4D78C	7135E26D	CD789AF-	DE26BC4E	-9AF1357	
	EBC4D78D	7135E26D	CD789AF-	DE26BC5E	-9AF1357	
	EBC4D78D	7135E26D	CD789AF-	DE26BC5E	-9AF1357	
	EBC4D78D	7135E26D	CD789AF-	DE26BC5E	-9AF1357	
	EBC4D78D	7135E26D	CD789AF-	DE26BC5E	-9AF1357	
b_0	-----1	-----	-----	-----	-----	2^{-4}
	-----1	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
a_1	-----1	-----	-----	-----	-----	
	-----1	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
b_1	-----1	-----	-----	-----	-----	2^{-4}
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----1-	-----	-----	-----	
	-----	-----	-----	-----	-----	
a_2	-----1	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----1-	-----	-----	-----	
	-----	-----	-----	-----	-----	

Table A.17: 2-Round KECCAK-f[800] differential path with probability 2^{-8}

a_0	357C9AF1	78DABC4D	2693135E	AF2-D789	C4D9E26B	
	357C9AF1	78DABC4D	2693135E	AF2-D789	C4D9E26B	
	357C9AF1	78DABC4D	2693135E	AF2-D789	C4D9E26B	
	35FC9AF1	78D2BC4D	2693135E	AF2-D789	C4D9E26B	
	357C9AF1	78DABC4D	2693135E	AF2-D789	C4D9E26B	
b_0	-----	-----	-----	-----	-----	2^{-4}
	-----	-----	-----	-----1	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----1	-----	
a_1	-----	-----	-----	-----	-----	
	-----	-----	-----	-----1	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----1	-----	
b_1	-----	-----	-----	-----	-----	2^{-4}
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-1-----	
	-----	--8-----	-----	-----	-----	
a_2	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-1-----	
	-----	--8-----	-----	-----	-----	
b_2	-----	-----	-----4	---4---	-----	2^{-24}
	--2-----	-----	-----	-----	--1-----	
	-----	2-----	---4---	-----1	-----	
	-----	-----	-----	-----4-	---2-----	
	--2-----	---1----	-----	-----	-2-----	
a_3	-----	-----	-----4	---4---	-----	
	--2-----	-----	-----	-----	--1-----	
	-----	2-----	---4---	-----1	-----	
	-----	-----	-----	-----4-	---2-----	
	--2-----	---1----	-----	-----	-2-----	

Table A.18: 3-Round KECCAK-f[800] differential path with probability 2^{-32}

The best 4-round path is the first 4 rounds of this path with 2^{-109} .

a_0	F2BC4561	93135BFC	B-D78F4A	11E26A5B	D49AE288	
	E2BC4571	93135BFC	B-D78F4A	11E26A5B	D49AF-88	
	F2BC4579	93135BFC	B8D78F4A	11E26A5B	D49AE-88	
	F2BC4571	93135BFC	B-D78F4A	11E26A5B	D49AE-88	
	F2BC4571	93135BFC	B-D78F4A	11E26A5B	D49AE-88	
b_0	-----1-	-----	-----4-	-----	-----	2^{-12}
	-----	-----1	-----4-	-----	-----	
	-----	-----	-----	-----	-----	
	-----1-	-----1	-----	-----	-----	
	-----	-----	-----	-----	-----	
a_1	-----1-	-----	-----4-	-----	-----	
	-----	-----1	-----4-	-----	-----	
	-----	-----	-----	-----	-----	
	-----1-	-----1	-----	-----	-----	
	-----	-----	-----	-----	-----	
b_1	-----1-	----1---	-----	-----	-----	2^{-12}
	-----	-----	-----	---2---	-----	
	-----	----1---	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----1-	-----	-----	---2---	-----	
a_2	-----1-	----1---	-----	-----	-----	
	-----	-----	-----	---2---	-----	
	-----	----1---	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----1-	-----	-----	---2---	-----	
b_2	-----1-	-----	-----	-----	-----	2^{-12}
	-----	-----	-----	-----	-----	
	---2---	-----	-----	-----	--4----	
	-----	-----	--4----	-----	-----2-	
	-----	-----1-	-----	-----	-----	
a_3	-----1-	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	---2---	-----	--4----	-----	--4----	
	-----	-----	--4----	-----	-----2-	
	-----1-	-----1-	-----	-----	-----	
b_3	--4--1-	-2-----	---8--2	-8--1--	1-----	2^{-73}
	---8---4	-----4	-2-1----	-4-----	-----2	
	---4---	-----4--	8--1----	--4-2--	--4--1--	
	---2--	-4-----	--8-----	---8--2-	4---8---	
	-----4	2--4---	2-2-----	8-----	---8-4-	
a_4	--4--1-	-2-----	---8--2	-8--1--	1-4-----	
	---8---4	-6-----4	-2-1--2	-4-8---	-----2	
	---4---	-----4--	8--1----	---2---	--4--1--	
	---2--	-4-----	--8-----	---8--2-	4---8---	
	---4--4	---4---	A-2-8---	8-----	---8-4-	
b_4	5-4-895A	8-21443-	-9222CC-	549-55--	2D1-2324	2^{-350}
	48AA-128	4-28C9-3	8--64A52	--42-821	B7-41488	
	8E1--42-	8-491106	-9-54-45	9-B4--CC	25394--3	
	-4E681A-	--8894E5	2-184114	12228C4-	84-2A--2	
	C6--2911	424354-1	681A8-46	-11694A-	1421-841	
a_5	5-6-A5-A	C--1--3-	4-22-BC-	4-8-54-4	2D1-66-4	
	8884-828	4-28C1-2	---242D2	-24--C-9	B7-41482	
	-E1--422	8-E811-2	-831----	11944-CD	293-4-C5	
	-4E6--1-	-28-9-B1	2-184114	12428C6-	8--42--2	
	C612A91-	-2414-81	68-9-8-6	411694A-	122-1-41	

Table A.19: 5-Round KECCAK-f[800] differential path with probability 2^{-459}

A.7 Plot for the pattern-match problem

Figure A.1: Average number of non empty buckets when a pattern-match occurs for $x_B = x_F$ and $s = 320$

A.8 Distributions for the KECCAK- $f[1600]$ rebound attack

In this section, we give all the probability distributions necessary to compute p_{match} and N_{match} . We give also some intermediary results.

A.8.1 P_{out}

P_{out} is the measured probability distribution at the output of an Sbox when all Sboxes are active. Recall that due to the properties of the DDT, we can consider every Sbox independently. The probability distribution is shown in Table A.20.

A.8.2 $\Pr[\text{Hw}_{\text{total}} = w | \text{full}]$

The computation of p_{match} depends greatly on the Hamming weight distribution at the input of the Sboxes of the inbound phase. We measured this distribution by taking a fair amount of samples and noticed that it behaves like a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, with mean μ and variance σ^2 . This behavior is shown in Figure A.2. For the backward paths we are using

y	$P_{\text{out}}(y)$	y	$P_{\text{out}}(y)$
0	0.00000000	10	0.01610571
1	0.01859357	11	0.04363845
2	0.01868644	12	0.04744619
3	0.04682032	13	0.02290569
4	0.01784977	14	0.04548532
5	0.04445059	15	0.02096362
6	0.04309371	16	0.01997166
7	0.02287948	17	0.04545379
8	0.01802242	18	0.04506549
9	0.04543792	19	0.02241770
a	0.04612980	1a	0.01806761
b	0.02005349	1b	0.04705496
c	0.04726458	1c	0.02056551
d	0.02075547	1d	0.04800789
e	0.02080299	1e	0.04603740
f	0.04910975	1f	0.01086272

Table A.20: Distribution of P_{out} for our attack on KECCAK- $f[1600]$

in our first attack (Section 4.3.2), i.e., when $X = 8$ and $k = 9$, this distribution behaves as $\mathcal{N}(847.88, 666.34)$. In the improved attack (Section 4.8, i.e., when $X = 8$ and $k = 8$, this distribution behaves as $\mathcal{N}(847.47, 651.51)$.

A.8.3 $\mathbb{E}[N_{\text{SBox}} | \text{match}, \text{Hw}_{\text{SBox}} = i]$

This values is used to compute N_{match} . It is defined as

$$\frac{1}{\binom{5}{i}} \sum_{\substack{v \in \{0,1\}^5: \\ \text{Hw}(v)=i}} \text{DDT}[v] .$$

For $i = \{1, 2, 3, 4, 5\}$ respectively, $\mathbb{E}[N_{\text{SBox}} | \text{match}, \text{Hw}_{\text{SBox}} = i] = \{8, 4, 3, 2, 2\}$.

A.8.4 Behavior of N_w and $p_{\text{match}}(w)$

Figure A.3 shows how N_w (resp. $p_{\text{match}}(w)$) decreases (resp. increases) exponentially when w grows.

Figure A.2: Hw_{total} (in gray) is distributed following a Gaussian distribution (in black).

Figure A.3: Behavior of N_w and $p_{\text{match}}(w)$

A.8.5 Different $N_{w_{\text{max}}}$ and p_{match} for relevant w_{max}

Given the complexity of the backward and forward differential path, we need to ensure that we generate enough values from the inbound phase. Table A.21 shows the resulting worst case N_{match} and p_{match} for some relevant w_{max} . For p_{match} the first Hamming weight distribution is used, *i.e.*, the distribution when $X = 8$ and $k = 5$.

$w_{\max}$	$\log(N_{w_{\max}})$	$\log(p_{\text{match}})$	$w_{\max}$	$\log(N_{w_{\max}})$	$\log(p_{\text{match}})$
935	520.65	-503.58	968	503.23	-495.57
936	520.11	-503.29	969	502.71	-495.38
937	519.58	-503.00	970	502.19	-495.19
938	519.05	-502.72	971	501.67	-495.00
939	518.51	-502.44	972	501.15	-494.82
940	517.98	-502.16	973	500.64	-494.64
941	517.45	-501.88	974	500.12	-494.46
942	516.91	-501.61	975	499.60	-494.29
943	516.38	-501.34	976	499.09	-494.12
944	515.85	-501.07	977	498.57	-493.95
945	515.32	-500.81	978	498.05	-493.78
946	514.79	-500.55	979	497.54	-493.62
947	514.26	-500.29	980	497.02	-493.46
948	513.73	-500.03	981	496.51	-493.30
949	513.20	-499.78	982	496.00	-493.15
950	512.67	-499.53	983	495.48	-493.00
951	512.14	-499.29	984	494.97	-492.85
952	511.61	-499.05	985	494.46	-492.70
953	511.09	-498.81	986	493.94	-492.56
954	510.56	-498.57	987	493.43	-492.42
955	510.03	-498.34	988	492.92	-492.29
956	509.51	-498.11	989	492.41	-492.15
957	508.98	-497.88	990	491.90	-492.02
958	508.46	-497.65	991	491.39	-491.89
959	507.93	-497.43	992	490.88	-491.76
960	507.41	-497.21	993	490.37	-491.64
961	506.88	-497.00	994	489.87	-491.52
962	506.36	-496.78	995	489.36	-491.40
963	505.84	-496.57	996	488.85	-491.29
964	505.32	-496.37	997	488.35	-491.18
965	504.79	-496.16	998	487.84	-491.07
966	504.27	-495.96	999	487.33	-490.96
967	503.75	-495.77	1000	486.81	-490.86

Table A.21: Values for $N_{w_{\max}}$ and p_{match} for different relevant $w_{\max}$

A.9 Differential paths for KECCAK- $f[1600]$

The first 2 rounds form a best 2-round differential path with probability 2^{-8} .

a_0	F135E26BC449789A	4D789AF135A226BC	5E26BC4D78B5AF13	89AF135E268FC4D7	6BC4D789AF1D35E2	
	F135E26BC449789A	4D789AF135A226BC	5E26BC4D78B5AF13	89AF135E268FC4D7	6BC4D789AF1D35E2	
	F135E26BC449789A	4D789AF135A226BC	5E26BC4D78B5AF13	89AF135E268FC4D7	6BC4D789AF1D35E2	
	F135E26BC4C9789A	4D789AF135AA26BC	5E26BC4D78B5AF13	89AF135E268FC4D7	6BC4D789AF1D35E2	
	F135E26BC449789A	4D789AF135A226BC	5E26BC4D78B5AF13	89AF135E268FC4D7	6BC4D789AF1D35E2	
b_0	-----	-----	-----	-----	-----	2^{-4}
	-----	-----	-----	-----1	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----1	-----	
a_1	-----	-----	-----	-----	-----	
	-----	-----	-----	-----1	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----1	-----	
b_1	-----	-----	-----	-----	-----	2^{-4}
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-1-----	
	-----	-----	-----	-----	-----	
	-----	--8-----	-----	-----	-----	
a_2	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-1-----	
	-----	-----	-----	-----	-----	
	-----	--8-----	-----	-----	-----	
b_2	-----	-----	-----4-----	-----4-----	-----	2^{-24}
	-----2-----	-----	-----	-----	--1-----	
	-----	2-----	-----4-----	-----1-----	-----	
	-----	-----	-----	-----4-----	--2-----	
	--2-----	--1-----	-----	-----	-2-----	
a_3	-----	-----	-----4-----	-----4-----	-----	
	-----2-----	-----	-----	-----	--1-----	
	-----	2-----	-----4-----	-----1-----	-----	
	-----	-----	-----	-----4-----	--2-----	
	--2-----	--1-----	-----	-----	-2-----	

Table A.22: 3-round differential path for KECCAK- $f[1600]$ with probability 2^{-32}

The first 3 rounds forming a 3-round differential path with probability 2^{-36} , is used in the rebound attack as forward path.

a_0	D78BE9AF44D1AF44	E26A4BC4B5E3C4B5	9AF-D135F88935F8	BC4D2D79A6B579A6	135AFE262F16262F	
	D78BE9AF44D1AF44	E26A-BC4B5E3C4B5	9AF-D135F88935F8	BC4D2D79A6B579A6	135AFE262F16262F	
	D78BE9AF44D1AF44	E26ACBC4B5E3C4B5	9AF-D135F88935F8	BC4D2D78A6B579A6	135AFE262F16262F	
	D78BE9AF44D1AF44	E26A4BC4B5E3C4B5	9AF-D135F88935F8	BC4D2D79A6B579A6	135AFE262F16262F	
	D78BE9AF44D1AF44	E26A4BC4B5E3C4B5	9AF-D135F88935F8	BC4D2D79A6B578A6	135EFE262F16262F	
b_0	-----	-----4-----	-----	-----	-----1	2^{-12}
	-----	-----4-----	-----	-----	-----	
	-----	-----4-----	-2-----	-----	-----	
	-----	-----	-2-----	-----	-----1	
	-----	-----	-----	-----	-----	
a_1	-----	-----4-----	-----	-----	-----1	
	-----	-----4-----	-2-----	-----	-----	
	-----	-----4-----	-2-----	-----	-----1	
	-----	-----	-----	-----	-----	
	-----	-----	-----	-----	-----	
b_1	-----	-----	-----1-----	-----	-----	2^{-12}
	-----	-----	-----	-----	-----	
	-----8-----	-----	-----	-----1--	-----	
	-----8-----	-----	-----1-----	-----1--	-----	
	-----	-----	-----	-----	-----	
a_2	-----	-----	-----1-----	-----	-----	
	-----	-----	-----	-----	-----	
	-----8-----	-----	-----	-----1--	-----	
	-----8-----	-----	-----1-----	-----1--	-----	
	-----	-----	-----	-----	-----	
b_2	-----	-----	-----4-----	-----2-----	-----	2^{-12}
	-----	-----	-----2-----	-----	-----	
	-----	-----	-----	-----8-----	-----	
	-----4-----	-----	-----	-----1-	-----	
	-----	-----	-----	-----	-----	
a_3	-----	-----	-----4-----	-----2-----	-----	
	-----	-----	-----2-----	-----	-----	
	-----	-----	-----	-----8-----	-----	
	-----4-----	-----	-----	-----1-	-----	
	-----	-----	-----	-----	-----	
b_3	-----	-----8--	---1---8---12--	--48-----1--	--2-8-----4--2	2^{-106}
	26-----	--82-----1---8-	-----	-----1--	--2---8-----4	
	-----1-----	-4-----8--	-48-----	-8--82-----1--	--1-----	
	41-----8--4--	-----	-----2-----	--2---1---8	1-----24-----	
	--4---1-----8	-----12-----	---8---4---41-	-----2-----	-----	
a_4	-----	-----8--	---1---8---12--	--48-----1--	--2-8-----4--2	
	26-----	--82-----1---8-	-----	-----1--	--2---8-----4	
	-----1-----	-4-----8--	-48-----	-8--82-----1--	--1-----	
	41-----8--4--	-----	-----2-----	--2---1---8	1-----24-----	
	--4---1-----8	-----12-----	---8---4---41-	-----2-----	-----	

Table A.23: 4-round differential path for KECCAK-f[1600] with probability 2^{-142}

a_0	4---2A6981735E2	C---314BD2F789A	4----9913DC26BC	C---9C6D77FAF13	C---48BE251C4D7	
	4---2A6981735E2	C---314BD2F789A	4----9913DC26BC	4---9C6D77FAF13	C---48FE251C4D7	
	4---2A69A1735E2	C---314BD2B789A	4----9913DC26BC	C---9C6D77FAF13	C---48BE251C4D7	
	4---2A6989735E2	C---314BD27789A	4----9913DC26BC	C---9C6D77FAF13	C---48BE251C4D7	
	4---2A6981735E2	C---314BD2F789A	4----D913DC26BC	C---1C6D77FAF13	C---48BE251C4D7	
b_0	-----	-----	-----	-----	-----	2^{-16}
	-----	--4-----	-----1-----	-----1-----	-----8-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----1-----	-----	-----8-----	
	-----	--4-----	-----	-----1-----	-----	
a_1	-----	-----	-----	-----	-----	
	-----	--4-----	-----1-----	-----1-----	-----8-----	
	-----	-----	-----	-----	-----	
	-----	-----	-----1-----	-----	-----8-----	
	-----	--4-----	-----	-----1-----	-----	
b_1	-----	-----4-----	-----	-----	-----	2^{-16}
	-----	--8-----	-----	-----	-----	
	-----	-----4-----	-----	-----8-----	-----	
	-----	-----	-----	-----8-----	-1-----	
	-----	--8-----	-----	-----	-1-----	
a_2	-----	-----4-----	-----	-----	-----	
	-----	--8-----	-----	-----	-----	
	-----	-----4-----	-----	-----8-----	-----	
	-----	-----	-----	-----8-----	-1-----	
	-----	--8-----	-----	-----	-1-----	
b_2	-----	-----8-----	-----	-----1-----	-----4-----	2^{-16}
	-----	-----	-----	-----	-----	
	-----8-----	-----	-----1-----	-----1-----	-----	
	-----	-----	-----1-----	-----	-----	
	-----	-----	-----	-----	-2-----	
a_3	-----8-----	-----8-----	-----	-----1-----	-----4-----	
	-----	-----	-----	-----	-----	
	-----8-----	-----	-----1-----	-----1-----	-----	
	-----	-----	-----1-----	-----	-----	
	-----	-----	-----	-----	-2-----	
b_3	-2---18---4-	--2---2-----	--8---4---	-----12-8-2	-----8-	2^{-114}
	-----91-4-1-	-----	1---C---2-	--4---4-----	-----1-----	
	---4-1-----4-	---2-----	-----122-8-2-	-----	-4-----1---8-	
	-----2-----	---4-2---1-	-8-----8---	-8-4-----	9--4-1-----	
	-----2-----	48-2---8-----	-----	---8-4---2---	---8-----8-	
a_4	-2---18---4-	--2-----	--8---2-C---	-----12-8-2	-----8-	
	-----91-4-1-	-----	1---C1---2-	--4---14-----	-----	
	---4-11-----4-	---2-2-----	-----122-8-2-	-----	-4---1-1---8-	
	-----2-----1-	-8---4-2---1-	1-8-----8---	-8-4-----	9--4-1-----	
	-----2-----	48-2---8-----	---8-----	-----4---2---	---8-----8-	
b_4	12448C-8-1---A4-	-844--3--418-5-5	4A-9228---3--97-	18-441-A4425-1A-	---1-9A-282-83--	2^{-547}
	-3--8D2232--9--C	--4268-A--2-C--8	82266--8-8--52--	1-A8-16--8B--E-A	8A--1-C--18-282-	
	-6-483--A-A1-88-	--1B-43--5-C11-	8-441-A4424-12-1	-48-14268-A--29C	3-48-4--28--4112	
	2134-5-41-6-4--	14--B--1-448C191	-1-E-1C14211---C	-3--6--E-822-4-	212--9--C422-872	
	14--218--38-6-41	1-9--68-6-1B-429	C-54-1-62-4--A13	--16--2-891824-2	2C-9-621414211-1	
a_5	1--48C-8-5---24-	--41--3--4-8-425	42-963----11-8D-	-8--4--244-5--A-	---1-18-2C2-814-	
	-32-8D2832--1--4	--4--8-2---8--8	---66-48---5--8	1-A81---83-26-E	884-18E--18-382-	
	-6---32-A-A1188-	-4C1B-41--1-C18-	8-44--86424-5285	-28-12262--1-29C	3-48-44-88-----2	
	213-24-4142-4484	14--B6C1445BC195	-2-E--C12211--4C	-234-2-4A4A22-2-	---1D-1C4628862	
	141426-4-3816E4-	1-C--6A24-43-438	C-45-3-76-4--A11	181E--2-C89A-4-2	38-9--2161C231--	

Table A.24: 5-round differential path for KECCAK-f[1600] with probability 2^{-709}

A.10 Example of valid pairs for KECCAK-f[100]

Each 4-bit hex number represents the lane value with LSB for the first slice.

Position	Difference		Value 1					Value 2					Complexity
a_0	- - - 2 4		2	9	4	E	7	2	9	4	C	3	2^8
	- - - - -		4	5	2	D	9	4	5	2	D	9	
	- - - - -		-	F	7	7	8	-	F	7	7	8	
	- - - 2 4		3	6	8	5	4	3	6	8	7	-	
	- - - - -		E	B	3	3	F	E	B	3	3	F	
a_1	- - - 4 -		4	B	C	8	3	4	B	C	C	3	2^8
	2 - - - -		F	D	2	D	3	D	D	2	D	3	
	- - - 4 -		2	3	6	5	B	2	3	6	1	B	
	2 - - - -		1	5	6	-	6	3	5	6	-	6	
	- - - - -		3	6	F	C	1	3	6	F	C	1	
a_2	- - - - -		6	5	6	5	A	6	5	6	5	A	rebound
	4 - - 4 -		1	-	1	9	8	5	-	1	D	8	
	8 8 8 - -		C	3	B	D	F	4	B	3	D	F	
	2 2 - - 2		4	B	A	C	F	6	9	A	C	D	
	- 4 4 4 -		C	8	D	2	D	C	C	9	6	D	
a_3	F 6 5 B -		B	5	8	3	5	4	3	D	8	5	2^{11}
	7 E 1 B 1		D	2	C	E	E	A	C	D	5	F	
	D 6 1 B 2		3	A	3	6	A	E	C	2	D	8	
	7 E 1 B -		-	4	B	9	2	7	A	A	2	2	
	F 6 9 B -		2	E	2	2	F	D	8	B	9	F	
a_4	- 8 - - -		-	5	2	4	4	-	D	2	4	4	
	- - - 1 -		C	5	A	9	A	C	5	A	8	A	
	- - - - -		F	2	2	3	-	F	2	2	3	-	
	- 8 - - -		D	9	1	E	7	D	1	1	E	7	
	- - - 1 -		D	8	3	A	1	D	8	3	B	1	

Table A.25: Example of 4-round differential path for KECCAK-f[100] and valid pairs verifying this path

Appendix B

Meet-in-the-Middle Attacks

B.1 Derived sub-keys of KTANTAN

Two bits of the 80-bit $K = k_{79}k_{78} \dots k_1k_0$ are selected each round, we list in Table B.1 indices of these two bits per round.

rounds	$(k_{a,r}, k_{b,r})$									
1-10	63, 31	31, 63	31, 63	15, 47	14, 14	60, 76	40, 40	49, 17	35, 67	54, 22
11-20	45, 77	58, 26	37, 69	74, 10	69, 69	74, 10	53, 21	43, 43	71, 7	63, 79
21-30	30, 62	45, 45	11, 11	54, 70	28, 60	41, 41	3, 19	38, 70	60, 28	25, 73
31-40	34, 34	5, 21	26, 74	20, 52	9, 41	2, 18	20, 68	24, 56	1, 33	2, 2
41-50	52, 68	24, 56	17, 49	3, 35	6, 6	76, 76	72, 8	49, 17	19, 51	23, 55
51-60	15, 63	14, 46	12, 28	24, 72	16, 48	1, 49	2, 34	4, 20	40, 72	48, 16
61-70	17, 65	18, 50	5, 53	10, 58	4, 36	8, 8	64, 64	64, 0	65, 1	51, 19
71-80	23, 55	47, 47	15, 15	78, 78	76, 12	73, 9	67, 3	55, 23	47, 47	63, 31
81-90	47, 79	62, 30	29, 77	26, 58	5, 37	10, 26	36, 68	56, 24	33, 65	50, 18
91-100	21, 69	42, 42	5, 5	58, 74	20, 52	25, 57	3, 51	6, 38	12, 12	56, 72
101-110	16, 48	33, 33	3, 3	70, 70	60, 28	41, 41	67, 3	71, 71	78, 14	77, 13
111-120	59, 27	39, 39	79, 15	79, 79	62, 30	45, 45	59, 27	23, 71	46, 46	13, 29
121-130	42, 74	52, 20	41, 73	66, 2	53, 69	42, 42	53, 21	27, 75	38, 38	13, 13
131-140	74, 74	52, 20	25, 57	35, 35	7, 7	62, 78	44, 44	73, 9	51, 67	22, 54
141-150	29, 61	11, 43	6, 22	44, 76	72, 8	65, 65	50, 18	37, 37	75, 11	55, 71
151-160	46, 46	77, 13	75, 75	70, 6	61, 29	27, 59	39, 39	15, 31	46, 78	76, 12
161-170	57, 73	34, 34	69, 5	59, 75	38, 38	61, 29	43, 75	70, 6	77, 77	58, 26
171-180	21, 53	43, 43	7, 23	30, 78	44, 44	9, 25	18, 66	36, 36	9, 9	50, 66
181-190	36, 36	57, 25	19, 67	22, 54	13, 45	10, 10	68, 68	56, 24	17, 49	19, 51
191-200	7, 39	14, 30	28, 76	40, 40	1, 1	66, 66	68, 4	57, 25	35, 35	55, 23
201-210	31, 79	30, 62	13, 61	10, 42	4, 4	72, 72	48, 16	33, 33	51, 19	39, 71
211-220	78, 14	61, 77	26, 58	21, 53	11, 59	6, 54	12, 44	8, 24	32, 64	64, 0
221-230	49, 65	18, 50	37, 37	11, 27	22, 70	28, 60	9, 57	2, 50	4, 52	8, 40
231-240	0, 0	48, 64	32, 32	65, 1	67, 67	54, 22	29, 61	27, 59	7, 55	14, 62
241-250	12, 60	8, 56	0, 32	0, 16	16, 64	32, 32	1, 17	34, 66	68, 4	73, 73
251-254	66, 2	69, 5	75, 11	71, 7						

Table B.1: The list of derived sub-keys of the KTANTAN family

B.2 Derived sub-keys of DES

Round	Subkey bits
1	10 51 34 60 49 17 33 57 2 9 19 42 3 35 26 25 44 58 59 1 36 27 18 41 22 28 39 54 37 4 47 30 5 53 23 29 61 21 38 63 15 20 45 14 13 62 55 31
2	2 43 26 52 41 9 25 49 59 1 11 34 60 27 18 17 36 50 51 58 57 19 10 33 14 20 31 46 29 63 39 22 28 45 15 21 53 13 30 55 7 12 37 6 5 54 47 23
3	51 27 10 36 25 58 9 33 43 50 60 18 44 11 2 1 49 34 35 42 41 3 59 17 61 4 15 30 13 47 23 6 12 29 62 5 37 28 14 39 54 63 21 53 20 38 31 7
4	35 11 59 49 9 42 58 17 27 34 44 2 57 60 51 50 33 18 19 26 25 52 43 1 45 55 62 14 28 31 7 53 63 13 46 20 21 12 61 23 38 47 5 37 4 22 15 54
5	19 60 43 33 58 26 42 1 11 18 57 51 41 44 35 34 17 2 3 10 9 36 27 50 29 39 46 61 12 15 54 37 47 28 30 4 5 63 45 7 22 31 20 21 55 6 62 38
6	3 44 27 17 42 10 26 50 60 2 41 35 25 57 19 18 1 51 52 59 58 49 11 34 13 23 30 45 63 62 38 21 31 12 14 55 20 47 29 54 6 15 4 5 39 53 46 22
7	52 57 11 1 26 59 10 34 44 51 25 19 9 41 3 2 50 35 36 43 42 33 60 18 28 7 14 29 47 46 22 5 15 63 61 39 4 31 13 38 53 62 55 20 23 37 30 6
8	36 41 60 50 10 43 59 18 57 35 9 3 58 25 52 51 34 19 49 27 26 17 44 2 12 54 61 13 31 30 6 20 62 47 45 23 55 15 28 22 37 46 39 4 7 21 14 53
9	57 33 52 42 2 35 51 10 49 27 1 60 50 17 44 43 26 11 41 19 18 9 36 59 4 46 53 5 23 22 61 12 54 39 37 15 47 7 20 14 29 38 31 63 62 13 6 45
10	41 17 36 26 51 19 35 59 33 11 50 44 34 1 57 27 10 60 25 3 2 58 49 43 55 30 37 20 7 6 45 63 38 23 21 62 31 54 4 61 13 22 15 47 46 28 53 29
11	25 1 49 10 35 3 19 43 17 60 34 57 18 50 41 11 59 44 9 52 51 42 33 27 39 14 21 4 54 53 29 47 22 7 5 46 15 38 55 45 28 6 62 31 30 12 37 13
12	9 50 33 59 19 52 3 27 1 44 18 41 2 34 25 60 43 57 58 36 35 26 17 11 23 61 5 55 38 37 13 31 6 54 20 30 62 22 39 29 12 53 46 15 14 63 21 28
13	58 34 17 43 3 36 52 11 50 57 2 25 51 18 9 44 27 41 42 49 19 10 1 60 7 45 20 39 22 21 28 15 53 38 4 14 46 6 23 13 63 37 30 62 61 47 5 12
14	42 18 1 27 52 49 36 60 34 41 51 9 35 2 58 57 11 25 26 33 3 59 50 44 54 29 4 23 6 5 12 62 37 22 55 61 30 53 7 28 47 21 14 46 45 31 20 63
15	26 2 50 11 36 33 49 44 18 25 35 58 19 51 42 41 60 9 10 17 52 43 34 57 38 13 55 7 53 20 63 46 21 6 39 45 14 37 54 12 31 5 61 30 29 15 4 47
16	18 59 42 3 57 25 41 36 10 17 27 50 11 43 34 33 52 1 2 9 44 35 26 49 30 5 47 62 45 12 55 38 13 61 31 37 6 29 46 4 23 28 53 22 21 7 63 39

Table B.2: Derived sub-key bits for all the 16 rounds of DES