Contributions to Cryptanalysis:

Design and Analysis of Cryptographic Hash Functions

By

Przemysław Szczepan Sokołowski

A thesis submitted to Macquarie University

for the degree of

Doctor of Philosophy

Department of Computing, Faculty of Science

August 2012

Abstract

A cryptographic hash function is a mechanism producing a fixed-length output of a message of arbitrary length. It fullfils a collection of security requirements guaranteeing that a hash function does not introduce any weakness into the system to which it is applied. The example applications of cryptographic hash functions include digital signatures and message authentification codes. This thesis analyzes cryptographic hash functions and studies the design principles in the construction of secure cryptographic hash functions.

We investigate the problem of building hash functions from block ciphers and the security properties of different structures used to design compression functions. We show that we can build open-key differential distinguishers for Crypton, Hierocrypt-3, SAFER++and Square. We know that our attack on SAFER++ is the first rebound attack with standard differentials. To demonstrate the efficiency of proposed distinguishers, we provide formal proof of a lower bound for finding a differential pair that follows a truncated differential in the case of a random permutation. Our analysis shows that block ciphers used as the underlying primitive should also be analyzed in the open-key model to prevent possible collision attacks.

We analyze the `IDEA`-based hash functions in a variety of cipher *modes*. We present practical complexity collision search attacks and preimage attacks, where we exploit a null weak-key and a new non-trivial property of IDEA. We prove that even if a cipher is considered secure in the secret-key model, one has to be very careful when using it as a building block in the hashing *modes*.

Finally, we investigate the recent rotational analysis. We show how to extend the rotational analysis to subtractions, shifts, bit-wise Boolean functions, multi

additions and multi subtractions. In particular, we develop formulae for calculation of probabilities of preserving the rotation property for multiple modular additions and subtractions. We examine S-functions and its application to the rotational analysis. The findings are applied to BMW and SIMD. We also propose a new shift distinguisher and apply it to Shabal.

Contents

Acknowledgements

I would like to express my gratitude to my supervisors, Prof. Josef Pieprzyk for his knowledge and wisdom which he is always keen to share. His guidance and care have been priceless to me and I am forever indebted to him. I would like to thank Prof. Jerzy Jaworski, who inspired me to start research in the field of cryptology and has been my guide ever since. I am also grateful to Dr Ron Steinfeld for his encouragement and suggestions for improvements during my studies.

The completion of this thesis could not have been achieved without collaboration with Ivica Nikolić, Lei Wei and Thomas Peyrin. Their fresh look on analyzed problems and many discussions were an inspiration for my research.

I would like to thank my parents Maria and Stanisław for their unconditional support and understanding throughout my PhD candidature. Finally, I would like to express my gratitude to my wife Urszula, for her love and faith in me.

My research was supported by Macquarie University via a Cotutelle Macquarie University Research Scholarship.

Declaration

This thesis is submitted in fulfilment of the requirements of the degree of Doctor of Philosophy at Macquarie University and has not been submitted for a higher degree to any other university or institution. This thesis represents my original work and contributions. I certify that to the best of my knowledge, all sources and assistance received in the preparation of this thesis have been acknowledged.

Przemysław Szczepan Sokołowski

List of Publications

1. Ivica Nikolić, Josef Pieprzyk, Przemysław Sokołowski and Ron Steinfeld, *Known and Chosen Key Differential Distinguishers for Block Ciphers*, in Information Security and Cryptology - ICISC 2010, *Lecture Notes in Computer Science* **6829**, Springer 2011, pp. 29 – 48

2. Lei Wei, Thomas Peyrin, Przemysław Sokołowski, San Ling, Josef Pieprzyk and Huaxiong Wang *On the (In)Security of IDEA in Various Hashing Modes*, in Fast Software Encryption 2012 - FSE 2012, *Lecture Notes in Computer Science*, to be published. Extended version available as:

- Lei Wei, Thomas Peyrin, Przemysław Sokołowski, San Ling, Josef Pieprzyk and Huaxiong Wang *On the (In)Security of IDEA in Various Hashing Modes*, in IACR Cryptology ePrint Archive, Report 2011/420, 2011

3. Ivica Nikolić, Josef Pieprzyk, Przemysław Sokołowski and Ron Steinfeld *Rotational Cryptanalysis of (Modified) Versions of BMW and SIMD*, Available online at `https://cryptolux.org/mediawiki/uploads/0/07/Rotational_distinguishers_%28Nikolic%2C_Pieprzyk%2C_Sokolowski%2C_Steinfeld%29.pdf`

1. Introduction

A cryptographic hash function is a transformation that maps an arbitrary length input, called the "message", into a fixed-length output, the "message digest". It is designed to be easily computable and has to achieve certain security properties, e.g.: preimage resistance, second preimage resistance, and collision resistance.

Cryptographic hash functions are crucial parts of many cryptographic algorithms like digital signatures, message authentication algorithms and commitment protocols to name a few. For example, digital signatures can be of a fixed length no matter how long the signed messages are. This is normally done by signing the message digest (of a fixed length) instead of signing the whole message. Note that finding two messages that have the same digest immediately allows an adversary to replace the message with its colliding sibling as the receiver making it impossible to determine which of the two colliding messages is genuine.

1.1. Cryptographic Hash Function Properties

As previously mentioned, the fundamental properties of the cryptographic hash functions are:

1. preimage resistance – given digest $d = H(M)$ for a message M, it is computationally difficult to find any message that gives the digest,

2. second preimage resistance – given message M, it is computationally difficult to find a different message M' that gives the same digest, i.e. a message M' such that $H(M) = H(M')$,

3. collision resistance – it is computationally difficult to find two different messages M, M' that give the same digest, i.e. for two messages M and M', $H(M) = H(M')$,

where H is the hash function that takes a message M of arbitrary length and produces a fixed length digest (formal definition of the three properties is provided in Section 2.1). Computational difficulty means that an attack is infeasible to be conducted due to computational restrictions of available hardware. In the above cases it is assumed that the asymptotic lower bound for the intractability is $O(2^n), O(2^n), O(2^{\frac{n}{2}})$ hash operations for the n-bit digest, respectively. In practice, for fixed-sized digest hash functions it is assumed that the asymptotic bounds are instantiated to $2^n, 2^n$ and $2^{\frac{n}{2}}$ calculations of a hash function, respectively. For example, SHA-0 is a hash function with 160-bit digest and is expected to withstand any collision search attack with complexity less than 2^{80} hash calculations. However, the attack presented in [111] reveals collisions for SHA-0 with complexity 2^{39} hash operations, which breaks the collision resistance of the function.

Another set of properties required from cryptographic hash functions is collectively called "pseudorandomness". In particular the SHA-3 call [40] specifies that the future SHA-3 algorithm should support:

- construction of deterministic Pseudo Random Function (PRF) with use of HMAC,
- randomized hashing.

In the first case a PRF obtained from HMAC must be resistant to any distinguishing attack with complexity less than $2^{\frac{n}{2}}$ and significantly smaller than preimage attack. On the other hand the randomized hashing schema should withstand an attack defined in the challenge-response manner, that is the adversary should be unable to obtain a second message M' and an additional parameter r' for chosen M and random r that applied to the schema produce the same hash value. Finally, it is also required that a hash function do not reveal any nonrandom properties or fail any statistical test.

1.2. Classification of Hash Functions

In general hash functions can be divided into two main categories:

1. keyed hash functions, referred to as Message Authentication Codes (MAC),
2. unkeyed hash functions, referred to as Modification Detection Codes (MDC).

Informally, the class of keyed hash functions (see [92] for a formal definition) is defined as a family of hash functions H_K indexed by a secret key K with an additional property called the computation-resistance – given any set of pairs $(M_i, H_K(M_i))$ for some $i \in \mathbb{N} \cup \{0\}$ it is computationally difficult to find $H_K(M)$ for any $M \notin \{M_i : i \in \mathbb{N} \cup 0\}$ even if $H_K(M) \in \{H_K(M_i) : i \in \mathbb{N} \cup 0\}$. Their main purpose is to provide mechanisms for authenticity and integrity check. That is to say that only parties sharing the same secret key can verify if the message was altered and if the MAC was generated with the correct secret key.

The second class – Modification Detection Codes – is a class of unkeyed hash functions, which in contrast to MAC has only one parameter – message input. Its main purpose is to produce "unique" hash value for any message, which provides mechanisms for data integrity checks.

It is also possible to produce a MAC from an unkeyed hash function. An example of such construction is HMAC (short for Hash-based Message Authentication Code) which incorporates secret key K in calculations of MAC for M by following formula:

$$HMAC(K, M) = H((K \oplus opad)||H((K \oplus ipad)||M))$$

where H is unkeyed hash function and $opad, ipad$ are constants (for details see [69]).

The MDC class can be further divided into following subclasses:

- One-Way Hash Functions (OWHF),
- Collision Resistant Hash Functions (CRHF).

A OWHF is a hash function that fulfills two security requirements: preimage resistance and second preimage resistance. Some applications of OWHF are mes-

sage integrity validation, authentication or password verification. A CRHF is on the other hand a hash function that is collision resistant and second preimage resistant.

1.3. Hash Functions Applications

The main application of hash functions is to generate "unique" and fixed length sequence of data for a given input. Cryptographic hashing is used in many applications such as:

- digital signatures,
- integrity checking,
- message authentication codes,
- commitments,
- password storage,
- encryption algorithms,
- software protection.

Digital signatures. Digital signature algorithms based on asymmetric cryptosystems are computationally inefficient in case of long messages. In order to improve their performance and make signatures of a fixed length no matter how long the signed messages, cryptographic hash functions are used to produce a hash digest for the message, which is then signed. Verifing a signature is done for the digest of the message. Note that finding two messages that have the same digest immediately allows an adversary to replace the message with its colliding sibling as the receiver has no way to determine which of the two colliding messages is genuine. Hence, security of such digital signatures largely depends on security of used cryptographic hash function.

Integrity checking. Hash functions are also applied to verifing the integrity of data sent over error prone communication channel. The sender calculates a

hash value for a message and then sends a sequence of data to a receiver. At the same time, the message digest is sent over another channel which is reliable, so that the receiver can compare the digest with hash value of the received message. If the hash values are the same we might be assured that the message was not altered during communication. However, note that if finding another message with the same digest as the hash value of the original is easy, an adversary is able to manipulate with the last sent in the communication channel and receiver can be tricked.

Message authentication codes. In order to authenticate a message keyed hash functions can be utilized. The possible schema of message authentication is similar to the one for integrity checking. The difference is in the secret key provided by the sender in the hash value calculations. Assuming that cryptographic hash function is MAC, the receiver, who knows the secret used, can identify whether the message has been altered or if it was sent by the authorized sender. If the digest of received message under the secret key is different to the digest accompanying the message, the last one was changed or generated with different key. Of course digital signatures can also be used for message authentication but hash functions have advantage over them in having a much lower complexity to necessary calculations.

Commitments. Another application of cryptographic hash functions are commitment protocols. For example consider such a simple protocol described as follows (see [51]):

Alice is in possession of some initially secret information. Let it be the sentence "The answer to the fundamental question is: YES!". She does not want to reveal it to Bob at this moment, but she will need to prove in the future that she had already known this sentence. So Alice hashes concatenation of a randomly chosen pad and her secret, showing to Bob only the computed digest. Then Bob knows nothing except the digest of Alice's secret. When the right time comes, Alice can prove that she had known the secret sentence by providing the random pad she used earlier. Bob can verify Alice's veracity by comparing the digest he received at first with recomputed hash value of the concatenation of the random string and

the revealed secret.

If we assume that the hash function used in the protocol is preimage resistant, Bob is not able to efficiently guess Alice's secret information. Even though Bob cannot deceive Alice, she can "prove" she knew another secret, if she is able to find for it different random pad for which the digest of their concatenation is identical to the first hash value. However, if the hash function used in the schema is collision resistant the trick becomes computationally infeasible. Unfortunately the schema does not provide adequate security for Alice, because Bob can still distinguish between possible secrets (for more details see [51] where the example is discussed in more detail).

Password storage. Password authentication mechanism for controlling access to IT resources is one more application of cryptographic hash functions. For instance operating systems like Windows or Unix are storing hash values of users' passwords. In order to authenticate to the IT system a user has to provide a secret password, which hash value is then compared to one stored in the database. The user is granted access if both hash values are equal. The main reason for storing hash values of passwords and not exact passwords is to guarantee that access is granted only to authorized users even if an adversary controls the database. Hence, we require that hash functions be preimage resistant, so that stored hashes can not be easily inverted.

Encryption algorithms. Hash functions can also be used as building blocks of ciphers. For instance they can be used as nonlinear blocks within a cipher, for instance as F-function in Feistel network designs. Another way to utilize a hash function in encryption is to use it as a key generator from secret password. The hash value of the password is forwarded from key input to the cipher in order to execute cipher or decipher procedure.

Software protection. Protecting software from third party modification, e.g. viruses, or restricting execution of authenticated programs can also be achieved with use of cryptographic hash functions. The simplest way to achieve the first goal is to generate an hash digest of the program and distribute it with program. However, this solution does not protect from forging another hash digest for

modified program. A remedy is a digital signature for the hash value of the program. An example of such a mechanism is Microsoft Authenticate present in Windows family of operating systems.

1.4. Cryptanalysis

Analysis of any cryptographic primitive can be divided into two main streams:

- generic – independent on algorithm,
- algorithm specific.

1.4.1. Generic Analysis

The generic approach does not depend on the internal structure of the subject of analysis, which is treated as a black-box with input and output interface. The attacker might only provide input data, which might be altered depending on the observed results of black-box calculations. Hence, their general assumption is the pseudo-randomness of the analyzed hash function. We can select three main attack strategies of generic attacks on hash functions:

- brute force attack,
- birthday attack,
- meet-in-the-middle attack,

which will be briefly presented below while more details can be found in Section 2.5.

Brute Force Attack

The brute force attack is the simplest one of the three and the most expensive. In the attack, the attacker tries all possible input values until the expected output has been generated. For example, brute force preimage search attack needs to check on average 2^{n-1} values before the right message is found.

Birthday Attack

The second method is fundamental to analysis of cryptographic hash functions, especially by finding collisions. The birthday attack (described in [113]) and its altered version, the generalized birthday attack (see [47]) are improvements on the brute force attack. The complexity of the birthday attack (in the case of hash function which produces n-bit digest, its asymptotic complexity is $O(2^{\frac{n}{2}})$) provides an upper bound for the security of any cryptographic hash function against collision search attacker.

Meet-in-the-Middle Attack

The meet-in-the-middle strategy applies to iterated designs but also to one that can be divided into two independent parts. For instance, let $f_k = g_{k_2} \circ h_{k_1}$ be a such function, where k_1 and k_2 are independent parts of some parameter k. The general idea of this kind of attacks is to pick random k_1 and k_2 and compute $h(x)$ and $g^{-1}(y)$ for challenge pair (x, y). Due to independence of h and g the match is found with the birthday bound complexity.

1.4.2. Algorithm Specific Analysis

The second group of cryptographic tools are design dependent. They exploit internal structure of analyzed algorithm in order to detect its unwanted properties. We list some but not all attacks in the group:

- differential attack,
- rebound attack,
- linear attack,
- random graph theory attack,
- distinguishing attack.

Differential Analysis

In recent years it has also become obvious that differential analysis, originated by cryptanalysis of symmetric ciphers (see for details [17]), is also applicable in the case of cryptographic hash functions like MD4, MD5, SHA–0, SHA–1, etc. (see [88, 108, 109, 111]). Generally this method is based on finding a correlation between the differences (XOR or modular) in input and output of a cipher or a cryptographic hash function. In the case of hash functions the difference in output should be equal to zero to produce a full collision or differ slightly to obtain a so called near-collision.

Rebound Attack

Further improvements of differential paths is possible due to an application of the meet-in-the-middle approach. By merging two differential paths with use of available degrees of freedom, longer paths are produced. The example of such an approach is the rebound attack proposed by Mendel et al. in [91], which resulted in attacks on cryptographic hash functions like: Grøstl, Whirlpool, ECHO, etc. (see [90, 91]) and also allowed better cryptanalysis of AES e.g. [46].

Linear Analysis

A very interesting method of breaking hash functions is one based on approximating the internal states of the cryptographic hash function. This method also has its origin also in cipher analysis e.g. [87] and is similar to differential analysis. For example, [30] a linear attack was presented based on finding solutions of a system of non linear equations describing internal states of the LASH hash function (described in [9]).

Random Graph Theory Attacks

More sophisticated methods of collision search are $\rho-$ and $\lambda-$Pollard ($\rho-$Pollard method is described in [6, 101, 104] and $\lambda-$Pollard in [105, 107]). The methods are based on the structure of a random mapping directed graph (digraph), while

making the assumption of a uniform distribution of the image of the cryptographic hash function f, i.e. for any uniformly distributed input its image under the function is also uniformly distributed. Basically both require finding two distinct points in the same connected component on the function digraph such that there is no directed path between either of them.

Distinguishing Attacks

The above-mentioned assumption is also extended to the case where inputs to the function might be somehow related, e.g. the input distribution is not necessarily uniform. In this case so called distinguishing attacks are considered, see for example [67, 94], where a variety of properties are tested for instance: q-multicollisions [20], preservation of rotations [58], etc. A distinguisher plays a central role in the attack, which is basically a probabilistic algorithm interacting with two oracles: one that simulates an analyzed primitive and the other simulates an ideal primitive, for example random permutation. The aim of the distinguisher is to decide which of the two is the analyzed primitive, based on queries provided to the oracles. The attack is considered to be successful if the number of queries required to make a correct decision is below a well defined level.

1.5. Secure Hash Standards

The first standard of secure hashing [41] was adopted by the United States National Institute of Standards and Technology (NIST) in 1993. It is commonly referred to as SHA-0 (SHA stands for Secure Hash Algorithm). The SHA-0 hashing was based on the MD4 and MD5 algorithms that were designed by Ron Rivest. Unlike its predecessors MD4 and MD5 that produce 128-bit message digest, SHA-0 generates longer 160-bit digest. A few years after the adoption as the standard NIST replaced it with a new standard so called SHA-1 [42], what might suggest that some weaknesses of SHA-0 were discovered. It is interesting to see that the only difference between SHA-0 and SHA-1 is the rotation of bits in the message scheduling algorithm. The justification of the change in design was pub-

lished 3 years later in 1998 [28] together with the security analysis showing that the rotation operation significantly increased the complexity of the attacks. At the same time the output of the European RACE Integrity Primitives Evaluation project was RIPEMD, another example of the large MD family. Improvements in the cryptanalysis of hash functions was the driving force behind modifications of the proposed algorithms. For instance 128-bit RIPMED was upgraded to its 160-bit version.

In 2002, NIST proposed a new hash standard called SHA-2 [43]. SHA-2 which is in fact a family of hash functions indexed by the required length of the digest. The lengths are 224, 256, 384 and 512 bits. The situation has dramatically changed in 2004, when a group of researchers under the leadership of Professor Wang published a collection of papers (see [109–111]) in which most of the members of the MD family were broken. Apart from MD4 and MD5, the casualties included SHA-1. The Wang's group showed in [109] that in SHA-1, the collisions can be found in 2^{69} steps which is much faster than the expected complexity of the birthday attack that is 2^{80} steps.

The need for new standard and novel approach for constructing cryptographic hash functions is reflected in the Secure Hash Algorithm 3 (SHA-3) competition. It was originated by NIST in 2007 [40] not only to develop a new standard of secure hashing but also to stimulate the international community of cryptologists to find better a way of estimating a security level of cryptographic hash functions. The competition is organized in a similar way to the Advanced Encryption Standard selection process, where submissions are revised in a public forum by the cryptographic community. The process has been divided into three phases Round 1, Round 2 and the final being Round 3, with an October 31, 2008 submission deadline the competition started in December 2008. Of the 64 submissions 54 of them were publicly known, 14 of them advanced to Round 2 and only 5 advanced through to the the final round: BLAKE, Grøstl, JH, Keccak and Skein. The winner - Keccak - of the process was announced in October 2012.

Note that many attacks on hash functions are "theoretical" as they require extensive amount of steps for practical analysis. However, as time goes by, the

computers become faster and there is a continual upgrading in the attacking algorithms. Consequently, the developed attacks tend to be more and more efficient and at some point of time, many theoretical attacks become practical. Development of quantum computers is another factor that can change the analytical tools accessible to adversaries. So far it is not known if quantum computing is feasible. Nevertheless we know that some "classically" intractable problems (such as factorization) can be solved in polynomial time on a quantum computer. An example of a "quantum" attack on CubeHash, one of SHA-3 submissions, exploiting quantum algorithm for searching database has been presented in [78].

1.6. Thesis structure

Chapter 2 consists of introductory information on cryptographic hash functions. We start by formalizing the framework used, followed by describing the cipher-based *modes* of hashing. Next we present selected methods of hash functions and block ciphers analysis like differential and truncated differential analysis. In particular we define open-key distinguishers for block ciphers and present some techniques for differential trail construction. Then rotational analysis is discussed and its variant – shift analysis.

In Chapter 3 we investigate the differential properties of block ciphers in hash function *modes* of operation. First we show the impact of differential trails for block ciphers on collision attacks for various hash function constructions based on block ciphers. Further, we prove the lower bound for finding a pair that follows some truncated differential trails in case of a random permutation. As far as we know this is the first formal proof of the bound. Then we present open-key differential distinguishers for some well known round-reduced block ciphers: Crypton, Hierocript-3, SAFER++, Square and generic n-bit Feistel cipher. Our rebound distinguishers substantially improve number of attacked rounds by means of key bits manipulation.

In Chapter 4 we present practical complexity attacks on `IDEA`-based hash functions in variety of cipher *modes* of hashing used, where we exploit null weak-key

and new non-trivial property of `IDEA`, that we called almost half-involution. The attacks are another example showing that application of block ciphers in *modes* of hashing requires more caution in comparison to their analysis in secret key model.

In Chapter 5 we extend the application of rotational distinguishers to classes of primitives that besides additions, rotations and XORs, may have subtractions, shifts, bit-wise Boolean functions and a combination of multi additions and multi subtractions. We use a concept of rotational analysis with corrections and provide formal framework for calculating accompanying probabilities. This allows us to launch rotational attacks on the compression functions of a SHA-3 candidates: BMW and SIMD. We also introduce a new form of attack – shift cryptanalysis, and apply it to the permutation of round 1,2 Shabal.

Finally, we conclude the thesis with some design guidelines for constructing hash functions and propose possible research directions in the field of cryptographic hash functions.

2. Cryptographic Hash Functions

This chapter is an introduction to the cryptographic hash function theory. First we give the definition of hash functions and provide a collection of security properties required from them. Next, we briefly discuss various applications of hashing. Further we provide the notations used in this thesis. This is followed by a study of different approaches in designing cryptographic hashing. This section concludes with an overview of attacks against hash functions. We start from differential analysis and its variant truncated differential analysis, then we discuss rotational analysis, followed by its modification – the so-called shift analysis. Finally we also recall T-function and S-function representation of Addition-Rotation-XOR designs.

2.1. Introduction to Cryptographic Hash Functions

Cryptographic hash functions are indispensable for an efficient digital signature. They provide a fixed-length digest for messages of arbitrary lengths (from very short to very long). Instead of a message, the signature is then generated for the message digest. This obviously also has some security implications. An adversary who would like to forge a signature may try to find two messages that are hashed to the same digest (we say that the messages collide). Note that the signature produced in this way is going to pass the verification for the colliding messages. For more formal definition of cryptographic hash function, we follow Menezes et al. [92] and introduce the notation $\{0,1\}^* = \bigcup_{i=1}^{\infty} \{0,1\}^i$, that is $\{0,1\}^*$ is a set of non-empty bit-sequences of any bit-length.

Definition 2.1. *A hash function* $F\colon \{0,1\}^* \to \{0,1\}^n$ *is a function that fulfills*

at least the two following requirements:

1. *for an arbitrary length input $x \in \{0,1\}^*$ the image y under F is of fixed bit-size n,*

2. *for given F and input x, $F(x)$ is "easily" computable.*

The two properties are referred to as *compression* and *ease of computation* respectively (see Definition 9.1 in [92]). However, the first term is somewhat misleading. While hash function compresses messages that are longer than the digest, it "expands" messages that are very short.

A cryptographic hash function is a hash function that is designed to achieve certain security properties. The collection of required security properties includes:

1. preimage resistance,

2. second preimage resistance,

3. collision resistance.

They are defined as follows:

Definition 2.2. *A hash function $F\colon \{0,1\}^* \to \{0,1\}^n$ is called:*

1. ***preimage resistant*** *– if given hash value H of some unknown message, it is computationally "hard" to find such a message M whose hash value is equal to H, i.e. $F(M) = H$,*

2. ***second preimage resistant*** *– if given one message M_1 it is "hard" to find other M_2, $M_1 \neq M_2$ such that hash values of both M_1 and M_2 are equal, i.e. $F(M_1) = F(M_2)$,*

3. ***collision resistant*** *– if it is "hard" to find different messages M_1 and M_2 $(M_1 \neq M_2)$ such that hash values of both are equal, i.e. $F(M_1) = F(M_2)$.*

Remark 1. *Collision resistant hash function is also second preimage resistant. However, collision resistance does not imply preimage resistance. What is more preimage resistance does not imply second preimage resistance and reverse implication does not hold either.*

The "hardness" in the Definition 2.2, which might as well be replaced with "computational infeasibility" like in [92] can be understood in many ways. The two most common interpretations are in terms of:

- asymptotic complexity,
- static complexity.

In the first case the difficulty of a particular problem is defined for an infinite family of functions indexed by digest length n. The problem is considered "easy" if there is a polynomially bounded in time and size algorithm that solves it for infinitely many instances. On the other hand it is "hard" if fraction of solved instances tends to 0 for any polynomially bounded algorithm. The obvious limitation of this approach is that the results obtained in this model are asymptotic for $n \to \infty$ and apply to infinite families of hash functions.

The second approach is derived from the so called concrete security (see for instance [22]), which is also based on some family of functions i.e. $\mathbb{F} = \{F \colon \mathbb{K} \times \mathbb{M} \to \{0,1\}^n\}$, where $\mathbb{K}$ is the space of indexes, $\mathbb{M}$ is the space of messages. The central element of this model is a probabilistic algorithm called adversary, which interacts with randomly chosen functions from $\mathbb{F}$. The strength of primitive is then obtained by calculating the probability that the adversary finds, for example, a collision in the hash function (the security bound on collision finding adversaries is $2^{-\frac{n}{2}}$). However, in practical applications we are dealing with one specific design what leads to slight abuse of the notation. For example, an upper bound for collision search attacks on 256-bit hash function is 2^{128}.

2.2. Designing Hash Functions

There are variety of different approaches for constructing a hash function. The most common design strategies are:

1. Merkle and Damgard construction,
2. wide-pipe (double pipe) construction,

3. fast wide-pipe construction,

4. Merkle tree,

5. sponge construction.

Merkle-Damgård Construction. The first and probably the most common approach for building cryptographic hash functions is the chaining of the so called compression functions together. This approach is used in both SHA-1 and SHA-2. The compression function has fixed size domain, range of size smaller than the prior and is easily computable. A more formal definition is as follows:

Definition 2.3. *A compression function* $f\colon \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$ *is a transformation that maps fixed length input* (x, y) *where* $x \in \{0,1\}^n$ *is called the "previous chaining value", and* $y \in \{0,1\}^k$ *is called the "message block", into fixed-length output, the "next chaining value"* $z \in \{0,1\}^n$*, such that* $z = f(x, y)$*.*

A cryptographic compression function also has to fulfill similar like cryptographic hash function security requirements. An example of such an approach is the Merkle-Damgård construction of cryptographic hash function which can be defined as follows (compare the Figure 2.1):

Definition 2.4. *Let* $h_0 = IV$ *where IV, so called "initial vector", is a constant value from* $\{0,1\}^n$*. Let* $M \in \{0,1\}^*$ *be a message for which hash value is computed,* $|M|$ *is its bit-length and* $|M|$ *is multiplicity of* k*.* M *is represented as a concatenation of message blocks* $m_0, m_1, \ldots, m_l$ *where each block has length* k *and* $l = |M|/k$*. In addition there is defined additional block* m_{l+1} *which consists of* k*-bit-representation of* $|M|$*. The Merkle-Damgård construction is then defined as* $F(M) = h_{l+1}$*, where* $h_i = f(h_{i-1}, m_{i-1})$ *for* $i \geq 1$*.*

The Merkle-Damgård (MD) construction guarantees that when the underlying compression function is collision resistant, the resulting hash function is also collision resistant. However, the method has some drawbacks, which were presented in Joux's multicollision attack (see [55]) that demonstrates better than generic attacks finding multicollisions. A possible alternative to the MD construction

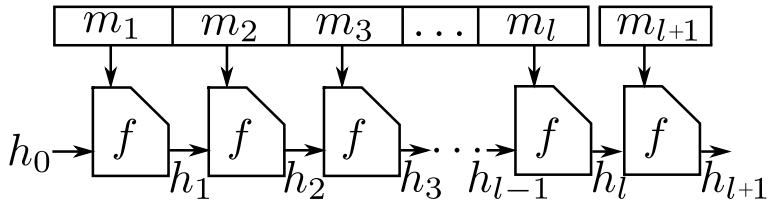

Figure 2.1.: Merkle-Damgård construction of a hash function.

is HAsh Iterative FrAmework (HAIFA) proposed in [12]. In this construction each compression depends on additional parameters: salt and number of already hashed bits. The compression function is then defined as $f\colon \{0,1\}^{n+m+s+b} \rightarrow \{0,1\}^n$ and each invocation of f is expressed as $h_i = f(h_{i-1}, m_{i-1}, \#bits, salt)$, where $\#bits$ is number of already processed bits of message at step i.

Wide Pipe Construction. This is an extension of Merkle-Damgård construction proposed in [85]. The construction aims at a complexity increase of internal collision search of the hash function by making the size of the chaining value larger in comparison to the hash digest. Figure 2.2 shows an example of double pipe design. The chaining value consists of two blocks (h_i, h'_i) and it is the input to the next execution of compression function f. The final transformation f' is the transforming of the last double-chaining value into single block digest.

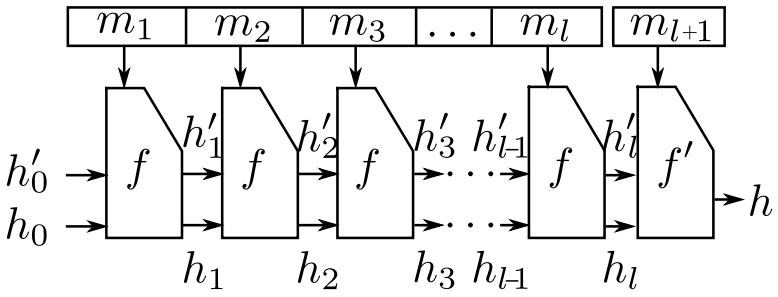

Figure 2.2.: Double pipe construction of a hash function.

Fast Wide Pipe Construction. This mode of operation is a variant of wide pipe design and was first presented in [97]. The underlying function f has only one chaining block input (apart from message block input), while it outputs two chaining value blocks (h_i, h'_i). Next invocation of f takes as the input next message block m_{i+1} and $h_i \oplus h'_{i-1}$, that is $(h_{i+1}, h'_i) = f(h_i \oplus h_{i-1}, m)$. In a similar

way to wide pipe design the last invocation of f' outputs final digest value of one chaining block size. The advantage of this approach is a speed up by at most factor 2 in comparison to the double pipe construction.

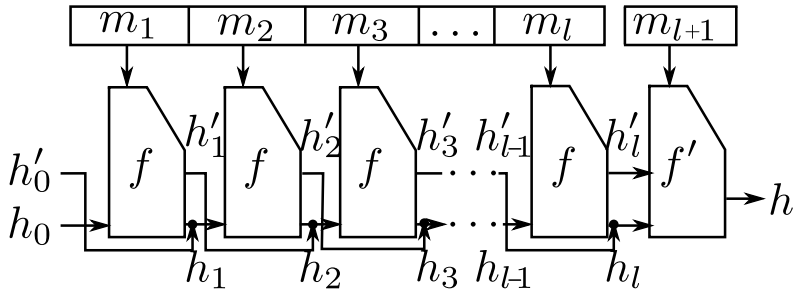

Figure 2.3.: Fast widepipe construction of a hash function. The • symbol represents XOR operation.

Merkle Tree Construction. The construction was first proposed by Merkle [93] in order to solve problem of signing multiple messages efficiently. The idea behind this design strategy is to build a binary tree with message blocks as leaves and traverse the three layer by layer to the root by compressing two lower nodes (compare the Figure 2.4). This approach allows straightforward paralellization and improvement in speed of hashing.

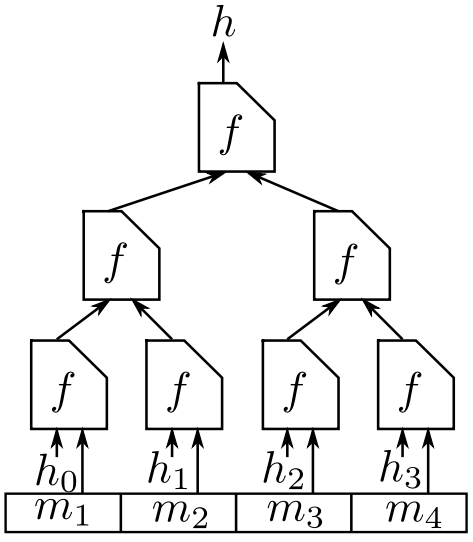

Figure 2.4.: Merkle tree construction of a hash function for 4-block message.

Sponge Construction. This iterative construction (presented in [11]), in contrast to previously presented constructions which used compression functions,

is based on fixed length permutation π. The internal state of the sponge function presented in the Figure 2.5 is of size $b = k + c$ bits. The first $l + 1$ invocations of π are called absorbing phase when consecutive message blocks are XORed with the first r bits of the previous chaining value and the whole state is transformed through π. After this phase the so called squeezing phase follows. This phase consists of a required number of π invocations, when for each invocation first r output bits of the state are returned as hash value block.

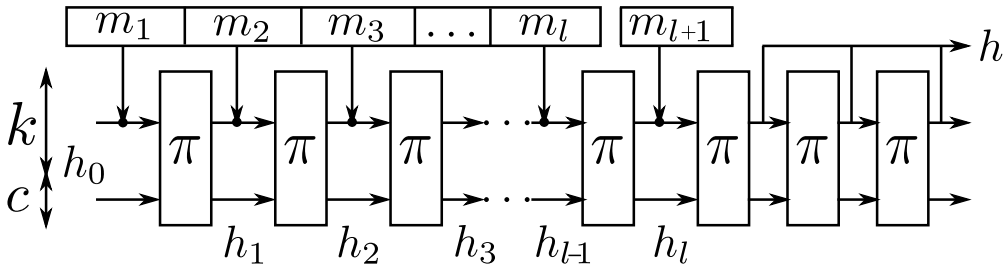

Figure 2.5.: Sponge construction of a hash function. The •represents XOR operation.

Constructions of Compression Functions. There are three main approaches for constructing compression functions for hash functions, that is:

1. dedicated designs,
2. block cipher based designs,
3. intractable-problem based designs.

The functions in the first class are built mainly in order to achieve high performance in hardware and software with minimal utilization of resources. The group is represented by a large MD-family, which includes functions like MD4, MD5, SHA-0, SHA-1, etc. Thanks to the focus it puts on the performance it is probably the most popular approach for designing hash functions. The drawback of this approach is the lack of "proof of security". The security is heuristic and it is argued by showing that the hash function is secure against all known attacks. It can be seen on the example of evolution of MD family that new designs were build in order to fix some "design weaknesses", but does not give mathematical explanation for introduced changes.

The second approach in the hashing design is based on block ciphers. The strong point of this approach is the fact that the block ciphers are the oldest and most scrutinized cryptographic primitives. In order to transform a cipher hash *modes* of operation were proposed to construct compression functions (more details can be found in Section 2.3). Due to small size of internal block of a cipher compared to size of the output digest of a hash function, except of "simple" *modes* there were also proposed double length *modes*, which produce digest of twice the block size. This approaches weakness is that security properties of block ciphers are not in general aligned well with the properties expected from secure hashing. In the case of block ciphers it is assumed that the encryption/decryption key is secret in classical security models. At the same time hash functions give much more freedom to the attacker, who can control key input to underlying block cipher and better exploit internal structure of the algorithm.

The last approach uses a (believed) intractable mathematical problem to design a hash function. The selling point of these constructions is a mathematical proof that demonstrates how an algorithm that breaks a hash function property can be used to solve efficiently the underlying (assumed to be intractable) problem. An example of such construction is the VSH hashing [31] that is based on intractability of finding nontrivial modular square roots of very smooth numbers modulo n-bit composite, There is, however, a drawback of this approach. All known constructions are much slower than other constructions. Nevertheless, it seems that this approach is getting more and more attention and the efficiency issue may be addressed by using different intractability assumptions.

2.3. Block Ciphers in Hash Function Modes of Operation

Block ciphers play an important role in symmetric cryptography providing the basic tool for encryption. They are the oldest and most scrutinized cryptographic tool. Consequently, they are the most trusted cryptographic algorithms that are often used as the underlying tool to construct other cryptographic algorithms.

One such application of block ciphers is for building compression functions for the hash functions.

Single-Block Hash Modes. There are many constructions (also called hash function *modes*) for turning a block cipher into a compression function. Probably the most popular is the well-known Davies-Meyer mode (*mode* 5 in Table 2.1). Preneel et al. in [102] have considered all possible modes that can be defined for a single application of n-bit block cipher in order to produce an n-bit compression function. They have found that there are 12 modes that are resistant against generic attacks (see 5 in Table 2.1). Later these findings have been formally proven in [22].

Table 2.1.: The table lists all provably secure hash function *modes*. The numbers are from [22].

mode ($\imath$)	h'	*mode* ($\imath$)	h'	*mode* ($\imath$)	h'
1	$E_h(m) \oplus m$	5	$E_m(h) \oplus h$	9	$E_{h\oplus m}(m) \oplus m$
2	$E_h(h \oplus m) \oplus h \oplus m$	6	$E_m(h \oplus m) \oplus h \oplus m$	10	$E_{h\oplus m}(h) \oplus h$
3	$E_h(m) \oplus h \oplus m$	7	$E_m(h) \oplus h \oplus m$	11	$E_{h\oplus m}(m) \oplus h$
4	$E_h(h \oplus m) \oplus m$	8	$E_m(h \oplus m) \oplus h$	12	$E_{h\oplus m}(h) \oplus m$

Double-Block Hash Modes. To make hash functions and compression functions resistant against the birthday-paradox attack, it is better to use double-block modes. This approach allows to square the complexity of the birthday attack in comparison to the attack on single application of primitive. Basic double-block modes have been proposed in [24, 54, 72, 75, 100] (Table 2.2 lists them all). In the first column A-DM, T-DM, Hirose and MDC-2 are abbreviations of Abrest DM, Tandem DM, Hirose's Double-Block-Length and Modification Detection Code 2, respectively (see [72] for the first two, [54] for the third and [24] for the last). In case of *Peyrin et al.(II)* [100] E_i are some independent functions built from the cipher, for example $E_i(x, y, z) = E_{x\|y}(z \oplus i) \oplus z$. For MJH-Double [75] f is involution with no fixed points and d is a natural number, $d \neq 1$.

(paragraph split) Note that the Tandem-DM mode has been proven to be collision resistant in [44], while a weakness in MDC-2 was found in [66]. The MJH-Double mode is described in [75].

Table 2.2.: The table lists double-block hash modes presented in [24, 54, 72, 75, 100].

mode	(h', g')
A-DM	$h' = E_{g,m}(h) \oplus h$ $g' = E_{m,h}(\bar{g}) \oplus g$
T-DM	$h' = E_{g,m}(h) \oplus h$ $g' = E_{m,E_{g,m}(h)}(g) \oplus g$
Hirose	$h' = E_{h\|m}(g \oplus c) \oplus g \oplus c$ $g' = E_{h\|m}(g) \oplus g$
MDC-2	$h' = (E_h(m) \oplus m)^L \parallel (E_g(m) \oplus m)^R$ $g' = (E_g(m) \oplus m)^L \parallel (E_h(m) \oplus m)^R$
Peyrin et al.(II)	$h' = E_1(h, g, m_1) \oplus E_2(h, g, m_2) \oplus E_3(h, m_1, m_2)$ $g' = E_3(h, m_1, m_2) \oplus E_4(h, g, m_1) \oplus E_5(g, m_1, m_2)$
MHJ-Double	$h' = E_{m_2\|g}(h \oplus m_1) \oplus h \oplus m_1$ $g' = d \cdot [E_{m_2\|g}(f(h \oplus m_1)) \oplus f(h \oplus m_1)] \oplus h$

2.4. Security Notation for Cryptographic Hash Functions

Ideal Ciphers vs. Hash Functions. Proofs of security of the above modes are performed under the assumption that the underlying block cipher is ideal. However, this assumption is not satisfied if the cipher is used to build hash functions. Note that the ideal cipher is related to the concept of pseudo-random permutation, where the adversary does not know the cryptographic key. Roughly speaking, for the unknown key, the permutation of the cipher is chosen at random. Clearly, the cryptanalyst in case of compression functions based on block ciphers has a much easier task as the block cipher is no longer a random permutation. The adversary fully controls the key input and can therefore select the permutation.

A known-key model [67] was proposed in order to bridge the gap between analysis of the two primitives. The model assumes that the attacker knows the encryption key and the aim is to distinguish the cipher from a random permutation on a message space by querying the primitive with messages constructed in a way to detect unwanted property. A good candidate for such a property would be one that can be easily checked and achieved in case of specific cipher while hardly detectable in case of random primitive. 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.

What differs these two approaches is that unlike the secret-key model, where the complexity of an attack is usually bounded by the size of the key space (i.e. 2^k for a k-bit key), the attacks in the open-key model are bounded by the size of the state space (i.e. 2^n for an n-bit state). Therefore, some of the published attacks in the secret-key model (precisely, the attacks with a complexity higher than 2^n) become worse than simple generic attacks, when applied in the open-key model.

2.5. Methods of Hash Functions Analysis

The analysis of cryptographic hash functions can be divided into two main categories. The first are generic attacks which are applicable to any design and are independent on the internal structure of the function. In this case the analyzed algorithm is considered as a black-box primitive for which input and output is specified. The internal behavior is not relevant in this context. On the other hand we can lift the previous assumption about black-box behavior of analyzed function and exploit internal structure of it, which is of course known to the attacker. These kind of attacks are no more applicable to any design, but are largely dependent on the algorithm.

2.5.1. Generic Attacks

Generic attacks are applicable to any primitive for which the internal structure is not known to the attacker. The primitive is treated as a black-box with input and output that can be provided and observed, respectively. The general assumption of these kind of attacks is the pseudo-randomness of the analyzed primitive, so in our case a block cipher or a hash function. We will describe in more detail two following generic attacks:

- brute force attack,
- birthday attack.

Brute Force Attack. This is the naïve method of breaking a cryptographic hash function by finding preimages for a given target hash value. The attacker is challenged with the hash value and its aim is to find a message that has the same digest. The complexity of this attack for a function on $\{0,1\}^n$ is 2^{n-1}. The illustration of the problem is following example:

Let assume that we are challenged with some date, let it be the 2nd day of a year. How many people have to be in a room in order to find at least one born on the same day of a year with probability of at least 0.5? We only distinguish birth dates by the number of a day in a year, so for instance people born on the 1st of January 1888 and the 1st of January 1999 are born on the same day of a year. For simplicity in the example we are not considering leap years. Hence, we assume a year has 365 days.

When we calculate the probability for the event of finding a person born on the 2nd day of a year we calculate it, that it is equal to $1 - (\frac{364}{365})^k$ where k is the number of additional people in the room. Elementary calculations show that $k > 252.65$, that is at least 253 people are required in order to achieve the goal. If we generalize the problem to a hash function with range $\{0,1\}^n$ we obtain that at least 2^{n-1} elements have to be checked in order to obtain required preimage with probability of at least 0.5.

Birthday Attack. The birthday attack on the other hand can be used in collisions search attacks for a hash function with a complexity significantly smaller, that is $2^{\frac{n}{2}}$. The birthday paradox is defined as follows:

How many people have to be in the room in order to find at least two of them born on the same day of a year with probability of at least 0.5? At first glance it might seem that the answer to the question is the same as before. This is partially true, the earlier number of occupants of the room will guarantee finding the required pair. However, we can do it much better. In the first case the day is fixed while in the later with each considered person the size of the set of possible dates of birth for finding a match/collision has increased. Hence, the success probability should be much higher with each new considered person. The other difference we can detect immediately (and is going to be more visible in the

following part) is that the first probability is increasing linearly with each person, because we consider their date of birth separately, while in the second case we are dealing with pairs of people what increases the number of possible success events in quadratic manner. When we calculate the exact probability we discover it is equal to $1 - \frac{364}{365} \cdot \ldots \cdot \frac{365-k+1}{365}$ and tedious calculation show that $k > 22.49$. In this case only 23 people are needed in the room to find a collision in comparison to the 253 in the earlier case. When we generalize the example to a hash function on $\{0,1\}^n$ we obtain that at least $2^{\frac{n}{2}}$ elements have to be checked.

In the following sections we present examples of algorithm specific attacks. We start with a definition of open-key distinguishers for block ciphers and description of some techniques for differential trail construction. Next rotational analysis and shift analysis is presented. The chapter ends with definition of T-functions which have application in analysis of Addition-Rotaion-XOR designs.

2.5.2. Differential Analysis

The differential analysis was introduced by Biham and Shamir in [16] and successfully used for the DES analysis. The idea is to follow the propagation of a difference in the state of the cipher throughout consecutive rounds. When the input-output differences can be predicted with a sufficiently high probability, than the cipher can be distinguished from a pseudo-random permutation. This concept can trivially be adjusted for the case, where the adversary knows/controls the key of the cipher (open-key differential distinguishers). The goal of adversary in this case would be to find an input-output pair of differences for the cipher that can be predicted with a probability higher than in a random permutation.

A natural consequence of constructions presented in Section 2.3 is that block ciphers methods of cryptanalysis are also applicable for attacking hash functions. The differential analysis is one of best known tools for cipher analysis and it has also been successfully applied for hash functions analysis, first examples are [28, 111]. What is more the discussion in Section 2.4 clearly shows that available degrees of freedom make differential attack much stronger tool in hash function context.

In this Section we first recall notation and basic differential attack technique. Next we describe some techniques for design of differential trails for block ciphers. Then we define open-key distinguishers for block ciphers and in particular open-key differential distinguishers.

Differential Analysis. We will focus our analysis on substitution-permutation (SP) block ciphers. Each round of such ciphers consists of two types of transformations:

- a non-linear layer of S-boxes (S),
- a linear-diffusion layer (LD).

The non-linear layer operates on bytes, i.e. the inputs to the S-boxes are bytes of the state. The linear-diffusion layer may apply different transformations such as multiplications of the columns/rows of the state matrix by a fixed diffusion matrix, transpositions of rows/columns, rotations of elements of the state matrix, subkey additions, and others.

Differential trails for ciphers are given as a sequence of input-output word differences of each transformation of the state. Since SP ciphers are usually byte-oriented, these trails can be given as a sequence of active bytes, i.e. bytes that have differences. Depending on the properties of the S-box layer and the linear-diffusion layer, the adversary can built two types of trails.

The first type is a *standard* differential trail, where the exact values of the input-output differences for each layer and for each round of the trail are fixed. The probability of these trails depends on the differential properties of the S-boxes, i.e. the probability that a given input difference to the S-box will produce a given output difference. Note that when these differences are fixed, then the trail in the linear-diffusion layer hold a probability 1.

The second type is a *truncated* differential trail (proposed in [64]). In this trail only the position of the active bytes is important, while the actual difference values are ignored. Since, the S-box operates on a single byte, it means it cannot change an active byte to a non-active and vice-versa. Hence the adversary concentrates only on the linear-diffusion layer and finds the probability of a

particular configuration of input-output active bytes.

Techniques for Differential Trail Constructions. A major improvement in the analysis of SP cryptographic algorithms was the introduction of the rebound attack [91]. The idea is as follows. If we assume that the adversary controls the input to the S-boxes in round i (S_i), then any input-output difference to this layer can be obtained for free (simple table lookups). More specifically only half of the input-output differences are possible, but for each of them there are two different input values and that is why on average this is true. In other words, when Δ_1, Δ_2 are fixed differences, then it is easy to find X such that $S(X + \Delta_1) \oplus S(X) = \Delta_2$. In two consecutive middle rounds (round i and $i + 1$) the adversary first fixes both the input differences Δ'_i of the LD_i layer in the i-th round, and the output differences Δ'_2 of the LD_{i+1} layer of the $(i + 1)$-th round. Then he goes forward through the LD_i layer and backwards through the LD_{i+1} layer. He ends up with fully determined differences Δ_1 and Δ_2, since the layers are linear. In between there is only one S-box (S_i) layer (composed of a number of S-boxes), which can be passed for free when the adversary fixes the values, i.e. when he finds the proper solutions for X of the above equation. Therefore, at the beginning of the i-th round, and at the end of the $(i + 1)$-th round, not only the differences, but now also the values have been fixed. The rounds that precede and follow the two middle rounds are passed probabilistically with probability $p_1 \times p_2$ dependent on the LD transformation. Compare simple example in Figure 2.6. The example cipher block consists of 16 bytes which are transformed with LD and S layers – each round starts with LD followed by S layer. The trail consists of 4 rounds from $i - 1$ to $i + 2$. Darkened square represents nonzero difference while not filled one – zero, so for example Δ'_1 is a difference with non-zero values only in the first column. Because key addition is not exploited for simplicity it is omitted in the Figure.

The technique of the rebound attack was improved with the Super-Sbox cryptanalysis [36, 46, 73]. When the round diffusion is incomplete then two layers of S-boxes can be passed for free using a precomputed lookup tables. The idea is similar to the one of the original rebound attack, but bigger lookup tables are

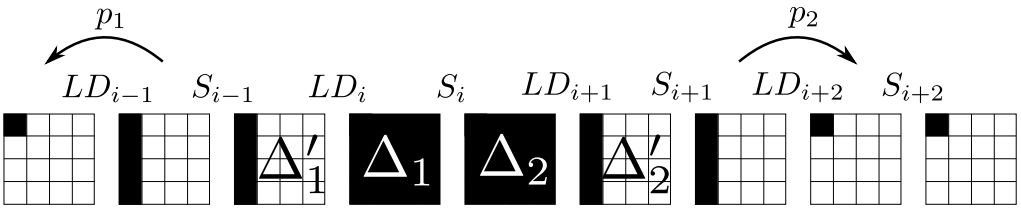

Figure 2.6.: Example of the application of the rebound attack to a substitution-permutation block cipher.

used.

The key can be used to gain an additional degree of freedom, which in return can lead to more S-box layers passed for free. When the adversary controls the key, then the rebound attack can be extended to one or two additional rounds, depending on the size of the key. The subkey (round key) is XOR-ed in each round of the cipher. The first S-box layer can be passed for free using the previous rebound technique (by fixing not only the difference, but the exact values as well). The second S-box layer can be passed for free as well if the adversary controls the input values to this layer by solving the appropriate equations. These values can be manipulated with the subkey, i.e. the adversary can choose a proper subkey such that the inputs to the S-box layer can be of arbitrary value (yet, their difference is fixed). Hence, the adversary can pass the second S-box layer for free if he controls the subkey of this round.

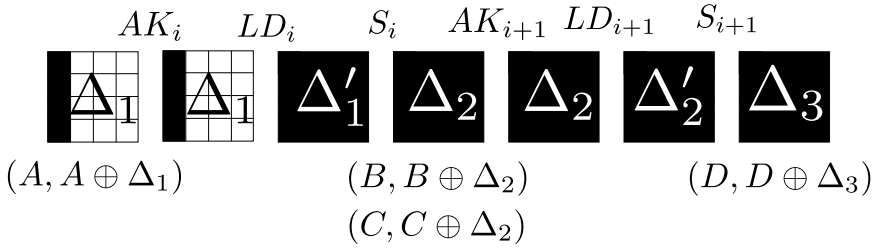

Figure 2.7.: Example of application of the rebound attack to a substitution-permutation block cipher in chosen-key model.

Let us explain the idea with an example (see Figure 2.7). The cipher block consists of 16 bytes which are transformed with LD and S layers – each round starts with LD followed by S layer. The trail consists of 2 rounds i and $i + 1$. Key addition operation is marked with AK. Let $\Delta_1 \to \Delta_2 \to \Delta_3$ be an arbitrary

two-round differential trail. First the adversary finds (with the rebound attack) a pair of states that satisfies the differential trail of the i-th round, i.e. he finds a pair $(A, A \oplus \Delta_1)$ that produces $(B, B \oplus \Delta_2)$ on the output. Then independently, he finds a pair of states for the $(i+1)$-th round, i.e. he finds $(C, C \oplus \Delta_2)$ that produces the output $(D, D \oplus \Delta_3)$. In the last step he has to fix a proper subkey k_{i+1} for the $(i+1)$-th round key addition operation AK_{i+1}, which will connect the output of the first round and the input of the second round. To do so, the adversary fixes $k_{i+1} = B \oplus C$, and as the result he obtains a pair of states $(A \oplus k_i, A \oplus k_i \oplus \Delta_1)$ that satisfy the two round differential trail.

Similarly, the adversary can pass more S-box layers when he controls the subkeys of these layers. An obvious requirement for the subkeys of these additional rounds is that they need to be independent. Otherwise, a change in a subkey in one round will change the value of a subkey in another round, which might lead to incorrect input values for the S-box layer of this second round. A second requirement is an invertible key schedule. Since the adversary controls the values of the subkeys of some middle rounds, he has to be able to produce the values of the subkeys of the rounds that precede and follow these rounds, hence he has to find the master key from the fixed subkeys. It is important to note that this technique requires a negligible memory.

Building the Differential Trails. For each of the techniques discussed above, the adversary first builds a trail that may have a plenty of active S-boxes in some middle rounds and a few at the ends of the trail. Then, a pair of values that follows the differential trail only in these middle rounds is found with complexity 1. The rest of the rounds, before and after the middle rounds, are found probabilistically since the adversary has no degree of freedom left.

In the case of byte oriented ciphers, finding the optimal differential trails with no difference in the key can be done automatically. Hence, in case of block ciphers of block size of b bytes the search space is reduced to only 2^{16} possible starting values.

Some of the ciphers are based on the so-called wide trail strategy [35], and provide an efficient method for estimating the probability of the best round-

reduced standard differential trails. These estimates are based on the differential properties of the S-boxes and the diffusion properties of the LD layers, which are often maximum distance separable mappings.

Open-key Distinguishers for Block Ciphers. A distinguisher is one of the weakest cryptographic attacks that can be launched against a secret-key cipher. In this attack, there are two oracles: one that simulates the cipher for which the cryptographic key has been chosen at random and the other simulates a truly random permutation. The adversary can query both oracles and their task is to decide which oracle is the cipher (or random permutation). The attack is considered to be successful if the number of queries required to make a correct decision is below a well defined level.

The idea of open-key distinguishers was introduced by Knudsen and Rijmen in [67] for analysis of AES and a class of Feistel ciphers. They examined the security of these block ciphers in *a model where the adversary knows the key.* Later, the same approach was used in the attack on 8-round reduced AES-128 [46] and for analysis of Rijndael with large blocks [94], where the authors defined a new security notion for a known-key cipher. The idea of chosen-key distinguishers was introduced in the attack on the full-round AES-256 [20]. This time *the adversary* is assumed to have *a full control over the key.* A chosen-key attack was launched on 8-round reduced AES-128 in [21]. When we assume that the adversary controls only differences in the key, a chosen-key attack is called a related-key attack.

Both the known-key and chosen-key distinguishers are collectively known *open-key distinguishers.* The adversary has the knowledge of the key or even can choose a value of the key. To succeed, the adversary has to discover some property of the attacked cipher that holds with a probability higher than for a random permutation.

Open-key Differential Distinguishers. Differential distinguishers in the open-key model are defined in similar way as in the secret-key model. However, their main application is for hash function analysis, because in secret key setting the known-key attacks are not regarded as a threat. The adversary builds a

differential trail $(\Delta_P, \Delta_K) \rightarrow \Delta_2$ for the block cipher $E_K(P)$. In other words, he finds a pair of plaintexts (P_1,P_2) and a pair of keys (K_1, K_2), together known as a differential pair, such that $P_1 \oplus P_2 = \Delta_P, K_1 \oplus K_2 = \Delta_K$ and $E_{K_1}(P_1) \oplus E_{K_2}(P_2) = \Delta_2$. In fact the adversary can build many pairs of plaintexts and keys. The pair (Δ_P, Δ_K) is the input difference, while Δ_2 is the output difference. At least one of Δ_P and Δ_K has to be non-zero. For example, the trails given in [21, 46, 94] have differences only in the plaintext, while the trail from [20] has differences in both the key and the plaintext.

2.5.3. Rotational Analysis

Rotational distinguishers exploit the fact that some transformations produce rotated outputs for rotated inputs. In case of rotational analysis, opposed to differential analysis where propagation of the difference $x \oplus y$ is tracked, the adversary is analyzing propagation of a rotational pair $(x, x \lll_r)$. The technique was first introduced in [10, 65, 103] while it was formally proposed in [58].

A cyclic rotation on n bits to the left by r bits $(0 < r < n)$ of a given binary string $a = (a_0, \ldots, a_{n-1})$ is defined as:

$$a \lll_r = (a \ll_r) \oplus (a \gg_{(n-r)}),$$

that is

$$a \lll_r = (a_r, \ldots, a_{n-1}, a_0, \ldots, a_{r-1}),$$

where $\ll$ and $\gg$ are left and right bit shifts, respectively. In the same manner we can define a cyclic rotation on n to the right by r bits:

$$a \ggg_r = (a \ll_{(n-r)}) \oplus (a \gg_r),$$

that is

$$a \ggg_r = (a_{n-r}, \ldots, a_{n-1}, a_0, \ldots, a_{n-r-1}).$$

The hash functions we consider in dissertation, besides the ARX operations, use

subtractions, shifts, bit-wise Boolean functions, multi additions and a combination of multi additions and multi subtractions. The following lemmas characterize the probability that a given transformation may preserve the rotational property. The proofs of the lemmas are presented in Appendix A.

Lemma 2.1 (Addition and Subtraction)**.** *Given a pair of n-bit words x, y and a positive integer r, then*

$$\Pr((x \star y) \lll_r = x \lll_r \star\ y \lll_r) = \frac{1}{4}(1 + 2^{r-n} + 2^{-r} + 2^{-n}),$$

where $\star \in \{+, -\}$.

Lemma 2.2 (Shifts)**.** *Given an n-bit word x and two positive integers r, s, then*

$$\Pr((x \ll_s) \lll_r = (x \lll_r) \ll_s) = 2^{-2t},$$

$$\Pr((x \gg_s) \lll_r = (x \lll_r) \gg_s) = 2^{-2t},$$

where $t = min(r, s, n - r, n - s)$.

Lemma 2.3 (Boolean function)**.** *Given a bit-wise Boolean function $f \in \{\wedge, \vee, \neg\}$, then*

$$\Pr(f(x) \lll_r = f(x \lll_r)) = 1,$$

where x is a n-bit word and r is some positive integer.

Lemma 2.4 (Multiplication)**.** *Given a pair of n-bit words x, y and positive integers r, s, then*

$$\Pr(x \lll_r \cdot\ y \lll_s = (x \cdot y) \lll_{(r+s)}) \geq 2^{-2(r+s)}.$$

A pair $(X, X \lll_r)$ for $X \in \{0,1\}^{m \times n}$ where $X = (X_1, \ldots, X_m)$ is called a rotational pair with rotational amount r, where $X \lll_r$ is understood as element wise rotation of X, that is:

$$X \lll_r = (X_1 \lll_r, \ldots, X_m \lll_r).$$

Given a transformation $F\colon \{0,1\}^n \to \{0,1\}^n$ and a rotational pair of inputs $(X_1, X_1 \lll_r)$, $X_1 \in \{0,1\}^n$ we say that F preserves the rotational property if the equality

$$F(X_1) \lll_r = F(X_1 \lll_r)$$

is fulfilled. Hence, F preserves rotational property if for $(X, X \lll_r)$, pair $(F(X_1), F(X_1 \lll_r))$ is rotational. A system $\Phi(X)\colon \{0,1\}^{m\times n} \to \{0,1\}^{k\times n}$ that consists of the transformations $F_1, \ldots, F_k\colon \{0,1\}^{m\times n} \to \{0,1\}^n$, i.e. $\Phi = (F_1, \ldots, F_k)$, preserves the rotational property if it produces a rotational output pair for a rotational input pair.

We would like to address two important issues in rotational analysis:

- in case of the differential analysis, the adversary may introduce differences on a part of the input, while in the rotational analysis, all the input pairs of words have to be rotational.

- there are only a few transformations that preserve the rotational property for any input pair and in majority of cases, for an arbitrary input X, the condition $F(X) \lll_r = F(X \lll_r)$ holds with a probability p_F.

The probability p_F is further called a rotational probability of F and it depends on the integer r. If we assume that the outputs of the transformations are independent, then a system Φ composed of transformations $F_1, \ldots, F_k$ preserves the rotational property with the probability $p_\Phi = p_{F_1} \cdot p_{F_2} \cdot \ldots \cdot p_{F_k}$. Therefore, in order to find the probability that a system preserves the rotational property, one only has to find the probabilities that each instance of the underlying transformations preserves this property. For a random system with n-bit output, the probability that a rotational input will produce rotational output is 2^{-n}. Therefore, if a system Φ with n-bit output, has a rotational probability $p_\Phi > 2^{-n}$, then this system can be distinguished from a random system.

A pair of n-bit words (X, Y) can be fully rotational, i.e. $X \lll_r \oplus Y = 0$, or only on t bits, i.e. $h_w(X \lll_r \oplus Y) = n - t$, where h_w is the hamming weight function. Basically, we require the output pairs of all internal transformations to be fully

rotational. We make an exception for the last transformation, where it is enough to have t rotational output bits as they can be used to build a distinguisher.

For some transformation, instead of taking rotational input pairs, it is better to introduce correction by XOR-ing some low hamming weight word to the second input, i.e. instead of $(X, X \lll_r)$, we take $(X, X \lll_r \oplus\delta)$. If X is input to another transformation, then the correction most likely has to be canceled (often by XOR-ing the same correction to some other input). Otherwise, a non-rotational input pair may significantly decrease the probability of rotational output pair for the second transformation.

Since most of the transformations preserve the rotational property only with some probability, we can observe errors in the cases when the property does not hold. A rotational error e_F of the transformation F is defined as $e_F = F(X) \lll_r -F(X \lll_r)$. Depending on the actual value of X, different values for the rotational error may be produced. The errors may cancel each other as well. For example, let the output pairs of two distinct transformations have the same rotational errors, but with opposite sign, i.e. $F_1(X) \lll_r -F_1(X \lll_r) = e, F_2(Y) \lll_r -F_2(Y \lll_r) = -e$. If the outputs of F_1, F_2 are inputs to addition, then the output pair of addition will be rotational (with the rotational probability of addition), since $(F_1(X) + F_2(Y)) \lll_r = F_1(X) \lll_r + F_2(X) \lll_r = F_1(X \lll_r) + e + F_2(Y \lll_r) - e = F_1(X \lll_r) + F_2(Y \lll_r)$.

2.5.4. Shift Analysis

In line with the rotational analysis, a similar yet distinct form of attack, which we call a shift analysis, is available. Whereas in rotational analysis the adversary follows the propagation of the rotational pair $(x, x \lll_r)$, in shift analysis he follows the pair $(x, x \ll_s)$ or $(x, x \gg_s)$, where s is the shift amount. There is a significant difference in the shift probabilities of various transformations depending if the adversary considers shifts to the left or to the right. For our purposes we will consider shifts to the left, i.e. the shift pair is defined as $(x, x \ll_s)$. In this case, when the analyzed primitive lacks addition of constants, the shift analysis might be more efficient (has a higher probability) than the rotational analysis.

This comes from the fact that the rotational and shift probabilities for certain transformations are not equal. For example, rotations and modular additions have different rotational and shift (to the left) probabilities.

2.5.5. T-functions and S-functions

T-functions (see [61–63]), which is short for triangular functions, are very common building blocks in cryptography. The most popular examples of such functions are XOR and modular addition. T-functions are characterized by the fact that their output at position k depends only on the input positions k-th and lower. The formal definition of T-function is:

Definition 2.5. *([38]) A function* $f\colon \{0,1\}^{m\times n} \to \{0,1\}^{l\times n}$ *is called a* ***T-function*** *if the* k*-th column of the output* $[f(x)]_{k-1}$ *depends only on the first* k *columns of the input* $[x]_{k-1}, \ldots, [x]_0$ *where:*

$$\begin{bmatrix} [x]_0 \\ [x]_1 \\ \vdots \\ [x]_{n-1} \end{bmatrix}^T \to \begin{bmatrix} f_0([x]_0) \\ f_1([x]_0, [x]_1) \\ \vdots \\ f_{n-1}([x]_0, \ldots, [x]_{n-1}) \end{bmatrix}^T .$$

Except for the mentioned XOR and modulo addition, the condition from the above definition is also met by all bit-wise Boolean functions and modulo multiplication of integers. What is more, composition of T-functions is also a T-function. It has been shown in [96] that triangular property of hash function design might facilitate its cryptanalysis.

When we consider for instance an additional modulo 2^n, we can observe interesting property of this particular T-function. The k-th column of output can be calculated only based on k-th input column and knowledge of carry from the $(k-1)$-th column. In general the group of T-functions can be defined as follows:

Definition 2.6. *([38]) A T-function is called* w***-narrow function*** *if there are mappings:* $\alpha_1\colon \{0,1\}^m \to \{0,1\}^w$ *and* $\alpha_k\colon \{0,1\}^{m+w} \to \{0,1\}^w$ *for* $k = 2, \ldots, n-1$*, auxiliary variables:* $a_1 = \alpha_1([x]_0)$ *and* $a_k = \alpha_k([x]_{k-1}, a_{k-1})$ *for* $k = 2, \ldots, n-1$

such that f can be written as:

$$\begin{bmatrix} [x]_0 \\ [x]_1 \\ \vdots \\ [x]_{n-1} \end{bmatrix}^T \rightarrow \begin{bmatrix} f_0([x]_0) \\ f_1([x]_1, a_1) \\ \vdots \\ f_{n-1}([x]_{n-1}, a_{n-1}) \end{bmatrix}^T .$$

The smallest w such that some f is w-narrow is called the ***narrowness*** *of f.*

In [27, 77, 95] the w-narrow functions are referred as S-functions which is short for state functions. The auxiliary variables a_i from the Definition 2.6 are called there states. Leurent provides also freely available tool for analysis ARX primitives with use of S-function toolkit accompanying [77].

3. Open Key Differential Analysis for Block Ciphers

In this chapter, we study the security of hash functions based on block ciphers with respect to differential attack. The study applies open-key differential distinguishers. For 16 hash function modes based (see Tables 2.1 and 2.2) on block-ciphers, we determine which collision finding attack variants (collisions, pseudo collisions, semi-free start collisions, or free start collisions) are feasible, assuming that the adversary is given a specific differential trail for the underlying block cipher in the open-key model. We target all Preneel-Govaerts-Vandewalle (PGV) single-block-length compression modes, as well as four double-block-length modes. We examine several well known block ciphers (Crypton, Hierocrypt-3, SAFER++, Square, and generic Feistel ciphers) and for each of them, we present new known-key and chosen-key differential distinguishers. Our distinguishers use the rebound attack [91] as a starting point, but we obtain substantial improvements in the number of attacked rounds by exploiting some cipher-specific properties that allow us to manipulate bits of the subkeys (a similar technique was used in the context of analysing the Whirlpool function [73]). In the chosen-key model, for substitution-permutation (SP) ciphers, we obtain an explicit formula for the number of additional rounds that can be attacked for free, when the cipher has an invertible key schedule.

To show the efficiency of our distinguishers, we give proof of a lower bound on the complexity of differential distinguishers in the case of a black-box random permutation. Although this bound has been used for a while (mainly as an upper bound, e.g. in [46] it is called a limited-birthday distinguisher), as far as we know,

it has never been formally proved.

Organization. The chapter is organized as follows. In Section 3.1, we present our findings about the impact of block cipher differential trails on the security of hash function modes. Section 3.2 contains our lower bound on the complexity of differential distinguishers for black-box random permutations. In Section 3.3, we present our cipher specific known-key and chosen-key differential distinguishers for various block ciphers. Section 3.4 concludes the chapter.

3.1. Impact of Block Cipher Known Key Differential Trails on Hash Modes

The most popular design of cryptographic hash is based on iterative use of a compression function. This construction is also known as the Merkle-Damgård (MD) structure. Early compression functions used block ciphers as the main building block. Assume that we have a single instance of a block cipher $E_K(P)$ and wish to design a compression function that takes a $2n$-bit input (h, m) and outputs a n-bit string $f(h, m)$. This problem has been investigated in [22, 102] and it has been shown that there are 12 structures (modes) that are secure. An example of one such structure is the well-known Davies-Meyer (DM) mode that is defined as $f(h, m) = E_m(h) \oplus h$, where h and m are the chaining value and the message, respectively.

In this work, we consider four types of collision attacks against the compression functions:

1. Collisions - for a fixed chaining value h_0, the adversary tries to find two distinct messages m_1, m_2 such that $f(h_0, m_1) = f(h_0, m_2)$.

2. Pseudo collisions - for a message m, the adversary wishes to find two distinct chaining values h_1, h_2 such that $f(h_1, m) = f(h_2, m)$.

3. Semi-free start collisions - the adversary attempts to find two distinct messages m_1, m_2 and a chaining value h such that $f(h, m_1) = f(h, m_2)$.

4. Free start collisions - the adversary tries to find two distinct chaining values h_1, h_2, and two distinct messages m_1, m_2 such that $f(h_1, m_1) = f(h_2, m_2)$.

We investigate the resistance of compression functions based on block ciphers against the attacks described above. We assume that the adversary can build a differential trail for the cipher with differences not only for the plaintext, or for the key, but also for both the plaintext and the key. For example, for the DM compression function, this means that the adversary can find a pair of chaining values (h_1, h_2) and a pair of messages (m_1, m_2) (possibly in one of the pairs the two values are equal) such that $h_1 \oplus h_2 = \Delta_h, m_1 \oplus m_2 = \Delta_m$ and $f(h_1, m_1) \oplus f(h_2, m_2) = \Delta_h \oplus \Delta_2$. Hence, when the adversary can build *some* trail, i.e. when he cannot control the exact values of the differences Δ_h, Δ_2, then he can find a differential distinguisher for the DM compression function. On the other hand, when the adversary can build a *specific* trail for the cipher with a difference in the plaintext (h is the plaintext input to the cipher), such that $\Delta_h \oplus \Delta_2 = 0$, then he can find: 1) free-start collisions, if $\Delta_m, \Delta_h \neq 0$, 2) pseudo-collisions, if $\Delta_m = 0, \Delta_h \neq 0$, 3) collisions or semi-free start collisions, if $\Delta_m \neq 0, \Delta_h = \Delta_2 = 0$ (note that this implies that there are key collisions in the cipher since in DM, the message is the key).

The same approach can be applied to the other 11 modes. We try to find all possible collision attacks under the assumption that the adversary can control the relation between the input and the output differences of a trail in the cipher. Our findings are presented in Table 3.1. The first column consists of numbers from [22]. The entries in the plaintext columns, key, plaintext and key show the best collision attacks for the modes when there is difference only in the plaintext, only in the key or both in the plaintext and key, respectively. The abbreviations C, PC, SFSC, FSC stand for collision, pseudo-collision, semi-free start collision, free start collision, respectively.

Often the block size of a cipher is too small to be secure in the compression mode. Hence, there is a class of compression functions, also called double-block-length ones, whose output size is two times bigger than the block size of the

Table 3.1.: Summary of findings for single block modes.

mode ($\imath$)	h'	plaintext	key	plaintext and key
1	$E_h(m) \oplus m$	C, SFSC	PC[a]	FSC
2	$E_h(h \oplus m) \oplus h \oplus m$	C, SFSC	PC	PC, FSC
3	$E_h(m) \oplus h \oplus m$	C, SFSC	PC	FSC
4	$E_h(h \oplus m) \oplus m$	C, SFSC	PC	PC, FSC
5	$E_m(h) \oplus h$	PC	C[a], SFSC[a]	FSC
6	$E_m(h \oplus m) \oplus h \oplus m$	PC	FSC	C, SFSC, FSC
7	$E_m(h) \oplus h \oplus m$	PC	C, SFSC	FSC
8	$E_m(h \oplus m) \oplus h$	PC	FSC	C, SFSC, FSC
9	$E_{h\oplus m}(m) \oplus m$	FSC	PC[a]	C, SFSC, FSC
10	$E_{h\oplus m}(h) \oplus h$	FSC	C[a], SFSC[a]	PC, FSC
11	$E_{h\oplus m}(m) \oplus h$	FSC	PC	C, SFSC, FSC
12	$E_{h\oplus m}(h) \oplus m$	FSC	C, SFSC	C, PC, FSC

[a]When key collisions exist in the cipher.

underlying cipher. We investigate the security of such functions proposed by Lai-Massey in [72], Hirose in [54] and Bracht et al. in [24]. Our results are presented in Table 3.2. In the first column A-DM, T-DM and Hirose are abbreviations of Abrest DM, Tandem DM and Hirose's Double-Block-Length, respectively. The abbreviations C, PC, SFSC, FSC stand for collision, pseudo-collision, semi-free start collision, free start collision, respectively.

Table 3.2.: Summary of findings for double block modes.

mode	(h', g')	plaintext	key	plaintext and key
A-DM	$h' = E_{g,m}(h) \oplus h$ $g' = E_{m,h}(\bar{g}) \oplus g$	FSC	C, SFSC	PC, FSC
T-DM	$h' = E_{g,m}(h) \oplus h$ $g' = E_{m,E_{g,m}(h)}(g) \oplus g$	FSC	C, SFSC	PC, FSC
MDC-2	$h' = (E_h(m) \oplus m)^L \parallel (E_g(m) \oplus m)^R$ $g' = (E_g(m) \oplus m)^L \parallel (E_h(m) \oplus m)^R$	C, SFSC	PC[a]	FSC
Hirose	$h' = E_{h\parallel m}(g \oplus c) \oplus g \oplus c$ $g' = E_{h\parallel m}(g) \oplus g$	PC	C, PC, SFSC, FSC	PC, FSC

[a]When key collisions exist in the cipher.

Although we have analyzed the collision resistance of the above modes, the differential trails for the underlying ciphers in the open-key model can be used as a standalone cryptanalytical result for the compression functions.

3.2. Lower Bound on Complexity of Differential Distinguisher for Random Permutations

In this section we present a lower bound on the complexity of differential distinguishers for a black-box random permutation. This allows us to make a fair comparison of our cipher-specific distinguisher complexities in Section 2.5.2 to the best possible black-box distinguisher. Although a similar *upper* bound has been used before (see, e.g. [46]), our result proves that it is indeed close to the best possible. To our knowledge, such a lower bound has not been published before, and may be of independent interest.

When the key is fixed, a block cipher becomes a permutation. An open-key differential distinguisher with no difference in the key is valid if the complexity of finding a differential pair is less than the complexity of finding such pair in a random permutation. When the input and output differences are fully fixed, in n-bit random permutation the complexity of finding a differential pair is 2^n, hence any differential distinguisher with a probability higher than 2^{-n} is valid. When the input difference is fixed, and the output difference can take values from a set of the cardinality 2^c, then for a random permutation, a differential pair can be found after performing 2^{n-c} encryptions. The general case when both the input and the output differences are taken from sets of fixed cardinalities, is discussed in the following lemma.

Lemma 3.1. *Let D_I, D_O denote subsets of $\{0,1\}^n$, which are closed under $\oplus$, i.e. $x \oplus y \in D_I$ (respectively D_O) for $x, y \in D_I$ (resp. D_O). For any attacker making queries to a random n-bit permutation π and its inverse π^{-1}, the complexity (measured in expected number of oracle queries) of finding a pair of inputs (x, y), where $x \oplus y \in D_I, |D_I| = 2^{c_I}$, such that $\pi(x) \oplus \pi(y) \in D_O, |D_O| = 2^{c_O}$, is lower bounded as $Q \geq \min(2^{\frac{n}{2}-2}, 2^{n-(c_I+c_O)-3})$.*

Proof. Since D_I and D_O are closed under $\oplus$, we may partition $\{0,1\}^n$ into input sets $A_1, \ldots, A_N$, where each $|A_i| = |D_I| = 2^{c_I}$, $N = \frac{2^n}{|D_I|} = 2^{n-c_I}$, such that $x \oplus y \in D_I$ for $x, y \in A_i$ for $i = 1, \ldots, N$. Similarly, we have a partition into

output sets $B_1, \ldots, B_M$ where $|B_j| = |D_O| = 2^{c_O}$, $M = \frac{2^n}{|D_O|} = 2^{n-c_O}$ for all $j = 1, \ldots, M$.

Let us define the following game G_0: attacker $\mathcal{A}$ has an access to a random permutation oracle $\pi\colon \{0,1\}^n \to \{0,1\}^n$ and its inverse π^{-1}, making a total of q queries to these oracles.

In the following games G_k $(k = 0, 1, 2)$, let E_k be the following event: $\mathcal{A}$ finds $x \neq y$ with $x, y \in A_i$ and $\pi(x), \pi(y) \in B_j$ for some i, j while interacting with game G_k.

We show below the following upper bound:

$$\Pr(E_0) \leq \frac{q^2}{2^n} + \frac{q}{2^{n-(c_O+c_I)}}. \tag{3.1}$$

Before we explain the formal proof, we remark that the intuition for this result is as follows. The first term $\frac{q^2}{2^n}$ is the upper bound on the collision probability error due to the fact that we simplify the problem by replacing the random permutation π with a random function. The last term arises because at each query to π (resp. π^{-1}) which is in some input set A_i (resp. output set B_j) there are at most 2^{c_I} points in A_i whose image under π is already defined (resp. at most 2^{c_O} points in B_j whose image under π^{-1} is already defined), thus occupying at most 2^{c_I} out of the 2^{n-c_O} output sets (resp. at most 2^{c_O} out of the 2^{n-c_I} input sets).

We first show that (3.1) implies the claimed expected complexity bound. In game G_0, let T denote the random variable defined as the number of oracle queries until the event E_0 occurs. We lower bound the expected value $Q = \mathrm{E}(T)$ as follows. Let $p(q)$ denote the right hand side of (3.1), and let q^* be such that $p(q^*) = \frac{1}{2}$. Since $\Pr(T \leq q) \leq p(q)$, we have

$$Q \geq \sum_{q > q^*} \Pr(T = q) \cdot q \geq q^* \cdot \Pr(T > q^*) \geq \frac{q^*}{2}. \tag{3.2}$$

Now, for $i \in \{1, 2\}$, let q_i denote the value of q such that the ith term on the right hand side of (3.1) is equal to $\frac{1}{4}$. Since there are 2 terms in (3.1), we may take $\min(q_1, q_2)$ as lower bound for q^*. Since $q_1 = 2^{\frac{n}{2}-1}$ and $q_2 = 2^{n-(c_I+c_O)-2}$,

the claimed lower bound on Q follows.

It still remains to prove (3.1). We will do this by building a chain of games, starting with G_0, which are similar until *bad* is set (for further details of this methodology see for example [8]).

First define a game G_1 to be similar to G_0 except that the permutation π is replaced by a relation $P \subset \{0,1\}^n \times \{0,1\}^n$ that is injective and functional, but not necessary defined in the whole domain. According to naming convention in [8] relation P is called partial permutation, whereas injectivity and functional conditions together are named "permutation constraint". Initially P is empty and through execution of G_1 its values are being sampled randomly with respect to "permutation constraint". Whenever $P(x)$ (resp. $P^{-1}(y)$) is needed first it is checked if P (resp. P^{-1}) is defined on x (resp. y). If this is the case then appropriate value is returned, otherwise $P(x)$ (resp. $P^{-1}(y)$) is sampled uniformly at random from $\overline{img(P)}$ (resp. $\overline{img(P^{-1})}$), where $\overline{img(P)}$ is complement of image of P. Because the sampling is the same as in the Game G_0, we have

$$\Pr(E_0) = \Pr(E_1). \tag{3.3}$$

Next we define game G_2 which is the same as G_1 except "permutation constraint" for P does not need to be fulfilled. That means the values $P(x)$ (resp. $P^{-1}(y)$) are sampled at random from $\{0,1\}^n$, but the game stops immediately when the "permutation constraint" is not satisfied. Unless the "permutation constraint" is violated by the occurrence of a collision between a new output value returned by P and a previous output value of P or input value queried to P^{-1} (resp. a collision between a new output value returned by P^{-1} and an previous output value of P^{-1} or input value queried to P), the games G_1 and G_2 proceed identically. Since at each query there are at most q previous P (resp. P^{-1}) output values already defined, we have

$$|\Pr(E_2) - \Pr(E_1)| \leq \frac{q^2}{2^n}. \tag{3.4}$$

At this stage we stop building chain of games and we upper bound the probability $\Pr(E_2)$ directly. We claim that

$$\Pr(E_2) \leq \frac{q}{2^{n-(c_O+c_I)}}. \tag{3.5}$$

Let x denote the qth query of the attacker, define the following variables for $i = 1, \ldots, N$ and $j = 1, \ldots, M$:

- a_i^F = number of P oracle queries made so far which are in A_i,
- a_i^R = number of P^{-1} oracle answers given so far which fell in A_i,
- b_j^F = number of P^{-1} oracle queries made so far which are in B_j,
- b_j^R = number of P oracle answers given so far which fell in B_j.

Suppose that x is a query to P and that $x \in A_i$ for some i. We have so far $a_i^F + a_i^R$ points in A_i whose B_j sets are already defined. Hence the event E_2 will occur only if the uniformly random (in $\{0,1\}^n$) answer of P falls in one of those output sets, so it will happen in this query with probability $\leq \frac{a_i^F + a_i^R}{M} \leq \frac{|D_I|}{M} = \frac{1}{2^{n-(c_I+c_O)}}$, using $a_i^F + a_i^R \leq |D_I|$ (since the game has not stopped so far). Similarly, if x is a query to P^{-1} and $x \in B_j$ for some j, then E_2 will occur in this query with probability $\leq \frac{b_j^F + b_j^R}{N} \leq \frac{|D_O|}{N} = \frac{1}{2^{n-(c_I+c_O)}}$. It follows that E_2 occurs among the first q queries with probability bounded by (3.5), as claimed. This completes the proof of the Lemma. □

3.3. Differential Trails for Specific Block Ciphers

We have searched for differential trails in the following ciphers: Crypton, Hierocrypt-3, SAFER++, and Square. Especically, we have tried to build standard and/or truncated trails, which can be used in a rebound-type attack. For some of the ciphers, the probabilities for both the standard and the truncated differential trails to be higher than in a random permutation. In this case, only the trails (which are usually truncated) with higher probability are presented.

The trails for the chosen-key distinguishers were built upon the trails for the known-key distinguishers by increasing the number of the full active middle rounds which can be covered for free when a proper subkey is fixed. When n-bit key is used, with an invertible key schedule that produces s-bit subkeys, then the chosen-key distinguisher has $\lfloor \frac{n}{s} \rfloor$ more rounds than the known-key distinguisher.

3.3.1. Crypton, Hierocrypt-3 and Square

Crypton [82], Hierocrypt-3 [32], and Square [34] are 128-bit SP block ciphers and have a various number of internal rounds depending on the length of the key. The best published attacks in the secret-key model are on 8 rounds of Crypton [60], 3-3.5 rounds of Hierocrypt-3 [7], and 8 rounds of Square [68].

The internal state of each cipher can be seen as 4×4 matrix of bytes, while a round consists of three types of transformations of the state:

1. byte-wise application of a non-linear S-box,
2. matrix-wise linear-diffusion (LD) layer that applies different linear transformations of various bytes of the matrix to introduce a sufficient diffusion among the bytes of the state,
3. subkey addition – a simple XOR of the round key to the matrix.

A round of Crypton consists of an S-box layer γ, LD layer composed of two transforms π and τ, and subkey addition σ. Hierocrypt-3 has six round transforms: two S-box layers $[S]$, two LD layers $[MDS_L]$ and $[MDS_H]$, and two subkey additions $[AK]$. A round of Square consists of four transforms: S-box layer γ, LD layer with two transforms θ and π, and a subkey addition σ. It is important to notice that all three ciphers have a non-linear, but invertible, key schedule. The 256-bit key versions of Crypton and Hierocrypt-3, have a key schedule such that each two consecutive 128-bit subkeys are independent.

For each cipher, we can build 7-round truncated differential trails (7 S-box layer trail in case of Hierocript), that have a full active state in the middle round, but only a few active S-boxes in the rest of the 3+3 rounds (S-box layers of Hierocript).

These trails can be used to construct known-key distinguishers on 7 rounds of the ciphers, based on the rebound technique. Since the ciphers have invertible key schedules, we can increase the number of attacked rounds by switching from the known-key to the chosen-key attacks and using the degrees of freedom of the subkeys. Hence, we can construct a chosen-key differential distinguisher on 8 rounds of Crypton with 128-bit keys, and 9 rounds of Crypton with 256-bit keys (the additional round comes from extra 128-bit freedom of the key; the chosen-key has $\lfloor \frac{256}{128} \rfloor = 2$ more rounds than the known-key, see Section 2.5.2). For Hierocrypt-3, the result is a chosen-key distinguisher on 8 S-box layers = 4 rounds for 128-bit keys, and on 9 S-box layers=4.5 rounds for 256-bit keys. Square only supports 128-bit keys, hence the chosen-key distinguisher works on 8 rounds, which is indeed the total number of rounds of this cipher.

The trails used in the chosen-key distinguishers for 9, 4.5 and 8 rounds of Crypton, Hierocrypt-3, and Square, respectively, are given in the Figure 3.1, the Figure 3.2 and the Figure 3.3, respectively.

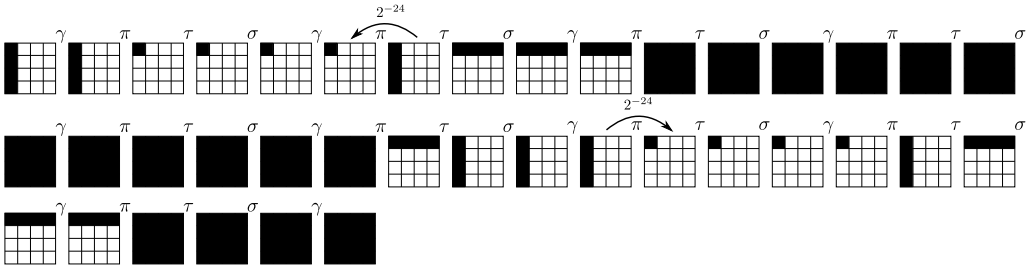

Figure 3.1.: Truncated differential trail for 9 rounds of Crypton for chosen-key distinguisher and 256-bit key.

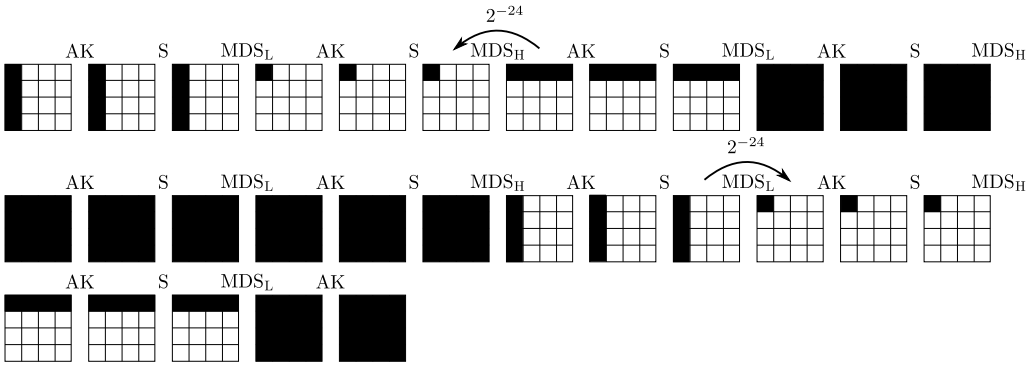

Figure 3.2.: Truncated differential trail for 4.5 rounds of Hierocrypt for chosen-key distinguisher and 256-bit key.

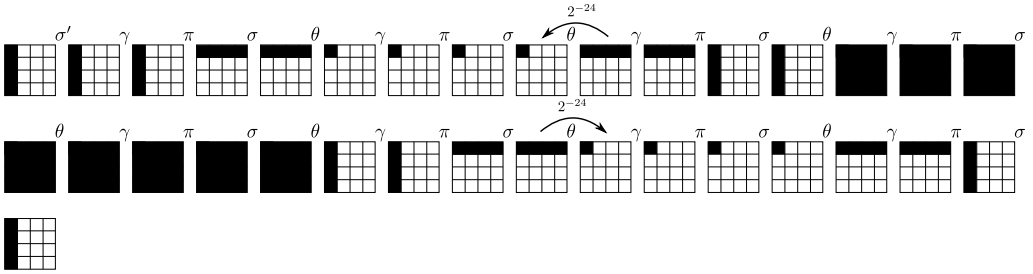

Figure 3.3.: Truncated differential trail for 8 rounds of Square for chosen-key distinguisher ($\sigma' = \sigma(\theta(k_0))$).

Since the middle full-active state round(s) are covered by the rebound attack and by fixing the subkeys used in these rounds, we can assume that the probability of the trails in these rounds is 1. Hence, we count only the probability of the rest of the rounds. In each of the three trails, we have twice 2^{-24} – that is the probability that the linear-diffusion transformation will turn four active bytes into one active byte. The probability of the trail in the rest of the layers is 1. Therefore, to find pairs of plaintexts and ciphertexts that will follow the truncated differential trails, one has to start with 2^{48} pairs of states that pass the middle rounds (each pair can be build with negligible complexity). Out of 2^{48} pairs, 2^{24} will produce four-to-one active byte in the first half of the trail, leading to a plaintext difference as the one in the trail. Out of these 2^{24}, one will produce four-to-one active in the second half of the trail and a ciphertext difference as the one in the trail. Now, let us try to compare our complexity of 2^{48} encryptions to the complexity in a case of a random permutation. By Lemma 3.1, to find this complexity we have to find the cardinalities of the plaintext and the ciphertext differences in the truncated trails. Although some of the plaintext/ciphertext differences in the trails have full active states, they are obtained by a linear transformation of some state with a four active bytes. Hence the cardinalities in all cases are $2^{4 \cdot 8} = 2^{32}$, and the complexity of producing a pair for a random permutation, that follows the trails, is at least $\min(2^{\frac{128}{2}-2}, 2^{128-(32+32)-3}) = 2^{61}$ encryptions.

To test the correctness of our results, we have constructed a chosen-key distinguisher on mCrypton [83], which has the same design as Crypton, but instead of bytes (8-bit words), it works with nibbles (4-bit words), and uses a non-invertible key schedule. The above distinguishers for Crypton can easily be applied to a

modified version of mCrypton with a (invertible) key schedule identical to the one of Crypton. The chosen-key distinguisher for 9 rounds of this modified mCrypton was implemented on a PC, and a differential pair was found, an example of which is presented in the Figure 3.4. The columns in the table represent: i - round number, $A[i]$ - value of the state in round i, $D[i]$ - difference between two states in round i, $D_\gamma[i]$ - difference between two states after γ in round i, $D_{\pi\circ\gamma}[i]$ - difference between two states after $\pi \circ \gamma$ in round i, $D_{\tau\circ\pi\circ\gamma}[i]$ - difference between two states after $\tau \circ \pi \circ \gamma$ in round i, $K[i]$ - subkey in round i. The trail was obtained for $K =$ `679ff202d5834e529d9cf7013a4d8218`.

i	$A[i]$	$D[i]$	$D_\gamma[i]$	$D_{\pi\circ\gamma}[i]$	$D_{\tau\circ\pi\circ\gamma}[i]$	$K[i]$
1	0 1 d 3 7 7 0 1 1 b 6 5 e a 3 b	6 1 b 9 e 7 1 f 4 b b 3 c f 4 5	c 0 0 0 c 0 0 0 8 0 0 0 4 0 0 0	c 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	c 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	9 8 2 2 e 6 1 5 a 7 a 2 d d 3 4
2	1 1 a c 9 1 3 7 5 9 d 2 e 7 1 4	c 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	8 0 0 0 9 0 0 0 9 0 0 0 1 0 0 0	8 9 9 1 0 0 0 0 0 0 0 0 0 0 0 0	e 0 8 d c a 4 2 a 5 4 1 2 b c 7
3	7 9 5 2 c a 2 8 6 a 3 d 6 8 a 0	8 9 9 1 0 0 0 0 0 0 0 0 0 0 0 0	9 a f d 0 0 0 0 0 0 0 0 0 0 0 0	8 8 b 5 9 a 7 c 9 2 e d 1 a d 9	8 9 9 1 8 a 2 a b 7 e d 5 c d 9	3 5 3 4 b 8 b 0 c 8 1 9 e 2 9 a
4	6 c b 1 a d 6 f f 9 b 8 1 6 8 2	8 9 9 1 8 a 2 a b 7 e d 5 c d 9	d 5 4 b 5 e 9 3 1 1 4 2 d b 2 4	d 4 5 1 1 e a d 1 1 f c 9 a b e	d 1 1 9 4 e 1 a 5 a f b 1 d c e	5 9 5 7 c e d 2 f 3 6 0 d b 2 4
5	8 6 f 0 e 0 d 0 0 0 7 0 0 d 0 7	d 1 1 9 4 e 1 a 5 a f b 1 d c e	1 6 a 7 b c 4 8 e 9 b 4 2 2 9 5	1 f d a e e 6 b c 1 4 d 5 1 3 2	1 e c 5 f e 1 1 d 6 4 3 a b d 2	3 d 5 0 3 8 e a 5 4 4 0 b f 1 7
6	0 0 0 5 0 0 6 6 0 0 0 0 7 2 3 0	1 e c 5 f e 1 1 d 6 4 3 a b d 2	e 8 2 7 c a 6 e a 2 6 d 6 a 4 b	e a 6 f 0 0 0 0 0 0 0 0 0 0 0 0	e 0 0 0 a 0 0 0 6 0 0 0 f 0 0 0	9 e 0 c 8 a e a b 0 4 6 b f b 1
7	d e 7 f b 7 3 7 0 d 4 d 3 3 0 2	e 0 0 0 a 0 0 0 6 0 0 0 f 0 0 0	e 0 0 0 c 0 0 0 a 0 0 0 6 0 0 0	e 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	e 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	3 e 4 0 1 7 8 d 1 f 6 c 5 e 4 7
8	1 b 5 0 f 0 e c 1 7 1 2 9 2 4 7	e 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	6 0 0 0 4 0 0 0 2 0 0 0 6 0 0 0	6 4 2 6 0 0 0 0 0 0 0 0 0 0 0 0	9 8 c 2 d a b 9 4 0 a 9 5 0 a 7
9	7 7 3 d e e e d 0 0 2 a b b c 0	6 4 2 6 0 0 0 0 0 0 0 0 0 0 0 0	f 9 c c 0 0 0 0 0 0 0 0 0 0 0 0	e 9 8 4 d 9 4 c b 1 c c 7 8 c 8	e d b 7 9 9 1 8 8 4 c c 4 c c 8	3 3 3 4 e 6 8 0 6 b 9 2 5 b 9 9
10	8 5 d 0 9 0 2 6 9 e 5 2 5 e a 9	e d b 7 9 9 1 8 8 4 c c 4 c c 8	7 2 9 9 a 8 d d 1 c f e 6 2 7 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 3 c 3 8 2 0 9 8 8 3 a 8 c 1 e

Figure 3.4.: Example of differential pair for mCrypton.

The detailed description of modified version of mCrypton can be found in

Appendix B.

3.3.2. SAFER++

SAFER++ [86] is a 128-bit SP block cipher. The version with 128-bit key has 7 rounds and the best published attack works for 5.5 rounds [18]. A round of SAFER++ consists of: 1) a byte-wise subkey addition, 2) a byte-wise S-box layer, 3) a byte-wise subkey addition, and 4) a state-wise linear-diffusion layer in the form of four 4-PHT. The subkey additions are modular and XOR, and two different S-boxes are used. After the last round, there is an extra subkey addition. The key schedule is linear.

When the subkeys are fixed, then the S-box layer can be merged with the subkey additions to form another S-box layer, with the same input and output size. In other words, the subkey addition together with S-box and the subkey addition can be seen simply as some S-box (since the bytes of the subkeys are different, the S-boxes are also different). Hence, we can assume that a round of the cipher is composed of an S-box layer and a linear-diffusion layer, and all the additions in the cipher are modular.

Our automatic search for the best round-reduced standard differentials has found that there exist only two three-round trails with 10 active S-boxes (the rest of the trails have more than 10 active S-boxes). The first trail has 4,2,4 while the second has 2,3, and 5 active S-boxes in the first, the second, and the third round, respectively. We have used two 4-2-4 trails in our standard differential attack (see Figure 3.5 and Figure 3.6 for detail values of differentials).

We attack 6.5 rounds of SAFER++, which is the full cipher, except for the first round, where the three transforms: subkey addition, S-box and subkey addition, are missing. As far as we know this is the first rebound attack with standard differentials. Therefore, we will describe it in more details.

First, to cancel the effects of the last extra subkey addition, we fix the MSB of the bytes 1, 3, 9, 12, 13, 14, 15, 16 of the last subkey to zero, while the values for the other bits of the subkey are randomly chosen. Then, from the mentioned subkey, we find the value of the master key, and the values for all remaining

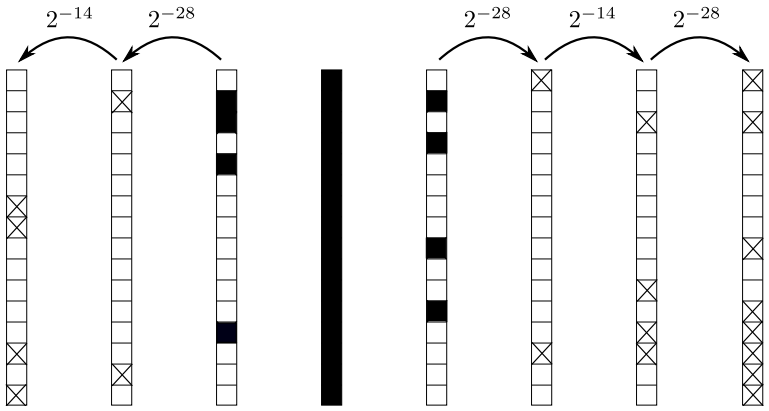

Figure 3.5.: Standard differential trail for 6.5 rounds of SAFER++ for the chosen-key distinguisher and 128-bit key. The first round is without the S-box layers, crossed square represents fixed 8-bit difference.

subkeys. Now we are ready to start the rebound attack.

We assign differences to the bytes 2, 3, 5, 13 (and no difference to other bytes) of the state before the linear layer in round 3. The differences should be such that after the linear layer all bytes are active (this holds for almost any assigned value). Similarly, we assign differences to bytes 2, 4, 9, 12 of the state after the linear layer in round 4, go backwards through the linear layer and obtain a full active state. In between the top and the bottom active states, there is only the S-box layer, hence we match the differences through this layer, i.e. we fix the values of the bytes such that all the input differences produce all the output differences. Since the values of the full state have been fixed, the rest of the rounds are passed probabilistically. There are 2, 4, 4, 2, 4 active S-boxes (16 in total) in the rounds 2, 3, 5, 6, 7, respectively.

If we assume that the differential propagation through all of the S-boxes occurs with the probability 2^{-7} then the complexity of the whole attack is $2^{7 \cdot 16} = 2^{112}$ encryptions. Note that for a fixed key, we have 2^{64} starting values for the rebound attack. We can choose different keys (such that the last subkey has the MSB of the mentioned above bytes fixed to zero) to get the necessary number of starting pairs for the differential attack. Since the input and output differences of the differential pair are fully fixed, such a pair in a random permutation can be found with 2^{128} encryptions.

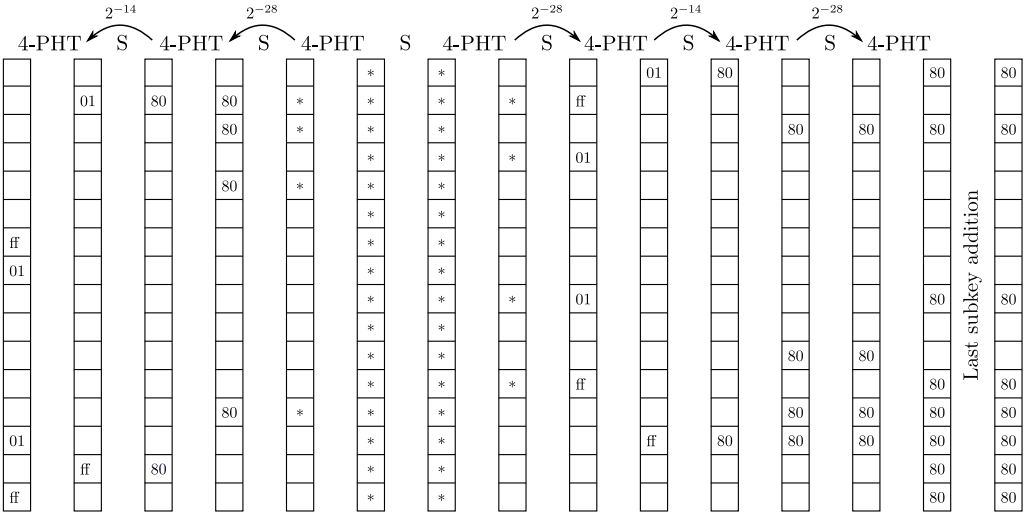

Figure 3.6.: Standard differential trail for 6.5 rounds of SAFER++ for chosen-key distinguisher and 128-bit key. The first round is without the S-box layers, * represents any non-zero 8-bit difference.

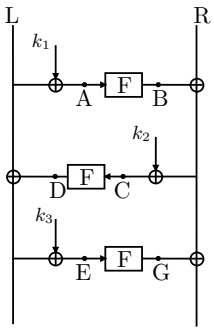

Figure 3.7.: Chosen-key distinguisher for 3-round Feistel ciphers.

3.3.3. Feistel Ciphers

Feistel ciphers with a SP round function can have a number of rounds covered for free in the known and chosen-key differential attacks. When the key is known, the S-box layers of two consecutive rounds can be attacked independently since the round function uses only half of the input. For a given two-round differential, first a pair of input states that satisfy the differential of the first round function is fixed, and then a pair of states of the second round function. Therefore, in an known-key attack, any differential trail can be extended by two additional rounds (this should not be confused with the distinguishers on 7-round Feistel ciphers proposed in [67]).

Assume that the adversary can control the key in a Feistel cipher. As the size of the input to the round function and the size of the round key are (usually) half as big as in the SP ciphers, the number of rounds that can be attacked for free is twice as big as for the SP ciphers. Let us examine the possibility of obtaining a pair of states for a three-round differential. Let n-bit Feistel cipher has an invertible key schedule that generates $\frac{n}{2}$-bit subkeys. To find a pair of states that follows some three-round differential:

$$(\Delta_1^L, \Delta_1^R) \to (\Delta_2^L, \Delta_2^R) \to (\Delta_3^L, \Delta_3^R) \to (\Delta_4^L, \Delta_4^R)$$

(the pair of states is $(L, R), (L \oplus \Delta_1^L, R \oplus \Delta_1^R)$), the adversary builds, as in the rebound attack, three pairs of states, separately for each round, that satisfy the one-round differentials, i.e. he finds the values A, C, E, such that

$$\begin{aligned} F(A) \oplus F(A \oplus \Delta_1^L) &= \Delta_1^R \oplus \Delta_2^L, \\ F(C) \oplus F(C \oplus \Delta_2^L) &= \Delta_2^R \oplus \Delta_3^L, \\ F(E) \oplus F(E \oplus \Delta_3^L) &= \Delta_3^R \oplus \Delta_4^L. \end{aligned}$$

Let $F(A) = B, F(C) = D, F(E) = G$. Then, in order to connect these three one-round differentials, the following conditions for the subkeys k_1, k_2, k_3 apply:

$$\begin{aligned} L \oplus k_1 &= A, \\ R \oplus B \oplus k_2 &= C, \\ L \oplus D \oplus k_3 &= E. \end{aligned}$$

From the first and the third equation, we get the relation $k_1 \oplus k_3 = A \oplus D \oplus E$ (note that the adversary does not control the values of A, D, E because they are fixed by the rebound attack). To satisfy this relation, the keys k_1, k_3 have to be independent (or be linearly dependent – but this is not common for ciphers). Once this is satisfied, the solution (L, R, k_1, k_2, k_3) for the system can be found in linear time. Hence in general, the master key has to be at least $\frac{3n}{2}$-bit long.

A similar analysis applies to cases when a higher number of rounds has to be covered for free. The only difference is that the resulting system has more equations. When r rounds are fixed, the system has r equation and $r+2$ unknowns: $L, R, k_1, \ldots, k_r$. In order to find the solution in linear time, for any invertible key schedule, the subkeys have to be independent. Hence, to attack an additional r rounds of a n-bit Feistel cipher the key has to be at least $\frac{rn}{2}$-bit long.

3.4. Summary

We have examined the application of the differential trails in analysis of ciphers that are used for compression function constructions. We have considered both the known-key and chosen-key models. We have especially analyzed the collision resistance of all compression functions based on single block ciphers as well as the four known double-block compression functions, when specific differential trails for the underlying ciphers can be built. Furthermore, we have presented differential distinguishers for Crypton, Hierocrypt-3, SAFER++, and Square. For these ciphers, we have shown that when the attack model is switched from secret-key to open-key, the number of rounds that can be attacked increases. We have also given as well a formal proof of lower bound of constructing pair that follow a truncated trail in the case of a random permutation. Our results are summarized in Table 3.3. The "Encryptions" column gives the expected number of encryptions in the case of a SP cipher, while the "Lower bound" column – the expected number of encryptions required in the case of a random permutation. In case of n-bit Feistel cipher r is a number of covered rounds, and 2^c is the complexity of some differential attack.

The area of open-key distinguishers is largely unexplored. Finding similar distinguishers based on related-key differentials remains an open problem.

Table 3.3.: Summary of attacks on the ciphers examined in the chapter.

Cipher	Distinguisher	Rounds	Encryptions	Lower bound	Reference
Crypton	Known-key	7	2^{48}	2^{61}	Section 3.3.1
	Chosen-key	9	2^{48}	2^{61}	Section 3.3.1
Hierocrypt-3	Known-key	3.5	2^{48}	2^{61}	Section 3.3.1
	Chosen-key	4.5	2^{48}	2^{61}	Section 3.3.1
SAFER++	Known-key	6.5	2^{120}	2^{128}	Section 3.3.2
	Chosen-key	6.5	2^{112}	2^{128}	Section 3.3.2
Square	Known-key	7	2^{48}	2^{61}	Section 3.3.1
	Chosen-key	8	2^{48}	2^{61}	Section 3.3.1
n-bit Feistel	Differential attack	r	2^c		
with k-bit key	Known-key	$r+2$	2^c		Section 3.3.3
	Chosen-key	$r+\lfloor\frac{2k}{n}\rfloor$	2^c		Section 3.3.3

4. IDEA in Various Hashing Modes

A potential candidate for hashing is the 64-bit block cipher the International Data Encryption Algorithm (IDEA) [70, 71] that uses 128-bit keys. While a single-block hashing *mode* would only provide a 64-bit hash output, insufficient for most of today's security applications, a double-block length construction (DBL) would allow 128-bit hash outputs which can be appropriate in some applications. As IDEA handles double-length keys, more flexibility in the constructions is possible. In fact, the well known Abreast-DM and Tandem-DM modes were especally created to perform hashing with IDEA (see page 2 and Section 6 of [70] and Table 2.2). These modes were later studied in much details [44, 45, 74, 76], but the security they provide when instantiated with IDEA remains a 20-year-old open question. In classical "secret key" setting, IDEA has already been studied a lot [5, 13–15, 19, 29, 33, 39, 52] and is still considered as a secure cipher despite its age and despite the current best attack [15]. The attack 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. The attack on the full cipher from [15] is very marginal with $2^{126.8}$ computations and the one from [57] requires 2^{126} computations and 2^{52} chosen plaintexts. One can also cite the work of [19], 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 [56] where a 3-round chosen-key attack is described and in [29] where the authors show how to find a free-start near collision (a free-start collision defined in Section 3.1 for which only a subset of the output collides) when IDEA is plugged into the Hirose DBL mode [29] (and also a free-start collision if the internal constant c is controlled by the attacker).

In this chapter, we study the security of the IDEA block cipher [70, 71] when

plugged into various block cipher based compression function constructions, such as the classical Davies-Meyer mode (refer *mode* 5 in Table 2.1), also DBL constructions such as Hirose, Abreast-DM, Tandem-DM, Peyrin *et al.*(II) or MJH-Double, listed in Table 2.2. Even if this cipher is still considered as secure in the classical "secret key" setting, its security remains an open problem in the 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 describe a method to compute collisions for the whole hash function.

Organization. This chapter is organized as follows. In Section 4.1, we provide description of IDEA block cipher. Next we discuss some properties of the cipher in Section 4.2, in particluar we present a novel and non-trivial almost half-involution property for IDEA. In Section 4.3 we demonstrate collisions attacks on a variety of hashing *modes* and in the following, Section 4.4, we exploit almost half-involution in order to improve previous attacks. Section 4.5 is summarizing our preimage attacks obtained with use of T-function framework. Section 4.6 summarizes the chapter.

4.1. The IDEA block cipher

The International Data Encryption Algorithm (IDEA) is a 64-bit block cipher handling 128-bit keys and designed by Lai and Massey [70, 71] in 1990. While its use is reducing in recent years, it remains deployed in practice and has not been broken yet despite its advanced age. 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. 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 subkeys used during the i-th round (only the first four subkeys for the last half round).

The KA layer simply incorporates four subkeys:

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

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:

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

All the subkeys are simply determined by choosing consecutive bits in the 128-bit master key according to the Table 4.1.

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
OT	22-37	38-53	54-69	70-85		

Table 4.1.: Key bits used for subkeys $Z_j^{(i)}$ in the i-th round of IDEA

Finally, ciphering the plaintext P with IDEA to obtain the ciphertext C is defined as: $C = \mathrm{KA} \circ \mathrm{S} \circ \{\mathrm{S} \circ \mathrm{MA} \circ \mathrm{KA}\}^8(P)$. Figure 4.1 provides a schematic view of one round of IDEA.

Currently, the best cryptanalysis work published on IDEA [15] 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

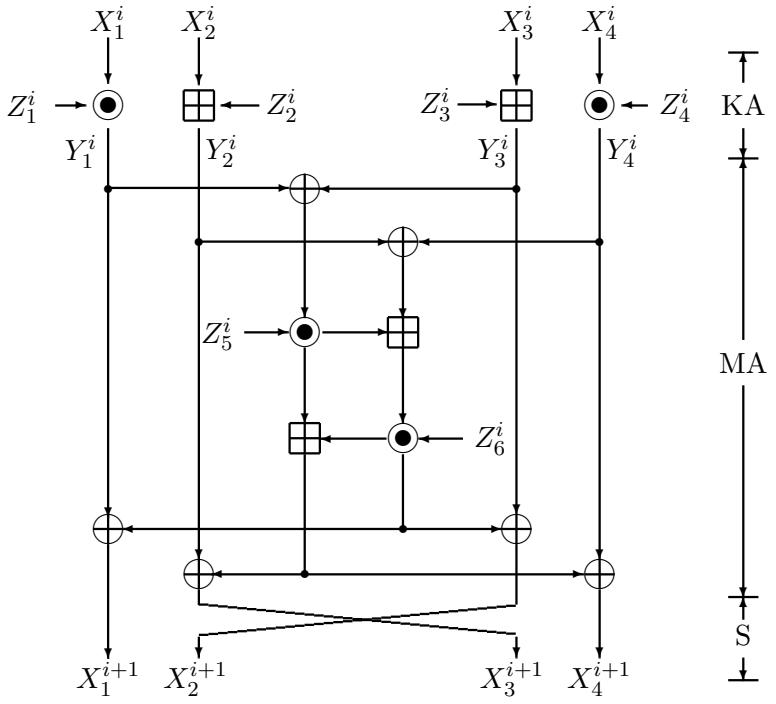

Figure 4.1.: One round of IDEA

published in [19].

4.2. Weak-keys for IDEA

Weak-keys for IDEA have already been studied in details [19, 33, 52], but what we are looking for is slightly different. Indeed, for block cipher cryptanalysis, since the attacker cannot 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.

4.2.1. Analysis of the Internal Functions

When looking at the internal round function of IDEA, one might wonder what a weak-key would be. 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 different bit positions. On the contrary, XOR operations are linear and do not provide any diffusion between bit positions, while the additions in $\mathbb{Z}_{2^{16}}$ can be easily approximated linearly. On the other hand, 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 stated in [33], a good choice is also 0, since

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

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

4.2.2. Weak-keys Classes

Due to the fact that the operand 0 is very weak for both multiplications and additions, Daemen *et al.* [33] generated a class of weak-keys. The first obvious candidate is the null key (all bits set to zero), which will force all the subkeys to zero as well. As a consequence, all subkeys additions can be simply removed and all subkeys 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

into the Most Significant Bit (MSB) of the four 16-bit plaintext input words, the attacker is ensured that the MSB of the four output words will be the only one to contain a difference. Even better, the mapping from an MSB input difference pattern to an MSB output difference pattern is completely deterministic (it is linear since no carry is propagated on the MSB). Such a property is largely sufficient to consider the null key as being weak. This reasoning can be generalized by observing that the attacker does not necessarily need all subkeys 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 subkeys are relaxed and one finally gets 2^{35} weak-keys.

4.2.3. The null Weak-key

We have shown 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 subkeys will be null as well. Then, all rounds layers will be the same and we will write KA_0 and MA_0 the KA and MA layers with null subkeys. A nice practical feature of `IDEA` is that the decryption is done using the very same algorithm as encryption, but with different subkeys. The decryption subkeys for the MA layer are the same as the encryption ones since the MA layer is an involution (i.e. $\mathrm{MA}=\mathrm{MA}^{-1}$). The decryption subkeys for the KA layer are the respective multiplicative and additive inverses of the encryption subkeys. However, note that the null subkey is both its own multiplicative and additive inverse and the KA layer becomes an involution as well (i.e. $\mathrm{KA}_0=\mathrm{KA}_0^{-1}$). To summarize, using the null key, we are ensured that $\mathrm{KA}_0=\mathrm{KA}_0^{-1}$ and $\mathrm{MA}_0=\mathrm{MA}_0^{-1}$. Note that we trivially have $\mathrm{S}=\mathrm{S}^{-1}$.

Now, since the KA layer and S layer commute, `IDEA` with the null key can be

rewritten as

$$\begin{aligned} C &= \mathrm{KA}_0 \circ \mathrm{S} \circ \{\mathrm{S} \circ \mathrm{MA}_0 \circ \mathrm{KA}_0\}^8(P) \\ &= \mathrm{KA}_0 \circ \mathrm{S} \circ \{\mathrm{S} \circ \mathrm{MA}_0 \circ \mathrm{KA}_0\}^3 \circ \mathrm{S} \circ \mathrm{MA}_0 \circ \mathrm{KA}_0 \circ \{\mathrm{S} \circ \mathrm{MA}_0 \circ \mathrm{KA}_0\}^4(P) \\ &= \underbrace{\mathrm{KA}_0 \circ \mathrm{MA}_0 \circ \{\mathrm{S} \circ \mathrm{KA}_0 \circ \mathrm{MA}_0\}^3}_{\sigma^{-1}} \circ \underbrace{\mathrm{KA}_0 \circ \mathrm{S}}_{\theta} \circ \underbrace{\{\mathrm{MA}_0 \circ \mathrm{KA}_0 \circ \mathrm{S}\}^3 \circ \mathrm{MA}_0 \circ \mathrm{KA}_0}_{\sigma}(P) \end{aligned}$$

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[1] property in Section 4.4 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 provided evidence that all operations remaining are either XOR or additions. These operations are triangular functions [61] (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 [96], this property might be very dangerous in a hash function design. In Section 4.5 we will explain how to exploit this weakness and compute preimages by guessing the input words bit layer by bit layer.

[1] The name "almost half-involution" was coined due to almost symmetric transformation $U \to P$ and $U \to C$ through σ and $\sigma \circ \theta$, respectively, which differ with additional composition with θ.

4.3. Simple Collision Attacks

As shown by Daemen *et al.* [33], 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 δ_{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

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

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 are omitting this generalization here since the methods described in forthcoming sections provide much better attacks.

Davies-Meyer. Finding a free-start collision on Davies-Meyer mode instantiated with IDEA is very easy. 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. An example of such a free-start colision is presented in Table 4.2.

Table 4.2.: An example of free-start collision in Davies-Mayer mode instantiated with IDEA.

CV_i	M	$CV_{i+1} = H(CV_i, M)$
0x9efc 0x14ef 0x85d6 0xc557	0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000	0x7f11 0x83f1 0x7617 0x8af3
0x1efc 0x94ef 0x05d6 0x4557	0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000	0x7f11 0x83f1 0x7617 0x8af3

Hirose. If we apply the method used for the Davies-Meyer mode to the Hirose

mode, it will prove to be an efficient step if we want 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. An example of such a free-start collision is presented in Table 4.3, where used as constant c the first 64 output bits of the `SHA-2` computation of the string "IDEA": `SHA-2`("$IDEA$") = "`9f8c7b26cde59ca3dacc74ec7afda737ac1d15aa5239206416f79019 dbd7ec37`" that is $c =$ `0x9f8c 0x7b26 0xcde5 0x9ca3`.

Table 4.3.: An example of free-start collision in Hirose *mode* instantiated with `IDEA`.

$CV1_i$	$CV2_i$	M	$CV1_{i+1}$	$CV2_{i+1}$
0x93e8 0x4d86	0x0000 0x0000	0x0000 0x0000 0x0000 0x0000	0x2101 0x23c9	0x0009 0x0401
0x45a5 0xa829	0x0000 0x0000	0x0000 0x0000 0x0000 0x0000	0xde42 0xdc96	0x3d38 0x3934
0x13e8 0xcd86	0x0000 0x0000	0x0000 0x0000 0x0000 0x0000	0x2101 0x23c9	0x0009 0x0401
0xc5a5 0x2829	0x0000 0x0000	0x0000 0x0000 0x0000 0x0000	0xde42 0xdc96	0x3d38 0x3934

Abreast-DM. 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.

Tandem-DM. This technique seems impossible to apply to the Tandem-DM mode for the exact same reasons as for Abreast-DM.

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 once, 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 re-keying 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 the 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.

MJH-Double. 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.

4.4. 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.

4.4.1. Exploiting the Almost Half-Involution

We have already shown in Section 4.2 that when the key is null, the `IDEA` encryption process can be rewritten as

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

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

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 subkeys).

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 the null subkey. Therefore, we need to study the probability that a random difference δ is mapped to itself through a multiplication by the null subkey.

Let a be randomly choosen and $a' = a \oplus \delta$. 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

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

where $b = a \oplus \texttt{0xffff}$. 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). On the whole, 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 for all the bits layers, we will obtain that the success probability is equal to

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

Hence, we have $\Pr[(\delta_L, \delta_M, \delta_M, \delta_R) \xrightarrow{\theta} (\delta_L, \delta_M, \delta_M, \delta_R)] = 2^{-3.17}$.

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 backwards from internal states U to P or forth 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 into 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 theoretically compute the probability of observing the same difference on P and C and hence we will leave this as an open question. Therefore, we have 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 cannot separate them because the two events are not independent).

To conclude, the probability that two randomly chosen internal state values U and U' will 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 subkeys are always null, the strengthening of the cipher would mean that $\sigma = \{\mathrm{MA}_0 \circ \mathrm{KA}_0 \circ \mathrm{S}\}^t \circ \mathrm{MA}_0 \circ \mathrm{KA}_0$ 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, similarily to the method presented in the previous section, the attacks using this almost half-involution property will work for any number of rounds.

4.4.2. Improving Collision Attacks

Davies-Meyer. The first obvious application of having the same difference in P and C is the collision search on Davies-Mayer mode, where the feed-forward will cancel the two differences in the output. The attack finds 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, so we can avoid controling IV, like in our free-start collision from Section 4.3, because the almost half-involution property can be applied to any random difference in 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 estimated in the previous section that using 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. An example of such a collision is presented in Table 4.4, where we used as the initial value IV the first 64 output bits of the `SHA-2` computation of the string "IDEA": `SHA-2`("$IDEA$") = "`9f8c7b26cde59ca3dacc74ec7afda737ac1d15aa5239206416f79019dbd7ec37`" that is IV_1 = `0x9f8c`, IV_2 = `0x7b26`, IV_3 = `0xcde5`, IV_4 = `0x9ca3`. 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.

Table 4.4.: An example of collision in Davies-Meyer mode instantiated with `IDEA`, obtained with almost half involution technique.

$M1_i$	$CV1_i = H(IV, M1_i)$	$CV2_i = H(CV1_i, 0)$
0xdacc 0xdacc 0xdacc 0xdacc 0xdacc 0xdacc 0xcadc 0x0282	0xb782 0x4583 0x83b6 0x0bef	0xdffd 0x3ffd 0x8e7d 0x6e7d
0xdacc 0xdacc 0xdacc 0xdacc 0xdacc 0xdacc 0xcade 0x1a3f	0x1ce2 0x8553 0xe656 0x4387	0xdffd 0x3ffd 0x8e7d 0x6e7d

Hirose. We have already shown 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.

Abreast-DM. One can derive a free-start collision attack for the Abreast-DM 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 of achieving 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 absolutely 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.

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 the left and the 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 they would 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 emcryption. 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.

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 $M1$ will lead 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 the 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 twice before computing the top $CV2$ output. Two times of the same difference will cancel themselves out and we will 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.

4.5. Preimage Attacks

We showed in Section 4.2 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 [96] 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 [96] 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 measured[2] 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. Ideally, with $C = 2^{25.5}$ computations allowed, an attacker should only be able to gener-

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

ate $A = 2^{25.5-64} = 2^{-38.5}$ solutions on average for an ideal 64-bit compression function.

We have provided an example of a preimage in Table 4.5. Since a random 64-bit challenge has preimage(s) with a probability p, we show the 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`("$IDEA$") = "`9f8c7b26cde59ca3dacc74ec7afda737ac1d15aa5239206416f79019dbd7ec37`" and the challenge is $CV_{i+1} = H($`0x9f8c7b26cde59ca3`$, 0) =$ `0x20ad1fc924e61ba2`.

Table 4.5.: An example of preimage for Davies-Meyer mode instantiated with `IDEA`.

$CV_{i+1} = H(CV_i, M)$	M	CV_i
`0x20ad 0x1fc9 0x24e6 0x1ba2`	`0x0000 0x0000 0x0000 0x0000` `0x0000 0x0000 0x0000 0x0000`	`0x1860 0x002e 0x2d82 0x0200`

The CV_i in Table 4.5 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. 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 finds 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 $2^{-102.5}$.

4.6. Summary

Table 4.6.: Summary of results for block cipher to compression function modes when instantiated with `IDEA`.

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 [92]	64	2^1		$2^{25.5}$ $(2^{17.5}, 2^{-17.5})$	$2^{16.13}$
Hirose [53, 54]	128	2^1		$2^{25.5}$ $(1, 2^{-64})$	
Abreast-DM [70, 72]	128	$2^{48.13}$		$2^{25.5}$ $(1, 2^{-64})$	
Tandem-DM [70, 72]	128	$2^{48.13}$		$2^{25.5}$ $(1, 2^{-64})$	
Peyrin *et al.*(II) [100]	128	2^1 / $2^{48.131}$	2^1 / $2^{48.131}$	$2^{25.5}$ $(1, 2^{-64})^1$	
MJH-Double [75]	128	$2^{32.26}$	$2^{32.26}$	$2^{25.5}$ $(2^{17.5}, 2^{-17.5})$	

In this chapter, we have shown 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 have 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 con-

jectured 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 secure in the secret key model. In particular, our answer is the negative to 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 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` extensively, 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.

In Table 4.6 we have depicted our results for the block cipher to compression function modes considered in this chapter when instantiated with `IDEA`. We did not include MDC-2 as it does not provide ideal collision resistance. The preimage complexity results find s preimages on average with a certain probability p, for a total average of $A = s \cdot p$ solutions. The results for Peyrin *et al.*(II) construction, marked with a *, depend on the instance considered (see relevant parts of Sections 4.3, 4.4 and 4.5 for more details).

5. Analysis of Addition-Rotation-XOR Designs

Rotational analysis is a relatively new type of attack. The technique was mentioned and applied in [10, 65, 103]. A formal treatment of rotational analysis is given in [58]. Note that in the differential analysis, for a pair of inputs (x, y), the adversary follows the propagation of the difference $x \oplus y$. In contrast, in the rotational analysis, the adversary examines the propagation of a rotational pair of inputs $(x, x \lll_r)$. Khovratovich and Nikolić in [58] have analyzed the primitives composed of only three operations: addition, rotation, XOR (ARX). For these primitives, they prove that the probability that a rotational pair of inputs will produce a rotational pair of outputs depends on the number of additions only.

We extend the application of rotational analysis to primitives that have transformations other than ARX. In particular, we provide the rotational probabilities of subtraction, shifts, Boolean functions and combination of additions and subtractions. As an example of usage we will apply the rotational analysis to two versions of Blue Midnight Wish 512 (abbreviated to BMW-512) 1st submitted to SHA-3 competition and 2nd tweaked by the designers in round 2 of the competition, and two versions of SIMD-512, 1st round [80] and 2nd round [81] tweaked by its designers. We find that round 1 BMW-512 [48] is susceptible to the rotational analysis. Also, the 2nd round BMW-512 [49], with a slightly altered constant, can be attacked using this method. For SIMD-512, we present various rotational distinguishers on round-reduced original and modified versions.

Rotational analysis with corrections exploits fact that composition of considered transformations produce some rotational error, which might be cancelled

with use of additional constants – corrections. The method has been simultaneously applied by Khovratovich et al. in [59] for Skein analysis where XOR corrections were considered. We apply XOR corrections in our analysis and provide formal framework for calculating exact rotational probabilities in this scenario.

We also present a new type of distinguishers – shift distinguishers. Here the adversary examines the propagation of a shift pair $(x, x \ll_s)$ or $(x, x \gg_s)$, where $\ll_s, \gg_s$ is shift to the left and right on s bits. Interestingly, although the rotational and shift analysis are very similar, for particular transformations the probabilities they preserve the rotational and shift property are different. We apply the shift analysis to the permutation used in the round 1,2 SHA-3 candidate Shabal and we obtain a shift distinguisher for this permutation.

Our distinguishers are for the compression functions only, and they do not contradict the security claims for the whole hash functions.

Organization. The chapter is organized as follows. In Section 5.1 we will introduce new results on rotational properties of multi additions and multi subtractions. In Section 5.2 we will extend rotational analysis to the case when XOR corrections are introduced and present how the S-function toolkit can be utilized to obtain exact probabilities. Section 5.3 contains our rotational distinguishers for the first submission of BMW and a modified version of tweaked BMW in the second round. In Section 5.5 we will present rotational distinguishers for modified SIMD-512 reduced to 24 rounds, with linearized key schedule, and for SIMD-512 reduced to 12 rounds. The distinguishers do not depend on the tweak introduced in the 2nd round submission. In Section 5.6 we will present our new kind of distinguisher based on a shift analysis which impact is presented on the example of Shabal. Section 5.7 concludes the chapter.

5.1. Rotational Properties of Multi Additions and Subtractions

Some basic facts on rotational analysis have been already presented in 2.5.3. Now let us focus on multi additions and multi subtractions.

Additions and subtractions are basic blocks in many cryptographic hash functions. We call addition of k integers $x_1, \ldots, x_k$ multi additions and in case of subtraction – multi subtraction. We are interested in calculating probability that multi additions are preserving rotational property, that is:

$$\Pr((x_1 + \ldots + x_k) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r)$$

and that combination of multi additions and subtractions is preserving rotational property:

$$\Pr((x_1 + \ldots + x_k - y_1 - \ldots - y_l) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r - y_1 \lll_r - \ldots - y_l \lll_r)$$

where $y_1, \ldots, y_l$ are integers.

The common approach to approximate the first probability is to estimate the lower bound in case of k summands (that is $k-1$ additions) under the assumption of independence of each operation and calculate it as the product of appropriate probabilities. For instance, in case of rotational amount 1 and 64-bit numbers calculation of lower bound for rotational property would be based on rotational probability of one addition, which is equal to $2^{-1.41}$. If calculated this way the lower bound is $2^{-1.41 \cdot (k-1)}$ and, for example for 2 additions, it equals to $2^{-2.82}$, whereas the value obtained with a computer simulation is $2^{-2.58}$ (the same result is calculated with the next described methodology). This shows how distant the bound from the exact value of the probability might be. Next, we will present the formulas for rotational probabilities of multi additions and subtractions.

For that purpose let define $N_k(i,t)$ such that:

$$N_k(i,t) = \sum_{j=0}^{\lfloor \frac{i}{t+1} \rfloor} (-1)^j \binom{k}{j} \binom{i-j(t+1)+k-1}{i-j(t+1)}.$$

Then the following lemmas hold.

Lemma 5.1 (Multi additions)**.** *Given n-bit words $x_1, \ldots, x_k$ and a positive integer r, then*

$$\Pr((x_1 + \ldots + x_k) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r) =$$
$$= \frac{1}{2^{nk}} \sum_{j=0}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} N_{k+1}(j2^n + 2^r - 1, 2^r - 1) \cdot \sum_{j=0}^{\lfloor \frac{k-1}{2^r} \rfloor} N_{k+1}(j2^n + 2^{n-r} - 1, 2^{n-r} - 1).$$

Multi additions. Let each x_i be represented as a concatenation of two numbers a_i, b_i, i.e. $x_i = a_i||b_i$, where bit-length of a_i and b_i is r and $n-r$ respectively. That is the r most significant bits of x_i is a_i and the rest $n-r$ last significant bits is b_i. Then the equation

$$(x_1 + \ldots + x_k) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r \tag{5.1}$$

can be rewritten as $(a_1||b_1 + \ldots + a_k||b_k) \lll_r = (a_1||b_1) \lll_r + \ldots + (a_k||b_k) \lll_r$, and because $(a_i||b_i) \lll_r = b_i||a_i$, hence, Equation (5.1) is equivalent to:

$$(a_1||b_1 + \ldots + a_k||b_k) \lll_r = b_1||a_1 + \ldots + b_k||a_k. \tag{5.2}$$

The sum on the left $a_1||b_1 + \ldots + a_k||b_k$ can be rewritten as $(a_1 + \ldots + a_k + C_{b_1,\ldots,b_k})||(b_1 + \ldots + b_k)$, where $C_{b_1,\ldots,b_k}$ is the carry from the sum $b_1 + \ldots + b_k$. Similar relation holds for $b_1||a_1 + \ldots b_k||a_k$, i.e. $b_1 + \ldots + b_k + C_{a_1,\ldots,a_k}||a_1 + \ldots + a_k$, where $C_{a_1,\ldots,a_k}$ is the carry from the sum $a_1 + \ldots + a_k$. Applying the above equations to the Equation (5.2), we obtain:

$$b_1 + \ldots + b_k||a_1 + \ldots + a_k + C_{b_1,\ldots,b_k} = b_1 + \ldots + b_k + C_{a_1,\ldots,a_k}||a_1 + \ldots + a_k,$$

that holds when

$$b_1 + \ldots + b_k \equiv b_1 + \ldots + b_k + C_{a_1,\ldots,a_k} \pmod{2^{n-r}},$$
$$a_1 + \ldots + a_k + C_{b_1,\ldots,b_k} \equiv a_1 + \ldots + a_k \pmod{2^r},$$

that is when $C_{a_1,\ldots,a_k} \equiv 0 \pmod{2^{n-r}}$ and $C_{b_1,\ldots,b_k} \equiv 0 \pmod{2^r}$. These two conditions can be further rewritten as:

$$a_1 + \ldots + a_k \in \bigcup_{j=0}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} [j2^n, j2^n + 2^r - 1], \quad b_1 + \ldots + b_k \in \bigcup_{j=0}^{\lfloor \frac{k-1}{2^r} \rfloor} [j2^n, j2^n + 2^{n-r} - 1].$$

Then the initial probability can be now expressed as:

$$\begin{aligned}
&\Pr((x_1 + \ldots + x_k) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r) = \\
&= \Pr(a_1 + \ldots + a_k \in \bigcup_{j=0}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} [j2^n, j2^n + 2^r - 1]) \cdot \\
&\quad \cdot \Pr(b_1 + \ldots + b_k \in \bigcup_{j=0}^{\lfloor \frac{k-1}{2^r} \rfloor} [j2^n, j2^n + 2^{n-r} - 1]) = \\
&= \sum_{j=0}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} \Pr(a_1 + \ldots + a_k \in [j2^n, j2^n + 2^r - 1]) \cdot \\
&\quad \cdot \sum_{j=0}^{\lfloor \frac{k-1}{2^r} \rfloor} \Pr(b_1 + \ldots + b_k \in [j2^n, j2^n + 2^{n-r} - 1]) = \\
&= \sum_{j=0}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} \sum_{i=j2^n}^{j2^n+2^r-1} \Pr(a_1 + \ldots + a_k = i) \cdot \sum_{j=0}^{\lfloor \frac{k-1}{2^r} \rfloor} \sum_{i=j2^n}^{j2^n+2^{n-r}-1} \Pr(b_1 + \ldots + b_k = i),
\end{aligned}$$

where $a_i \in [0, 2^r - 1]$ and $b_i \in [0, 2^{n-r} - 1]$.

Let us notice that $N_k(i,t) = \#\{(z_1, \ldots, z_k) \in [0,t]^k \colon z_1 + \ldots + z_k = i\}$, hence

$\Pr(z_1 + \ldots + z_k = i) = \frac{N_k(i,t)}{(t+1)^k}$, and we get:

$$\Pr((x_1 + \ldots + x_k) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r) =$$
$$= \sum_{j=0}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} \sum_{i=j2^n}^{j2^n+2^r-1} \frac{N_k(i, 2^r-1)}{2^{rk}} \cdot \sum_{j=0}^{\lfloor \frac{k-1}{2^r} \rfloor} \sum_{i=j2^n}^{j2^n+2^{n-r}-1} \frac{N_k(i, 2^{n-r}-1)}{2^{(n-r)k}} =$$
$$= \frac{1}{2^{nk}} \sum_{j=0}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} N_{k+1}(j2^n + 2^r - 1, 2^r - 1) \cdot \sum_{j=0}^{\lfloor \frac{k-1}{2^r} \rfloor} N_{k+1}(j2^n + 2^{n-r} - 1, 2^{n-r} - 1),$$

where at the last stage, we use the recursion $N_{k+1}(i,t) = \sum_{j=0}^{t} N_k(i-j,t)$, and precisely, derived from it recursion $N_{k+1}(t+T,t) = \sum_{i=T}^{t+T} N_k(i,t)$ for some T. □

At this point we would like to discuss the case of multi additions and subtractions. The approach for calculating the rotational probability of $k-1$ additions and l subtractions that assumes it is equal to the rotational probability of $k+l-1$ additions, evaluates only the lower bound on the exact probability. For instance in case of one addition and one subtraction, rotational amount 1 and 32-bit numbers, a computer simulation (verified by the next lemma) shows that the probability is approximately equal to $2^{-1.58}$, which is a much better result comparing to the heuristic $2^{-2.58}$ (rotational probability of two additions).

Lemma 5.2 (Multi additions and subtractions)**.** *Given n-bit words $x_1, \ldots, x_k, y_1, \ldots, y_l$ and a positive integer r, then*

$$\Pr((x_1 + \ldots + x_k - y_1 - \ldots - y_l) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r - y_1 \lll_r - \ldots - y_l \lll_r) =$$
$$= \frac{1}{2^{n(k+l)}} \sum_{j=-\lceil \frac{l}{2^{n-r}} \rceil}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} N_{k+l+1}(j2^n + (l+1)(2^r-1), 2^r-1) \cdot \sum_{j=-\lceil \frac{l}{2^r} \rceil}^{\lfloor \frac{k-1}{2^r} \rfloor} N_{k+l+1}(j2^n + (l+1)(2^{n-r}-1), 2^{n-r}-1).$$

Multi additions and subtractions. This proof is parallel to the previous one, so we will only provide a sketch of it.

Let $x_i = a_i || b_i$ and $y_i = a_{k+i} || b_{k+i}$ then

$$(x_1 + \ldots + x_k - y_1 - \ldots - y_l) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r - y_1 \lll_r - \ldots - y_l \lll_r$$

can be transformed to

$$b_1 + \ldots + b_k - b_{k+1} - \ldots - b_{k+l} || a_1 + \ldots + a_k - a_{k+1} - \ldots - a_{k+l} + C'_{b_1,\ldots,b_{k+l}} =$$
$$= b_1 + \ldots + b_k - b_{k+1} - \ldots - b_{k+l} + C'_{a_1,\ldots,a_{k+l}} || a_1 + \ldots + a_k - a_{k+1} - \ldots - a_{k+l}, \tag{5.3}$$

where $C'_{b_1,\ldots,b_{k+l}}$ is the carry from $b_1 + \ldots + b_k - b_{k+1} - \ldots - b_{k+l}$, and $C'_{a_1,\ldots,a_{k+l}}$ from $a_1 + \ldots + a_k - a_{k+1} - \ldots - a_{k+l}$, however this time negative carries are possible. The above Equation (5.3) is equivalent to conditions: $C'_{a_1,\ldots,a_{k+l}} \equiv 0 \pmod{2^{n-r}}$ and $C'_{b_1,\ldots,b_{k+l}} \equiv 0 \pmod{2^r}$ that can be rewritten as:

$$a_1 + \ldots - a_{k+l} \in \bigcup_{j=-\lceil \frac{l}{2^{n-r}} \rceil}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} [j2^n, j2^n + 2^r - 1],$$
$$b_1 + \ldots - b_{k+l} \in \bigcup_{j=-\lceil \frac{l}{2^r} \rceil}^{\lfloor \frac{k-1}{2^r} \rfloor} [j2^n, j2^n + 2^{n-r} - 1].$$

Then the initial probability can be expressed as:

$$\Pr((x_1 + \ldots + x_k - y_1 - \ldots - y_l) \lll_r = x_1 \lll_r + \ldots + x_k \lll_r - y_1 \lll_r - \ldots - y_l \lll_r) =$$
$$= \sum_{j=-\lceil \frac{l}{2^{n-r}} \rceil}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} \sum_{i=j2^n}^{j2^n+2^r-1} \Pr(a_1 + \ldots - a_{k+l} = i) \cdot \sum_{j=-\lceil \frac{l}{2^r} \rceil}^{\lfloor \frac{k-1}{2^r} \rfloor} \sum_{i=j2^n}^{j2^n+2^{n-r}-1} \Pr(b_1 + \ldots - b_{k+l} = i),$$

where $a_i \in [0, 2^r - 1]$ and $b_i \in [0, 2^{n-r} - 1]$. Let us now

$$N_{k,l}(i,t) = \#\{(z_1, \ldots, z_k, z_{k+1}, \ldots, z_{k+l}) \in [0,t]^{k+l} \colon z_1 + \ldots + z_k - z_{k+1} - \ldots - z_{k+l} = i\},$$

for $1 \leq k, 0 \leq l$, then of course we have $N_{k,l}(i,t) = N_{k+l}(i+tl,t)$ and

$$\Pr((x_1+\ldots+x_k-y_1-\ldots-y_l) \lll_r = x_1 \lll_r +\ldots+x_k \lll_r -y_1 \lll_r -\ldots-y_l \lll_r) =$$

$$= \sum_{j=-\lceil \frac{l}{2^{n-r}} \rceil}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} \sum_{i=j2^n}^{j2^n+2^r-1} \frac{N_{k,l}(i, 2^r-1)}{2^{r(k+l)}} \cdot \sum_{j=-\lceil \frac{l}{2^r} \rceil}^{\lfloor \frac{k-1}{2^r} \rfloor} \sum_{i=j2^n}^{j2^n+2^{n-r}-1} \frac{N_{k,l}(i, 2^{n-r}-1)}{2^{(n-r)(k+l)}} =$$

$$= \frac{1}{2^{n(k+l)}} \sum_{j=-\lceil \frac{l}{2^{n-r}} \rceil}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} \sum_{i=j2^n}^{j2^n+2^r-1} N_{k+l}(i+l(2^r-1), 2^r-1) \cdot$$

$$\cdot \sum_{j=-\lceil \frac{l}{2^r} \rceil}^{\lfloor \frac{k-1}{2^r} \rfloor} \sum_{i=j2^n}^{j2^n+2^{n-r}-1} N_{k+l}(i+l(2^{n-r}-1), 2^{n-r}-1) =$$

$$= \frac{1}{2^{n(k+l)}} \sum_{j=-\lceil \frac{l}{2^{n-r}} \rceil}^{\lfloor \frac{k-1}{2^{n-r}} \rfloor} N_{k+l+1}(j2^n+(l+1)(2^r-1), 2^r-1) \cdot$$

$$\cdot \sum_{j=-\lceil \frac{l}{2^r} \rceil}^{\lfloor \frac{k-1}{2^r} \rfloor} N_{k+l+1}(j2^n+(l+1)(2^{n-r}-1), 2^{n-r}-1).$$

□

Note that the probabilities in Lemmas 5.1,5.2 are efficiently computable in time polynomial in $k+l, n$.

5.2. Rotational Pairs with Corrections

In the general scenario, we assume that the rotational pair is of the type $(a, a \lll_r)$. Although this approach is very effective in case of composition of transformations as discussed before (e.g. additions), it might produce worse estimates when an analyzed design includes some operations influenced by constants. For instance in case of a nonzero constant addition, the resulting rotational probability is decreased for rotational pair $(a, a \lll_r)$. However, we can insert a correction, i.e. we can analyze the pair $(a, a \lll_r \oplus\alpha)$, where α is the correction for the pair (note that the initial rotational pair can be seen as a pair with a correction equal to zero). This approach results in better rotational probabilities compared to the previous one, compare the next examples.

For a transformation $F(x, y)$ with fixed rotational input pairs with corrections $(a, a \lll_r \oplus\alpha)$ and $(b, b \lll_r \oplus\beta)$ the correction of the output pair is defined as:

$$\gamma = F(a, b) \lll_r \oplus F(a \lll_r \oplus\alpha, b \lll_r \oplus\beta).$$

Depending on the rotational probability of the the function $F(x, y)$ this correction can possibly take different values for different pairs (a, b) and fixed corrections α and β.

For the rotational pairs with corrections, the rotational probabilities of XOR and rotation do not change – they are still equal to 1. Let $(a, a \lll_r \oplus\alpha)$, $(b, b \lll_r \oplus\beta)$ be two rotational pairs with corrections. For the correction c of XOR we have:

$$\gamma = (a \oplus b) \lll_r \oplus(a \lll_r \oplus\alpha \oplus b \lll_r \oplus\beta) = \alpha \oplus \beta$$

Hence, the XOR of two pairs with corrections α, β produces another pair with a correction $\alpha \oplus \beta$ with a probability 1.

Similarly, for the correction of rotations we have:

$$a \lll_r \lll_{r_2} \oplus(a \lll_r \oplus\alpha) \lll_{r_2} = \alpha \lll_{r_2},$$

i.e. the output composes a rotational pair with a correction $\alpha \lll_{r_2}$ with probability 1.

Finally, let us see the impact of an addition.

5.2.1. Definition of Problem

For the correction γ of addition $a+b$ we have to study the following expression:

$$\gamma = (a + b) \lll_r \oplus[(a \lll_r \oplus\alpha) + (b \lll_r \oplus\beta)]. \tag{5.4}$$

It might be tempting to analyze this case as follows: if we assume that $(a + b) \lll_r = a \lll_r + b \lll_r$ (which holds with the rotational probability of one

addition), and introduce the annotation $a' = a \lll_r, b' = b \lll_r$, we get:

$$\gamma = (a' + b') \oplus [(a' \oplus \alpha) + (b' \oplus \beta)]$$

Hence, the problem of finding the correction, as well as its probability, will be reduced to the problem of finding the differential property of the addition (which has been solved in [84]). However, it can be easily checked that for instance:

$$\begin{aligned}
&\Pr(0x7 = (a+b) \oplus [(a \oplus 0x1) + (b \oplus 0x6)]) \cdot \Pr((a+b) \lll_r = a' + b') = \\
&\quad = 2^{-3} \cdot 2^{-1.41} = 2^{-4.41} \\
&\Pr(0x7 = (a' + b') \oplus [(a' \oplus 0x1) + (b' \oplus 0x6)]) = 2^{-3.42}
\end{aligned}$$

for rotational amount $r = 1$, which shows incorrectness of such an approach. The reason why we would obtain a wrong probability is caused by some rotational pairs $(a, a \lll_r)$ and $(b, b \lll_r)$, that do not fulfill the first assumption, but produce solutions to Equation 5.4. Of course there might occur the opposite situation, there would be much less solutions to the equation. Hence, the estimation cannot serve either as upper or lower bound on required probability.

To solve this problem we will apply the S-functions methodology described in [27, 95]. A state function (abbreviation: S-function) is a function of k bits $x_i^0, \ldots, x_i^{k-1}$ and a state S_i, for $i = 0, \ldots, n-1$, that outputs bit y_i and next state value S_{i+1}, that is:

$$f(x_i^0, \ldots, x_i^{k-1}, S_i) = (y_i, S_{i+1}),$$

which is in consequence equivalent to some transformation of k n-bit words $x^0, \ldots, x^{n-1}$ and a sequence of states $\{S_i\}_{i=0,\ldots,n-1}$ into an n-bit word y.

In order to represent our problem with use of S-function we can express Equation (5.4) in bit-wise manner. Let

$$\gamma = G \oplus (A + B),$$

where $G = (a + b) \lll_r, A = a \lll_r \oplus\alpha, B = b \lll_r \oplus\beta$, then

$$G_{i+r} = a_i \oplus b_i \oplus s_i^1, \tag{5.5}$$

$$A_{i+r} = a_i \oplus \alpha_{i+r}, \tag{5.6}$$

$$B_{i+r} = b_i \oplus \beta_{i+r}, \tag{5.7}$$

$$\gamma_{i+r} = G_{i+r} \oplus A_{i+r} \oplus B_{i+r} \oplus s_{i+r}^2, \tag{5.8}$$

where state S_i consists of values of two carries from $(i-1)$-th bit position of $a+b$ and $(i+r-1)$-th bit position of $A+B$, respectively. That is $S_i = (s_i^1, s_{i+r}^2)$, where $s_i^1 = C_{a_{i-1},b_{i-1},s_{i-1}^1)}$, $s_{i+r}^2 = C_{A_{i+r-1},B_{i+r-1},s_{i+r-1}^2}$ and the initial state $S_0 = (s_0^1, s_r^2)$ is defined in the following way:

$$s_0^1 = 0, \tag{5.9}$$

$$s_r^2 = C_{A_{r-1},B_{r-1},s_{r-1}^2}, \tag{5.10}$$

where $C_{x,y,z}$ is the carry from the sum $x+y+z$. The S-function defined by (5.5)-(5.10) updates the state S in the following way:

$$f(a_i, b_i, \alpha_{i+r}, \beta_{i+r}, S_i) = (\gamma_{i+r}, S_{i+1}) \text{ for } 0 \leq i < n.$$

In order to compute the state S_0 the value of s_{r-1}^2 is required. In a similar way as in [27] we will iterate over two possible values of s_{r-1}^2 and at the step $i = n-1$ of computations we are going to discard the states that do not match the chosen value s_{r-1}^2. Moreover, the state S_{n-r} has to be treated in a different way, because the carry s_{n-r-1}^2 (the carry from addition of two most significant bits of A and B) has to be omitted, so that the following conditions are fulfilled:

$$s_{n-r}^1 = C_{a_{n-r-1},b_{n-r-1},s_{n-r-1}^1},$$
$$s_0^2 = 0.$$

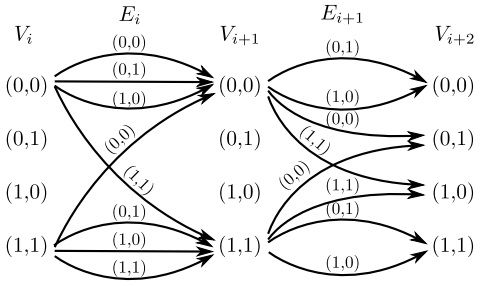

Figure 5.1.: A subgraph $\mathbf{S}_i \cup \mathbf{S}_{i+1}$ for $\alpha_{i+r} = \beta_{i+r} = \gamma_{i+r} = 0, \alpha_{i+r+1} = \beta_{i+r+1} = 1, \gamma_{i+r+1} = 1$.

Let

$$rot_r(\alpha, \beta \to \gamma) = \Pr(\gamma = (a \lll_r + b \lll_r) \oplus [(a \lll_r \oplus \alpha) + (b \lll_r \oplus \beta)])$$

where a and b are random n-bit variables.

5.2.2. Calculation of Probabilities of Rotational Pairs with Corrections for Addition

The problem of evaluating the probability $rot_r(\alpha, \beta \to \gamma)$ can be reduced to enumeration of paths in a special n-partite graph. The graph is composed of bipartite subgraphs $\mathbf{S}_i = (V_i \cup V_{i+1}, E_i)$ whose vertices V_i and V_{i+1} are all possible values for states S_i and S_{i+1}, respectively. The set of edges E_i represents possible transformations of state S_i into S_{i+1} in accordance to relations (5.5)-(5.10) for specific values of correction bits $\alpha_{i+r}, \beta_{i+r}, \gamma_{i+r}$. An edge of E_i is labeled with values for a_i and b_i that produce appropriate values of S_{i+1} from S_i. For instance a subgraph $\mathbf{S}_i \cup \mathbf{S}_{i+1}$ for $\alpha_{i+r} = \beta_{i+r} = \gamma_{i+r} = 0$, $\alpha_{i+r+1} = \beta_{i+r+1} = 1, \gamma_{i+r+1} = 0$ is shown in Fig. 5.1. For 8 possible bipartite subgraphs we have constructed their adjacency matrices (compare matrices in case of xdp$^+$ in [95]):

$$A_{000} = \begin{bmatrix} 3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 3 \end{bmatrix}, A_{100} = A_{010} = A_{001} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix},$$

$$A_{110} = A_{101} = A_{011} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}, A_{111} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

where matrices are indexed with concatenation of i-th bits of α, β, γ, that is $\alpha_i||\beta_i||\gamma_i$. Let $w(i) = [\alpha_i||\beta_i||\gamma_i]$, $L_0 = (1,0,1,0), L_1 = (0,1,0,1)$, $C_0 = (1,0,0,0)^T, C_1 = (0,1,0,0)^T$ and

$$R = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Following the reasoning from Section 5. in [27] we conclude that

$$rot_r(\alpha, \beta \to \gamma) = 4^{-n} \sum_{j=0,1} L_j A_{w(n-1)} \cdots A_{w(n-r)} R A_{w(n-r-1)} \cdots A_{w(0)} C_j. \quad (5.11)$$

Let us verify the formula in case $j = 0$ ($j = 1$ is shown in a similar way). The number of paths in n-partite graph constructed in the above-mentioned way from bipatite graphs $\mathbf{S}_{w(i)}$ can be calculated with the use of adjacency matrices $A_{w(i)}$ in the following way:

$$(1,1,1,1) \cdot A_{w(n-1)} \cdots A_{w(0)} \cdot (1,1,1,1)^T.$$

In our case for $j = 0$ vectors C_0 and L_0 restrict considered paths to the ones starting from the vertex $(0,0)$ from V_0 and ending at vertex $(0,0)$ or $(1,0)$ from V_{n-1}, that corresponds to the guess $s_r^2 = 0$. Finally the matrix R is an adjacency matrix for an artificially added bipartite subgraph, which links each vertex (v_1, v_2) for $v_1, v_2 \in \{0,1\}$ with vertex $(v_1, 0)$, that simulates cancellation of the carry from the most significant bit of $A + B$.

The following lemma is analogue to Theorem 1. in [27] and can be proven in a similar way.

Lemma 5.3. *Let P_0 be a set of all paths from $(s_0^1, s_0^2) = (0,0)$ to any of two vertices $(s_{n-1}^1, s_{n-1}^2) \in \{(0,0),(1,0)\}$ and P_1 be a set of all paths from $(s_0^1, s_0^2) = (0,1)$ to any of two vertices $(s_{n-1}^1, s_{n-1}^2) \in \{(0,1),(1,1)\}$. Then there is exactly*

one path in $P_0 \cup P_1$ *for every pair* (a, b) *that fulfills* (5.4).

The straightforward consequence of Lemma 5.3 is formula (5.11) for calculating probability $rot_r(\alpha, \beta \to \gamma)$.

The rotational pairs with corrections are useful when analysing constructions that have some type of constant additions/XORs since this class is closed under addition of any constants, whereas the class of pairs without corrections is closed only if the constants are rotational.

When XOR of a constant K is used, the correction takes the form

$$\gamma = (a \oplus K) \lll_r \oplus a \lll_r \oplus \alpha \oplus K = K \oplus K \lll_r \oplus \alpha$$

with a probability 1.

When the constant K is added (modularly), we have:

$$\gamma = (a + K) \lll_r \oplus[(a \lll_r \oplus \alpha) + K] \tag{5.12}$$

and this part can be dealt with in a similar manner as in case of XOR corrections. The S-function for Equation (5.12)

$$(\gamma_{i+r}, S_{i+1}) = g(a_i, b_i, \alpha_{i+r}, \beta_{i+r}, S_i) \text{ for } 0 \leq i < n.$$

is defined by system of equations:

$$\begin{aligned} H_{i+r} &= a_i \oplus K_i \oplus s_i^1, \\ A_{i+r} &= a_i \oplus \alpha_{i+r}, \\ \gamma_{i+r} &= H_{i+r} \oplus A_{i+r} \oplus K_i \oplus s_{i+r}^2, \\ s_i^1 &= C_{a_{i-1}, K_{i-1}, s_{i-1}^1}, \\ s_{i+r}^2 &= C_{A_{i+r-1} + K_{i-1} + s_{i+r-1}^2}, \end{aligned}$$

for $A = a \lll_r \oplus \alpha, H = (a + K) \lll_r$ (Equation (5.12) has then form $\gamma = H \oplus (A + K)$).

The corrections in rotational attacks play the role of differences in differential

attacks. To find the rotational probability of an ARX primitive, one only has to concentrate on the additions in the primitive. The rotational probabilities of these additions depend on the corrections in the rotational pairs. To launch a rotational attack, one has to find optimal corrections such that the total probability of all corrections is high.

5.3. Rotational Analysis of BMW-512

In this section we will present rotational distinguishers for the compression functions of the submitted to the SHA-3 competition [98] BMW-512 [48], further denoted as BMWv1, and for the second, tweaked by the designers, version of BMW-512 [49], denoted as BMWv2. Thomsen in [106] described pseudo-collision and pseudo-preimage attacks on BMWv1. Recently, practical differential distinguishers on BMWv2 were presented in [3, 50]. Our attack on BMWv1 is for the original design, while the attack on BMWv2 is for a modified version of the compression function, where one byte of the constant used in the internal function f_1 has been altered.

5.3.1. Rotational Properties of Some BMW-512 Transforms

The rotational probabilities of the shifts s_i in BMW, presented in Table 5.1, can be found by the method used to prove Lemmas 2.2-2.4.

Table 5.1.: Rotational probabilities of the functions s_i used in BMW

Function	Definition	Prob. log_2
$s_0(x)$	$SHR^1(x) \oplus SHL^3(x) \oplus ROTL^4(x) \oplus ROTL^{19}(x)$	-4
$s_1(x)$	$SHR^1(x) \oplus SHL^2(x) \oplus ROTL^8(x) \oplus ROTL^{23}(x)$	-4
$s_2(x)$	$SHR^2(x) \oplus SHL^1(x) \oplus ROTL^{12}(x) \oplus ROTL^{25}(x)$	-4
$s_3(x)$	$SHR^2(x) \oplus SHL^2(x) \oplus ROTL^{15}(x) \oplus ROTL^{29}(x)$	-4
$s_4(x)$	$SHR^1(x) \oplus x$	-2
$s_5(x)$	$SHR^2(x) \oplus x$	-4

Remark The rotational probabilities of the different shifts to the left and to the right of XH in f_2 of BMW are computed simply as 2^{-4} or 2^{-2} (the rotation amount is 2, see Lemma 2.2). A more careful analysis shows that indeed the

rotational probability of all 8 shifts of XH is 2^{-14} instead of 2^{-30} used in the attacks.

For any rotation amount, the rotational probability of addition is not lower than 2^{-2}, while the probabilities of XOR and rotation are 1 (see [58]). Further in our analysis, the rotation amount will be fixed to 1 or 2. For these cases, the rotational probability of addition is $2^{-1.41}$ and $2^{-1.68}$ respectively in case of 64-bit numbers.

5.3.2. Analysis of BMWv1-512

BMWv1 takes two 1024-bit inputs: the message M and the chaining value H and produces an 1024-bit output. The compression function is constructed using three functions f_0, f_1 and f_2. Next, we will give a short description of each function f_i. A complete specification of the functions can be found in [48]. All words in BMW-512 are 64-bit long. Assume that the message is $M = (M_0, \ldots, M_{15})$ and the chaining value is $H = (H_0, \ldots, H_{15})$.

The f_0 function takes as its input the pair: message M and chaining value H and produces an output $(Q_0, \ldots, Q_{15})$ as follows.

1. First intermediate words $W_0, \ldots, W_{15}$ are obtained as a bijective transformation of $M \oplus H$ defined below as

$$W_j = (M_{j_1} \oplus H_{j_1}) \star (M_{j_2} \oplus H_{j_2}) \star (M_{j_3} \oplus H_{j_3}) * (M_{j_4} \oplus H_{j_4}) \star (M_{j_5} \oplus H_{j_5}),$$

   where $\star \in \{+, -\}$, $j = 0, \ldots, 15$ and $j_1, j_2, \ldots, j_5 \in \{0, \ldots, 15\}$.

2. The words W_i undergo a bijective transformation and the output of f_0 are produced, i.e. $Q_j = s_j(W_j)$, where $s_j(x), j = 0, \ldots, 15$, are XORs of shifts and rotations of x (see Table 5.1).

The f_1 function takes the following pair as its input: message M and output of f_0, and produces words $(Q_{16}, \ldots, Q_{31})$ on its output. For $j = 16, 17$, Q_j are defined as:

$$Q_j = expand_1(j) = s_1(Q_{j-16}) + \ldots + s_0(Q_{j-1}) + AddElement(j - 16),$$

while for $j = 18, \ldots, 31$, they are defined as:

$$Q_j = expand_2(j) = Q_{j-16} + \ldots + s_5(Q_{j-1}) + AddElement(j - 16),$$

where $s_k(x)$ are the same functions as in f_0, and $AddElement(j) = M_j + M_{j+3} - M_{j+10} + K_{j+16}$. The constants K_j are obtained from the initial constant $C = 0x0555555555555555$ by multiplication, i.e. $K_j = j \cdot C$.

The last function (the f_2 one) produces 16 words of the new chaining value. As its input, it takes message M and $(Q_0, \ldots, Q_{31})$ (the outputs of f_0, f_1). First, it produces the words $XL = Q_{16} \oplus \ldots \oplus Q_{23}$ and $XH = XL \oplus Q_{24} \oplus \ldots \oplus Q_{31}$. Then, the first 8 words (out of 16) of the new chaining value are defined as[1]:

$$H_j = (SHL^{i_j}(XH) \oplus SHR^{k_j}(Q_{j+16}) \oplus M_j) + (XL \oplus Q_{j+24} \oplus Q_j),$$

where $j = 0, \ldots, 7$ and SHL^k, SHR^k are shifts to left and right by k bits.

We will build a rotational distinguisher for BMWv1 such that the input pairs of chaining values and message words will compose a rotational pair, but with some corrections. The output pairs of f_0 and f_1 will be rotational for all 1024 bits, while the output pairs of f_2 will be rotational for at least 384 bits.

Analysis of f_1. We will start from f_1 because this is the only function that applies additions of constants. Note that in general, we cannot create a rotational pair for constants since their values are fixed. To overcome this technical difficulty, either constant has to be rotational, i.e. $K_j = K_j \lll_r$, or the errors from the constants have to be canceled with some other errors. In our attack, we will use the fact that the constants are almost rotational, and we will use small errors, coming from other words, to make the outputs fully rotational. Recall that the outputs $Q_j, j = 16, \ldots, 31$ of f_1 are defined as[2] $Q_j = AddElement(j - 16) + s_1(Q_{j-16}) + \ldots + s_0(Q_{j-1})$. Let $T_j = s_1(Q_{j-16}) + \ldots + s_0(Q_{j-1})$ and $\tilde{T}_j = s_1(Q_{j-16} \lll_r) + \ldots + s_0(Q_{j-1} \lll_r)$. To obtain rotational outputs Q_j, we have to find an input message pair $(M, \tilde{M})$) for the following system of 16

[1] We will use only these 8 words in our attack. Therefore we omit the definition of the next 8 words.

[2] The case when $expand_2(j)$ is used can be analyzed similarly.

equations:

$$[M_j + M_{j+3} - M_{j+10} + K_{j+16} + T_{j+16}] \lll_r = \tilde{M}_j + \tilde{M}_{j+3} - \tilde{M}_{j+10} + K_{j+16} + \tilde{T}_{j+16}, \tag{5.13}$$

$j = 0, \ldots, 15$. If we take into account the distributive properties of addition and rotation, then with some probability (that will be estimated later) this system can be rewritten as:

$$\begin{aligned} M_j \lll_r + M_{j+3} \lll_r - M_{j+10} \lll_r + K_{j+16} \lll_r + T_{j+16} \lll_r = \\ = \tilde{M}_j + \tilde{M}_{j+3} - \tilde{M}_{j+10} + K_{j+16} + \tilde{T}_{j+16}. \end{aligned} \tag{5.14}$$

If we denote $M_j^{'} = M_j \lll_r - \tilde{M}_j$, then we will obtain the following system:

$$M_j^{'} + M_{j+3}^{'} - M_{j+10}^{'} = K_{j+16} - K_{j+16} \lll_r + \tilde{T}_{j+16} - T_{j+16} \lll_r, \tag{5.15}$$

for $j = 0, \ldots, 15$. When the amount of rotation equals 2, then the words $K_{j+16} - K_{j+16} \lll_2$, $j = 0, \ldots, 15$ have zeroes in all bytes except for the first and the last (the exact values are given in Table 5.2).

Table 5.2.: Rotational properties of the constants of f_1 in BMWv1 and BMWv2

	BMWv1			BMWv2		
i	K_i	$K_i \lll_2$	$K_i - K_i \lll_2$	K_i	$K_i \lll_2$	$K_i - K_i \lll_2$
16	5555...5550	5555...5541	0000...000f	5555...5550	5555...5541	0000...000f
17	5aaa...aaa5	6aaa...aa95	f000...0010	aaaa...aaa5	aaaa...aa96	0000...000f
18	5fff...fffa	7fff...ffe9	e000...0011	ffff...fffa	ffff...ffeb	0000...000f
19	6555...554f	9555...553d	d000...0012	5555...554f	5555...553d	0000...0012
20	6aaa...aaa4	aaaa...aa91	c000...0013	aaaa...aaa4	aaaa...aa92	0000...0012
21	6fff...fff9	bfff...ffe5	b000...0014	ffff...fff9	ffff...ffe7	0000...0012
22	7555...554e	d555...5539	a000...0015	5555...554e	5555...5539	0000...0015
23	7aaa...aaa3	eaaa...aa8d	9000...0016	aaaa...aaa3	aaaa...aa8e	0000...0015
24	7fff...fff8	ffff...ffe1	8000...0017	ffff...fff8	ffff...ffe3	0000...0015
25	8555...554d	1555...5536	7000...0017	5555...554d	5555...5535	0000...0018
26	8aaa...aaa2	2aaa...aa8a	6000...0018	aaaa...aaa2	aaaa...aa8a	0000...0018
27	8fff...fff7	3fff...ffde	5000...0019	ffff...fff7	ffff...ffdf	0000...0018
28	9555...554c	5555...5532	4000...001a	5555...554c	5555...5531	0000...001b
29	9aaa...aaa1	6aaa...aa86	3000...001b	aaaa...aaa1	aaaa...aa86	0000...001b
30	9fff...fff6	7fff...ffda	2000...001c	ffff...fff6	ffff...ffdb	0000...001b
31	a555...554b	9555...552e	1000...001d	5555...554b	5555...552d	0000...001e

On the other hand, for random $Q_j, j = 0, \ldots, 15$, the difference $\tilde{T}_{j+16} - T_{j+16} \lll_2$ takes the values $0x16, 0x17$ with probability $2^{-8.4}$ when $expand_1(j)$ is applied, and $2^{-5.6}$ when $expand_2(j)$ is applied (this result is obtained ex-

perimentally, with 2^{27} trials). Hence, we can assume that the constant terms of System (5.15), have only two non-zero bytes – the first (MSB) and the last (LSB). For specific values of these terms (different values can be obtained since $\tilde{T}_{j+16} - T_{j+16} \lll_2$ takes two values, and there are 16 equations, therefore one can get 2^{16} systems), the words $M_j^{'}$ of the solution also have only two non-zero bytes, i.e. $M_j^{'}$ can be represented as $M_j^{'} = msb_j \cdot 2^{56} + lsb_j$, where $msb_j, lsb_j < 256$. The exact values of these bytes are given in Table 5.3.

Table 5.3.: Constant terms and solutions for the systems in f_1 of BMWv1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\tilde{T}_{j+16} - T_{j+16} \lll_2$	16	16	16	16	16	16	16	16	17	16	16	16	16	17	16	17
LSB of R_j[a]	25	26	27	28	29	2a	2b	2c	2e	2d	2e	2f	30	32	32	34
MSB of R_j[a]	00	0f	0e	0d	0c	0b	0a	09	08	07	06	05	04	03	02	01
msb_j	59	18	e7	76	c5	d4	93	52	21	d0	cf	ce	ad	fc	0b	ea
lsb_j	28	28	29	29	30	32	31	28	25	22	2c	32	34	32	2f	2d

[a] $R_j = K_{j+16} - K_{j+16} \lll_2 + \tilde{T}_{j+16} - T_{j+16} \lll_2$

Once we have $M_j^{'}$, we can find the message pair. We choose the message words $M_j, \tilde{M}_j$ such that $\tilde{M}_j = M_j \lll_2 \oplus \delta_j$ (rotational with corrections δ_j). Since all $M_j^{'}$ were fixed by the system, we get the following equations:

$$M_j \lll_2 - M_j \lll_2 \oplus \delta_j = msb_j \cdot 2^{56} + lsb_j, j = 0, \ldots, 15. \tag{5.16}$$

We would like to find many solutions (that we will use later) for this system. To do that, we will fix the MSB of $M_j \lll_2$ and δ_j to msb_j and the LSB to lsb_j, i.e. $(M_j \lll_2)_{MSB} = (\delta_j)_{MSB} = msb_j, (M_j \lll_2)_{LSB} = (\delta_j)_{LSB} = lsb_j$. If we fix the middle 6 bytes of δ_j to 0, then the message words $M_j \lll_2 = msb_j \cdot 2^{56} + X_j \cdot 2^8 + lsb_j$, where $X_j < 2^{48}$, are solutions of (5.16). It is important to notice, that δ_j have only two non-zero bytes and therefore the input pairs of message words are rotational for 6 bytes. Hence, we can easily find $2^{16 \cdot 6 \cdot 8} = 2^{768}$ input rotational pairs of messages such that if the inputs $Q_0, \ldots, Q_{15}$ of f_1 are rotational, then the outputs $Q_{16}, \ldots, Q_{31}$ are rotational as well.

Let us estimate the total probability of obtaining these rotational outputs. First and foremost, let us find the probability that System (5.13) is equivalent to System (5.14). For one equation the probability (obtained experimentally)

is $2^{-3.8}$ and hence for the whole system it is 2^{-61}. Now, let us concentrate on the transformations in f_1. Since there are 2 applications of $expand_1(j)$ and 14 applications of $expand_2(j)$, the probability of obtaining the required differences $\tilde{T}_{j+16} - T_{j+16} \lll_2$ for all 16 outputs is $2^{-2 \cdot 8.4 - 14 \cdot 5.6} = 2^{-95.2}$. Therefore, the total rotational probability (obtained heuristically) of f_1 is $2^{-61-95.2} = 2^{-156.2}$.

Analysis of f_0. The function f_0 uses the words $(M_i \oplus H_i)$ as inputs. Since the message pair is $(M_i, M_i \lll_2 \oplus\delta_i)$, instead of taking simply rotational inputs for the chaining values H_i, we will also introduce corrections. To obtain a fully rotational input for f_0 we will take the chaining value pairs $(H_i, H_i \lll_2 \oplus\delta_i)$. Then the input pair for f_0 is $(M_i \oplus H_i, M_i \lll_2 \oplus\delta_i \oplus H_i \lll_2 \oplus\delta_i) = (M_i \oplus H_i, (M_i \oplus H_i) \lll_2)$, hence it is rotational. Now let us find the probability that the outputs $Q_0, \ldots, Q_{15}$ are also rotational. These words are produced in two phases:

1. the words $W_0, \ldots, W_{15}$ are generated as linear combinations of five terms of a type $M_i \oplus H_i$ or $-(M_i \oplus H_i)$,
2. each Q_i is obtained from W_i as $Q_i = s_i(W_i)$.

The rotational probabilities of the words W_i are given in Table 5.4. The theoretical basis for these numbers is provided by the Lemmas 5.1 ,5.2. Note that since we consider rotation amount $r = 2$ and the number of additions and subtractions in W_i is limited to 4, the counter j of the sums in the lemmas takes only value 0, hence the formulas for the probabilities can be significantly simplified for these specific values. The total probability of the first phase is $2^{-60.8}$. To compute the probability of phase 2 of f_0 we only have to find the rotational probabilities of functions s_i (see Table 5.1). There are 4, 3, 3, 3, 3 applications of s_0, s_1, s_2, s_3, s_4 respectively. Therefore, the probability of phase 2 is $2^{-4 \cdot 4 - 3 \cdot 4 - 3 \cdot 4 - 3 \cdot 4 - 3 \cdot 2} = 2^{-58}$, and hence, the total rotational probability of f_0 is $2^{-118.8}$.

Analysis of f_2. Function f_2 takes the M message and the words $Q_0, \ldots, Q_{31}$ (outputs of f_0, f_1) as an input, and produces the next chaining value. We can assume the words $Q_0, \ldots, Q_{31}$ to be rotational with some probability (the combined rotational probabilities of f_0 and f_1). The terms XL and XH are rotational with

Table 5.4.: Rotational probabilities of the words in f_0 of BMWv1 and BMWv2

	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7
log_2	-3.82	-1.68	-3.82	-1.68	-1.68	-1.68	-3.82	-10.01
	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}	W_{14}	W_{15}
log_2	-3.82	-1.68	-3.82	-3.82	-1.68	-10.01	-3.82	-3.82

probability 1 since they are produced as XORs of rotational words. We require rotational outputs from the shifts of XH and the shifts of $Q_j, j = 16, \ldots, 23$. The rotational probability of all the shifts of XH is $2^{-4-4-4-2-4-4-4-4} = 2^{-30}$, (see Remark in Section 5.3.1), while for the shifts of Q_j we pay $2^{-4-4-4-4-4-4-4-4} = 2^{-28}$ (see Table 5.1). Since the message pair words are rotational in 6 bytes, it follows that the words $SHL^{j_k}(XH) \oplus SHR^{j_l}(Q_j) \oplus M_j$ are also rotational in 6 bytes (all bytes except MSB and LSB). Let $P_j = SHL^{j_k}(XH) \oplus SHR^{j_l}(Q_j) \oplus M_j$ and $R_j = XL \oplus Q_{j+24} \oplus Q_j$. Then the chaining values are defined as $H_j = P_j + R_j$. Note that P_j is rotational for 6 bytes, and R_j is fully rotational. For the error of the new chaining value we have $(P_j + R_j) \lll_2 -[(P_j \lll_2 \oplus\delta_j) + R_j \lll_2]$ $^{2^{-1.68}}=$ $P_j \lll_2 + R_j \lll_2 -(P_j \lll_2 \oplus\delta_j) - R_j \lll_2 = P_j \lll_2 -(P_j \lll_2 \oplus\delta_j)$, hence the error can occur in the MSB and LSB only if there are no carries (with probability 2^{-1}) in the LSB. Therefore, for the rotational properties of the first 8 chaining values in 6 bytes we have to pay in total $2^{-8\cdot(1.68+1)}$. If we take into account the previous probabilities of the shifts, we will get the total rotational probability of f_2, which is $2^{-30-28-21.5} = 2^{-79.5}$. The output pair is rotational in at least $8 \cdot 6 = 48$ bytes, or 384 bits.

The Attack on the Full BMWv1.

The relations between the pairs of input message words and the chaining value words are fully fixed. For the first input, the message words M_j are chosen randomly, except their MSB and LSB which are fixed as explained above. The chaining values H_j are chosen randomly as well. Then, the message $\tilde{M}$ and the chaining value $\tilde{H}$ of the second input are defined as $\tilde{M}_j = M_j \lll_2 \oplus\delta_j, \tilde{H}_j = H_j \lll_2 \oplus\delta_j$. The probability that such an input pair will produce an output pair of the chaining values, rotational in 384 bits, is the combined probability

of f_0, f_1, f_2 which is $2^{-156.2-118.8-79.5} = 2^{-354.5}$. On the other hand, the same probability for a random function is 2^{-384}, hence BMWv1 can be distinguished from a random function.

Note that we can obtain non-random properties for the last 8 chaining values $H_8, \ldots, H_{15}$ as well[3]. We only have to take into account the terms $ROTL^{(j+1)}(H_k)$ which are rotational in 6 bytes. Also, the complexity of the whole attack can be reduced to $2^{223.5}$ compression function calls by using more advanced techniques as explained in the next Section.

5.4. Lower Complexity Attack on the Full BMWv1

Further in this dissertation, we will present an algorithm for finding the rotational output pair with a lower complexity.

1. Take random values for $A_0, \ldots, A_{15}$, where $A_j = M_j \oplus H_j$ and produce the outputs $Q_0, \ldots, Q_{15}$ of f_0. Assign $\tilde{A}_j = A_j \lll_2$ and produce another outputs $\tilde{Q}_0, \ldots, \tilde{Q}_{15}$. Check if the pairs $(Q_j, \tilde{Q}_j), j = 0, \ldots, 15$ are rotational. If they are not, repeat step 1.

2. Fix the MSB i LSB of $M_j \lll_2$ to msb_j and lsb_j, the middle 6 bytes to zero. Obtain the second message (by rotation and XOR of δ_i). Produce the first output pair of f_1, i.e. $(Q_{16}, \tilde{Q}_{16})$, and check if it is rotational. If not, go to step 1.

3. Assign random values to the middle 6 bytes of the message words $M_j \lll_2$ (the MSB and LSB are still fixed to the previous values) and obtain the second message (by rotation and XOR). Produce the other outputs $Q_{17}, \ldots, Q_{31}$ of $f_1(M_0, \ldots, M_{15}, Q_0, \ldots, Q_{15})$, and the outputs $\tilde{Q}_{17}, \ldots, \tilde{Q}_{31}$ of $f_1(M_0 \lll_2 \oplus\delta_0, \ldots, M_{15} \lll_2 \oplus\delta_{15}, \tilde{Q}_0, \ldots, \tilde{Q}_{15})$. If $(Q_j, \tilde{Q}_j), j = 17, \ldots, 31$, are not rotational then repeat step 3.

4. Produce the new pair of chaining values – outputs of f_2 and check if they are rotational in the 384 bits. If not, go to step 3.

[3]We omit the description since we already have an attack.

Let us estimate the total complexity of finding the output pair of rotational chaining values in 384 bits. For passing step 1, we have to try $2^{118.8}$ different values for $A_0, \ldots, A_{15}$. Step 2 calls step 1 around $2^{12.2}$ times: $2^{8.4}$ to get the proper difference $\tilde{T}_0 - T_0 \lll_2$, and $2^{3.8}$ times to get equivalence of (5.13) and (5.14) for $j = 16$. This is done because: 1) the difference $\tilde{T}_{16} - T_{16} \lll_2$ depends only on $Q_0, \ldots, Q_{15}$, 2) when the amount of rotation is fixed to 2, the equivalence of system (5.13) and (5.14) depends on the values of the 2 most significant bits of $M_0, M_3, M_{10}, \tilde{M}_0, \tilde{M}_3, \tilde{M}_{10}, T_{16}, \tilde{T}_{16}$ – they are all fixed or depend on the values of $Q_0, \ldots, Q_{15}$. On the hand, by varying the messages words, the values of $Q_{17}, \ldots, Q_{31}$ can vary, hence we can obtain rotational pairs for these outputs of f_1 by simply taking different message words. As a result, we have to to take $2^{156.2-12.2} = 2^{144}$ different messages to pass step 3. Finally, step 4 calls step 3 around $2^{79.5}$ times. Hence the total complexity of finding the rotational pair of output chaining values equals $2^{118.8+12.2} + 2^{144+79.5} \approx 2^{223.5}$ computations.

5.4.1. Analysis of Modified Version of BMWv2-512

The compression function BMWv2 is similar to the one of BMWv1, but a few tweaks are introduced by the designers. We will only describe the differences between these two functions. The first tweak is in f_0, where the words Q_j are produced as $Q_j = s_j(W_j) + H_{j+1}$. The second tweak is in f_1. Now this function takes the chaining value H as an additional input. The tweak of f_1 is in the AddElement function, which is defined as follows

$$AddElement(j) = (M_j \lll_{j+1} + M_{j+3} \lll_{j+4} - M_{j+10} \lll_{j+11} + K_{j+16}) \oplus H_{j+7}.$$

We attack a modified version of BMWv2, denoted as BMWv2$_C$, where the above round constants K_{j+16} are obtained by multiplying the round indexes $(j + 16)$ by the constant $C = 0x5555555555555555$. In the original version the value of the constant is $C = 0x0555555555555555$.

The Attack on BMWv2$_C$.

We will take a different approach for producing rotational pairs in BMWv2$_C$ although the analysis uses the results of the previous section. The input pairs of messages and chaining values will be fully rotational, while the output chaining values will be rotational in the first 8 words (512 bits).

Let us fix a random message $(M_0, \ldots, M_{15})$ and a chaining value $(H_0, \ldots, H_{15})$ for the first input of f_0, and the pair $(M_0 \lll_2, \ldots, M_{15} \lll_2)$, $(H_0 \lll_2, \ldots, H_{15} \lll_2)$ for the second. Since f_0 in BMWv2 differs from f_0 in BMWv1 only in the extra additions of H_j in BMWv2, in order to find the rotational probability of f_0, we only have to consider these 16 additions. Thus, the probability of the rotational output pair for f_0 is $2^{-118.8-16\cdot 1.68} = 2^{-145.7}$.

Now, let us focus on f_1. When the constant C is fixed to $0x5555555555555555$ then the values of the differences $K_{j+16} - K_{j+16} \lll_2$ are only one byte (see Table 5.2).

On the other hand, all of these differences (rotational errors of the constants) can be canceled since both the addition and the rotation are not fully distributive. For example, for some x, y the following holds $(x + y) \lll_2 = x \lll_2 + y \lll_2 + 1$. When more terms are added, these errors can be bigger, i.e. for some $x_1, \ldots, x_k$ it holds $(x_1 + \ldots + x_k) \lll_2 = x_1 \lll_2 + \ldots + x_k \lll_2 + e_+$, where $e_+ \in \{1, 2, \ldots\}$. We have found that all differences $K_{j+16} - K_{j+16} \lll_2$ can be canceled with these errors coming from the additions/rotations. When the input pairs of words $Q_0, \ldots, Q_{15}, M_0, \ldots, M_{15}, H_0, \ldots, H_{15}$ are rotational, then the probabilities of rotational output pairs for $expand_1(j), j = 16, 17$ and $expand_2(j), j = 18 \ldots, 31$ (obtained experimentally) are given in the Table 5.5.

Table 5.5.: Rotational properties of the words in f_1 (without the shifts) in BMWv1

	Q_{16}	Q_{17}	Q_{18}	Q_{19}	Q_{20}	Q_{21}	Q_{22}	Q_{23}
log_2	-2.37	-2.38	-3.93	-3.95	-3.97	-3.97	-3.98	-4.00
	Q_{24}	Q_{25}	Q_{26}	Q_{27}	Q_{28}	Q_{29}	Q_{30}	Q_{31}
log_2	-4.00	-4.02	-4.03	-4.03	-4.03	-4.03	-4.03	-4.04

The total probability of obtaining all the 16 rotational outputs from these

transformations, i.e. the rotational probability of f_1 is around 2^{-61}.

Finally, let us analyze f_2. We require rotational outputs only for the first 8 new chaining values, i.e. $H_0, \ldots, H_7$. Similarly as for BMWv1, the probability can be estimated simply by counting the number of shifts and additions required for producing these 8 values, i.e. the probability is $2^{-30-28-8\cdot 1.68} = 2^{-71.5}$. Note that now there are no corrections in the message words, hence the 512-bit output is fully rotational.

For the whole BMWv2$_C$, the probability that rotational inputs of messages and chaining values will produce rotational outputs in the first 8 words is equal to $2^{-145.7-61-71.5} = 2^{-278.2}$. On the other hand, a rotational input in a random function will produce a rotational output in 8 words (512 bits) with probability 2^{-512}. The probability of our distinguishers most likely can be raised most likely if the message modification technique is applied. Then the first phase of f_0 (probability 2^{-61}) can be passed for free.

The low attack complexity allows us to launch rotational distinguishers for the 384-bit version of BMWv2$_C$ as well. Note that, increasing the number of applications of $expand_1(j)$ (which is considered to be stronger) from 2 to all 16 does not stop the attack because the rotational probability of $expand_1(j)$ is higher than the one of $expand_2(j)$. Also, the probability of rotational output pairs, does not seems to change significantly, when the order of s_j in f_0 and f_1 is changed.

5.5. Rotational Analysis of SIMD-512

In this section we will present rotational distinguishers for the compression function of SIMD-512 [80], later revised in [81]. For simplicity, we refer to SIMD-512 as SIMD for the remainder of this chapter. SIMD passed through to the second round of the SHA-3 competition [98]. Differential distinguishers for the full round 1 compression function of SIMD were presented in [89]. Recently, a high probability distinguisher, exploiting the symmetric properties of round 1,2 SIMD has been found [79]. Our rotational distinguishers work for the compression functions of the both (round) versions of SIMD, hence we will not make distinc-

tions between the two versions. First, we shall launch a rotational distinguisher for 24 rounds of the modified version of SIMD, where the message expansion is linearized. Next, we will present a rotational distinguisher for the original, but reduced to 12 rounds, compression function.

The general idea of the construction of SIMD is based on a modified Davies-Meyer construction with a multiple-pipe Feistel-like block cipher and affine-code based on a Reed-Solomon code for message expansion. In each iteration of the compression function, one message block M of 1024 bits is processed by expanding it in 8192 bits. The expansion is done using an affine code that applies the following three operations: a number theoretic transformation (NTT), a concatenated code and a permutation. The expanded message is used as a key for Feistel-like 32-round block cipher that transforms 32 state words initialized with XOR-ed message M and chaining value H, and the result is finally transformed to 32 words (H') with similar Feistel-like 4-round block cipher with H as the key (hence in total there are 36 rounds).

The three message expansion operations are defined in the following way:

1. *NTT* – a 1024-bit input message block, represented as $x = (x_0, x_1, \ldots, x_{127}) \in (\mathbb{Z}_{2^8})^{128}$, is mapped into $y = (y_0, y_1, \ldots, y_{255}) \in (\mathbb{F}_{257})^{256}$ and

$$y_i = \sum_{j=0}^{127} x_j \beta^{ij} + \beta^{255i}, i = 0, \ldots, 255, \tag{5.17}$$

   where $\beta = 41$ (β is a 256th root of unity in $\mathbb{F}_{257}$),

2. *Concatenated Code* – a pair $(x, y) \in (\mathbb{F}_{257})^2$ is mapped into word $I_C(x) + 2^{16} I_C(y)$, where $I_C \colon \mathbb{F}_{257} \to \mathbb{Z}_{2^{16}}$, and $I_C(x) = C \cdot \tilde{x}$, where C takes one of the two values 185 or 233, and according to the reference implementation [81] $\tilde{x} = x$, when $x \leq 128$, and $\tilde{x} = x - 257$, otherwise.

3. *Permutation* – message words $W_j^{(i)} = Z_j^{(P(i))}$ are expended, where $Z_j^{(i)} = I_m(y_{t_i}, y_{t_j})$, where $m \in \{185, 233\}, t_i, t_j \in [0, 255]$.

Observation: The expanded message words in the first 8 rounds of SIMD are obtained using the mapping I_{185} only.

The main part of one round (out of 36) of the compression function consists of 8 step functions (pipes) processed in parallel. One step function updates four 32-bit words of state, further denoted as $A_j^{(i)}, B_j^{(i)}, C_j^{(i)}, D_j^{(i)}$ for $j = 0, 1, \ldots, 7$ and rounds $i = 0, 1, \ldots, 35$ in the following way:

$$A_j^{(i)} = (D_j^{(i-1)} + W_j^{(i-1)} + \phi^{(i)}(A_j^{(i-1)}, B_j^{(i-1)}, C_j^{(i-1)})) \lll_{s^{(i)}} + A_{p^{(i)}(j)}^{(i-1)} \lll_{r^{(i)}},$$
$$B_j^{(i)} = A_j^{(i-1)} \lll_{r^{(i)}}, \qquad C_j^{(i)} = B_j^{(i-1)}, \qquad D_j^{(i)} = C_j^{(i-1)},$$

where $\phi^{(i)}$ is either bit-wise MAJ function or bit-wise IF function depending on the round, $p^{(i)}$ are round dependant permutations, $r^{(i)}, s^{(i)}$ are round dependant rotations amounts and $W_j^{(i)}$ are expanded message blocks or in the case of the last four rounds, the old chaining values. For a detailed description of the SIMD compression function see [81].

In the sequel, first we will find the rotational probability of the Feistel transformation, and then we will obtain rotational probabilities for the message expansion. We will analyze two versions of the message expansion with and without additions of the constants in the NTT part of the message expansion.

5.5.1. Analysis of the Feistel of SIMD

In the Feistel transformation of SIMD, there are only three different operations, namely, additions, rotations, and Boolean functions. More importantly, the Feistel does not apply any constant. To find the rotational probability of one round of the Feistel transformation, we only have to count the number of additions, which is 24: 8 pipes, each with 3 additions. In the rotational pairs we will use the rotation amount of 1, hence for each addition we will have the probability $2^{-1.41}$. Therefore, the rotational probability of one round of the Feistel transformation is $2^{-1.41 \cdot 3 \cdot 8} = 2^{-33.8}$.

When the expanded message words $W_j^{(i)}$, used as a key for the Feistel, equal zero, then the number of additions per pipe drops from 3 to 2. Therefore, one round has 16 additions (instead of 24), and the rotational probability of one round becomes $2^{-1.41 \cdot 16} = 2^{-22.6}$.

5.5.2. Analysis of Round-reduced Linearized SIMD

First, we will consider SIMD with linearized message expansion, which can be achieved by changing the constant β^{255} in the NTT transformation to 0. This modification allows us to fix message block m to zero which changes the expanded message words $W_j^{(i)}$ also to zeros. In other words, on zero input, all the three message expansion parts, produce zero output. Also, value 0 is rotational with respect to any amount (including our target amount of 1).

A rotational distinguisher for this modified version of SIMD works as follows. The message input in both pairs is fixed to 0, i,e, the message pair (M_1, M_2) is $(0^{1024}, 0^{1024})$. The chaining value input pair (H_1, H_2) is simply rotational with amount 1, i.e. the input pair is $(H_1, H_1 \lll_1)$, where the values of the words $H_1^l, l = 0, \ldots, 31$ of the vector H_1 are chosen arbitrarily, and the values of the words for the second vector H_2 are fixed as $H_2^l = H_1^l \lll_1, l = 0, \ldots, 31$. Then the input to the Feistel $(M \oplus H)$ is also rotational. The expanded message words are all zeroes, therefore the rotational probability of each round except for the last four (where the input is the chaining value) is $2^{-22.6}$. As a result, we can launch a rotational distinguisher on 20 round-reduced linearized SIMD: the rotational probability of the first 16 rounds is $2^{-22.6 \cdot 16}$, and of the last 4 rounds (the feedforward rounds) is $2^{-33.8 \cdot 4}$, hence the total rotational probability of the distinguisher is 2^{-497}. On the other hand, in a random function, the probability that a rotational input will produce rotational output in 1024 bits is 2^{-1024}.

The previous distinguisher can be extended for 4 additional rounds. The first round can be passed for free, i.e. the rotational probability of the first round can be equal to 1, if we fix the values of the most and last significant bits of each chaining value word[4] – this is in line with the message modification techniques. Three additional rounds can be passed for free, if we take into account that we can produce output pairs rotational in only 512 bits (instead of all 1024 bits). In the last 4 (feedforward) rounds, we produce rotational pairs only in the first 2 of these 4 rounds, i.e. $A_j^{(k+1)}, A_j^{(k+2)}, B_j^{(k+1)}, B_j^{(k+2)}, C_j^{(k+1)}, C_j^{(k+2)}, D_j^{(k+1)}, D_j^{(k+2)}$

[4]The pair is still rotational, only some bits are fixed.

are rotational. Then, due to the property of the Feistel, $C_j^{(k+4)} = A_j^{(k+2)} \lll_{r^{(k+2)}}$, $D_j^{(k+4)} = B_j^{(k+2)}$, hence the words C, D of the last round are rotational. Therefore, the output pair is rotational in 512 bits. The total probability of the 24 round distinguisher is $2^{0+19\cdot(-22.6)+2\cdot(-33.8)+2\cdot 0} = 2^{-497}$. The same probability in a random function is 2^{-512}.

5.5.3. Analysis of Round-reduced SIMD

The designers of SIMD were aware of a potentially dangerous property of the linear NTT function (see [80] page 8) and intentionally introduced addition of some powers of β^{255} in the definition of NTT making the message expansion affine. This modification no more allows us to obtain any expanded message words all equal to 0. However, for the first 8 rounds, 128 expanded message words are constrained by a system of 128 equations in 128 variables (each of them refers to consecutive 8 bits of input message) and by forcing these 128 words to zero we obtain a system (see Equations (5.17), where $y_i = 0$ for $i = 0, 1, \ldots, 127$), which has only one solution in $(\mathbb{F}_{257})^{128}$ that can be mapped to $(\mathbb{Z}_{2^8})^{128}$. The existence of such a solution allows us to build a distinguisher for 12 rounds of SIMD (8 rounds with words from message expansion and 4 rounds with the chaining value words).

The expanded message words $W_j^{(i)}$ are zeroes but the original message words $x_0, x_1, \ldots, x_{127}$ are not zeroes. We require the input pair $(M_1^l \oplus H_1^l, M_2^l \oplus H_2^l)$ to the Feistel transformation to be rotational, i.e. $M_2^l \oplus H_2^l = (M_1^l \oplus H_1^l) \lll_1$. Since we take the same message input in the pair (because this input will produce expanded message words equal to zero, hence rotational), i.e. $M_1^l = M_2^l$, then for the pair of chaining values, we will obtain $H_2^l = H_1^l \lll_1 \oplus(M_1^l \lll_1 \oplus M_1^l)$. Note that now the chaining values are rotational only in 514 bits (out of 1024) because the sum of the hamming weights of the words $M^l \lll_1 \oplus M^l$ is 510. After 8 rounds of the Feistel, where $W_j^{(i)}$ are zeroes, all the intermediate state words $A_j^{(8)}, B_j^{(8)}, C_j^{(8)}, D_j^{(8)}$ are rotational with probability $2^{8\cdot(-22.6)} = 2^{-181}$. In the ninth round, the inputs $W_j^{(i)}$ are the chaining values (which are rotational in

roughly half of the bits). Let us find the rotational error $e_{A_j^{(9)}}$ of $A_j^{(9)}$ (the terms H_1, H_2 in the following formulae denote the inputs $W_j^{(i)}$ in the first and in the second instance of the rotational pair), which is

$$\begin{aligned} e_{A_j^{(9)}} = [(D_j^{(8)} + \phi_j^{(9)} + H_1) \lll_{s^{(9)}} + A^{(8)}_{p^{(9)}(j)} \lll_{r^{(9)}}] \lll_1 - \\ - [(D_j^{(8)} \lll_1 + \phi_j^{(9)} \lll_1 + H_2) \lll_{s^{(9)}} + (A^{(8)}_{p^{(9)}(j)} \lll_1) \lll_{r^{(9)}}], \end{aligned}$$

where $\phi_j^{(9)} = \phi^{(9)}(A_j^{(8)}, B_j^{(8)}, C_j^{(8)})$). With the probability $2^{2\cdot(-1.41)}$ (rotational probability of two additions), the error can be rewritten as

$$\begin{aligned} e_{A_j^{(9)}} = ((D_j^{(8)} + \phi_j^{(9)}) + H_1) \lll_{(s^{(9)}+1)} + (A^{(8)}_{p^{(9)}(j)} \lll_{r^{(9)}}) \lll_1 - \\ - [((D_j^{(8)} + \phi_j^{(9)}) \lll_1 + H_2) \lll_{s^{(9)}} + (A^{(8)}_{p^{(9)}(j)} \lll_1) \lll_{r^{(9)}}] = \\ = ((D_j^{(8)} + \phi_j^{(9)}) + H_1) \lll_{(s^{(9)}+1)} - ((D_j^{(8)} + \phi_j^{(9)}) \lll_1 + H_2) \lll_{s^{(9)}} . \end{aligned}$$

The rotational probability of addition for any rotation amount is at least 2^{-2}. Hence, with probability $2^{2\cdot(-2)}$, the error can be rewritten as

$$\begin{aligned} e_{A_j^{(9)}} = (D_j^{(8)} + \phi_j^{(9)}) \lll_{(s^{(9)}+1)} + H_1 \lll_{(s^{(9)}+1)} - (D_j^{(8)} + \phi_j^{(9)}) \lll_{(s^{(9)}+1)} - \\ - H_2 \lll_{s^{(9)}} = H_1 \lll_{(s^{(9)}+1)} - H_2 \lll_{s^{(9)}} . \end{aligned}$$

If we take into account that $H_2 = H_1 \lll_1 \oplus K$ (the constant K depends on the message words), then for the rotational error we will get that $e_{A_j^{(9)}} = X - X \oplus K^j$, where $X = H_1 \lll_{(s^{(9)}+1)}$, and $K^j = (M_1^j \lll_1 \oplus M_1^j) \lll_{s^{(9)}}$. For a random X, the value of $e_{A_j^{(9)}}$ is K^j with the probability $2^{-hamming(K^j)}$ – X has to have 1 in the bits where K^j has 1. Since we control the bits of the chaining value, we can fix the required bits to 1, and get $e_{A_j^{(9)}} = K^j$ with probability 1. Note that we have to fix roughly half of the bits of the chaining value but we still have enough freedom (in 512 bits) to launch an attack. As the result, in a single pipe, the probability that $e_{A_j^{(9)}} = K^j$ is $2^{2\cdot(-1.41)+2\cdot(-2)} = 2^{-6.8}$.

The values of new chaining words $D_j^{(12)}$ are defined as $D_j^{(12)} = A_j^{(9)} \lll_{r^{(9)}}$. Let $(D_j^{(12)}, \tilde{D}_j^{(12)})$ be the output pair of words, obtained with a rotational input pair

specified as above. Then in all 8 pipes

$$(D_j^{(12)} \ggg_{r^{(9)}}) \lll_1 -(\tilde{D}_j^{(12)} \ggg_{r^{(9)}}) = e_{A_j^{(9)}} = K^j$$

with probability $2^{-181-6.8\cdot 8} = 2^{-236}$. The same property in a random function holds with probability 2^{-256} (eight 32-bit words).

5.6. Shift Distinguishers on Shabal

Shabal [25] is a hash function submitted to the SHA-3 competition and it passed to the second round. There are various published distinguishers on the permutation of Shabal [2, 4, 65, 99] and among them is a rotational distinguisher proposed by Van Assche in [1].

The compression function of Shabal is based on a keyed permutation $\mathcal{P}_{M,C}(A, B)$, where $B, C, M \in \{0,1\}^{l_m}, A \in \{0,1\}^{l_a}$ (recommended values for the parameters are $l_m = 512, l_a = 384$). The inputs can be seen as arrays $A_i, B_j, C_j, M_j, i = 0, \ldots, 11, j = 0, \ldots, 15$ of 32-bit words. The permutation $\mathcal{P}$ outputs the new values of A_i, B_i and is defined as:

for $i = 0$ **to** 15 **do**

$B_i \leftarrow B_i \lll_{17}$

end for

for $j = 0$ **to** $\mathbf{p} - 1$ **do**

for $i = 0$ **to** 15 **do**

$A_{i+16j \bmod \mathbf{r}} \leftarrow \mathcal{U}(A_{i+16j \bmod \mathbf{r}} \oplus C_{8-i \bmod 16} \oplus \mathcal{V}(A_{i-1+16j \bmod \mathbf{r}} \lll_{15}))$

$A_{i+16j \bmod \mathbf{r}} \leftarrow A_{i+16j \bmod \mathbf{r}} \oplus M_i$

$A_{i+16j \bmod \mathbf{r}} \leftarrow A_{i+16j \bmod \mathbf{r}} \oplus B_{i+13 \bmod 16} \oplus (B_{i+9 \bmod 16} \wedge \overline{B_{i+6 \bmod 16}})$

$B_i \leftarrow B_i \lll_1 \oplus \overline{A_{i+16j \bmod \mathbf{r}}}$

end for

end for

for $j = 0$ **to** 35 **do**

$A_{j \bmod r} \leftarrow A_{j \bmod \mathbf{r}} + C_{j+3 \bmod 16}$

end for

where $\mathcal{U}(x) = 3x$, $\mathcal{V}(x) = 5x$ and the recommended values for the parameters are $(\mathbf{r}, \mathbf{p}) = (12, 3)$.

The following lemmas specify the shift probabilities of the transforms used in $\mathcal{P}$.

Lemma 5.4. *Given n-bit words x, y and positive integers r, s, then*

$$\Pr((x + y) \ll_s = x \ll_s + y \ll_s) = 1,$$
$$\Pr((x \oplus y) \ll_s = x \ll_s \oplus y \ll_s) = 1,$$
$$\Pr(\mathcal{U}(x \ll_s) = \mathcal{U}(x) \ll_s)) = 1,$$
$$\Pr(\mathcal{V}(x \ll_s) = \mathcal{V}(x) \ll_s)) = 1,$$
$$\Pr((x \wedge \bar{y}) \ll_s = (x \ll_s) \wedge \overline{y \ll_s}) = 1,$$
$$\Pr((x \lll_r) \ll_s = (x \ll_s) \lll_r) = 2^{-2t},$$

where $t = min(r, s, n - r, n - s)$.

Lemma 5.5 (Updates of B_i)**.** *Given n-bit words x, y, then*

$$\Pr((x \lll_1 \oplus \overline{y}) \ll_1 = ((x \ll_1) \lll_1 \oplus \overline{y \ll_1})) = 2^{-2}.$$

The proofs for the lemmas are analogous to the ones for rotational property.

Lemma 5.6 (Multiplication)**.** *Given a pair of n-bit words x, y and positive integers r, s, then*

$$\Pr(x \ll_r \cdot\ y \ll_s = (x \cdot y) \ll_{(r+s)}) = 1.$$

Let us consider that each n-bit word z can be represented as a concatenation of two words a, b, i.e. $z = a||b$, where a are the s most significant bits of z, and b the $n - s$ least significant bits of z. Let $x = a_1||b_1, y = a_2||b_2$ be such a representation

of x, y. With $C_{u,v}$ we denote the carry from $u + v$. Then for addition we have:

$$(x + y) \ll_s = (a_1||b_1 + a_2||b_2) \ll_s = (a_1 + a_2 + C_{b_1,b_2}||b_1 + b_2) \ll_s = b_1 + b_2||\underbrace{0 \ldots 0}_{s},$$

$$x \ll_s + y \ll_s = (a_1||b_1) \ll_s + (a_2||b_2) \ll_s = b_1||\underbrace{0 \ldots 0}_{s} + b_2||\underbrace{0 \ldots 0}_{s} = b_1 + b_2||\underbrace{0 \ldots 0}_{s}.$$

A similar reasoning can be applied to XOR as well. Let now $x = x_1 2^{n-r} + x_2$ and $y = y_1 2^{n-s} + y_2$, then

$$x \ll_r \cdot\, y \ll_s \equiv x_2 y_2 2^{r+s} \pmod{2^n},$$
$$(x \cdot y) \ll_{r+s} = (x_1 y_1 2^{2n-r-s} + x_1 y_2 2^{n-r} + x_2 y_1 2^{n-s} + x_2 y_2) \ll_{r+s} \equiv x_2 y_2 2^{r+s} \pmod{2^n},$$

that proves $x \ll_r \cdot\, y \ll_s \equiv (x \cdot y) \ll_{r+s} \pmod{2^n}$.

Let us now provide a proof for $\mathcal{U}$ (the proof for $\mathcal{V}$ is analogous).

$$\mathcal{U}(x) \ll_s = (3x) \ll_s = (2x + x) \ll_s = (2x) \ll_s + x \ll_s = x \ll_{s+1} + x \ll_s,$$
$$\mathcal{U}(x \ll_s) = 3(x \ll_s) = 2(x \ll_s) + x \ll_s = x \ll_{s+1} + x \ll_s .$$

For the function $x \wedge \bar{y}$:

$$(x \wedge \bar{y}) \ll_s = (x \ll_s) \wedge (\overline{y} \ll_s) = (a_1||b_1 \ll_s) \wedge (\overline{a_2||b_2} \ll_s) = b_1||\underbrace{0 \ldots 0}_{s} \wedge \bar{b}_2||\underbrace{0 \ldots 0}_{s} =$$

$$= b_1 \wedge \bar{b}_2||\underbrace{0 \ldots 0}_{s},$$

$$(x \ll_s \wedge \overline{y \ll_s}) = ((a_1||b_1) \ll_s \wedge \overline{(a_2||b_2) \ll_s} = b_1||\underbrace{0 \ldots 0}_{s} \wedge \bar{b}_2||\underbrace{1 \ldots 1}_{s} = b_1 \wedge \bar{b}_2||\underbrace{0 \ldots 0}_{s}.$$

For the proof of the shift probability of rotation compare proof of rotational probability of shift. Finally, let us prove the Lemma 5.5, i.e. let find the shift probability (with $s = 1$) of the function $f(x, y) = x \lll_1 \oplus \bar{y}$. We assume that

$x = x_{n-1}x_{n-2}\dots x_0, y = y_{n-1}y_{n-2}\dots y_0$, where x_i, y_i are bits.

$$\begin{aligned} f(x,y) \ll_1 &= (x \lll_1 \oplus \bar{y}) \ll_1 = (x \lll_1) \ll_1 \oplus \bar{y} \ll_1 = \\ &= (x_{n-2}\dots x_0 x_{n-1}) \ll_1 \oplus \overline{y_{n-2}\dots y_1 y_0}0 = \\ &= x_{n-3}\dots x_0 x_{n-1}0 \oplus \bar{y}_{n-2}\dots\bar{y}_1\bar{y}_0 0, \\ f(x \ll_1, y \ll_1) &= (x \ll_1) \lll_1 \oplus \overline{y \ll_1} = \\ &= (x_{n-2}\dots x_0 0) \lll_1 \oplus \overline{y_{n-2}\dots y_1 y_0 0} = \\ &= x_{n-3}\dots x_0 0 x_{n-2} \oplus \bar{y}_{n-2}\dots\bar{y}_1\bar{y}_0 1. \end{aligned}$$

Therefore $f(x,y) \ll_1 = f(x \ll_1, y \ll_1) \iff x_{n-1} = 0$ and $x_{n-2} = 1$, hence the probability is 2^{-2}.

From the lemmas we can see that among all the transformations in $\mathcal{P}$ it is only the rotation and the update function for B_i that have a probability less than 1 and therefore to find the shift probability of the whole $\mathcal{P}$, i.e. to find the probability that $\mathcal{P}(A,B,C,M) \ll_s = \mathcal{P}(A \ll_s, B \ll_s, C \ll_s, M \ll_s)$ for a random A, B, C, M, we only have to count the number of rotations and updates of B_i used in $\mathcal{P}$. Their number is $16 + 3 \cdot 16 + 3 \cdot 16 = 112$. Therefore, if we fix the shift amount s to 1, then the shift probability of one rotation and update of B_i is 2^{-2}, and we obtain a shift distinguisher for the whole $\mathcal{P}$ with a probability $2^{-2 \cdot 112} = 2^{-224}$. If we manipulate the exact values of the inputs B_i we can pass the beginning 16 rotations (with $r = 17$)) and the 16 rotations in the updates of B_i (with $r = 1$) of the first round, i.e. $j = 0$. Then the total probability of the shift distinguisher will drop to 2^{-160}. Furthermore, we can as well control the input values of C_i and pass for free the rotations in $\mathcal{V}$ (with $r = 15$) in the first round. In this case, the total probability of our shift distinguisher will equal 2^{-128}.

5.7. Summary

The invention of the rotational analysis has created a new avenue in cryptanalysis and has invigorated the evaluation process of the SHA-3 competition.

Table 5.6.: Summary of distinguishers for BMW and Shabal.

Compression Function	Distinguisher	Complexity	Reference
BMWv1	differential	1	[50]
BMWv1	rotational	$2^{223.5}$	Section 5.4
BMWv2	differential	2^{19}	[3]
BMWv2	rotational	$2^{278.2}$	Section 5.4.1
Shabal	differential	1	[2]
Shabal	differential	1	[65]
Shabal	differential	2	[4]
Shabal	differential	2^{21}	[99]
Shabal	shift	2^{128}	Section 5.6
Shabal	rotational	2^{159}	[1]
SIMD	symmetric	1	[23]
SIMD	differential	2^{-398}	[112]
SIMD	rotational	2^{-236}	Section 5.5.3
SIMD	rotational	2^{-497}	Section 5.5.2

However, the theory and the application of the rotational analysis and distinguishers is largely unexplored. In this thesis, we have made a step towards extending the area of applying the rotational analysis to primitives that besides ARX, may have subtractions, shifts, Boolean functions and a combination of additions and subtractions. We have derived the rotational probabilities from these operations. Our findings have allowed us to launch attacks on the modified and original compression functions of BMW-512 and SIMD-512 which were in the second round of the SHA-3 competition. Furthermore, we have proposed a new type of attack, a shift analysis. We have found distinguishers based on the shift analysis for the permutation of the Shabal hash function.

Our distinguishers do not contradict the security claims of all of the hash functions. Our findings demonstrate that some parts of the analyzed designs exhibit non random behaviour which might be exploited to mount successfull attack on full versions of them.

6. Conclusions

The study presented in this thesis has covered only a small part of wide subjects of design and analysis of cryptographic hash functions. We have discussed security of available hashing *modes* applied to block ciphers and illustrated our results with examples of attacks on instances of such constructions. Our results do not only have application in hash function analysis but have also improved some of the attacks on block ciphers: Crypton, Hierocrypt-3, `IDEA`, SAFER++ and Square in the open-key model. We have extended application of rotational distinguishers to larger class of primitives besides additions, rotations and XORs. Our findings have allowed to mount rotational attacks on two round 2 SHA-3 candidates: BMW and SIMD. We have proposed a new kind of shift distinguisher and applied it to third SHA-3 candidate: Shabal. The theoretical results we have obtained for: random permutations in terms of differential propagation and multi additions/subtractions in case of rotational analysis have demonstrated the beauty of mathematics applied to cryptanalysis of hash functions and block ciphers.

The SHA-3 competition concluded in 2012 and Keccak has been selected as a successful winner out of five final candidates: BLAKE, Grøstl, JH, Keccak or Skein. However, it is not the only output of the Secure Hash Standard completion. The 4-year selection process has enthused the international cryptographic community to search for better ways of estimating security level of cryptographic hash functions. New attacks have been devised and new "unwanted" properties of cryptographic primitives have been discovered. Without a doubt, the new SHA-3 attract more attention from the research community, which will bring more interesting cryptanalytic results and maybe answers to above questions.

6.1. Contributions

In Chapter 3, we investigated the problem of building hash functions from block ciphers and have discussed the results for commonly used *modes* of designing compression functions. We have considered both the known-key and chosen-key models. Specifically, we have analyzed the collision resistance of compression functions based on single block ciphers as well as the double-block compression functions, when specific differential trails for the underlying ciphers can be built. We have shown that we can build open-key differential distinguishers for some well known block ciphers: Crypton, Hierocrypt-3, SAFER++ and Square. As far as we know, the attack on SAFER++ is the first rebound attack with standard differentials. For these ciphers, we have shown that when the attack model is switched from secret-key to open-key, the number of rounds that can be attacked increases. In order to demonstrate efficiency of proposed distinguishers, we have provided formal proof of a lower bound for finding a differential pair that follows some truncated differential in case of a random permutation. Our hash analysis has shown that block ciphers used as underlying primitives in considered *modes* should be analyzed also in the open-key model in order to prevent possible collision attacks.

In Chapter 4, we studied the security of the `IDEA` block cipher when applied to following block cipher hashing *modes*: Davies-Meyer, Hirose, Abreast-DM, Tandem-DM, Peyrin *et al.*(II) and MJH-Double. We have shown free-start-collision and preimage attacks for all the *modes* and semi-free start collisions attacks for Peyrin *et al.*(II) and MJH-Double *modes*. Finally, we have constructed collision search attack for the whole hash function in Davies-Meyer *mode*. We have exploited weak-keys of `IDEA`, in particular we have used the fact that the key 0 in `IDEA` is extremely weak, actually rendering the whole encryption process to a T-function, already known as dangerous for building a hash function [96]. All our attacks were based on weak-keys utilization, but in contrary to the secret key model where the goal of the attacker is to exhibit the biggest weak-key class possible, in the hashing *mode* the goal is to find and exploit the weakest of all

keys. While weak-keys are already known to be dangerous for block cipher-based hash functions, our method used a novel and non-trivial almost half-involution property for `IDEA`. We have found that even strengthened versions of the cipher with any number of rounds can be attacked with about the same complexities. Our analysis has once again demonstrated in a similar way as in Chapter 3 that even if this cipher is considered secure in the secret-key model, its security remains an open problem in a hashing *mode*. 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.

In Chapter 5, we investigated recently proposed rotational analysis and rotational distinguishers. Our findings allowed to extend rotational analysis (applied to additions, rotations and XORs) to wider range of primitives, that is: subtractions, shifts, Boolean functions and combination of additions and subtractions. We have derived the rotational probabilities from these operations. In particular we have presented exact formulas for calculation of rotational probability for multi additions and multi subtractions. We have also applied S-function framework for rotational analysis with corrections and provided an algorithmic way to calculate exact probabilities for fixed corrections. We have applied our findings to rotational analysis of compression functions of SHA-3 candidates: BMW and SIMD. We have found that round 1 BMW [48] is susceptible to the rotational analysis. Also, the round 2 BMW [49], with a slightly altered constant, can be attacked using this method. For SIMD, we have presented various rotational distinguishers on round-reduced original and modified versions. We have also proposed new shift distinguisher and applied it to the permutation of the SHA-3 candidate Shabal. The invention of rotational analysis has created a new avenue in cryptanalysis and has invigorated the evaluation process of the SHA-3 competition. However, the theory and application of rotational analysis and distinguishers is largely unexplored. Even though our analysis did not provide best attacks on analyzed functions: BMW, SIMD or Shabal, we have improved the rotational framework and have shown possible ways of its further development.

6.2. Design Guidelines

Our analysis has demonstrated many successful attacks on a variety of block ciphers in hashing *modes* and three SHA-3 candidates. The results were exploiting "weaknesses" of analyzed primitives and might be wrongly considered as negative. However, they are in a sense warning signs for designers of new constructions. The tools applied to cryptanalysis in the thesis, that is: differential distinguishers, rotational distinguishers, almost half-involution property, T/S-functions, etc. can be as well used for testing of hash function security margin. Of course such an approach will not replace mathematical proof, but might detect flaws in new designs and grow designer confidence in a chosen strategy.

What is important, our findings introduced in Chapter 3 and Chapter 4 demonstrate that block ciphers applied in hashing *modes* do not only have to be secure in the secret-key model but should as well be analyzed in the open-key model. In particular, one should strictly avoid block ciphers for which any weak-key exists for compression function construction.

Our rotational analysis of BMW and SIMD, and shift analysis of Shabal in Chapter 5 proved to be effective to some extent, even though the preservation of rotational/shift pairs does not seem to be probable for such complicated designs. What we have observed is that "good" constants might even improve our attacks. In the light of SHA-3 competition requirements, the observed property of the compression functions is not (pseudo) random, hence it might raise some concern, even if it is not clear how the property is extended to the entire function. On the other hand, the tools we have developed for this kind of analysis can help determine "bad" for attacker constants and alignment of internal operations of build primitive.

6.3. Open Problems and Future Research Directions

The open-key model and its variants, the known-key and the chosen-key models are still highly researched. Even if the attacks on block ciphers demonstrated in these models are not considered as a real threat for this primitive, the application

of block ciphers in the construction of compression functions will result in the designers willing to consider these models seriously. The open problem in this field of analysis is how the gap between the two models: the secret-key and the open-key can be closed. That is finding the relation for attacks designed in both cases.

The second problem we would like to present is the existence of attacks on hash functions or block ciphers based on available highly efficient distinguishers. For many cryptographic primitives there exist high probability distinguishers, for example in case of Shabal there are two differential distinuishers with probability 1 on its keyed permutation [2, 65]. However, there is no straightforward way to extend them to any kind of non-distinguishing attack on the whole hash function. The problem has been researched to some extent, for example designers of Shabal introduced in their analysis [26] a distinguishing oracle, and demonstrated that the advantage of an attacker interacting with the oracle equipped with effective distinguisher for compression function is negligible.

Another interesting field of research is the application of the S-function framework for rotational analysis with corrections. The obtained results allow to establish exact probability for fixed corrections. However, finding optimal configuration of input/output corrections in polynomial time is an open problem. An efficient algorithm could drastically improve rotational analysis. This is to say that the results would have an immediate application to differential probabilities of modular addition, and might as well lead to improvements to differential characteristics of ARX designs.

A. Proofs of Rotational Analysis Lemmas

The Lemma 2.1 was proven in [37].

Shifts. We will provide evidence for shifts to the right. The case for shifts to the left can be proven similarly. Recall that for n-bit word x, rotation is defined as

$$x \lll_r = (x \ll_r) \oplus (x \gg_{(n-r)}).$$

Then,

$$(x \gg_s) \lll_r = [(x \gg_s) \ll_r] \oplus [(x \gg_s) \gg_{(n-r)}],$$
$$(x \lll_r) \gg_s = [(x \ll_r) \oplus (x \gg_{(n-r)})] \gg_s = [(x \ll_r) \gg_s] \oplus [(x \gg_{(n-r)}) \gg_s]$$

Hence, we have to find the probability that $L \equiv (x \gg_s) \ll_r = (x \ll_r) \gg_s \equiv R$ holds. Let $x = x_{n-1} \dots x_0$, where x_i are bits of x. When $s \geq r$, then,

$$L \equiv (x \gg_s) \ll_r = \underbrace{0 \dots 0}_{s-r} x_{n-1} \dots x_s \underbrace{0 \dots 0}_{r},$$

$$R \equiv (x \ll_r) \gg_s = \underbrace{0 \dots 0}_{s} x_{n-1-r} \dots x_{s-r}$$

If $0 \leq r \leq s \leq \frac{n}{2}$, then $L = R$ with probability 2^{-2r}: the bits $x_{n-1}, \dots, x_{n-r}$ (r bits in total) and the bits $x_{s-1}, \dots, x_{s-r}$ (again r bits) have to be zeros, and $r = \min(r, s, n-r, n-s)$.

If $0 \leq r \leq \frac{n}{2} < s \leq n$, then $L = R$ with probability $2^{-2\min(r,n-s)}$: if $r \leq n-s$ then we have an analogous situation as above mentioned, whereas if $n - s < r$

then the bits $x_{n-1}, \ldots, x_{n-s}$ ($n-s$ bits in total) and the bits $x_{n-1-r}, \ldots, x_{s-r}$ ($n-s$ bits in total) have to be zeros, and $n-s = \min(r, s, n-r, n-s)$.

If $\frac{n}{2} < r \leq s \leq n$, then $n-1-r < \frac{n}{2}$ and therefore, the probability that $L = R$ is $2^{-2(n-s)}$: the bits $x_{n-1}, \ldots, x_s$ (total $n-s$ bits) and the bits $x_{n-1-r} \ldots x_{s-r}$ (total $n-s$ bits) have to be zeros, and $n-s = \min(r, s, n-r, n-s)$.

Obviously, the case when $s < r$ can be reduced to the above case. □

Boolean function. Let x be the n-bit input, i.e. $x = x_{n-1}x_{n-2}\ldots x_0$. Then $f(x) = f(x_{n-1})\ldots f(x_0)$. Therefore, we have $f(x \lll_r) = f(x_{n-1-r})\ldots f(x_{n-r}) = [f(x_{n-1})\ldots f(x_0)] \lll_r = f(x) \lll_r$. □

Multiplication. Let $x = x_1 2^{n-r} + x_2$ and $y = y_1 2^{n-s} + y_2$, then

$$(x \cdot y) \lll_{r+s} = (x_1 y_1 2^{2n-r-s} + x_1 y_2 2^{n-r} + x_2 y_1 2^{n-s} + x_2 y_2) \lll_{r+s},$$
$$x \lll_r \cdot\, y \lll_s = (x_2 2^r + x_1) \cdot (y_2 2^s + y_1) = x_2 y_2 2^{r+s} + x_2 y_1 2^r + x_1 y_2 2^s + x_1 y_1.$$

If $x_1 = 0$ and $y_1 = 0$ then the above equations simplify to

$$(x \cdot y) \lll_{r+s} = (x_2 y_2) \lll_{r+s},$$
$$x \lll_r \cdot\, y \lll_s = x_2 y_2 2^{r+s},$$

respectively. Because $(x_2 y_2) \lll_{r+s} = x_2 y_2 2^{r+s}$ with probability 2^{-r-s}, as also $x_1 = 0$ and $y_1 = 0$ with the same probability, the claimed lower bound is obtained.

□

B. mCrypton

mCrypton is a 64-bit block cipher with three possible key sizes 64-bit, 96-bit and 128-bit, proposed by Lim and Korkishko [83]. It is compact version of Crypton [82] intended for resource-constrained hardware implementations. A round transformation of mCrypton consists of an S-box layer γ, linear diffusion layer composed of two transforms π and τ and subkey addition σ. The encryption involves 12 rounds applied to an internal state of the cipher.

The internal state of mCrypton can be represented as 4×4 matrix of nibbles (4-bit words) a_{ij} or alternatively as 4 columns $A_c[0], \ldots, A_c[3]$ or 4 rows $A_r[0], \ldots, A_r[3]$, where $A_c[i] = (a_{0i}, \ldots, a_{3i})^T$ and $A_r[i] = (a_{i0}, \ldots, a_{i3})$.

Table B.1.: mCrypton internal state representation:

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

(a) nibble-wise

$A_c[0]$	$A_c[1]$	$A_c[2]$	$A_c[3]$

(b) column-wise

$A_r[0]$
$A_r[1]$
$A_r[2]$
$A_r[3]$

(c) row-wise

The encryption process consists of number of rounds build of four types of transformations:

1. nonlinear substitution γ (S-box),
2. linear π,
3. linear τ,
4. key addition σ,

described in more detail in the following sections. The i-th round transformation ρ_i is a composition of above, that is $\rho_i = \sigma_{K_i} \circ \tau \circ \pi \circ \gamma$, where K_i is an i-th

round key obtained with a key schedule. The key schedule described in B.5 is a modified version of the Crypton key schedule (compare [82]).

B.1. Nonlinear Substitution γ

This is the only nonlinear part of the cipher. It applies combination of four nonlinear S-boxes $\gamma_0, \gamma_1, \gamma_2, \gamma_3$ to 16-nibble state presented in Table B.3, where γ_i is defined in Table B.5.

Table B.3.: mCrypton nonlinear substitution γ

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

$\longrightarrow$

$\gamma_0(a_{00})$	$\gamma_1(a_{01})$	$\gamma_2(a_{02})$	$\gamma_3(a_{03})$
$\gamma_1(a_{10})$	$\gamma_2(a_{11})$	$\gamma_3(a_{12})$	$\gamma_0(a_{13})$
$\gamma_2(a_{20})$	$\gamma_3(a_{21})$	$\gamma_0(a_{22})$	$\gamma_1(a_{23})$
$\gamma_3(a_{30})$	$\gamma_0(a_{31})$	$\gamma_1(a_{32})$	$\gamma_2(a_{33})$

Table B.5.: Definition of mCrypton S-boxes $\gamma_0, \ldots, \gamma_3$.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
γ_0	4	f	3	8	d	a	c	0	b	5	7	e	2	6	1	9
γ_1	1	c	7	a	6	d	5	3	f	b	2	0	8	4	9	e
γ_2	7	e	c	2	0	9	d	a	3	f	5	8	6	4	b	1
γ_3	b	0	a	7	d	6	4	2	c	e	3	9	1	5	f	8

B.2. Column-Wise Bit Permutation π

π is a linear transformation of columns $A_c[i]$. It is defined in the following way:

$$\pi_i(A_c[i]) = (b_{0i}, b_{1i}, b_{2i}, b_{3i})^T$$

where each b_{ij} is calculated as follows:

$$b_{ji} = \oplus_{k=0}^{3}(m_{(i+j+k)mod4} \wedge a_{ik})$$

where $\oplus$ and $\wedge$ are bit-wise logical operations XOR and AND, respectively. The constants used to express b_{ij} are: $m_0 = 1110_2, m_1 = 1101_2, m_2 = 1011_2, m_3 = 0111_2$ represented as binary numbers.

B.3. Column-To-Row Transposition τ

This is also a linear transformation of mCrypton state, which realizes transposition of the state, that is i-th column is transformed to i-th row.

Table B.6.: mCrypton column-to-row transposition τ

$A_c[0]$	$A_c[1]$	$A_c[2]$	$A_c[3]$

$\longrightarrow$

$A_c[0]^T$
$A_c[1]^T$
$A_c[2]^T$
$A_c[3]^T$

B.4. Key Addition σ

Let $K_i = (K_i[0], K_i[1], K_i[2], K_i[3])$ be 64-bit i-th round key, each $K_i[j]$ consists of 4 nibbles. Key addition σ_{K_i} is a row-wise state transformation that XORs j-th row with j-th element of K_i, that is $K_i[j]$.

Table B.8.: mCrypton key addition σ

$A_r[0]$
$A_r[1]$
$A_r[2]$
$A_r[3]$

$\longrightarrow$

$A_r[0] \oplus K_i[0]$
$A_r[1] \oplus K_i[1]$
$A_r[2] \oplus K_i[2]$
$A_r[3] \oplus K_i[3]$

B.5. Altered Key Schedule

The altered key schedule adopted from Crypton is defined in the following way:

Let K be a 128-bit encryption key and $K = k_0 \dots k_{31}$ where each k_i is four-bit nibble for $i = 0, \dots, 31$. At first two temporal values U and V are derived from K so that $U[i] = k_{8i}k_{8i+2}k_{8i+4}k_{8i+6}$ and $V[i] = k_{8i+1}k_{8i+3}k_{8i+5}k_{8i+7}$ for $i = 0, 1, 2, 3$. Next for $U' = \gamma(U)$ and $V' = \gamma(V)$ the eight expanded keys are evaluated as:

$$E[i] = \bigoplus_{j \neq i} U'[j] \qquad E[i+4] = \bigoplus_{j \neq i} V'[j]$$

for $i = 0, 1, 2, 3$ with use of which the 13 subkeys for each encryption round are generated according to the following procedure:

1. for the first and the second round:

$$K_0[i] = E[i] \oplus C[0] \oplus MC_i \qquad K_1 = E[i+4] \oplus C[1] \oplus MC_i$$

for $i = 0, 1, 2, 3$,

2. for the remaining eleven rounds ($r = 2, \ldots, 12$) two steps are executed alternatively:

   a) for r even:

$$\{E[0], E[1], E[2], E[3]\} \leftarrow \{E[1]^{\ll 12}, E[2]^{\ll 8}, E[3]^{\ll_b 3}, E[0]^{\ll_b 3}\},$$
$$K_r[i] = E[i] \oplus C[r] \oplus MC_i,$$

   b) for r odd:

$$\{E[4], E[5], E[6], E[7]\} \leftarrow \{E[7]^{\ll_b 1}, E[4]^{\ll_b 1}, E[3]^{\ll 4}, E[0]^{\ll 8}\},$$
$$K_r[i] = E[i+4] \oplus C[r] \oplus MC_i,$$

for $i = 0, 1, 2, 3$,

where $C[0] = \texttt{0xf53a}, C[k] = C[k-1] + \texttt{0xf372} \bmod 2^{16}$ for $k = 1, \ldots, 12$, $MC_0 = \texttt{0xacac}, MC_k = MC_{k-1}^{\ll_b 1}$ for $i = 0, \ldots, 3$ and $\ll_b a$ represents bit-left-rotation by a bits within each four-bit nibble.

B.6. Encryption

The encryption procedure E takes as an input: 64-bit plaintext P and 128-bit secret key K. The key is expanded to 13 round keys $K_0, \ldots, K_{12}$ with use of key schedule described in Section B.5. The encryption function E_K for given K is defined as follows:

$$E_K = \phi \circ \rho_{12} \circ \cdots \circ \rho_1 \circ \sigma_{K_0}$$

where i-th round function ρ_i is defined as:

$$\rho_i = \sigma_{K_i} \circ \tau \circ \pi \circ \gamma.$$

The last part of encryption process the ϕ function is a linear transformation and $\phi = \tau \circ \pi \circ \tau$.

Bibliography

[1] Gilles Van Assche. A rotational distinguisher on Shabal's keyed permutation and its impact on the security proofs. Available online at `http://gva.noekeon.org/papers/ShabalRotation.pdf`, 2010.

[2] Jean-Philippe Aumasson. On the pseudorandomness of Shabal's keyed permutation. Available online at `http://131002.net/data/papers/Aum09.pdf`, 2009.

[3] Jean-Philippe Aumasson. Practical distinguisher for the compression function of Blue Midnight Wish. Available online at `http://131002.net/data/papers/Aum10.pdf`, 2010.

[4] Jean-Philippe Aumasson, Atefeh Mashatan, and Willi Meier. More on Shabal's permutation. OFFICIAL COMMENT, 2009.

[5] Eyüp Serdar Ayaz and Ali Aydin Selçuk. Improved DST Cryptanalysis of IDEA. *LNCS* **4356**, 2006, pp. 1–14.

[6] Eric Bach. Toward A Theory of Pollard's Rho Method. *Inf. Comput.*, **90**, 1991, pp. 139–155.

[7] Paulo S. L. M. Barreto, Vincent Rijmen, Jorge Nakahara Jr., Bart Preneel, Joos Vandewalle, and Hae Yong Kim. Improved SQUARE Attacks against Reduced-Round HIEROCRYPT. *LNCS* **2355**, 2002, pp. 165–173.

[8] Mihir Bellare and Phillip Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. *LNCS* **4004**, 2006, pp. 409–426.

[9] K. Bentahar, D. Page, J.H. Silverman, M.-J.O. Saarinen, and N.P. Smart. LASH. *Second Cryptographic Hash Workshop*, 2006.

[10] Daniel J. Bernstein. Salsa20. Technical Report 2005/025. In *eSTREAM, ECRYPT Stream Cipher Project (2005),* `http://cr.yp.to/snuffle.html`, 2005.

[11] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. *LNCS* **4965**, 2008, pp. 181–197.

[12] Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. IACR Cryptology ePrint Archive, Report 2007/278, 2007.

[13] Eli Biham, Orr Dunkelman, and Nathan Keller. New Cryptanalytic Results on IDEA. *LNCS* **4284**, 2006, pp. 412–427.

[14] Eli Biham, Orr Dunkelman, and Nathan Keller. A New Attack on 6-Round IDEA. *LNCS* **4593**, 2007, pp. 211–224.

[15] Eli Biham, Orr Dunkelman, Nathan Keller, and Adi Shamir. New Data-Efficient Attacks on Reduced-Round IDEA. IACR Cryptology ePrint Archive, Report 2011/417, 2011.

[16] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *LNCS* **537**, 1990, pp. 2–21.

[17] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, **4**, 1991, pp. 3–72.

[18] Alex Biryukov, Christophe De Cannière, and Gustaf Dellkrantz. Cryptanalysis of SAFER++. *LNCS* **2729**, 2003, pp. 195–211.

[19] Alex Biryukov, Jorge Nakahara Jr., Bart Preneel, and Joos Vandewalle. New Weak-Key Classes of IDEA. *LNCS* **2513**, 2002, pp. 315–326.

[20] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. *LNCS* **5677**, 2009, pp. 231–249.

[21] Alex Biryukov and Ivica Nikolić. A New Security Analysis of AES-128. *CRYPTO 2009 rump session*, 2009.

[22] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. *LNCS* **2442**, 2002, pp. 320–335.

[23] Charles Bouillaguet, Pierre-Alain Fouque, and Gaëtan Leurent. Security analysis of SIMD. *LNCS* **6544**, 2011, pp. 351–368.

[24] Bruno O. Brachtl, Don Coppersmith, Myrna M. Hyden, Stephen M. Matyas Jr, Carl H. W. Meyer, Jonathan Oseas, Shaiy Pilpel, and Michael Schilling. Data authentication using modification detection codes based on a public one way encryption function. US Patent no. 4,908,861. Assigned to IBM. Filed August 28, 1987, March 13, 1990.

[25] Emmanuel Bresson, Anne Canteaut, Benoît Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-François Misarsky, Marìa Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet, and Marion Videau. Shabal, a Submission to NIST's Cryptographic Hash Algorithm Competition. Submission to NIST, 2008.

[26] Emmanuel Bresson, Anne Canteaut, Benoît Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-François Misarsky, Marìa Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet, and Marion Videau. Indifferentiability with Distinguishers: Why Shabal Does Not Require Ideal Ciphers. IACR Cryptology ePrint Archive, Report 2009/199, 2009.

[27] Christophe De Cannière, Nicky Mouha, Vesselin Velichkov, and Bart Preneel. The Additive Differential Probability of ARX, 2011. *LNCS* **6733**, 2011, pp. 342–358.

[28] Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. *LNCS* **1462**, 1998, pp. 56–71.

[29] Donghoon Chang. Near-Collision Attack and Collision-Attack on Double Block Length Compression Functions based on the Block Cipher IDEA. IACR Cryptology ePrint Archive, Report 2006/478, 2006.

[30] Scott Contini, Krystian Matusiewicz, Ron Steinfeld, Josef Pieprzyk, Jian Guo, San Ling, and Huaxiong Wang. Cryptanalysis of LASH. *LNCS* **5086**, 2008, pp. 207-223.

[31] Scott Contini, Arjen K. Lenstra, and Ron Steinfeld. VSH, an Efficient and Provable Collision-Resistant Hash Function. *LNCS* **4004**, 2006, pp. 165–182.

[32] Toshiba Corporation. Specification of Hierocrypt-3. *submitted to the First Open NESSIE Workshop*, 13-14 November,Leuven, Belgium 2000.

[33] Joan Daemen, René Govaerts, and Joos Vandewalle. Weak Keys for IDEA. *LNCS* **773**, 1994, pp. 224–231.

[34] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. *LNCS* **1267**, 1997, pp. 149–165.

[35] Joan Daemen and Vincent Rijmen. The Wide Trail Design Strategy. *LNCS* **2260**, 2001, pp. 222–238.

[36] Joan Daemen and Vincent Rijmen. Understanding Two-Round Differentials in AES. *LNCS* **4116**, 2006, pp. 78–94.

[37] Magnus Daum. *Cryptanalysis of Hash Functions of the MD4-Family.* PhD thesis, Ruhr-Universität Bochum, May 2005.

[38] Magnus Daum. Narrow T-Functions. *LNCS* **3557**, 2005, pp. 50–67.

[39] Hüseyin Demirci, Ali Aydin Selçuk, and Erkan Türe. A New Meet-in-the-Middle Attack on the IDEA Block Cipher. *LNCS* **3006**, 2004, pp. 117–129.

[40] Federal Register. Federal Register / Vol. 72, No. 212. Government Printing Office, 2007.

[41] FIPS PUB 180. SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES, 1993.

[42] FIPS PUB 180-1. SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES, 1995.

[43] FIPS PUB 180-2. SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES, 2001.

[44] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. On the Security of Tandem-DM. *LNCS* **5665**, 2009, pp. 84–103.

[45] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Security of Cyclic Double Block Length Hash Functions including Abreast-DM. IACR Cryptology ePrint Archive, Report 2009/261, 2009.

[46] Henri Gilbert and Thomas Peyrin. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. *LNCS* **6147**, 2010, pp. 365–383.

[47] M. Girault and M. Campana. A generalized birthday attack. *Lecture Notes in Computer Science on Advances in Cryptology-EUROCRYPT'88*, 1988, pp. 129–156.

[48] Danilo Gligoroski, Vlastimil Klima, Svein Johan Knapskog, Mohamed El-Hadedy, Jørn Amundsen, and Stig Frode Mjølsnes. Cryptographic Hash Function BLUE MIDNIGHT WISH. *Submission to NIST (Round 1)*, 2008. Available online at `http://people.item.ntnu.no/~danilog/Hash/BMW/Supporting_Documentation/BlueMidnightWishDocumentation.pdf`.

[49] Danilo Gligoroski, Vlastimil Klima, Svein Johan Knapskog, Mohamed El-Hadedy, Jørn Amundsen, and Stig Frode Mjølsnes. Cryptographic Hash Function BLUE MIDNIGHT WISH. *Submission to NIST (Round 2)*, 2009. Available online at `http://people.item.ntnu.`

no/~danilog/Hash/BMW-SecondRound/Supporting_Documentation/BlueMidnightWishDocumentation.pdf.

[50] Jian Guo and Søren S. Thomsen. Distinguishers for the Compression Function of Blue Midnight Wish with Probability 1, 2010. Available online at http://www2.mat.dtu.dk/people/S.Thomsen/bmw/bmw-distinguishers.pdf.

[51] Shai Halevi and Silvio Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. *LNCS* **1109**, 1996, pp. 201–215.

[52] Philip Hawkes. Differential-Linear Weak Key Classes of IDEA. In *EUROCRYPT*, pp. 112–126, 1998.

[53] Shoichi Hirose. Provably Secure Double-Block-Length Hash Functions in a Black-Box Model. *LNCS* **3506**, 2004, pp. 330–342.

[54] Shoichi Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. *LNCS* **4047**, 2006, pp. 210–225.

[55] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. *LNCS* **3152**, 2004, pp. 306–316.

[56] John Kelsey, Bruce Schneier, and David Wagner. Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. *LNCS* **1109**, 1996, pp. 237–251.

[57] Dmitry Khovratovich, Gaëtan Leurent, and Christian Rechberger. Narrow-Bicliques: Cryptanalysis of Full IDEA. *LNCS* **7237**, 2012, pp. 392–410.

[58] Dmitry Khovratovich and Ivica Nikolić. Rotational Cryptanalysis of ARX. *LNCS* **6147**, 2010, pp. 333–346.

[59] Dmitry Khovratovich, Ivica Nikolić, and Christian Rechberger. Rotational Rebound Attacks on Reduced Skein. *LNCS* **6477**, 2010, pp. 1–19.

[60] Jongsung Kim, Seokhie Hong, Sangjin Lee, Jung Hwan Song, and Hyungjin Yang. Truncated Differential Attacks on 8-Round CRYPTON. *LNCS* **2971**, 2003, pp. 446–456.

[61] Alexander Klimov and Adi Shamir. Cryptographic Applications of T-Functions. *LNCS* **3006**, 2004, pp. 248–261.

[62] Alexander Klimov and Adi Shamir. New Cryptographic Primitives Based on Multiword T-Functions. *LNCS* **2005**, 2004, pp. 1–15.

[63] Alexander Klimov and Adi Shamir. New Applications of T-Functions in Block Ciphers and Hash Functions. *LNCS* **3557**, 2005, pp. 18–31.

[64] Lars R. Knudsen. Truncated and Higher Order Differentials. *LNCS* **1008**, 1994, pp. 196–211.

[65] Lars R. Knudsen, Krystian Matusiewicz, and Søren S. Thomsen. Observations on the Shabal keyed permutation. OFFICIAL COMMENT, 2009. Available online at `http://www.mat.dtu.dk/people/S.Thomsen/shabal/shabal.pdf`.

[66] Lars R. Knudsen, Florian Mendel, Christian Rechberger, and Søren S. Thomsen. Cryptanalysis of MDC-2. *LNCS* **5479**, 2009, pp. 106–120.

[67] Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. *LNCS* **4833**, 2007, pp. 315–324.

[68] Bonwook Koo, Yongjin Yeom, and Junghwan Song. Related-Key Boomerang Attack on Block Cipher SQUARE. IACR Cryptology ePrint Archive, Report 2010/073, 2010.

[69] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997.

[70] Xuejia Lai. On the Design and Security of Block Ciphers. Hartung-Gorre Verlag, Konstanz, 1992.

[71] Xuejia Lai and James L. Massey. A Proposal for a New Block Encryption Standard. In *EUROCRYPT*, 1990, pp. 389–404.

[72] Xuejia Lai and James L. Massey. Hash Function Based on Block Ciphers. *LNCS* **658**, 1992, pp. 55–70.

[73] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. *LNCS* **5912**, 2009, pp. 126–143.

[74] Jooyoung Lee and Daesung Kwon. The Security of Abreast-DM in the Ideal Cipher Model. IACR Cryptology ePrint Archive, Report 2009/225, 2009.

[75] Jooyoung Lee and Martijn Stam. MJH: A Faster Alternative to MDC-2. *LNCS* **6558**, 2011, pp. 213–236.

[76] Jooyoung Lee, Martijn Stam, and John P. Steinberger. The Collision Security of Tandem-DM in the Ideal Cipher Model. *LNCS* **6841**, 2011, pp. 561–577.

[77] Gaëtan Leurent. Analysis of Differential Attacks in ARX Constructions. *LNCS* **7658**, 2012, pp. 226–243.

[78] Gaëtan Leurent. Quantum Preimage and Collision Attacks on CubeHash. IACR Cryptology ePrint Archive, Report 2010/506, 2010.

[79] Gaëtan Leurent. Symmetric Distinguishers on SIMD, 2010. Presented at the rump session of ECRYPT2 retreat, Paris.

[80] Gaëtan Leurent, Charles Bouillaguet, and Pierre-Alain Fouque. SIMD Is a Message Digest. Submission to NIST (Round 1), 2008.

[81] Gaëtan Leurent, Charles Bouillaguet, and Pierre-Alain Fouque. SIMD Is a Message Digest. Submission to NIST (Round 2), 2009.

[82] Chae Hoon Lim. A Revised Version of Crypton - Crypton V1.0. *LNCS* **1636**, 1999, pp. 31–45.

[83] Chae Hoon Lim and Tymur Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. *LNCS* **3786**, 2005, pp. 243–258.

[84] Helger Lipmaa and Shiho Moriai. Efficient Algorithms for Computing Differential Properties of Addition. *LNCS* **2355**, 2002, pp. 336–350.

[85] Stefan Lucks. Design Principles for Iterated Hash Functions. IACR Cryptology ePrint Archive, Report 2004/253, 2004.

[86] James Massey, Gurgen Khachatrian, and Melsik Kuregian. Nomination of SAFER++ as Candidate Algorithm for the New European Schemes for Signatures, Integrity, and Encryption (NESSIE). *First Open NESSIE Workshop*, November, 2000.

[87] Mitsuru Matsui. Linear Cryptoanalysis Method for DES Cipher. *LNCS* **765**, 1993, pp. 386–397.

[88] Krystian Matusiewicz and Josef Pieprzyk. Finding Good Differential Patterns for Attacks on SHA-1. *LNCS* **3969**, 2005, pp. 164–177.

[89] Florian Mendel and Tomislav Nad. A Distinguisher for the Compression Function of SIMD-512. *LNCS* **5922**, 2009, pp. 219–232.

[90] Florian Mendel, Thomas Peyrin, Christian Rechberger, and Martin Schläffer. Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. *LNCS* **5867**, 2009, pp. 16-35.

[91] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. *LNCS* **5665**, 2009, pp. 260–276.

[92] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[93] Ralph C. Merkle. A certified digital signature. *LNCS* **435**, 1989, pp. 218–238.

[94] Marine Minier, Raphael C.-W. Phan, and Benjamin Pousse. Distinguishers for Ciphers and Known Key Attack against Rijndael with Large Blocks. *LNCS* **5580**, 2009, pp. 60–76.

[95] Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The differential analysis of S-functions. *LNCS* **6544**, 2011, pp. 36–56.

[96] Frédéric Muller and Thomas Peyrin. Cryptanalysis of T-Function-Based Hash Functions. *LNCS* **4296**, 2006, pp. 267–285.

[97] Mridul Nandi and Souradyuti Paul. Speeding Up the Wide-Pipe: Secure and Fast Hashing. *LNCS* **6498**, 2010, pp. 144–162.

[98] National Institute of Standards and Technology. Cryptographic Hash Algorithm Competition. `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.

[99] Peter Novotney. Distinguisher for Shabal's Permutation Function. IACR Cryptology ePrint Archive, Report 2010/398, 2010.

[100] Thomas Peyrin, Henri Gilbert, Frédéric Muller, and Matthew J. B. Robshaw. Combining Compression Functions and Block Cipher-Based Hash Functions. *LNCS* **4284**, 2006, pp. 315–331.

[101] J.M. Pollard. Monte Carlo Methods for Index Computation mod *p*. *Mathematics of Computation*, **32**, 1978, pp. 918–924.

[102] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. *LNCS* **773**, 1994, pp. 368–378.

[103] François-Xavier Standaert, Gilles Piret, Neil Gershenfeld, and Jean-Jacques Quisquater. SEA: A Scalable Encryption Algorithm for Small Embedded Applications. *LNCS* **3928**, 2006, pp. 222–236.

[104] E. Teske. On random walks for Pollard's Rho method. *Math. Comput.*, **70**, 2001, pp. 809–825.

[105] E. Teske. Computing discrete logarithms with the parallelized kangaroo method. *Discrete Appl. Math.*, **130**, 2003, pp. 61–82.

[106] Søren S. Thomsen. Pseudo-cryptanalysis of Blue Midnight Wish, 2009. Available online at `http://www.mat.dtu.dk/people/S.Thomsen/bmw/bmw-pseudo.pdf`.

[107] Paul C. van Oorschot and Michael J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, **12**, 1999, pp. 1–28.

[108] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. IACR Cryptology ePrint Archive, Report 2004/199, 2004.

[109] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. *LNCS* **3621**, 2005, pp. 17–36.

[110] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. *LNCS* **3494**, 2005, pp. 19–35.

[111] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. *LNCS* **3621**, 2005, pp. 1–16.

[112] Hongbo Yu and Xiaoyun Wang. Cryptanalysis of the Compression Function of SIMD. *LNCS* **6812**, 2011, pp. 157–171.

[113] G. Yuval. How to swindle Rabin. *Cryptologia*, **3**, 1979, pp. 187–190.